UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

IMPLEMENTAR UN SERVICIO WEB QUE PERMITA CREAR MASHUPS PÚBLICOS O PRIVADOS PARA SER USADOS A TRAVÉS DE LA MENSAJERÍA DE TEXTO CELULAR

TRABAJO ESPECIAL DE GRADO

Presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR Kuster Chimienti, Willy Alberto
Rauseo Kafrouni, Orlando Anibal

PROFESOR GUIA Rodolfo Campos

FECHA Caracas, 19 de Febrero de 2009

ÍNDICE GENERAL

AGRADECIMIENTOS V

SINOPSIS VI

INTRODUCCIÓN 1

CAPÍTULO I. PROBLEMA 4

I.1 PLANTEAMIENTO DEL PROBLEMA 4
I.2 OBJETIVO GENERAL 5
I.3 OBJETIVOS ESPECÍFICOS 5
I.4 JUSTIFICACIÓN 5
I.5 ALCANCE 6
I.6 LIMITACIONES 7

CAPÍTULO II. MARCO TEÓRICO 9

II.1 WEB 2.0 9
II.1.1 RSS (REALLY SIMPLE SYNDICATION) 9
II.1.2 XML (EXTENSIBLE MARKUP LANGUAGE) 9
II.1.3 HTML (HYPERTEXT MARKUP LANGUAGE) 10
II.2 WEB MASHUPS 10
II.2.1 WEB SCRAPING 11
II.2.2 SERVICIO WEB 12
II.2.2.1 SOAP (SIMPLE OBJECT ACCESS PROTOCOL) 12
II.2.2.2 REST (REPRESENTATIONAL STATE TRANSFER) 12
II.3 PATRONES DE DISEÑOS DE SOFTWARE 13
II.3.2 CADENA DE RESPONSABILIDADES (CHAIN-OF-RESPONSIBILITY) 13
II.3.3 FACHADA (FACADE) 14
II.3.4 DAO (DATA ACCES OBJECT) 14
II.3.5 MVC (MODEL–VIEW–CONTROLLER) 14
II.4 JAVA 15
II.4.1 JAVA SERVLET API 15
II.4.2 STRUTS 2 16
II.4.3 SPRING 2 16
II.4.4 JPA (JAVA PERSISTENCE API) 17
II.4.4.1 ORM (OBJECT-RELATIONAL MAPPING) 17
II.4.4.2 HIBERNATE 17

II.4.5 GWT 17
II.5 SVN 18
II.6 SMS (SHORT MESSAGE SERVICE) 18
II.6.1 MÓDEMS GSM 18
II.6.2 SMSLIB 19
CAPÍTULO III. MARCO METODOLÓGICO 20
III.1 METODOLOGÍA UTILIZADA 20
III.1.1 ITERATIVA E INCREMENTAL 20
III.1.2 ESTRUCTURA DE LAS ITERACIONES 21
III.1.2.1 DISEÑO O REVISIÓN DE DISEÑO 21
III.1.2.2 DISEÑO E IMPLEMENTACIÓN DE LAS PRUEBAS 22
III.1.2.3 DESARROLLO DE LAS TAREAS 22
III.1.2.4 SISTEMA DE PUNTOS 23
III.1.3 *SVN* COMO HERRAMIENTA METODOLÓGICA 23
III.2 PLAN GENERAL 24
CAPÍTULO IV. DESARROLLO 25
IV.1 [ITERACIÓN 1]: INVESTIGACIÓN Y ANÁLISIS 25
IV.1.1 DESARROLLO DE LAS TAREAS 25
IV.1.1.1 INVESTIGACIÓN DE LOS TEMAS RELACIONADOS 25
IV.1.1.2 EL PROBLEMA Y LOS REQUERIMIENTOS 26
IV.1.1.3 SELECCIÓN DE LA PLATAFORMA 27
IV.1.1.3.1 SELECCIÓN DE FRAMEWORKS DE DESARROLLO 29
IV.1.1.3.2 SELECCIÓN DEL MEDIO DE PERSISTENCIA 30
IV.2 [ITERACIÓN 2]: DISEÑO DE LA ARQUITECTURA DEL SISTEMA 30
IV.2.1 DISEÑO O REVISIÓN DE DISEÑO 30
IV.2.1.1 CASOS DE USO 30
IV.2.1.2 ARQUITECTURA 32
IV.2.1.2.1 ENTIDADES 32
IV.2.1.2.2 EJECUCIÓN DE SERVICIOS 33
IV.2.1.2.2.1 LA CADENA DE RESPONSABILIDADES 36
IV.2.1.2.2.2 LAS ESTRUCTURAS DE EJECUCIÓN 37
IV.2.1.2.3 LA CAPA DE SERVICIOS 39
IV.2.1.2.3.1 LA CAPA DE PERSISTENCIA 39
IV.2.1.2.4 EL *SERVICIO WEB* DE LOS *SERVICIOS* 40
IV.3 [ITERACIÓN 3]: ESTABLECIMIENTO DEL AMBIENTE DE DESARROLLO 41

IV.3.1 DESARROLLO DE LAS TAREAS 41
IV.4 [ITERACIÓN 4]: ESCRITURA DE LAS CLASES E INTEGRACIÓN CON EL MEDIO DE PERSISTENCIA 42
IV.4.1 DESARROLLO DE LAS TAREAS 43
IV.5 [ITERACIÓN 5]: DESARROLLO DE *LA TRANSACCIÓN, EL TRADUCTOR* Y *EL VALIDADOR* 43
IV.5.1 DISEÑO O REVISIÓN DE DISEÑO 43
IV.5.2 DISEÑO E IMPLEMENTACIÓN DE PRUEBAS UNITARIAS 44
IV.5.3 DESARROLLO DE LAS TAREAS 45
IV.6 [ITERACIÓN 6]: DESARROLLO DEL *MOTOR DE EJECUCIÓN* Y *EL INSTANCIADOR* 46
IV.6.1 DISEÑO O REVISIÓN DE DISEÑO 47
IV.6.1.1 EXCEPCIONES 47
IV.6.1.2 CRITERIOS 49
IV.6.2 DISEÑO E IMPLEMENTACIÓN DE PRUEBAS UNITARIAS 50
IV.6.3 DESARROLLO DE LAS TAREAS 50
IV.7 [ITERACIÓN 7]: DESARROLLO DE LAS ESTRUCTURAS DE EJECUCIÓN 51
IV.7.1 DISEÑO O REVISIÓN DE DISEÑO 51
IV.7.2 DISEÑO E IMPLEMENTACIÓN DE PRUEBAS UNITARIAS 53
IV.7.3 DESARROLLO DE LAS TAREAS 55
IV.8 [ITERACIÓN 8]: INTEGRACIÓN DEL SISTEMA A *SMS* 56
IV.8.1 DISEÑO O REVISIÓN DE DISEÑO 57
IV.8.2 DISEÑO E IMPLEMENTACIÓN DE PRUEBAS UNITARIAS 58
IV.8.3 DESARROLLO DE LAS TAREAS 58
IV.9 [ITERACIÓN 9]: DISEÑO Y DESARROLLO DE LA INTERFAZ DE USUARIO GRÁFICA 60
IV.9.1 DISEÑO O REVISIÓN DE DISEÑO 60
IV.9.2 DESARROLLO DE LAS TAREAS 62
IV.10 [ITERACIÓN 10]: EJECUCIÓN DE LAS PRUEBAS FUNCIONALES DEL SISTEMA 63
IV.10.1 DESARROLLO DE LAS TAREAS 63

CAPÍTULO V. RESULTADOS 65

CAPITULO VI. CONCLUSIONES Y RECOMENDACIONES 66

VI.1 CONCLUSIONES 66
VI.2 RECOMENDACIONES 67

REFERENCIAS BIBLIOGRÁFICAS 68

GLOSARIO DE TÉRMINOS 71

APÉNDICE "A": REQUERIMIENTOS Y CASOS DE USO 73

APÉNDICE "B": DIAGRAMAS DE CLASES 78

APÉNDICE "C": DIAGRAMAS DE SECUENCIAS 82

APÉNDICE "D": PRUEBAS FUNCIONALES DEL SISTEMA 85

APÉNDICE "F": ITERACIONES 93

AGRADECIMIENTOS

WILLY KUSTER

Le doy gracias a todos los que creyeron en este proyecto desde el principio. A mi amada familia, por su cariño y apoyo incondicional, especialmente a mi madre Domenica y a mi tía Ivana por su inmensa ayuda. A Alejandro, Jeferson, Jonathan, Luis, Nelson, Rigoberto y Xenia, por su infinita amistad y apoyo. Y a Yoshelyn, por su profundo cariño y por creer tanto en mi.

ORLANDO RAUSEO

Quiero dar un especial agradecimiento a mis padres por todo el esfuerzo y el apoyo que me dieron en los buenos y malos momentos vividos durante la carrera ya que sin ellos este logro no fuera posible, a mi hermano Juan Carlos por su entusiasmo, a María Eugenia por todo su cariño y sacrificio realizado para superar las horas y días de dedicación a este trabajo.

AMBOS

Ambos queremos agradecerle a nuestro Tutor y amigo Rodolfo, por creer en nosotros desde el principio y por su increíble ayuda; sin él, la realización de este proyecto no hubiera sido posible.

A todos, ¡gracias!

SINOPSIS

El presente trabajo de grado consistió en realizar un sistema Web para la extracción, procesado y formateo de información de interés personal, orientado a la mensajería de texto celular.

El sistema ofrece a sus usuarios la posibilidad de crear, reutilizar o suscribirse a servicios capaces de obtener información de fuentes variadas, disponibles en la Red, a través de una interfaz Web que les permite seleccionar las fuentes de información deseadas, encadenar los procesos y por último aplicar el formato deseado al resultado obtenido.

Aprovechando la sencillez de la mensajería de texto celular, el sistema es capaz de ofrecer al usuario una interfaz sencilla de utilizar y que además goza de una alta disponibilidad, de tal manera que los usuarios pueden consumir los servicios disponibles a través de dicha interfaz.

El sistema abre posibilidades infinitas con respecto al consumo de información de interés personal. De esta manera, un corredor de bolsa podría crear un servicio que le permita obtener el valor de acciones bursátiles específicas en cualquier momento, o un economista podría consultar el valor de indicadores económicos en tiempo real y así sucesivamente, ofreciendo a cualquier potencial consumidor una vía directa hacia la información que requiera en un momento dado.

La metodología utilizada en el desarrollo del trabajo de grado fue iterativa e incremental, realizando iteraciones cortas a lo largo de todo el ciclo de vida del proyecto para así minimizar los riesgos y flexibilizar los requerimientos.

INTRODUCCIÓN

Una de las características más importantes de la sociedad actual es la presencia de un creciente consumo de información. El flujo constante y caudaloso de la misma se acentúa a medida que más personas, al igual que corporaciones, toman conciencia de que la información es uno de los recursos más valiosos al intentar conseguir una ventaja competitiva en cualquiera de las áreas concebibles.

¿Cómo formar parte de tal flujo de información?, ¿Cómo dominarlo o sacar provecho de él? La tarea es ciertamente abrumadora, considerando la inmensa cantidad de sucesos o eventos, novedades o simples unidades de conocimiento que brotan de dicho flujo por cada segundo que pasa.

De los problemas enunciados anteriormente nace una necesidad inmediata de nuevas tecnologías que facilitan y, en la mayoría de los casos, hacen posible la simbiosis que se puede observar hoy en día entre las personas y la información que las rodea. Filosofías tecnológicas como la *Web 2.0*, —en donde la premisa más importante consiste en que los sistemas de información obtienen e incrementan su valor a medida que las personas dan uso de ellos—, han asfaltado el camino hacia lo que conocemos hoy como *Facebook, MySpace, Twitter, Wikipedia* y demás redes sociales, cuya popularidad es consecuencia directa de la forma en la que manejan la información y, por sobre todo, la forma en la que se la presentan a sus usuarios.

La presentación o representación de la información es, y ha sido, uno de los temas más controvertidos en el mundo de la informática. Facilitar su consumo, velar por su disponibilidad, brindar seguridad e integridad, además de optimizar todo el proceso hasta alcanzar la máxima eficiencia no es tarea fácil. Sin embargo, los máximos exponentes de la *Web 2.0* mencionados anteriormente han logrado

dominar las técnicas de la presentación de la información no sólo gráfica, sino también de una larga serie de formatos y técnicas (*RSS, JSON, XML, Servicio Web* y muchos más). Éstas hacen posible el consumo de información, no sólo por parte de personas, sino también de otros sistemas, que a su vez alimentan a otros, dando evidencia del gran caudal de información que nos rodea en la actualidad.

La ocurrencia de sistemas de información alimentándose de otros no es nueva, sin embargo, recientemente se conceptualizó y tomó forma en lo que se conoce hoy como *Mashup. John Crupi* y *Chris Warner* [4], en su artículo *Enterprise Mashups: Bringing SOA to the People,* definen *Mashup* como *"Una micro-combinación, usuario-centralizada y basada en estándares de fuentes de información internas y externas".* De tal definición se puede entender que un *Mashup* consiste en la combinación de varias fuentes de información que convergen en una aplicación totalmente nueva, que ni los dueños de la data original, individualmente, pueden haber considerado posible.

Crupi y *Warner* resaltan un punto importante en su artículo, "usuario-centralizado". Explican que los *Mashups* son para el consumo de usuarios finales y que generalmente son creados por ellos mismos, sin embargo, siempre debe haber participación de los departamentos de tecnología y éstos deben velar por la presentación final de la información resultante de los *Mashups.* Sin embargo, lo anterior está tomado en un contexto empresarial, lo que conduce a la siguiente pregunta: ¿Cómo aprovechar todas las ventajas de los *Mashups* a nivel personal? Es en este contexto donde surge el presente trabajo especial de grado. Satisfacer la necesidad de las personas de poder construir, personalizar, reutilizar y consumir *Mashups* en el lugar y momento en que lo necesiten, de la forma más sencilla posible. Esto se lograría desarrollando un sistema capaz de cumplir con lo señalado

previamente, aprovechando las ventajas de las *Web 2.0* con respecto al incremento del valor del sistema por concepto de uso, y la simplicidad del protocolo de mensajería celular *SMS* como potencial interfaz para el consumo de *Mashups*.

El presente trabajo está estructurado de la siguiente manera:

El Capítulo I describe el Planteamiento del problema, el Objetivo general y los específicos, La justificación y Las limitaciones y alcance. El Capítulo II plantea los conceptos que dan cuerpo al marco teórico del trabajo. A continuación, el Capítulo III describe el marco metodológico utilizado durante el desarrollo del trabajo. El Capítulo IV explica las actividades realizadas durante el desarrollo. Seguidamente, el Capítulo V muestra los resultados obtenidos. El Capítulo VI contiene las conclusiones y recomendaciones; por último, se muestran las referencias bibliográficas y los apéndices.

CAPÍTULO I. PROBLEMA

I.1 Planteamiento del problema

Los *Web Mashups* forman parte del conjunto de elementos que conforman la *Web 2.0,* toda vez que promueven la colaboración e intercambio de información. Ofrecen a las personas una vía directa para obtener información en el formato que deseen, de tal manera que pueden desarrollar sus propios *Servicios Web*, los cuales ofrecen flujos de información para el uso y beneficio de sí mismos, como también para otros, sin la necesidad de tener conocimientos de programación.

Estos servicios extraen información a través de varios formatos o fuentes, tales como *HTML (Web Scraping), XML,* documentos de texto, archivos *CSV* u otros.

Para la creación de los *Mashups* se usan estructuras o filtros para el procesamiento de información. Estas estructuras permiten extraer, homogenizar, limpiar y organizar un conjunto de datos de distintas fuentes, cuya finalidad es la de facilitar la manipulación de los datos como un todo o individualmente.

En tal sentido, podemos encontrar distintos tipos de estructuras tales como las de procesamiento por palabras, expresiones regulares, ciclos o incluso *Mashups* en sí mismos.

Dichas estructuras tienen entradas y salidas llamadas elementos de información. Estos elementos de información están compuestos por variados atributos como cualquier estructura de información convencional. Las estructuras de procesamiento de información se comunican por medio de los elementos de información de forma muy similar a una *Tubería,* así pues, las salidas de una estructura de procesamiento son las entradas de otra estructura de procesamiento.

El acceso a la información servida por los *Web Mashups* puede variar, desde *RSS* hasta *Servicios Web* tipo *SOAP*, pero para poder utilizar estos medios se necesita un computador personal o equipos móviles con acceso a *Internet*, los cuales son costosos. Luego, los servicios de mensajería de texto celular ofrecen a los usuarios una forma de acceso a la información y comunicación muy utilizada y de bajos costos.

I.2 Objetivo general

Implementar un servicio Web que permita crear *Mashups* públicos o privados para ser usados a través de la mensajería de texto celular.

I.3 Objetivos específicos

1. Desarrollar e implementar un sistema Web encargado de la gestión de los *Mashups* que permita la creación, eliminación, modificación, publicación, suscripción y búsqueda de los mismos.
2. Desarrollar un módulo del sistema para la gestión de estructuras para el procesamiento de información (filtros).
3. Desarrollar un sitio Web que aloje la interfaz gráfica del sistema y ofrezca tutoriales, foros y documentación para facilitar la gestión de *Mashups*.
4. Desarrollar un motor para la ejecución de los *Mashups*.
5. Desarrollar una interfaz que permita la integración del sistema con un módem *SMS*.

I.4 Justificación

Los servicios de *Web Mashups* ofrecen una gran cantidad de funcionalidades que ayudan a los usuarios a obtener información puntual sobre temas de interés propio,

pero la accesibilidad a dicha información depende del uso de la *Internet* como medio de comunicación, para lo cual los usuarios deben disponer de equipos con acceso a la red.

Generalmente, los equipos móviles que permiten acceso a este medio son costosos, así como también el acceso al servicio. Por lo que resultaría atractivo desarrollar un servicio Web, basado en *Web Mashups* e integrarlo a los servicios de mensajería de texto celular para así facilitar el acceso y reducción de costos a una mayor cantidad de usuarios.

I.5 Alcance

1. Se tomará en cuenta autenticación y autorización de usuarios para el acceso a los *Mashups*.
2. El sistema extraerá información de las siguientes fuentes:
    a. Html (*Web Scraping*).
    b. *Mashups*.
    c. XML
3. El módulo de estructuras para el procesamiento de información tendrá lo siguiente:
    a- Procesamiento por palabras: Consiste en un filtro por palabras claves de los elementos de información.
    b- Procesamiento por expresiones regulares: Consiste en un filtro por medio de expresiones regulares con capacidades de reemplazo.
    c- Ciclos: Consiste en una estructura de control que permite ejecutar una acción por cada elemento de información entrante.
    d- Renombrado de atributos: Permite renombrar o copiar los atributos

de los elementos de información.

e- Reversión de orden: Permite revertir el orden de los elementos de información entrantes.

f- Ordenado: Permite definir la forma de ordenar los elementos de información entrantes.

g- Manejo de tipos de datos: Permite el fácil manejo de los tipos de datos más comunes como fechas, cadenas de caracteres y números.

h- Truncado: Permite definir el número de elementos de información que van a ser mostrados.

i- Formato de salida: Permite definir el formato de la salida en los mensajes de texto móviles.

j- Parámetros de entradas: Permite definir los parámetros de entrada del *Mashup*.

4. El sistema será desarrollado y probado con equipos móviles que cumplan con el estándar *SMS* (*Short Message Service*) que forma parte de la serie de estándares *GSM* (*Groupe Spécial Mobile*) concebidos en 1985 por la *CEPT*[1].

I.6 Limitaciones

1. Disponibilidad de un módem *SMS*, debido a que es un equipo especializado y costoso con el que cuentan las compañias de telecomunicaciones. Por lo que la demostración final puede realizarse con un equipo celular conectado a una computadora para el envío de los mensajes en caso de no poder conseguirse dicho equipo.
2. Disponibilidad de una gran variedad de equipos móviles, lo que dificulta las

[1] *Conférence européenne des administrations des postes et des télécommunications: Organización europea encargada de regular las comunicaciones en general. Ver: http://www.cept.org/*

pruebas del sistema a gran escala, por lo que las pruebas se limitarán a los siguientes dispositivos:

a- Apple iPhone (Primera generación)

b- Motorola RAZR

c- Motorola L7

d- Nokia N95

3. Escasa información disponible, dado que es un tema muy reciente y aún no está enfocado a la mensajería celular.

CAPÍTULO II. MARCO TEÓRICO

II.1 Web 2.0

La *Web 2.0* es un término que describe el uso de las tecnologias asociadas a la *Internet*, enfocadas a los aspectos de colaboración e intercambio de la información entre sus usuarios. Estos conceptos han promovido el desarrollo de comunidades Web como las redes sociales, wikis, blogs, entre otros.

El termino *Web 2.0* no se refiere a ninguna especificación técnica, sino más bien a un cambio en la filosofia de desarrollo y uso de las aplicaciones Web. Sin embargo, tecnologias, técnicas y protocolos como *RSS, HTML y XML* han jugado un papel importante en este movimiento.

Referencias consultadas: [17], [27].

II.1.1 RSS (Really Simple Syndication)

Es un sencillo formato de datos, utilizado para redifundir contenidos a suscriptores de un sitio Web. Permite distribuir contenido sin necesidad de un navegador, utilizando un software diseñado para leer estos contenidos.

RSS esta basado en *XML*, desarrollado para todo tipo de sitios Web, los cuales permiten compartir la información publicada en ellos y usarla en otros sitios Web o programas. A esto se le conoce como redifusión Web.

Referencias consultadas: [12], [21].

II.1.2 XML (Extensible Markup Language)

Es una especificación de uso general que define una forma de crear lenguajes de marcado personalizados. Su cualidad más importante es la "extensibilidad", que le

permite cambiar y crecer en complejidad con respecto al tiempo, produciendo un mínimo impacto sobre las aplicaciones o medios que lo usan.

Está pensado principalmente para la transmisión de información entre aplicaciones, pero su alcance es ilimitado, desde persistencia de datos hasta la representación de casi cualquier estructura de información.

II.1.3 HTML (HyperText Markup Language)

Es un lenguaje de marcado usado para la construcción de páginas Web. Permite describir la estructura y el contenido en forma de texto que usualmente es interpretado y representado gráficamente por una aplicación de software denominada Navegador Web. En la Figura 1 se puede contemplar un ejemplo del código *HTML* de un documento sencillo. Como se puede observar, esta compuesto por etiquetas que abren y cierran a la vez que aportan un significado semántico para con el *Navegador Web* que ha de interpretar el código.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Titulo</title>
  </head>
  <body>
    <p>Contenido</p>
  </body>
</html>
```

Figura 1. Ejemplo de HTML

Fuente: Elaborado por los autores.

II.2 Web Mashups

Son aplicaciones Web híbridas que usan contenidos de diversas fuentes para crear una herramienta integrada completamente nueva.

Suelen ser orientados al usuario final y en la mayoría de los casos son creados por los usuarios mismos con la debida intervención de los departamentos de tecnología para cubrir los aspectos técnicos. En la Figura 2 se muestra un esquema de cómo

funcionan los *Mashup*. Como se puede observar, varias fuentes provenientes de la *Internet* con distintos formatos convergen en un *Mashup*.

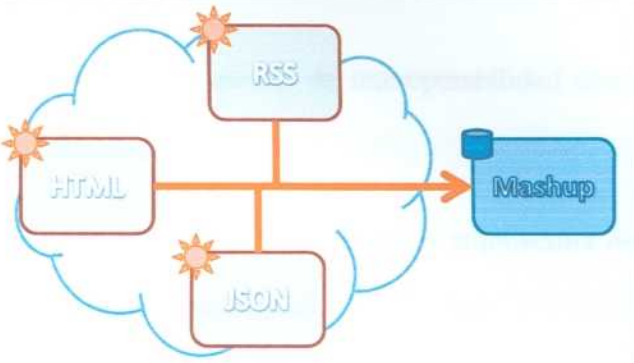


Figura 2. Esquema general de un Mashup.

Fuente: Elaborado por los autores.

Las fuentes disponibles para la creación de *Mashups* suelen ser *Servicios Web,* sin embargo, no se requiere de ningún formato específico, por lo que incluso una página web en *HTML* puede ser considerada como una posible fuente de información. La técnica de extracción de información a partir de páginas web se denomina *Web Scraping.*

Referencias consultadas: [14].

II.2.1 Web Scraping

Consiste en una serie de distintos medios de extracción de contenido de un sitio *Web* a través del protocolo *HTTP*, con el propósito de interpretar y transformar ese contenido para su posterior utilización en otro contexto. Esta técnica es comúnmente utilizada en la construcción de *Mashups*.

El *Web Scraping* difiere del *Screen Scraping* (técnica de análisis gráfico de pantallas) en que un sitio *Web* no es realmente una pantalla gráfica, sino más bien, un contenido basado en *HTML*, con una representación gráfica provista por un *Navegador Web*. Por lo tanto, *Web Scraping* no se involucra con la interfaz visual de la pantalla sino más bien con la estructura de objeto subyacente o *DOM* (*Document Object Model*) de *HTML*.

II.2.2 Servicio Web

El *W3C* [2] define *Servicio Web* como un medio estándar de interoperabilidad entre distintos sistemas o plataformas a través de una red.

Los *Servicios Web* son una consideración importante en el diseño y arquitectura de sistemas de software complejos debido a su extensibilidad y flexibilidad.

Suelen clasificarse en *Servicios Web SOAP* y *RESTful.*

Referencias consultadas: [28], [29].

II.2.2.1 SOAP (Simple Object Access Protocol)

Según el *W3C*, *SOAP* es un protocolo pensado para el intercambio de información estructurada en ambientes descentralizados y distribuidos.

Se caracteriza por una serie de reglas que deben cumplirse para hacer uso del protocolo, como la definición de la interfaz del servicio en el lenguaje *WSDL* (*Web Services Description Language*) y el uso del lenguaje *XML* para los mensajes del protocolo. También usualmente *SOAP* describe una lista de métodos remotos a diferencia de *REST* que propone disponer una serie de recursos.

Referencias consultadas: [29].

II.2.2.2 REST (Representational State Transfer)

Según *Roy Fielding* [6] en su disertación doctoral, *REST* es una abstracción de los elementos arquitecturales involucrados en un sistema distribuido o Web.

REST se caracteriza principalmente por introducir el concepto de "recurso" como

---

[2] *World Wide Web Consortium: Organización encargada de los estándares para la Internet. Ver: http://www.w3.org/*

unidad lógica en cualquier aplicación web. Le otorga importancia al "caché" como una tecnología para la eficiencia, propone que cada pedido por parte del cliente al servidor sea aislado y que la indexación de los recursos se maneje aprovechando al máximo el protocolo de comunicación utilizado.

En el caso específico de *HTTP* el mapeo a los recursos suele aprovechar los métodos *PUT*, *DELETE* y *MOVE*, además de los ya populares *GET* y *POST*.

Referencias consultadas: [6], [18].

II.3 Patrones de Diseños de Software

Son soluciones reutilizable y de comprobada efectividad que permiten resolver problemas específicos en el proceso de desarrollo de Software.

Según *Nicolás Tedeschi* [24] en su artículo *¿Qué es un Patrón de Diseño?*, comenta que los patrones de diseños "*brindan una solución ya probada y documentada a problemas de desarrollo de software que están sujetos a contextos similares*".

Referencias consultadas: [8], [24].

II.3.2 Cadena de Responsabilidades (Chain-of-responsibility)

Resuelve el problema que se presenta al definir una serie de entidades procesadoras esperando por comandos específicos, donde el comando no posee la información necesaria para mapearlo a su debida entidad procesadora. El patrón promueve un bajo acoplamiento garantizando que el comando llegue a la entidad procesadora sin necesidad de que las mismas se conozcan entre sí, ni de que haya un controlador encargado del mapeo de los comandos.

Referencias consultadas: [8].

II.3.3 Fachada (Facade)

Intenta minimizar la complejidad de uso de un *API* (*Application Programming Interface*), ofreciendo una interfaz unificada y sencilla que un cliente puede usar sin necesidad de conocer los subcomponentes más complejos.

Referencias consultadas: [8].

II.3.4 DAO (Data Acces Object)

Permite suministrar una interfaz común entre la aplicación y uno o más medios de almacenamiento de datos, tales como Base de Datos o archivos planos.

Promueve el bajo acoplamiento con respecto a dichos medios de almacenamiento, permitiendo así la migración entre distintos medios sin impactar la aplicación como tal.

Referencias consultadas: [5], [8].

II.3.5 MVC (Model–view–controller)

Es un patrón de diseño arquitectónico que permite separar la lógica de negocio de la interfaz de usuario, para así poder minimizar el impacto generado al realizar futuras modificaciones, tanto a nivel de la interfaz gráfica, como en la lógica de negocio.

El patrón define tres consideraciones expuestas en la Figura 3:

La vista: es lo que comúnmente se conoce como *UI* (*User Interface*). Representa la manera en como se comunica el usuario con el sistema.

El modelo: representa todas las entidades que dan vida a la lógica de negocio.

El controlador: su tarea es responder a acciones del usuario para invocar cambios en el modelo y en la vista.

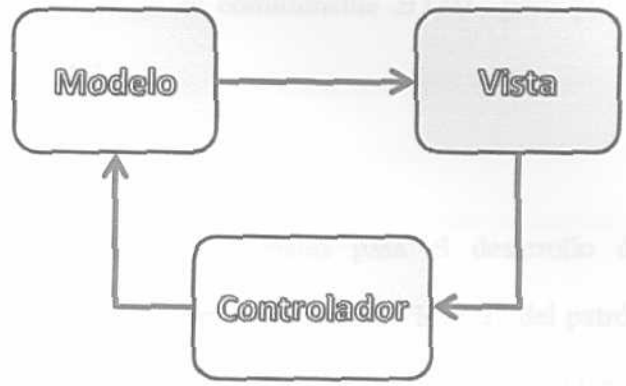


Figura 3. Esquema de MVC.

Fuente: Elaborado por los autores.

Referencias consultadas: [8], [15].

II.4 Java

Es un lenguaje de programación multiplataforma orientado a objetos desarrollado por *Sun Microsystems* [3].

Las aplicaciones Java están compiladas en lo que se conoce como *bytecode* o código intermedio, que luego es interpretado en tiempo de ejecución por una máquina virtual, lo que permite virtualmente a cualquier plataforma ejecutar aplicaciones *Java* siempre y cuando exista una implementación de la máquina virtual. También puede compilarse el *bytecode* a código nativo para su ejecución sobre una plataforma específica.

II.4.1 Java Servlet API

Es una especificación introducida por *Sun Microsystems* que permite a los desarrolladores de software, agregar contenido dinámico a un servidor Web usando

[3] *Sun Microsystems: Empresa informática productora de software y semiconductores. Ver: http://www.sun.com/*

la plataforma *Java*. El contenido generado es comúnmente *HTML*, pero puede manejar otros tipos de datos como *XML*.

II.4.2 Struts 2

Es un Framework de segunda generación diseñado para el desarrollo de aplicaciones web en *Java*, que promueve una implementación "limpia" del patrón *MVC*, introduciendo una serie de refinamientos arquitecturales que facilitan y automatizan una gran cantidad de tareas comunes para los desarrolladores web.

Referencias consultadas: [3].

II.4.3 Spring 2

Es un Framework de código abierto para el desarrollo de aplicaciones en *Java*.

Spring es único por varias razones:

- Se enfoca en proveer una manera eficiente de administrar los objetos de negocio.
- Posee una arquitectura en capas que brinda la libertad de poder escoger cual parte del Framework utilizar en el desarrollo, de manera que se pueda extraer el máximo valor de la curva de aprendizaje.
- Está diseñado para ayudar a escribir código fuente fácil de probar, gracias a su gran soporte para pruebas unitarias.
- Su principal característica es la *Inyección de Dependencias*, que incentiva el bajo acoplamiento entre los objetos de negocio.

Referencias consultadas: [11], [26].

II.4.4 JPA (Java Persistence API)

Es una *API* de persistencia desarrollada para la plataforma *Java EE* (*Enterprise Edition*) que busca unificar la manera en que funcionan las utilidades que proveen mapeo objeto-relacional. Es decir busca estandarizar la manera en que se mapean y persisten los datos entre los distintos *ORM*.

II.4.4.1 ORM (Object-Relational Mapping)

Es una técnica de programación que permite convertir objetos dinámicos en el contexto de lenguajes de programación orientados a objetos, a tablas escalares en el contexto de Bases de Datos Relacionales.

Referencias consultadas: [2], [16].

II.4.4.2 Hibernate

Hibernate es una herramienta *ORM* que facilita el mapeo de atributos entre una Base de Datos Relacional tradicional y el modelo de objetos de un sistema orientado a objetos. También ofrece un lenguaje de consulta denominado *HQL* (*High-level Query Lenguage*), que permite obtener información de la Base de Datos en un contexto orientado a objetos.

Referencias consultadas: [2].

II.4.5 GWT

Es un Framework de código abierto creado por *Google* [4] con la finalidad de facilitar tareas arduas como lo son: la creación, reutilización y mantenimiento de componentes *AJAX* y bases de código *JavaScript*, programando todo en código *Java*.

[4] *Google: Empresa proveedora de servicios online. Ver: http://www.google.com/intl/es/about.html*

Es decir, *GWT* ofrece un medio para transformar del lenguaje *Java* al lenguaje *JavaScript*.

Una de sus ventajas más notables es la compatibilidad con los distintos navegadores Web.

Referencias consultadas: [9].

II.5 SVN

Es un sistema de control de versiones iniciado por *CollabNet* [5], que facilita la colaboración y comunicación entre equipos de desarrollo de sistemas, manejando el control de las distintas versiones de los proyectos.

Referencias consultadas: [20].

II.6 SMS (Short Message Service)

SMS es un servicio que permite el envío de mensajes de texto cortos entre teléfonos móviles, teléfonos fijos y otros dispositivos de mano. Fue diseñado originalmente como parte del estándar de telefonía móvil digital *GSM* como un medio para enviar mensajes de hasta 160 caracteres incluyendo espacios.

Referencias consultadas: [22], [23], [25].

II.6.1 Módems GSM

Son dispositivos de interconexión entre computadores y teléfonos móviles que habilitan el envío y recepción de mensajes de texto celular desde y hacia dichos computadores.

---

[5] *CollabNet: Empresa informática que produce software de colaboración para equipos de desarrollo de software. Ver: http://www.collab.net/*

II.6.2 SMSlib

Es una librería para *Java* que permite el envío y recepción de mensajes *SMS* a través de un módem *GSM* o un teléfono móvil *GSM* utilizando un computador.

SMSlib ofrece una interfaz para la implementación de aplicaciones que requieran del manejo automatizado de mensajería celular.

CAPÍTULO III. MARCO METODOLÓGICO

III.1 Metodología utilizada

Para el desarrollo del presente proyecto se planteó el uso de una metodología iterativa e incremental orientada a pruebas, capaz de brindar suficiente flexibilidad durante la realización del proyecto y, a su vez, plantear las bases para la realización de un plan estructurado en etapas.

III.1.1 Iterativa e incremental

Considerando la naturaleza del proyecto y los objetivos planteados, es clara la necesidad de una metodología flexible, capaz de aceptar cambios durante el desarrollo del proyecto.

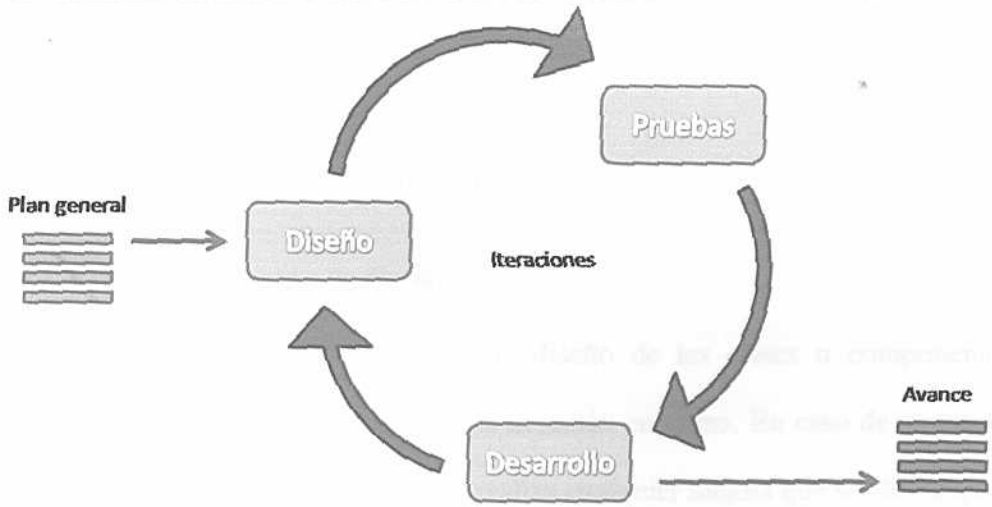


Figura 4. Diagrama de la metodología utilizada.

Fuente: Elaborado por los autores.

Una metodología iterativa e incremental promueve el desarrollo de proyectos por cortos lapsos de tiempo, con una mínima planificación, enfatizando el trabajo de equipo y la comunicación constante entre todos los miembros del proyecto.

El presente proyecto se realizó considerando las ventajas de dicha metodología, ésta se adaptó a las necesidades y al equipo de trabajo como se expone a continuación.

III.1.2 Estructura de las iteraciones

Las iteraciones cuentan con una serie de tareas especificas. Se implementó un sistema de puntos que son asignados a las tareas y permiten llevar un riguroso control del avance del proyecto, tanto a nivel individual de cada iteración, como a nivel general de todo el proyecto.

En la mayoría de los casos, cada iteración consta de tres partes:

- Diseño o revisión de diseño.
- Diseño e implementación de pruebas unitarias.
- Desarrollo de las tareas.

Sin embargo, la estructura de una iteración específica puede variar según convenga.

III.1.2.1 Diseño o revisión de diseño

Al inicio de cada iteración se realiza el diseño de las clases o componentes relacionados con las tareas asignadas a la iteración en curso. En caso de ya existir un diseño, se revisa detalladamente y se realiza cualquier mejora que se identifique.

También se considera el impacto que presenta el diseño que se está realizando con respecto al diseño general del proyecto, para así conservar una visión clara del plano abstracto del proyecto de manera incremental.

III.1.2.2 Diseño e implementación de las pruebas

En este proyecto se le da uso a una de las mejores prácticas de desarrollo de software en la actualidad.

TDD (Test Driven Development) [1] o desarrollo orientado a pruebas, consiste en el diseño e implementación de pruebas unitarias automatizadas, antes del desarrollo de las unidades funcionales a las que se refieren. Entre las ventajas de tal práctica destacan:

- La identificación de requerimientos que pueden haberse pasado por alto, ya que en el proceso de desarrollo de las pruebas las unidades funcionales a ser probadas se manipulan en un contexto muy similar al verdadero donde serán usadas.

- La integridad del proyecto entero a medida que se progresa. Dado que las pruebas son automatizadas, siempre se pueden y se deben ejecutar las pruebas ya existentes para garantizar que el progreso del proyecto no impacte en lo que ya se tiene desarrollado.

- Permite concretar los objetivos a corto plazo, por la simple razón de que si una prueba no pasa exitosamente la unidad funcional en desarrollo aún no está completa.

III.1.2.3 Desarrollo de las tareas

El diseño previamente realizado guiará el desarrollo del componente, ofreciendo una idea clara del objetivo a cumplir.

El desarrollo concreto de las tareas siempre se realiza contra las pruebas, es decir, luego de considerar "funcional" al componente en desarrollo, siempre se debe

ejecutar la prueba correspondiente para garantizar el correcto funcionamiento y en caso de fallar la prueba se sabrá que aún no se ha completado la tarea.

III.1.2.4 Sistema de puntos

Se implementó un sistema de puntos similar al propuesto en *Scrum* [19] para el control de las tareas realizadas. A cada tarea se le asigna cierta cantidad de puntos en función de su complejidad o tiempo requerido para su realización.

Se totalizan los puntos de la iteración en curso y a medida que se avanza, se deben quemar los puntos asignados a las tareas completadas.

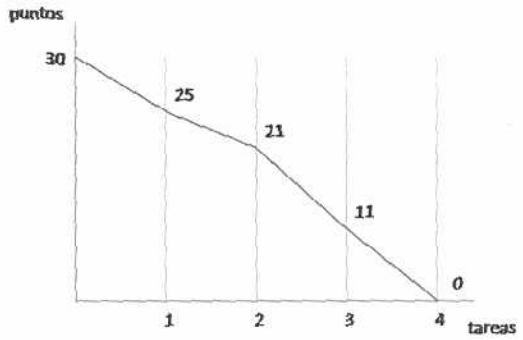


Figura 5. Ejemplo de un *Burn-Down Chart*.

Fuente: Elaborado por los autores.

A medida que se queman los puntos se registran en un artefacto conocido como *Burn-Down Chart,* similar al que se muestra en la Figura 5.

III.1.3 *SVN* como herramienta metodológica

SVN es una excelente herramienta para el control de versiones de proyectos de software. Sin embargo, según *Garrett Rooney* [20] —uno de los creadores de *SVN*—, la herramienta ofrece más que control de versiones. "*Es una forma de comunicar*", menciona en su presentación *Subversion Best Practices,* donde hace referencia a sus experiencias personales con la herramienta y explica, en su opinión, como debería ser utilizada.

En el presente proyecto, se definió la siguiente logística:

Antes de iniciar cada tarea, los miembros deben actualizar el contenido local del proyecto con respecto al servidor *SVN*. Al culminar cada tarea se deben concretar los cambios en el servidor *SVN*, siempre y cuando el código del proyecto compile y todas las pruebas unitarias se ejecuten exitosamente.

Al concretar los cambios, *SVN* permite agregar un mensaje de bitácora. Éste debe describir los cambios realizados y los puntos quemados en las tareas.

III.2 Plan general

Inicialmente se plantearon las siguientes iteraciones, con una duración de dos a tres semanas cada una:

1. Investigación y análisis.
2. Diseño de la arquitectura del sistema.
3. Establecimiento del ambiente de desarrollo.
4. Escritura de las clases e integración con el medio de persistencia.
5. Desarrollo de *La Transacción*, *El Traductor* y *El Validador*.
6. Desarrollo del *Motor de ejecución* y *El Instanciador*.
7. Desarrollo de las estructuras de ejecución.
8. Integración del sistema a *SMS*.
9. Diseño y desarrollo de la interfaz de usuario gráfica.
10. Ejecución de las pruebas funcionales del sistema.

CAPÍTULO IV. DESARROLLO

El desarrollo del presente trabajo se describe en función de la metodología expuesta anteriormente. Así pues, las iteraciones listadas en el plan general son descritas consecutivamente.

IV.1 [Iteración 1]: Investigación y análisis

Durante esta iteración no se desarrollaría ningún tipo de software, por lo tanto, no requirió de una etapa de diseño ni de pruebas unitarias.

IV.1.1 Desarrollo de las tareas

Las tareas de esta iteración consistieron mayormente en búsqueda y profundización de los temas relacionados al presente trabajo.

IV.1.1.1 Investigación de los temas relacionados

La mayor parte de la investigación realizada está descrita en el marco teórico del trabajo. Los temas de investigación más resaltantes, aupados por los objetivos del proyecto, se listan a continuación:

- *Web 2.0*
- *Mashups*
- *Tuberías o pipelines*
- Especificación *GSM/SMS*

De la *Web 2.0* fueron considerados el aspecto colaborativo y el incremento del valor del sistema por concepto de uso.

El concepto de *Mashup* probablemente es el más importante para el presente

trabajo. Se tomaron en cuenta todas las tecnologías involucradas en el concepto y se investigó el uso de las mismas en distintos casos.

Las *tuberías o pipelines*, son el medio que se tomó en consideración para que los usuarios del sistema puedan expresar y representar los servicios que deseen crear. Las tuberías son estructuras compuestas por "elementos de procesamiento" interconectados entre sí, de tal manera que la entrada de uno es la salida del otro, a excepción del primero y el último, que pasan a ser, correspondientemente, la entrada y la salida de la *tubería* entera. Estos "elementos de procesamiento" serían las estructuras que los usuarios dispondrían de forma interconectada, para representar así, un servicio ejecutable.

Uno de los objetivos más importantes del presente trabajo es la accesibilidad. La mensajería de texto *SMS* es una de las aplicaciones de datos más utilizada del planeta tierra, con más de 2.4 billones de usuarios activos enviando y recibiendo mensajes en sus móviles [22]. Es por ésta misma razón que se consideró a la mensajería *SMS* como la interfaz para el consumo de los servicios o *Mashups* del sistema. No sólo por la accesibilidad que brinda, sino también por el uso masivo de esta tecnología, lo que reduce en cierta medida la curva de aprendizaje con respecto al sistema propuesto.

IV.1.1.2 El problema y los requerimientos

El problema general del proyecto puede enunciarse de la siguiente manera: ¿Cómo facilitar el acceso a los *Mashups* como tecnología, minimizando la curva de aprendizaje de los potenciales usuarios?

Del problema anterior se identificaron los siguientes requerimientos:

- Una interfaz web, donde los usuarios puedan interactuar con el sistema, gestionar sus propios servicios o *Mashups*.

- Considerar el aspecto colaborativo, brindando a los usuarios la posibilidad de beneficiarse entre sí mismos. Ya que, aunque no lo parezca, éste requerimiento es clave para reducir la curva de aprendizaje de los usuarios para con el sistema.

- Un editor de *Mashups*, que permita a los usuarios crear y editar servicios que luego puedan consumir.

- Un componente de ejecución para los *Mashups*, capaz de interpretar y ejecutar las creaciones de los usuarios del sistema.

- Una especificación para el consumo de los *Mashups*. Dado que parte del problema es la accesibilidad, se le prestó especial atención a éste requerimiento.

IV.1.1.3 Selección de la plataforma

De todas las opciones disponibles actualmente para el desarrollo de software, *Java* de *Sun Microsystems* y *.NET Framework* de *Microsoft*, fueron las principales candidatas, debido a que las habilidades de los integrantes del proyecto predominaban en ambas tecnologías.

El *Microsoft .NET Framework* es una poderosa plataforma de desarrollo, pensada especialmente para asistir a los desarrolladores en la mayoría de los problemas comunes del desarrollo de software.

A pesar de su creciente popularidad, *.NET* presenta ciertas limitaciones:

- Actualmente la plataforma posee una madurez aceptable sólo en sistemas operativos de *Microsoft*.

- *.NET* ya es un Framework de desarrollo, lo que obliga en algunos casos a seguir la filosofía del Framework, limitando ciertas posibilidades.

- El *IDE* de desarrollo para *.NET* por excelencia es *Microsoft Visual Studio,* que a pesar de ser un excelente *IDE*, es necesario adquirir una licencia de un costo aproximado de $500 dólares americanos para usarlo.

Por su parte *Java* es una plataforma de desarrollo con más de 13 años de madurez. Ha formado parte de las tecnologías y aplicaciones que han sobresalido en los últimos 10 años, al punto de convertirse en la actualidad en líder del desarrollo de software corporativo. Una de sus cualidades más resaltantes es su portabilidad, dado que las aplicaciones realizadas con *Java* pueden ejecutarse en cualquier sistema operativo y cualquier configuración de hardware, siempre y cuando exista una implementación de la *JVM (Java Virtual Machine)* para el sistema operativo donde se desee ejecutar las aplicaciones.

Sin embargo *Java* también presenta una serie de desventajas:

- *Java SE (Standard Edition)* no incluye ningún Framework de desarrollo, los desarrolladores son responsables de seleccionar e integrar el Framework que requieran.

- En la mayoría de los casos se requiere de una inversión importante de tiempo y esfuerzo para la integración de los distintos Frameworks disponibles para *Java*.

Ambas alternativas fueron cuidadosamente analizadas y la decisión final fue a favor de *Java* luego del siguiente análisis:

- *Java* brinda una gran flexibilidad para utilizar Frameworks para distintos propósitos, mientras que *.NET* no ofrece tanta flexibilidad al respecto.
- Con Java es posible ejecutar el software en distintas plataformas, lo que flexibiliza la selección de la infraestructura para el despliegue del sistema.
- Existe una gran cantidad de *IDE* disponibles para desarrollar en *Java.* Uno de los cuales es el popular *Eclipse IDE,* que no sólo es un excelente *IDE* de desarrollo multipropósito, sino que es gratuito.

IV.1.1.3.1 Selección de Frameworks de desarrollo

Los Frameworks seleccionados para el desarrollo del proyecto se listan a continuación:

- *Struts 2*: *Framework* que ofrece las bases para el desarrollo de aplicaciones web utilizando el patrón *MVC*. Se seleccionó este *Framework* por su arquitectura limpia, modular y fácil de adoptar, sin mencionar su poder y flexibilidad.
- *Spring 2*: *Framework* de manejo de recursos. Se seleccionó debido a su capacidad para reducir el acoplamiento entre los objetos de negocio y asistir en el desarrollo de pruebas unitarias.
- *JPA/Hibernate*: Herramienta de mapeo objeto-relacional. Se seleccionó esta herramienta debido a su popularidad y superioridad sobre los demás.
- *GWT*: *Framework* para el desarrollo de aplicaciones *JavaScript* en lenguaje

Java. Se seleccionó debido a las facilidades que ofrece para el desarrollo de interfaces web dinámicas.

IV.1.1.3.2 Selección del medio de persistencia

Se consideró *PostgreSQL* como Manejador de Base de Datos por lo siguiente:

- Es accesible y gratuito.
- Sigue la filosofía de *Oracle,* uno de los manejadores líderes en el mercado actual.
- Es simple de configurar y es multiplataforma.
- Cuenta con soporte para *Hibernate*.
- Es considerablemente eficiente en el manejo de recursos.

IV.2 [Iteración 2]: Diseño de la arquitectura del sistema

Esta iteración conservó la etapa de diseño, pero no precisó de la etapa de pruebas ni de la etapa de desarrollo.

IV.2.1 Diseño o revisión de diseño

IV.2.1.1 Casos de uso

El diseño del sistema se inició como es usual, con los casos de uso. Los casos de uso más resaltantes, identificados y diseñados durante ésta etapa, se listan a continuación.

Nombre del caso de uso	Descripción
Explorar servicios	El usuario debe seleccionar la opción "explorar" del menú principal y escribir

	un término en el campo de búsqueda.
Registrar usuario	El usuario debe seleccionar la opción "registrarse" del menú principal y llenar el formulario de registro para luego presionar el botón "enviar".
Iniciar sesión	El usuario introduce su alias y su contraseña y presiona el botón "iniciar".
Suscribirse a servicio	Requiere: Iniciar sesión, Explorar servicios. El usuario debe presionar el botón "suscribirse" en el servicio de su preferencia de la lista de servicios.
Crear servicio	Requiere: Iniciar sesión. El usuario debe seleccionar la opción "crear servicio" del menú principal, agregar estructuras al lienzo del editor, interconectarlas y salvar el servicio.
Consumir servicio	Requiere: Registrar usuario. El usuario debe enviar un mensaje de texto al número del servicio cumpliendo con el formato: "aliasDelServicio:parametro1,..., parametroN".

Tabla 1. Lista de casos de uso generales.

Fuente: Elaborado por los autores.

La lista anterior de casos de uso se ramifica y se describe en detalle en el Apéndice "A": Requerimientos y casos de uso.

Los casos de uso más importantes son los relacionados con la gestión de los servicios. Estos son descritos en el siguiente diagrama.

Como se puede observar, el usuario debe ser capaz de crear, modificar, eliminar y pre-visualizar servicios. También debe ser capaz de suscribirse a un servicio, y al hacerlo, pasa a poseer un clon del servicio, al que puede asignar cualquier alias que

desee para su posterior consumo e incluso puede modificar su contenido. Es claro que para poder crear o modificar un servicio el usuario debe trabajar con el Editor de servicios. Y por último el usuario tiene la opción de publicar, o conservar privados, los servicios que cree.

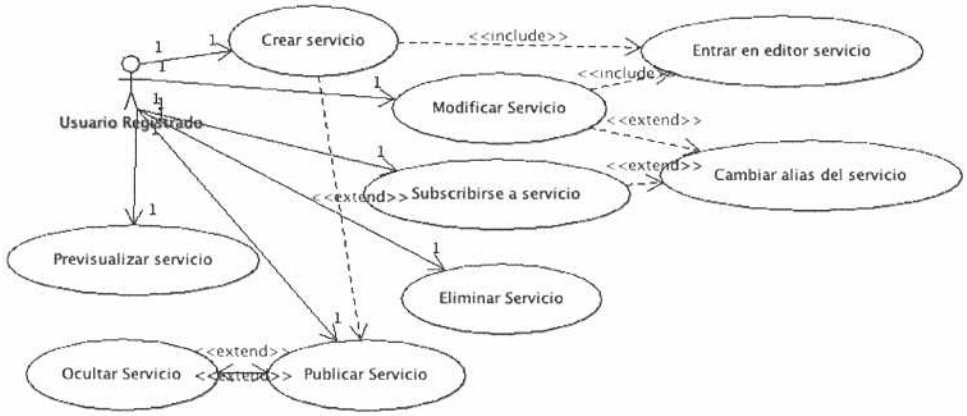


Figura 6. Casos de uso de gestión de servicios.

Fuente: Elaborado por los autores.

IV.2.1.2 Arquitectura

La arquitectura del sistema fue diseñada basada en módulos cuyo enfoque fue orientado a objetos. Éste ha demostrado ser de gran utilidad en la batalla contra la incertidumbre en el mundo del desarrollo de software, ya que permite separar la complejidad de un problema en elementos u objetos que interactúan entre sí, de tal forma que la solución se asemeja, en cierta medida, a la realidad tangible o intangible del problema. A continuación se describe en detalle la arquitectura concebida para este proyecto.

IV.2.1.2.1 Entidades

Siguiendo el enfoque orientado a objetos, se identificaron las entidades del dominio

del problema que se describen en la Figura 7.

Como se puede observar, un usuario tiene una cuenta que a su vez está relacionada con una lista de servicios. Los servicios a su vez poseen etiquetas que permiten clasificarlos (ej. deportes, comida, locales). Estas entidades deben persistir en la base de datos.

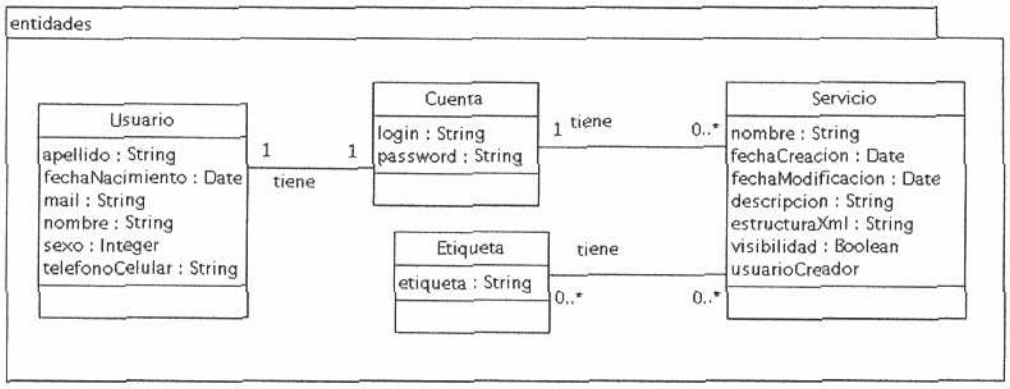


Figura 7. Diagrama de clases de las entidades.

Fuente: Elaborado por los autores.

IV.2.1.2.2 Ejecución de servicios

La entidad *Servicio* mostrada anteriormente representa el concepto de *Mashup* concebido para este sistema. En su interior, posee un atributo denominado "estructuraXml" que contiene la información de la estructura del *Mashup* en formato *XML*.

Dicha estructura describe una *tubería o pipeline* formada por elementos de ejecución interconectados entre sí. Esta *tubería* ha de ser ejecutada para la obtención del resultado que se espera del *Servicio*.

Se consideró un esquema de "mensajes" para la invocación y respuesta de los servicios. Así pues, un *Mensaje Entrante* invocaría la ejecución de un servicio, y el

resultado del mismo se representaría en una lista de *Mensajes Salientes.*

La ejecución de los servicios se separó en cuatro unidades funcionales:

- El ***Validador:*** encargado de validar los *Mensajes Entrantes.*
- El ***Instanciador:*** encargado de instanciar las *Estructuras de Ejecución.*
- El ***Motor de Ejecución:*** encargado de ejecutar las *Estructuras de Ejecución.*
- El ***Traductor:*** encargado de traducir el resultado a *Mensajes Salientes.*

Estas cuatro unidades funcionales están contenidas en lo que se denominó *La Transacción* y, por último, se designó un repositorio de información denominado *El Contexto de Ejecución*, donde las cuatro unidades funcionales intercambian la información necesaria durante el proceso de ejecución de los servicios.

Dentro del *Contexto de Ejecución* se manejan entidades como: el *Mensaje Entrante*, la *Petición*, el *Resultado,* la lista de *Estructuras de ejecución* y, por último, la lista de *Mensajes Salientes.*

El ***Validador*** se encarga de revisar e interpretar el contenido del *Mensaje Entrante*. Dicho contenido debe cumplir con cierta especificación, que es validada contra una expresión regular. Luego se verifica si existe un servicio que corresponda con el alias contenido en el mensaje y, por último, se concreta la *Petición,* que posee el servicio a ser ejecutado y los parámetros de entrada.

Luego el ***Instanciador*** toma la petición e inicia la interpretación de la estructura del servicio representada en *XML,* instanciando las *Estructuras de Ejecución* y colocando los parámetros de entrada en donde corresponda. Luego de finalizar, se concreta la lista de *Estructuras de Ejecución.*

El ***Motor de Ejecución*** es responsable de tomar la lista de *Estructuras de Ejecución* y ejecutar las estructuras siguiendo el comportamiento de una *tubería o pipeline*. En ella cada estructura recibe el producto de la estructura previa en la lista, realiza un determinado proceso y produce un resultado para la siguiente estructura en la lista. La primera estructura siempre será responsable de fungir como fuente de información y por tanto recibe un producto "vacío" por parte del ***Motor de Ejecución***.

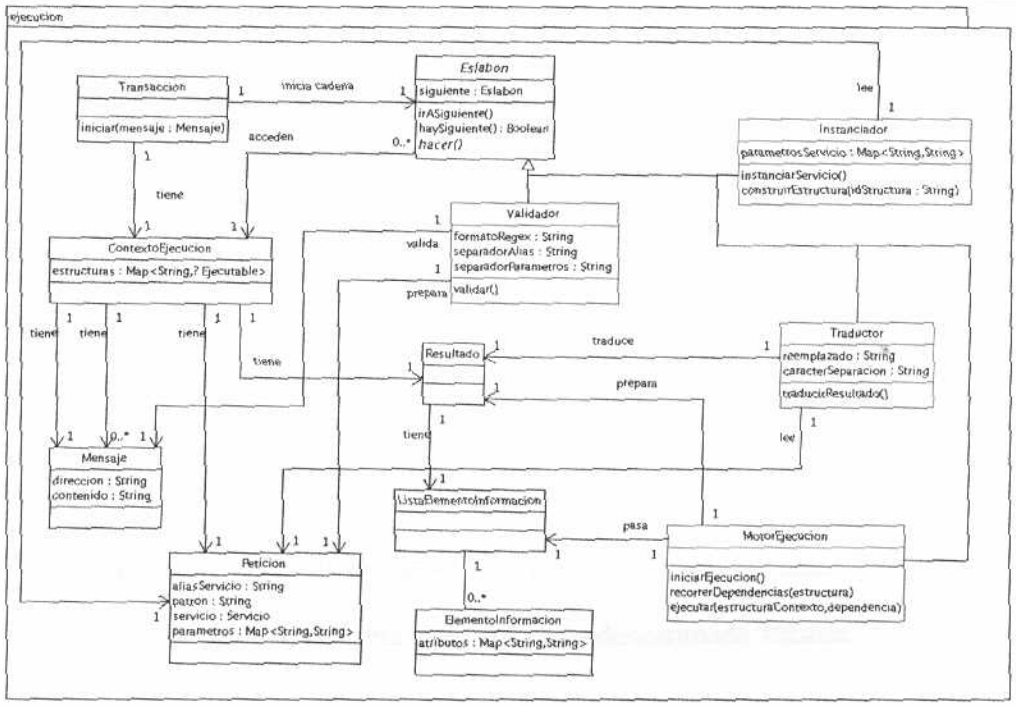


Figura 8. Diagrama de clases de ejecución.

Fuente: Elaborado por los autores.

Todas las figuras de este trabajo están disponibles en: http://soicimage.awardspace.com/

Al culminar con la ejecución de las estructuras en la lista, el ***Motor de Ejecución*** debe preparar un *Resultado,* que contiene el producto de la última *Estructura de Ejecución.*

Los "productos" de las *Estructuras de Ejecución* se identifican como *Elementos de*

Información. Estos pueden verse metafóricamente como el agua que atraviesa la *tubería* compuesta por las *Estructuras de ejecución.*

Los *Elementos de Información* cuentan con un mapa dinámico de atributos y valores que han de representar, en conjunto, a cualquier entidad que el usuario desee durante la creación de un servicio (ej. noticias, personajes, libros). Por lo tanto, dicho "producto" no es más que una *Lista de Elementos de Información,* que crece y cambia a medida que va recorriendo su camino.

El ***Traductor*** es el último eslabón del proceso de ejecución de los servicios. Es responsable de interpretar el *Resultado* y generar la lista de *Mensajes Salientes.* Debe traducir el resultado utilizando un formato, representado por una cadena de caracteres, provisto por el usuario creador del servicio.

Todas las entidades anteriores y sus interacciones se representan en la Figura 8, donde las cuatro unidades funcionales más importantes resaltan en color azul.

IV.2.1.2.2.1 La cadena de responsabilidades

En el diagrama de clases de la Figura 8 se puede observar una clase que aún no ha sido mencionada. Se trata de una clase abstracta denominada *Eslabón.*

Dado que las cuatro unidades funcionales del proceso de ejecución pueden ser vistas como una cadena de "eslabones" funcionales, es natural pensar que cada una de ellas se considere como un *Eslabón.*

Es por esto que se hizo uso del concepto de herencia provisto por el enfoque orientado a objetos, en donde las cuatro unidades funcionales heredan de la clase abstracta *Eslabón,* adquiriendo sus características. De esta manera, se puede delegar la interconexión de los eslabones a esta clase abstracta, logrando un alto grado de

desacoplamiento entre los eslabones mismos.

Esto permite que los eslabones se desempeñen de forma aislada, sin conocimiento alguno de los demás eslabones, reduciendo enormemente el esfuerzo necesario para agregar o remover eslabones en un futuro, según el proceso de ejecución lo requiera.

Lo descrito anteriormente es una adaptación del patrón de diseño *Cadena de Responsabilidades* o *Chain-of-responsibility,* del cual se aprovechó la idea de aislar las unidades funcionales para lograr el grado de desacoplamiento deseado.

IV.2.1.2.2.2 Las Estructuras de Ejecución

Como se ha dicho anteriormente, las *Estructuras de Ejecución* han de conformar una *tubería* que da cuerpo a un servicio. El usuario creador de un servicio deberá construir esta *tubería* interconectando distintas estructuras y supliendo a cada una de ellas de los parámetros de configuración que requieran para su funcionamiento.

Por lo tanto, las posibilidades de los usuarios en cuanto a la creación de los servicios vienen dadas por la selección de *Estructuras de Ejecución* que se hagan disponibles. Sin embargo, para incrementar las posibilidades de los usuarios, se consideró otro tipo de estructura capaz de realizar operaciones sencillas sobre datos concretos y que puedan depositar sus resultados en los parámetros de configuración de las estructuras principales del servicio.

Así pues, las estructuras principales se identificaron como *Estructuras Módulo*, ya que son los módulos de la *tubería* del servicio. Y las estructuras secundarias se identificaron como *Estructuras Operación*, ya que son responsables de realizar operaciones sencillas, sobres datos usualmente provistos por el usuario creador del

servicio.

Como se observa en la Figura 9, una interfaz denominada *Ejecutable* define el método "ejecutar" en el que cada *Estructura de Ejecución* concreta especificará su propia lógica de ejecución.

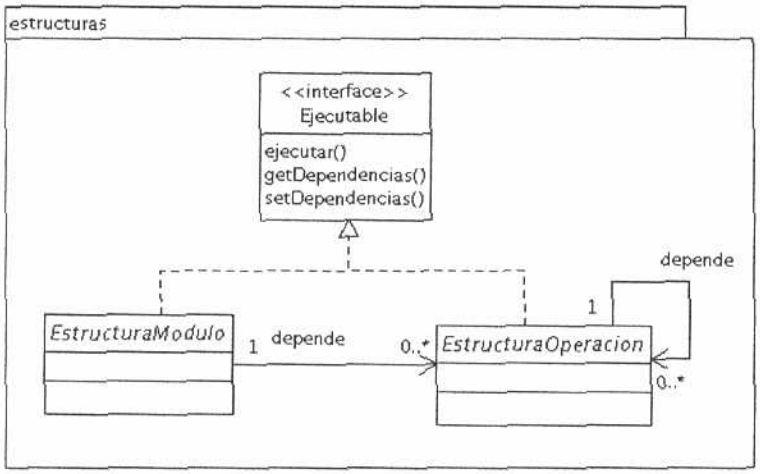


Figura 9. Diagrama de clases de las estructuras.

Fuente: Elaborado por los autores.

Luego, dos clases abstractas implementan la interfaz *Ejecutable*. Una representa a las *Estructuras Módulo*, la otra a las *Estructuras Operación*. En el diagrama se describe claramente como una *Estructura Módulo* puede depender de cero o más *Estructuras Operación* y, a su vez, una *Estructura Operación* puede depender de otras *Estructuras Operación*.

En todo caso, una dependencia ha de suplir un dato para un parámetro de configuración de otra estructura. Por lo que la lista de dependencias siempre es un mapa que relaciona el nombre de un parámetro con la dependencia misma.

Con este diseño se garantiza la extensibilidad del sistema, dado que lo único que se

requiere para agregar nuevas *Estructuras de Ejecución*, es heredar de alguno de los dos tipos de estructura e implementar la lógica correspondiente.

IV.2.1.2.3 La capa de servicios

El sistema se aprovecha de un patrón de diseño denominado *Fachada* o *Facade*, para facilitar el uso de las distintas acciones de las que dispone.

A esto se le denominó "la capa de servicios", que no ha de ser confundida con la entidad *Servicio* antes mencionada.

La capa de servicios no es más que una interfaz que provee métodos convenientes de uso general, a lo largo y ancho del sistema. Como es de esperarse, la mayor parte de los métodos convenientes de la capa de servicios son referentes al acceso de datos en el medio de persistencia.

IV.2.1.2.3.1 La capa de persistencia

Durante la primera iteración del proyecto se consideró *JPA/Hibernate* como herramienta de abstracción del medio de persistencia. Al ofrecer una abstracción del medio de persistencia, es natural pensar que esta tecnología absorbería cualquier impacto que pueda causar un cambio importante sobre el sistema (ej. cambio del manejador de base de datos).

Sin embargo, se hizo uso del patrón de diseño *DAO* para garantizar la protección del sistema a cualquier variación que pueda ocurrir con el medio de persistencia. La familia concreta de *DAO* sería la de los *DAO JPA*.

IV.2.1.2.4 El *Servicio Web* de los *Servicios*

La interfaz que provee el sistema para la invocación de los servicios es un *Servicio Web.* Esta decisión se tomó debido a que el sistema debe ser capaz de interactuar con cualquier otra tecnología al respecto de la invocación de los servicios. De tal manera que cualquier componente capaz de manejar alguna tecnología de mensajería, como por ejemplo *SMS,* pueda interactuar con el sistema, para que de esta manera los usuarios puedan utilizar el medio de mensajería que se desee integrar.

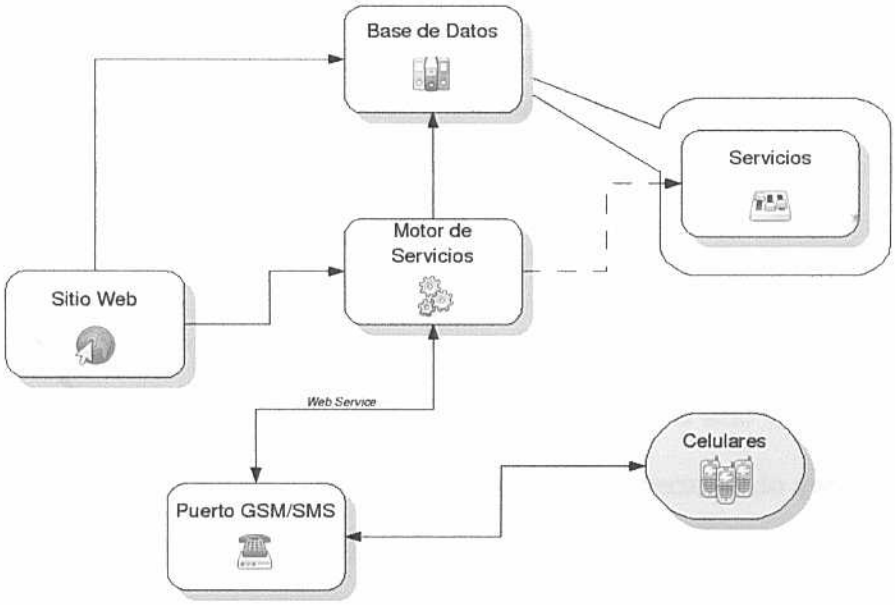


Figura 10. Diagrama general del sistema.

Fuente: Elaborado por los autores.

Así pues, cualquier medio de mensajería, desde correo electrónico hasta mensajes de voz, puede ser integrado con el sistema requiriendo una mínima inversión de

tiempo y esfuerzo. Esto se describe con mayor detalle en la Iteración 8.

En la Figura 10 se puede observar un sencillo diagrama de la arquitectura general del sistema. El *Sitio Web* claramente debe acceder a la *Base de Datos* para representar la información del sistema, así como debe acceder al *Motor de Servicios* para invocar la ejecución de los Servicios, cumpliendo el rol del *Servicio Web* de los *Servicios*. El *Motor de Servicios* debe acceder a la *Base de Datos* para extraer la estructura de los *Servicios* almacenados, y por último, el *Puerto GSM/SMS* es la interfaz entre los *Celulares* de los usuarios y el sistema. Éste debe invocar la ejecución de los *Servicios* en el *Motor de Servicios* a través del *Servicio Web* expuesto.

IV.3 [Iteración 3]: Establecimiento del ambiente de desarrollo

Esta iteración se modificó en su estructura debido a que no se desarrollaría ningún tipo de funcionalidad, por lo que no contó con una etapa de diseño, ni de pruebas unitarias.

IV.3.1 Desarrollo de las tareas

Se requirió de un contenedor de *Servlets* para el desarrollo del sitio web del sistema. *Apache Tomcat v6* resultó ser el más apropiado para el proyecto, dado que es de fácil acceso y muy sencillo de configurar.

Como *IDE* de desarrollo se consideró *Eclipse Ganymede,* puesto que está pensado principalmente para desarrolladores *Java* y posee integración nativa para *Apache Tomcat.*

Luego se creó un proyecto web nuevo y se inició la configuración del *Framework Struts 2* sobre ese proyecto. Esto fue un proceso de pruebas de concepto, donde progresivamente se ajustó la configuración de *Struts 2* hasta que cumpliera con los

requerimientos del proyecto.

Struts 2 es un *Framework* orientado a *Acciones,* donde cada acción se refiere a un comando que los usuarios pueden ejecutar contra el sistema. Las acciones se mapean a las vistas, de tal manera que una acción produzca un resultado que los usuarios puedan observar.

A continuación se inició la integración del *Framework Spring 2,* que permite el manejo de los objetos de negocio de una forma totalmente desacoplada y ofrece facilidades de uso general como el manejo transaccional para *JPA* y soporte para pruebas unitarias.

Luego se integró el *Framework GWT* a *Eclipse,* logrando así la posibilidad de realizar *debug* del código que se escriba en *GWT.* Ésto es un logro considerable, dado que *GWT* produce código en *JavaScript* a partir de código fuente en *Java,* lo que significa que durante el *debug,* el código que está corriendo es *JavaScript,* pero el código que el desarrollador observa es *Java.*

Por último se integró *JPA/Hibernate* con los archivos de configuración necesarios para aprovechar la auto-generación de esquemas de base de datos, que en pocas palabras, consiste en generar las tablas de la base de datos en función a las clases mapeadas.

IV.4 [Iteración 4]: Escritura de las clases e integración con el medio de persistencia

Esta iteración no precisó de revisión de diseño, ya que se pretendía escribir las clases del diseño obtenido previamente e integrar las entidades con el medio de persistencia. Tampoco contó con pruebas unitarias ya que no se implementaría ningún tipo de funcionalidad.

IV.4.1 Desarrollo de las tareas

La escritura de los esqueletos o "stubs" de las clases, dispone las bases para el desarrollo de las pruebas unitarias del sistema, ya que, las pruebas unitarias inicialmente manipularán las clases sin funcionalidad alguna.

Luego de escritas todas las clases del sistema, se inició el proceso de mapeo de las entidades a ser persistidas en la base de datos según lo dispone *JPA/Hibernate.* Se decidió utilizar la técnica de mapeo por *Annotations,* que consiste en utilizar esta característica provista por *Java* para "anotar" las clases con *metadata,* que luego un compilador o un programa puede leer y tomar acciones con respecto a esa *metadata*.

Para lo anterior se investigó en la documentación oficial de *Hibernate* y se escribieron los mapeos en las entidades del sistema exitosamente.

También se desarrolló un *DAO* genérico que implementa, haciendo uso de la técnica de *Reflection*, las tareas comunes de persistencia como búsqueda, inserción, actualización y eliminación de cualquier entidad del sistema.

IV.5 [Iteración 5]: Desarrollo de *La Transacción, El Traductor* y *El Validador*

Con esta iteración se dio inicio a la programación concreta del sistema por lo que contó con todas sus etapas.

IV.5.1 Diseño o revisión de diseño

Se diseñó un árbol de excepciones para cubrir los casos excepcionales identificados en las unidades funcionales *Transacción*, *Traductor* y *Validador.*

En cuanto al diseño de las unidades funcionales en cuestión, no se realizaron cambios significativos.

El diagrama de la Figura 11 expone las excepciones identificadas y son explicadas junto con el resto de las excepciones del proceso de ejecución en la Iteración 6.

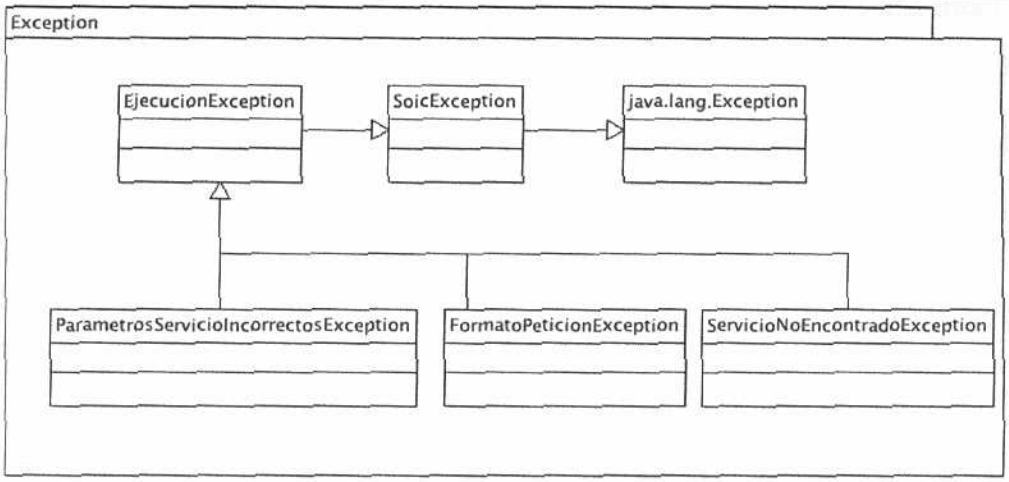


Figura 11. Excepciones para la *Transacción*, el *Validador* y el *Traductor*.

Fuente: Elaborado por los autores.

IV.5.2 Diseño e implementación de pruebas unitarias

Para probar la *Transacción* unitariamente se dispuso de dos "eslabones" ficticios cuya responsabilidad es registrar si son ejecutados o no.

Luego, al verificar que todos los eslabones fueron ejecutados, se comprueba que la transacción se está ejecutando completamente.

Gracias al soporte de *Spring* para las pruebas unitarias, se inyectó la cadena de eslabones conectados a la *Transacción* y se realizaron las verificaciones necesarias.

A continuación se diseñaron las pruebas del *Validador*, donde se dispuso de una serie de mensajes válidos e inválidos y de una capa de servicios ficticia para verificar la obtención de los *Servicios* asociados a los *Mensajes de Entrada* por parte del *Validador*.

Al verificar el resultado esperado del proceso de validación en cada uno de los casos se comprobaría el buen funcionamiento del *Validador.*

Por último se diseñaron las pruebas del *Traductor,* donde se dispuso de una lista de *Elementos de Información* llena, un formato para el proceso de traducción y una lista de *Mensajes Salientes* esperados, donde luego de ejecutar el proceso de traducción, se verifica el resultado contra la lista esperada. De ser iguales, se comprueba el funcionamiento correcto del *Traductor.*

IV.5.3 Desarrollo de las tareas

El diagrama de secuencia de la Figura 12 explica el orden de los eventos más importantes en le proceso de ejecución de los servicios contenido en una *Transacción.*

Sin embargo el diagrama no es capaz de representar la realidad completa del proceso de ejecución de la cadena de responsabilidades contenida por la *Transacción* y formada por las cuatro unidades funcionales ya mencionadas anteriormente.

Los eslabones están enlazados entre sí, lo que significa que el proceso de recorrido y ejecución es recursivo y posee un condicional que determina si se debe o no continuar la corrida. Este condicional simplemente pregunta si hay un eslabón siguiente al eslabón actual, en caso de no haber se detiene la recursión.

Luego de que las pruebas unitarias de la *Transacción* pasaron exitosamente, se procedió a la programación del *Validador* y el *Traductor* contra sus pruebas unitarias.

El desarrollo del *Validador* no presentó ocurrencia alguna digna de mención. En el caso del *Traductor,* lo que presentó la mayor complejidad fue el cuidado de

mantener la integridad de la información con respecto a los *Mensajes Salientes*. Dado que cada mensaje tiene un límite en su contenido, el traductor garantiza que la información no se disperse de forma inadecuada entre los distintos mensajes necesarios para completar la respuesta de un servicio.

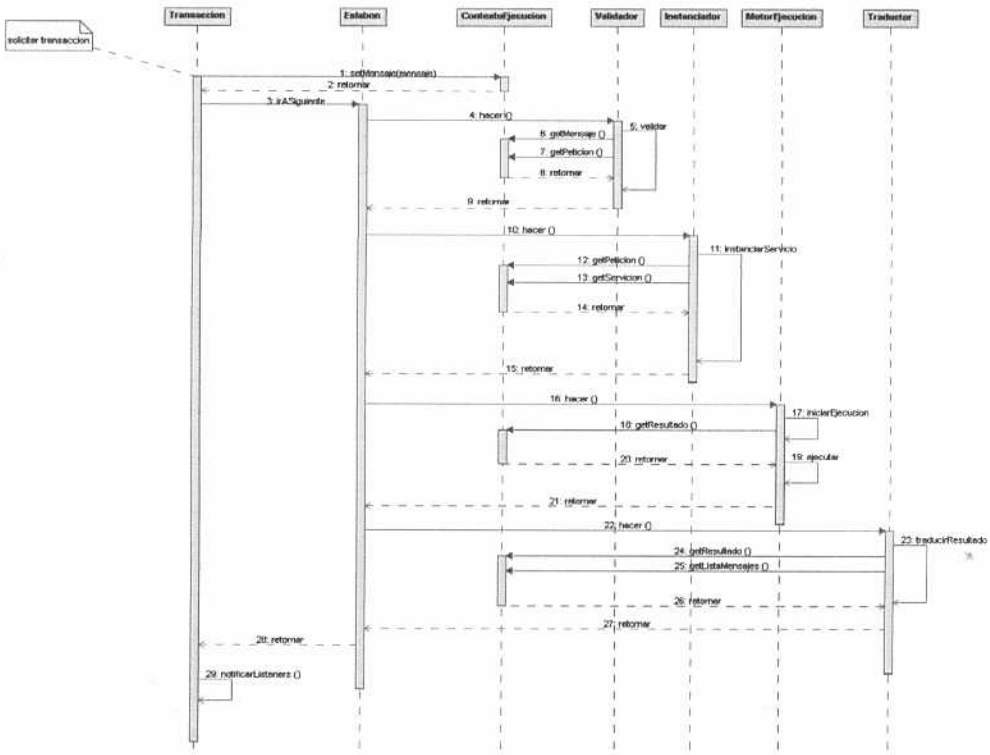


Figura 12. Diagrama de secuencia del proceso de ejecución.

Fuente: Elaborado por los autores.

IV.6 [Iteración 6]: Desarrollo del *Motor de ejecución* y *El Instanciador*

En esta iteración se desarrollaron los componentes más complejos e importantes del proceso de ejecución.

IV.6.1 Diseño o revisión de diseño

IV.6.1.1 Excepciones

El árbol de excepciones mostrado anteriormente se completó en esta iteración para así cubrir la totalidad de los casos excepcionales del proceso de ejecución.

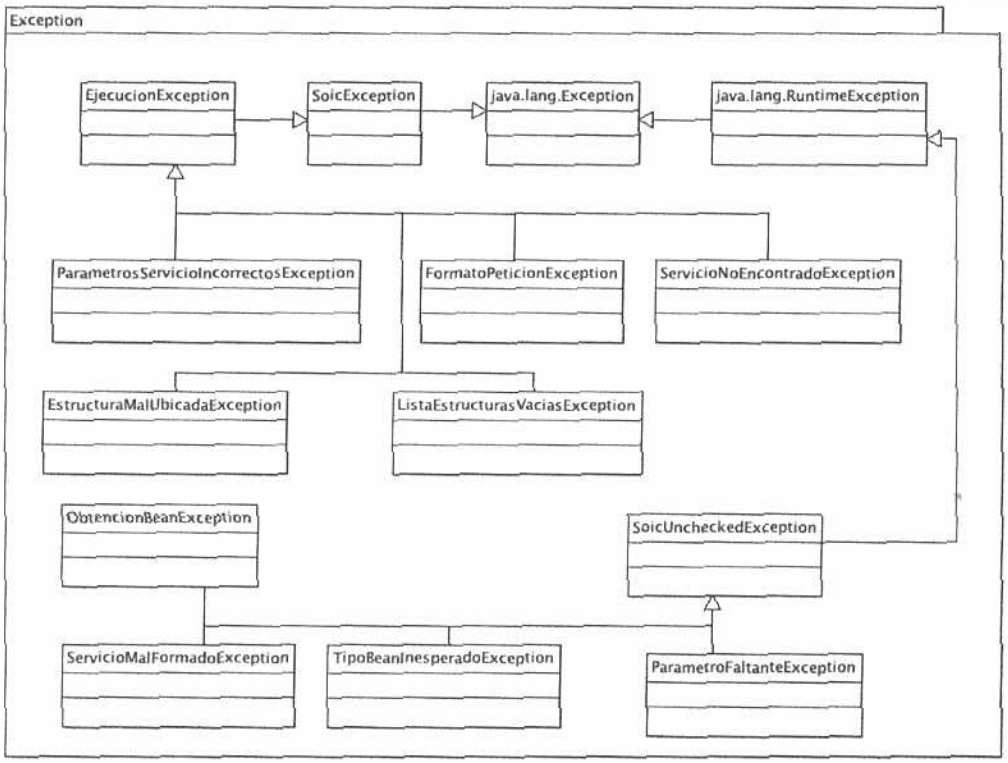


Figura 12. Diagrama de secuencia del proceso de ejecución.

Fuente: Elaborado por los autores.

En el diagrama de clases de la Figura 12 se muestran las clases *Exception* y *RuntimeException* de Java a modo de referencia, sin embargo las dos raíces del árbol de excepciones del proyecto son *SoicException* y *SoicUncheckedException,* donde la primera es la raíz de las excepciones que definen un contrato explícito con el usuario de los componentes a los que corresponden, mientras que la segunda es la

raíz de las excepciones de las que probablemente el usuario de los componentes no pueda recuperarse explícitamente.

ParametrosServicioIncorrectosException corresponde al caso en el que los parámetros en el *Mensaje Entrante* difieren en cantidad a los parámetros del servicio.

FormatoPeticionException corresponde al caso en el que el formato del contenido del *Mensaje Entrante* no corresponde con la especificación del sistema.

ServicioNoEncontradoException corresponde al caso en el que el alias del servicio en el *Mensaje Entrante* no corresponde a ningún servicio registrado en el sistema.

EstructuraMalUbicadaException corresponde al caso en el que se detecte un tipo de estructura inesperado en un contexto determinado.

ListaEstructurasVaciaException corresponde al caso en el que la lista de las estructuras a ejecutar en un servicio se encuentra vacía.

ObtencionBeanException corresponde al caso en el que la creación dinámica de instancias de cualquier clase del sistema falla por alguna razón.

ServicioMalFormadoException corresponde al caso en el que la estructura del servicio en formato *XML* no cumple con la especificación esperada.

TipoBeanInesperadoException corresponde al caso en el que un tipo de objeto creado dinámicamente no corresponde con el tipo de objeto esperado en un contexto determinado.

ParametroFaltanteException corresponde al caso en el que la clase de una estructura no posea un parámetro que se intenta inyectar.

IV.6.1.2 Criterios

Durante la revisión de diseño se identificó un problema con respecto a las estructuras cuyos parámetros de configuración eran dinámicos y se corrigió de la siguiente manera.

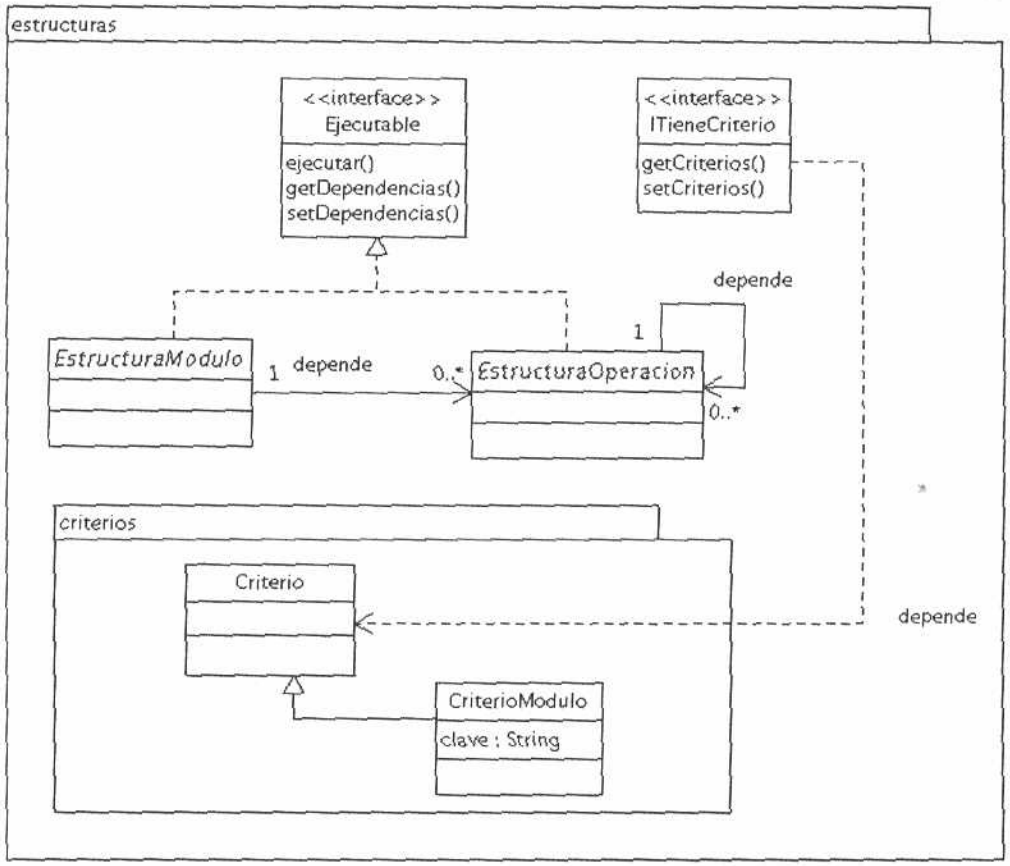


Figura 13. Diagrama de clases de estructuras con criterios.

Fuente: Elaborado por los autores.

Como se muestra en la Figura 13, se definió una interfaz denominada *ITieneCriterios* que debe ser implementada por las estructuras que requieran parámetros de configuración dinámicos. Luego se definió una clase *Criterio* que representa a los criterios o parámetros dinámicos de las *Estructuras de Ejecución,* y se

definió una clase especial de criterios para las *Estructuras Módulo* denominada *CriterioModulo,* que hereda de la clase *Criterio* y que define una cadena de caracteres que hará referencia a las claves de los *Elementos de Información* que atraviesen la *Estructura Módulo* con tal criterio.

IV.6.2 Diseño e implementación de pruebas unitarias

Para probar el *Instanciador* se dispuso de una estructura de *Servicio* en *XML*, luego se creó una lista de *Estructuras de Ejecución* esperada y se inyectó en la prueba unitaria utilizando *Spring.*

La lista de *Estructuras de Ejecución* se construyó tomando en cuenta la mayoría de casos posibles, así pues, se incluyeron estructuras con criterios, dependencias de criterios, estructuras sin dependencia alguna, etc.

Luego de ejecutar el proceso de instanciación, se verifica que las estructuras resultantes sean iguales a las estructuras esperadas, y en caso de serlo, se comprueba el correcto funcionamiento del *Instanciador.*

En el caso del *Motor de Ejecución,* se dispuso una lista de *Estructuras de Ejecución* y una lista de *Elementos de Información* esperada. Luego del proceso de ejecución de las estructuras se verifica la igualdad de la lista de *Elementos de Información* resultante con la esperada, y de ser iguales, se comprueba el funcionamiento adecuando del *Motor de ejecución.*

IV.6.3 Desarrollo de las tareas

El desarrollo del *Motor de Ejecución* consistió en la implementación de un proceso iterativo sobre la lista de *Estructuras de Ejecución*, donde por cada estructura, se ejecutan sus dependencias, se inyectan los valores resultantes y finalmente se

ejecuta la estructura en cuestión. A medida que va iterando, el *Motor de Ejecución* pasa una misma instancia de una lista de *Elementos de Información* a cada *Estructura Módulo* en la lista.

En el caso del *Instanciador,* se requirió de un componente para el acceso a los datos del *XML* que representa la estructura de los servicios. Este componente se denominó *DaoXML.*

El acceso a los datos en el *XML* se realizó a través de *XPath*, una tecnología que permite seleccionar información arbitrariamente, con una expresión representativa de la consulta deseada.

Luego de tener acceso al *XML* el *Instanciador* realiza un proceso iterativo con respecto a la información extraída y realiza la instanciación de las Estructuras de Ejecución a medida que inyecta los parámetros de entrada del servicio en donde se requiera según el *XML.*

La instanciación de las estructuras ocurre a través de *Spring*, utilizando una técnica denominada *Reemplazo de Métodos,* se logra delegar de forma limpia, el proceso de instanciación de objetos a *Spring*.

IV.7 [Iteración 7]: Desarrollo de las estructuras de ejecución

IV.7.1 Diseño o revisión de diseño

Las estructuras concretas diseñadas para el sistema se muestran en el diagrama de clases de la Figura 14. Estas estructuras fueron ideadas buscando abarcar la mayor cantidad de posibilidades para un usuario, pero considerando la simplicidad de las mismas como un objetivo primordial.

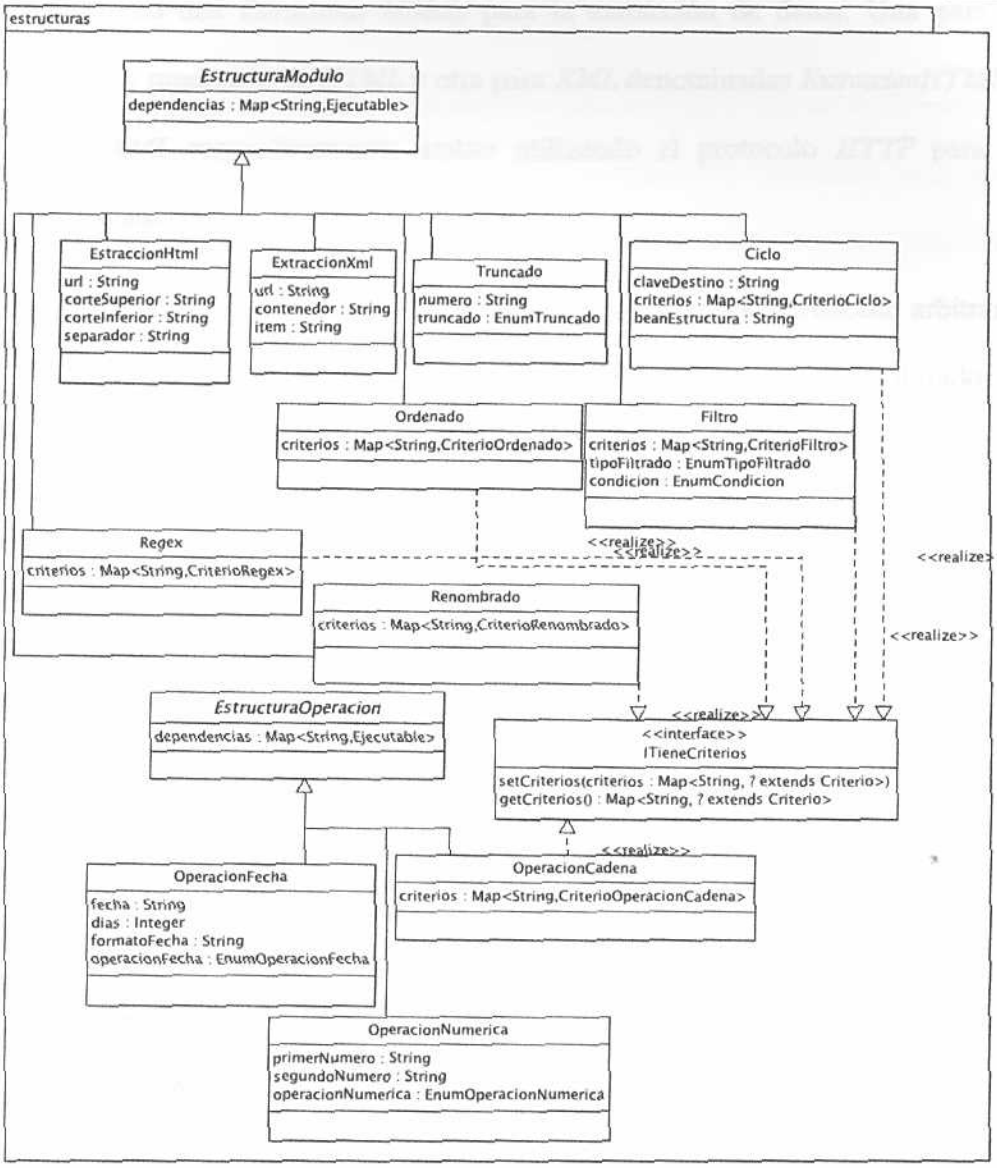


Figura 14. Diagrama de clases de estructuras completas.

Fuente: Elaborado por los autores.

Se tomó la decisión de colocar los atributos de cada estructura de tipo *String* y así liberar al *Instanciador* de realizar un proceso de conversión de datos a la hora de instanciar cada estructura, así pues, cada estructura es responsable de manejar y convertir sus atributos de *String* al tipo que requiera.

Se incluyeron dos *Estructuras Módulo* para la extracción de datos. Una para la extracción y procesado de *HTML* y otra para *XML* denominadas *ExtraccionHTML* y *ExtraccionXML* respectivamente, ambas utilizando el protocolo *HTTP* para la comunicación con las fuentes.

Se incluyó una *Estructura Módulo* denominada *Truncado* para el truncado arbitrario de *Elementos de Información*, una denominada *Filtro* para permitir el filtrado de *Elementos de Información* según un criterio determinado, una que permite ordenar los *Elementos de Información* denominada *Ordenado*, una que permite utilizar las *Estructuras Operación* directamente sobre los *Elementos de Información* denominada *Ciclo*, una denominada *Regex*, capaz de aplicar expresiones regulares sobre los *Elementos de Información*, y una denominada *Renombrado*, capaz de renombrar y copiar los atributos de los *Elementos de Información*.

Por último se agregaron tres *Estructuras Operación*, una denominada *Operación Fecha* para realizar operaciones sobre fechas, una denominada *Operación Cadena* para concatenación de una cantidad arbitraria de cadenas y una denominada *Operación Numérica* para operar dos números con doble precisión.

IV.7.2 Diseño e implementación de pruebas unitarias

Las pruebas unitarias fueron diseñadas e implementadas de la siguiente manera:

La prueba de *ExtraccionHTML* consistió en disponer de un documento *HTML* y de unos parámetros de configuración para el procesado del mismo, luego se dispuso de un resultado esperado y se comparó el resultado obtenido con el esperado.

La prueba de *ExtraccionXML* consistió en disponer de un documento *XML*, unos parámetros de configuración determinados y un resultado esperado. Luego se

comparó el resultado obtenido con el esperado.

La prueba de *Truncado* consistió en disponer una lista de *Elementos de Información,* una serie de parámetros de configuración y resultados esperados para cada conjunto de parámetros de configuración, luego se igualaban los resultados obtenidos con los esperados.

La prueba de *Ciclo* consistió en disponer una lista de *Elementos de Información*, una *Estructura Operación* específica y una lista de *Elementos de Información* esperada. Luego se comparó la lista obtenida con la esperada.

La prueba de *Filtro* consistió en disponer una lista de *Elementos de Información,* una serie de criterios y una serie de resultados esperados. Luego se compararon los resultados obtenidos en cada caso con los resultados esperados correspondientes.

La prueba de *Ordenado* consistió en disponer de una lista de *Elementos de Información*, una serie de criterios y una lista de *Elementos de Información* con el orden esperado. Luego se compararon la lista obtenida con la esperada.

La prueba de *Regex* consistió en disponer de una lista de *Elementos de Información*, una expresión regular y una lista de *Elementos de Información* esperada, que luego fue comparada con la lista resultante.

La prueba de *Renombrado* consistió en disponer de una lista de *Elementos de Información* y el campo a renombrar de los *Elementos de Información* junto con el nuevo valor del campo. Luego se compararon el valor del campo resultante con un valor esperado.

La prueba de *OperacionCadena* consistió en disponer de una serie de cadenas arbitrarias y una cadena esperada formada por la unión de cada una de ellas. Para

luego comparar el resultado con la cadena esperada.

La prueba de *OperacionNumerica* consistió en disponer de dos números, una serie de operaciones y un resultado esperado para cada operación, luego se verificó en cada caso el resultado esperado con el valor resultante de cada operación.

La prueba de *OperacionFecha* consistió en disponer de una fecha y una fecha esperada, una serie de criterios y un valor numérico con el cual operar la fecha. Luego se comparó la fecha resultante con la fecha esperada.

IV.7.3 Desarrollo de las tareas

En esta fase se desarrollaron cada una de las funcionalidades para las estructuras mencionadas anteriormente.

Para la estructura *ExtraccionHTML* se desarrolló un proceso en el cual, dada una *URL* para identificar la fuente de información, un separador para separar la data extraída y un corte superior e inferior para identificar el sector a extraer, se realiza una búsqueda iterativa de ambos cortes y si son encontrados entonces se genera una lista de *Elementos de Información* con la data extraída.

Para la estructura *ExtraccionXml* se desarrolló un proceso de extracción de información especificando una *URL*, un elemento contenedor de la información y el campo o ítem a extraer por medio *XPath*.

Para la estructura *Truncado* se desarrolló un proceso iterativo en el cual, dada una cantidad a truncar y un criterio, se aplica un truncado a la lista de *Elementos de Información*.

Para la estructura *Regex* se desarrolló un proceso iterativo sobre la lista de *Elementos*

de Información, dada una expresión regular y un criterio por el cual iterar.

Para la estructura *Filtro* se desarrolló un proceso iterativo en cual, según un criterio permisivo, restrictivo y unas condiciones para filtrado, se eliminan los *Elementos de Información* según el criterio.

Para la estructura *Renombrado* se desarrolló un proceso iterativo sobre la lista de *Elementos de Información* dado un criterio o valor de búsqueda y el nombre del nuevo atributo.

Para la estructura *Ciclo* se desarrolló un proceso iterativo en el cual, dada una *Estructura Operación,* se ejecuta dicha estructura por cada *Elemento de Información*.

Para la estructura *Ordenado* se desarrolló un proceso para ordenar la lista de *Elementos de Información* según criterios, utilizando el método "sort" de la clase "Collection" de *Java*.

Para la estructura *OperacionCadena* se desarrolló un proceso iterativo que va concatenando una serie de cadenas de caracteres.

Para la estructura *OperacioNumerica* se desarrolló un proceso que, dadas dos cadenas de caracteres, son transformadas a números tipo *Double* y operados según un criterio.

Para la estructura *OperacioFecha* se desarrolló un proceso en el cual, dada una fecha, una cantidad de días, y un criterio de operación, se opera la fecha inicial con la cantidad de días resultando en una nueva fecha.

IV.8 [Iteración 8]: Integración del sistema a *SMS*

En esta iteración se realizaron las tareas referentes al *Servicio Web* de los *Servicios* y

su papel en la integración con el sistema de mensajería *SMS*. Estas tareas son de vital importancia para el consumo de los *Servicios* por parte de los usuarios.

IV.8.1 Diseño o revisión de diseño

Los *Servicios Web* hacen posible la interacción entre sistemas de distintas naturalezas, permitiendo que diferentes plataformas se comuniquen en un lenguaje común y estándar.

La idea detrás del *Servicio Web* de los *Servicios* es de separar al sistema del medio de mensajería a utilizarse para la invocación y consumo de los *Servicios*. Por lo que se diseñó un *Servicio Web* tipo *REST* o *Servicio Web RESTful* cuyo lenguaje de comunicación sería *JSON*.

Se consideró la filosofía *REST* por su simplicidad, tanto de implementación, como de uso. Sin embargo, *REST* obliga, en la mayoría de los casos, a trabajar muy de cerca con el protocolo *HTTP* por el lado de los clientes de los *Servicios Web*, por lo que se diseñó un "conector" para los clientes, encargado de encapsular la complejidad de la serialización y deserialización de los *Mensajes Entrantes* y *Salientes* del proceso de invocación y consumo de los *Servicios*.

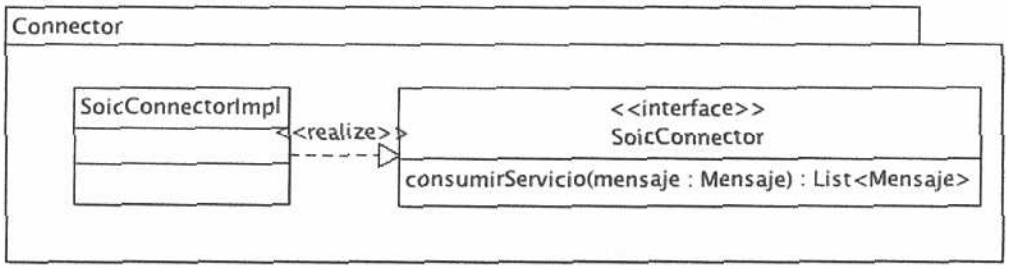


Figura 15. Diagrama de clases del conector.

Fuente: Elaborado por los autores.

El conector diseñado se puede observar en la Figura 15. Claro que sería desarrollado en *Java* y podría se utilizado sólo en clientes *Java*, sin embargo, se pueden desarrollar conectores para virtualmente cualquier plataforma, ya que lo único que requieren los *Servicios Web* es una conexión de red.

IV.8.2 Diseño e implementación de pruebas unitarias

Para probar el funcionamiento del "conector" con el *Servicio Web* se dispuso una capa de servicios ficticia que devuelve una serie arreglada de *Mensajes Salientes* en función a un *Mensaje Entrante* determinado. Luego de invocar el *Servicio Web* a través de la interfaz del "conector", se comparan los *Mensajes Salientes* obtenidos con los esperados, mientras que a su vez, del lado del *Servicio Web* se compara el *Mensaje Entrante* recibido con el esperado. Si ambas comparaciones resultan en igualdades, se comprueba el correcto funcionamiento del proceso de comunicación.

IV.8.3 Desarrollo de las tareas

Para el desarrollo del *Servicio Web* se utilizó un *plugin* de *Struts 2* denominado *JSON Plugin,* que facilita la serialización y deserialización de objetos arbitrarios dentro del contexto de una *Petición HTTP,* utilizando las "acciones" de *Struts* como punto de entrada para la lógica a ser realizada detrás del *Servicio Web.*

Esta "acción" de *Struts* llama a la capa de servicios del sistema y le provee un *Mensaje Entrante* para que ésta a su vez ejecute una *Transacción.* Luego se disponen los *Mensajes Salientes* de la *Transacción* para su posterior serialización.

El "conector" desarrollado se encarga de encapsular la complejidad del proceso de comunicación con el *Servicio Web* de los *Servicios.* De tal manera, se hizo uso de la clase *HttpConnection* de *Java* para la realización de la *Petición HTTP* necesaria para

la comunicación con el *Servicio Web*, y se utilizó la librería *XStream* de serialización de objetos a *XML*, aprovechando su soporte para *JSON*, para la serialización y deserialización de los *Mensajes Entrantes* y *Salientes*. Todo lo anterior se escondió detrás de una interfaz que expone un simple y sencillo método denominado "consumirServicio" que recibe un *Mensaje Entrante* y retorna una lista de *Mensajes Salientes* del tipo de *Mensajes* descritos en la Iteración 2.

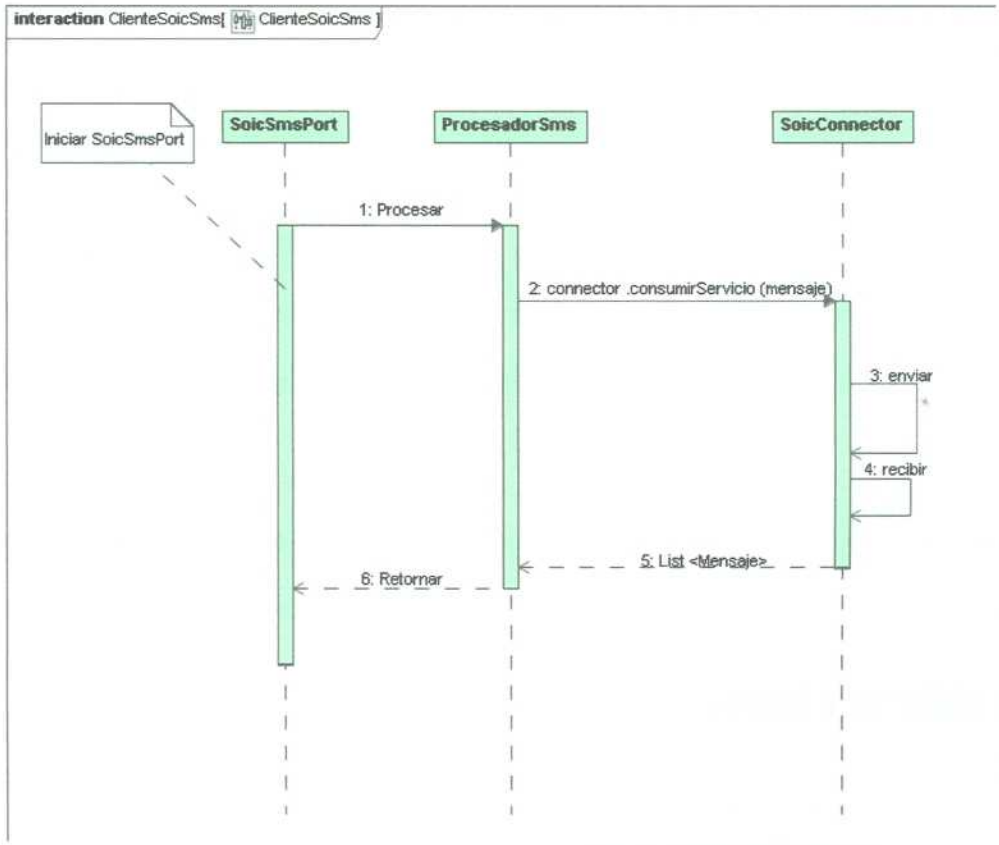


Figura 16. Diagrama de secuencia del conector.

Fuente: Elaborado por los autores.

El diagrama de secuencia de la Figura 16 explica el orden de los eventos más importantes del proceso interno del "conector".

Por último, se desarrolló el cliente concreto del *Servicio Web* de los *Servicios* denominado *SoicSmsPort,* y como su nombre lo indica, es un puerto *SMS* para el sistema. Este componente hace uso del "conector" desarrollado, para comunicarse con el sistema a través del *Servicio Web*, y aprovecha las ventajas de la librería *SMSLib* para posibilitar el envío y recepción de mensajes *SMS* a través de un *Módem GSM* o un teléfono móvil *GSM*.

IV.9 [Iteración 9]: Diseño y desarrollo de la interfaz de usuario gráfica

Esta iteración no contó con la fase de pruebas unitarias debido a su naturaleza funcional.

IV.9.1 Diseño o revisión de diseño

Para el diseño de la interfaz de usuario se utilizó una plantilla *Open Source* realizada por el grupo *NodeThirtyThree*, que fue modificada y adaptada para satisfacer las necesidades del sistema.

Se dividió la navegación en cinco (5) secciones expuestas en el menú principal de la pantalla en la Figura 17:

- *Inicio*: Donde se muestra una descripción del sistema y se brinda la posibilidad al usuario de iniciar sesión.
- *Explora*: Sección que permite al usuario explorar y subscribirse a los servicios públicos realizando una búsqueda incremental.
- *Aprende*: Sección que permite el alojamiento de tutoriales y foros sencillos para ayudar al usuario con la creación de los servicios.
- *Regístrate*: Sección que brinda la posibilidad a los usuarios de formar parte

del sistema, para luego gestionar y consumir los servicios a los cuales estén subscritos.

- *Mis Servicios*: Sección personal de cada usuario, donde pueden explorar y administrar sus servicio creados y suscritos.

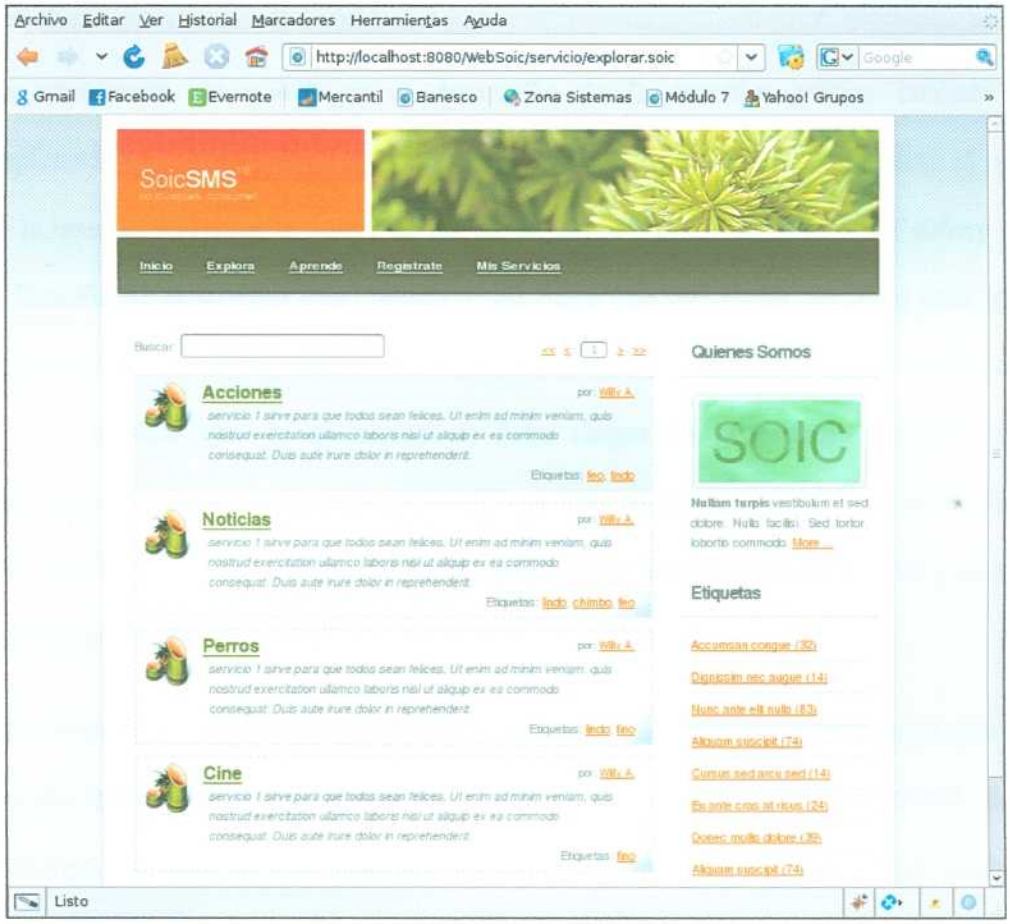


Figura 17. Pantalla de exploración de servicios públicos.

Fuente: Elaborado por los autores.

Además de las secciones ya descritas, se cuenta también con la pantalla para la creación de los servicios denominada *Editor de Servicios*, accesible desde la sección *Mis Servicios*.

IV.9.2 Desarrollo de las tareas

Para la adaptación de la plantilla escogida se utilizó *Tiles*, un Framework que permite construir de manera muy sencilla interfaces web utilizando el patrón de diseño "composite". Éste consiste en definir plantillas o "layouts" que pueden ser utilizados como esqueletos en donde piezas o "tiles" componen las diferentes pantallas de la interfaz gráfica, promoviendo la reutilización de las piezas definidas.

La utilización de *GWT* en el desarrollo de la interfaz gráfica facilitó la implementación de componentes altamente dinámicos que a su vez facilitan el uso de la interfaz por parte de los usuarios. Sin embargo, la filosofía de *GWT* difiere de la filosofía de desarrollo web convencional, dado que *GWT* está diseñado para que los desarrolladores programen la totalidad de la interfaz gráfica de forma dinámica, lo que implica que la aplicación se debe cargar en la primera petición del explorador o cliente web y toda la interfaz se debe generar en tiempo de ejecución, para que luego los datos a ser mostrados se carguen de forma asíncrona, tal y como lo propone la filosofía *AJAX*.

Esto presentaba un problema estructural en el caso del presente proyecto, ya que la interfaz gráfica se había diseñado y planteado de manera convencional. Sin embargo, se llegó a una solución eficiente y limpia para trabajar con ambas filosofías, aprovechando lo mejor de cada una de ellas.

Se desarrolló un controlador frontal o "Front Controler" que se encarga de ejecutar distintas acciones en función del *URL* presente en el explorador. Así pues, se logra un comportamiento determinístico que permite asignar una acción específica a una pantalla o página web con un *URL* especifico, evitando asi la necesidad de compilar un módulo *GWT* distinto por cada página web de la interfaz del sistema.

Por otra parte, el papel de *Struts 2* en el desarrollo de la interfaz gráfica fue el de facilitar la separación entre la capa de la vista y el modelo de negocio. Como ya fue mencionado, *Struts* es un Framework para *Java* que permite el desarrollo de aplicaciones web bajo el patrón *MVC*.

Struts es un Framework orientado a "acciones". Estas "acciones" sirven como intermediarias para la separación limpia de la vista y el modelo. Además, *Struts* cuenta con una figura denominada "interceptor" que sirve para asistir a las "acciones" en tareas comunes del desarrollo web.

Aprovechando la arquitectura flexible de *Struts,* se desarrolló un "interceptor" llamado *AuthenticationInterceptor* que verifica si un usuario ha iniciado sesión antes de que se ejecute una acción que lo requiera.

Lo interesante de las "acciones" de *Struts* es que se puede decir que comparten una relación de uno a uno con los casos de uso del sistema en desarrollo.

IV.10 [Iteración 10]: Ejecución de las pruebas funcionales del sistema

Esta iteración no contó con la fase de diseño ni con la fase de las pruebas unitarias, puesto que las pruebas funcionales no implican desarrollo de software alguno.

IV.10.1 Desarrollo de las tareas

Las pruebas funcionales del sistema se realizaron implantando el prototipo funcional desarrollado en un ambiente de producción simulado, donde se dispuso una serie de usuarios con la tarea de hacer uso de las distintas funcionalidades del sistema. A medida que los usuarios realizaban su tarea, se observaba cuidadosamente su desempeño y se registraba cualquier ocurrencia relevante.

Las tareas de los usuarios consistieron en la exploración de los servicios, la creación de cuentas personales, la suscripción, creación, modificación y eliminación de los servicios y por último el consumo de los mismos a través de equipos móviles con soporte para *SMS*.

Las pruebas realizadas y los resultados obtenidos son expuestos con mayor detalle en el Apéndice "D": Pruebas funcionales del sistema.

CAPÍTULO V. RESULTADOS

Luego de haber finalizado el desarrollo y la ejecución de las pruebas funcionales del sistema, se presentan a continuación los resultados obtenidos, respaldados por las pruebas funcionales expuestas en el Apéndice "D".

- Se obtuvo un sitio Web que brinda a los usuarios la posibilidad de suscribirse, crear, modificar y eliminar los *Servicios*.
- Se obtuvo un sitio Web que permite alojar tutoriales y foros para brindar ayuda a los usuarios del sistema.
- Se obtuvo un módulo que facilita la utilización y el manejo de estructuras para el procesamiento de información.
- Se obtuvo un motor de ejecución que permite ejecutar los servicios que son solicitados por los usuarios.
- Se obtuvo una interfaz que permite la integración del sistema con un módem *SMS* a través de *SMSlib* para *Java*.

CAPITULO VI. CONCLUSIONES Y RECOMENDACIONES

VI.1 Conclusiones

En base a los resultados obtenidos y mencionados anteriormente, se puede concluir lo siguiente:

- La arquitectura diseñada para el motor de ejecución ofrece posibilidades importantes, dado que permite agregar nuevas estructuras de ejecución sin impactar en otras áreas del sistema. Además, los módulos que componen al motor de ejecución son reemplazables, lo que posibilita realizar mejoras sustanciales en cualquiera de los mismos sin afectar a los demás.

- El desarrollo de una arquitectura extensible abre las puertas para que, en un futuro, los *Servicios* no sólo sean consumidos a través de la mensajería de texto celular, si no que también puedan ser consumidos a través de cualquier tecnología de mensajería, ofreciendo oportunidades de expansión y crecimiento en el área del consumo de los *Servicios*.

- El aspecto colaborativo del sistema ciertamente incrementa las posibilidades de los usuarios, debido a que pueden compartir y reutilizar sus creaciones. Esto permite que aquellos usuarios que no posean el tiempo o la disposición de crear nuevos servicios aún puedan utilizar las creaciones de otros.

- Gracias a la integración del sistema con los módem *SMS* es posible ofrecer a los usuarios la oportunidad de obtener información a través de un medio común, masivo y que en la actualidad es considerado una de las tecnologías más utilizadas.

VI.2 Recomendaciones

A continuación se presentan algunas mejoras y recomendaciones que pudieran ser tomadas en cuenta para el futuro:

- Agregar nuevas estructuras de extracción de información que permitan a los usurarios extraer datos de una mayor variedad de fuentes (ej. documentos *CSV*, *Word*, *Excel* y otros).

- Desarrollar una interfaz de usuario para el editor de servicios que reduzca la curva de aprendizaje, con la finalidad de que más usuarios puedan disfrutar de los beneficios ofrecidos por el sistema.

- Brindarle la posibilidad a los usuarios de observar el funcionamiento de los servicios al momento de la creación o modificación de los mismos, sin la necesidad de enviar un mensaje *SMS,* facilitando así la corrección de fallas.

- Brindar la posibilidad a los usuarios de crear servicios multi-lineales, capaces de tener una variedad de caminos posibles en su ejecución.

- El sistema podría utilizarse como intermediario o "middleware" entre otros sistemas debido a la extensibilidad de su arquitectura y la posibilidad de crear servicios que procesen una salida para transformarla en otra.

- En un futuro no muy lejano el sistema podría utilizase como herramienta de soporte para la Web 3.0, dado que podrían desarrollarse componentes que soporten lenguajes como OWL (*Web Ontology Language*) o RDF (*Resource Description Framework*), abriendo las puertas de esta nueva filosofía de Internet a las corporaciones.

REFERENCIAS BIBLIOGRÁFICAS

[1] Ambler S. W. (2007). *Introduction to Test Driven Design (TDD)*, [en línea]. USA: Ambysoft. Disponible en: http://www.agiledata.org/essays/tdd.html [Agosto 2008].

[2] Bauer, C. y King, G. (2007). *Java Persistence with Hibernate.* USA: Manning.

[3] Brown, D., Davis, C. M. y Stanlick, S. (2008). *Struts 2 in Action.* USA: Manning.

[4] Crupi, J. y Warner, C. (2008). *Enterprise Mashups Part I: Bringing SOA to the People*, [en línea]. USA: SOA Magazine. Disponible en: http://www.soamag.com /I18/0508-1.asp [Agosto 2008].

[5] *Data Access Object* (2002), [en línea]. USA: Sun Microsystems. Disponible en: http://java.sun.com/blueprints/patterns/DAO.html [Agosto 2008].

[6] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*, [en línea]. USA: Universidad de California. Disponible en: http://www.ics.uci.edu/ ~fielding/pubs/dissertation/top.htm [Agosto 2008].

[7] Fowler, M. (1999). *UML gota a gota*. México: Addison-Wesley.

[8] Gamma, E., Helm R., Johnson, R. y Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* USA: Addison-Wesley.

[9] Hanson, R. y Tacy, A. (2007). *GWT in Action.* USA: Manning.

[10] Henderson-Sellers, B. (1996). *Object-oriented metrics: measures of complexity*. New Jersey, USA: Prentice Hall.

[11] Johnson R. (2007). *Introduction to the Spring Framework 2.5,* [en línea]. USA: Enterprise Java Community. Disponible en: http://www.theserverside.com/tt/ articles/article.tss?l=IntrotoSpring25 [Agosto 2008].

[12] Libby, D. (1999). *RSS 0.91 Spec, revision 3*, [en línea]. USA: Netscape Communications. Disponible en: http://web.archive.org/web/20001204093600/ my.netscape.com/publish/formats/rss-spec-0.91.html [Agosto 2008].

[13] *Manifesto for Agile Software Development* (2001), [en línea]. Disponible en: http://www.agilemanifesto.org/ [Agosto 2008].

[14] *Mashup (web application hybrid)* (2008), [en línea]. Wikipedia. Disponible en: http:// en.wikipedia.org/wiki/Mashup_(web_application_hybrid) [Agosto 2008].

[15] *Model–view–controller* (2008), [en línea]. Wikipedia. Disponible en: http://en. wikipedia.org/wiki/Model-view-controller [Agosto 2008].

[16] *Object-relational mapping* (2008), [en línea]. Wikipedia. Disponible en: http: //en.wikipedia.org/wiki/Object-relational_mapping [Agosto 2008].

[17] O'Reilly, T. (2005). *What Is Web 2.0*, [en línea]. USA: O'Reilly Media. Disponible en: http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/ what-is-web-20.html [Agosto 2008].

[18] *Representational State Transfer* (2008), [en línea]. Wikipedia. Disponible en: http://en .wikipedia.org/wiki/Representational_State_Transfer [Agosto 2008].

[19] Rising, L. y Janoff N. S. (2000). *The Scrum Software Development Process for Small Teams*, [en línea]. USA: IEEE Software. Disponible en: http://members.cox .net/risingl1/Articles/IEEEScrum.pdf [Agosto 2008].

[20] Rooney, G. (2004). *Subversion Best Practices*, [en línea]. USA: electricjellyfish.net. Disponible en: http://electricjellyfish.net/garrett/talks/ oscon2004/svn-best-practices/index.html [Agosto 2008].

[21] *RSS* (2008), [en línea]. Wikipedia. Disponible en: http://en.wikipedia.org/ wiki/RSS [Agosto 2008].

[22] *Short message service* (2008), [en línea]. Wikipedia. Disponible en: http://en. wikipedia.org/wiki/Short_message_service [Agosto 2008].

[23] *Short Message Service / SMS Tutorial* (2008), [en línea]. USA: Developers' Home. Disponible en: http://www.developershome.com/sms/ [Agosto 2008].

[24] Tedeschi, N. (2007). *¿Qué es un Patrón de Diseño?*, [en línea]. Uruguay: Microsoft Developer Network. Disponible en: http://msdn.microsoft.com/es-es/library/bb972240.aspx [Agosto 2008].

[25] Trosby, F. (2004). *SMS, the strange duckling of GSM,* [en línea]. Noruega: Telektronikk. Disponible en: http://www.telenor.com/telektronikk/volumes/pdf/3.2004/Page_187-194.pdf [Agosto 2008].

[26] Walls, C. y Breidenbach, R. (2008). *Spring in Action* (2da ed.). USA: Manning.

[27] *Web 2.0* (2008), [en línea]. Wikipedia. Disponible en: http://en.wikipedia.org/wiki/Web_2.0 [Agosto 2008].

[28] *Web service* (2008), [en línea]. Wikipedia. Disponible en: http://en.wikipedia.org/wiki/Web_service [Agosto 2008].

[29] World Wide Web Consortium (2007). *SOAP Version 1.2 Part 1: Messaging Framework* (2da ed.), [en línea]. USA: W3C. Disponible en: http://www.w3.org/TR/soap12-part1/ [Agosto 2008].

GLOSARIO DE TÉRMINOS

AJAX: es una técnica para crear aplicaciones Web interactivas.

API: conjunto de clases, procedimientos, funciones o métodos agrupados para ser utilizado por otro software como una capa de abstracción.

Bytecode: código intermedio más abstracto que el código máquina que luego es interpretado por la *JVM* de *Java* para su ejecución.

Caché: tecnología que consiste en guardar información temporal para su posterior uso, mejorando la eficiencia de cualquier tipo de comunicación.

CSV: es un tipo de documento sencillo para representar datos en forma de tabla.

DOM: es una interfaz que proporciona un conjunto estándar de objetos para representar documentos *HTML* y *XML*.

Expresiones Regulares: son un medio muy poderoso para operar cadenas de texto a través de una expresión representativa.

Framework: es una pieza de software pensada para facilitar y automatizar tareas comunes del desarrollo de software.

GSM: es un sistema estándar, completamente definido, para la comunicación mediante teléfonos móviles que incorporan tecnología digital.

HQL: lenguaje para consulta y manipulación de datos en un *ORM*.

HTTP: protocolo de transporte para hipertexto, este protocolo permite la transmisión de un paquete de formato hipertexto a través de la red.

IDE: entornos de programación integrados con una serie de herramientas para

facilitar la labor del desarrollador.

Inyección de Dependencia: patrón arquitectónico en el cual se inyectan objetos a una clase, en lugar de que la propia clase cree los objetos para asií tener independencia entre objetos.

JavaScript: es un lenguaje de programación interpretado utilizado para el desarrollo de aplicaciones Web del lado del cliente.

JSON: es un formato estándar para el de intercambio de datos.

JVM: programa ejecutable encargado de interpretar y ejecutar instrucciones escritas en código binario (el *Java Bytecode*), generado por el compilador del lenguaje *Java*.

SOA: arquitectura de software orientada a servicios.

Tubería: conjunto de elementos de procesamiento de datos conectados en serie, de tal manera que la salida de un elemento es la entrada del siguiente.

URL: es un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en *Internet*, por su localización.

XPath: especificación que permite consultar documentos *XML* en función a una expresión representativa de la consulta.

XStream: librería para *Java* que serializa y deserializa objetos arbitrarios a *XML*.

APÉNDICE "A": REQUERIMIENTOS Y CASOS DE USO

A.1 Requerimientos

Código	Descripción
PUB-001a	Permitir explorar los servicios públicos.
PUB-001b	Exponer la opción entrar en sesión con el sistema o de registrarse.
PUB-001c	Registrar usuarios con nombre, apellido, contraseña, email, login, sexo, número de celular y fecha de nacimiento.
PUB-001d	Exponer tutoriales y foros.
PER-001a	Exponer los servicios de los usuarios separados por subscritos o creados
PER-001b	Exponer opciones de previsualización, modificación, publicación, eliminación, suscripción y creación de servicios.
PER-001c	Consumir los servicios a través de mensajes SMS

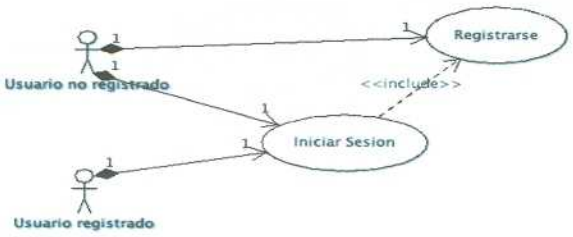
A.2 Casos de Uso

Codigo	Nombre del caso de uso	Actor	Descripción
CU-1	Explorar servicios	Usuarios Registrados y No registrados	El usuario debe seleccionar la opción "explorar" del menú principal y escribir un término en el campo de búsqueda.
CU-2	Registrar usuario	Usuario no Registrado	El usuario debe seleccionar la opción "registrarse" del menú principal y llenar el formulario de registro para luego presionar el botón "enviar".
CU-3	Iniciar sesión	Usuario Registrado	El usuario introduce su alias y su contraseña y presiona el botón "iniciar".
CU-4	Suscribirse a servicio	Usuario Registrado	Requiere: Iniciar sesión, Explorar servicios. El usuario debe presionar el botón "suscribirse" en el servicio de su preferencia de la lista de servicios.
CU-5	Crear servicio	Usuario Registrado	Requiere: Iniciar sesión. El usuario debe seleccionar la opción "crear servicio" del menú principal, agregar estructuras al lienzo del editor, interconectarlas y salvar el servicio.
CU-6	Consumir servicio	Usuario Registrado	Requiere: Registrar usuario. El usuario debe enviar un mensaje de texto al número del servicio cumpliendo con el formato: "aliasDelServicio:parametro1,..., parametroN".
CU-7	Modificar servicio	Usuario Registrado	Requiere: Iniciar sesión. El usuario debe seleccionar el servicio de la sección personal, seleccionar modificar, modificarlo y seleccionar guardar.
CU-8	Eliminar	Usuario	Requiere: Iniciar sesión.

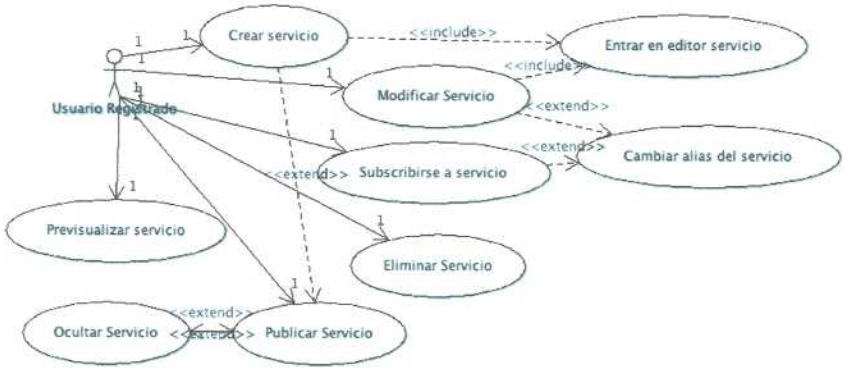
	servicio	Registrado	El usuario debe seleccionar el servicio de la sección personal y escoger la opción eliminar.
CU-9	Publicar servicio	Usuario Registrado	Requiere: Iniciar sesión. El usuario debe seleccionar el servicio de la sección personal y escoger publicar o al momento de la creación del servicio seleccionar la opción publicar.
CU-10	Ver Tutoriales	Usuarios Registrados y No registrados	El usuario debe seleccionar la opción Aprende del menú principal y luego seleccionar el tutorial de la lista de opciones.
CU-11	Acceder al foro	Usuarios Registrados y No registrados	El usuario debe seleccionar la opción Aprende del menú principal, luego seleccionar foros.
CU-12	Crear nuevo post	Usuarios Registrados y No registrados	Requiere: Acceder al foro. El usuario debe acceder al foro y escoger la opción crear nuevo post, escribir el mensaje y luego enviarlo
CU-13	Cambiar alias del servicio	Usuario Registrado	Requiere: Modificar servicio o subscribirse a servicio. El usuario debe seleccionar la opción subscribirse a servicio de la sección publica modificar el alias y guardar, o debe seleccionar la opción modificar servicio de la sección personal y cambiar el alias.
CU-14	Entrar al editor de servicio	Usuario registrado	Requiere: Iniciar sesión. El usuario debe seleccionar la opción crear servicio en la sección personal o modificar servicio de la lista de servicios propios

A.3 Diagramas de Casos de Uso

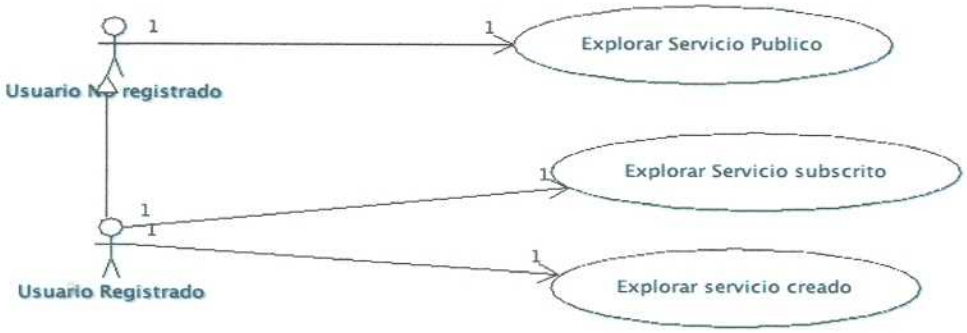
A.3.1 Iniciar Sesión



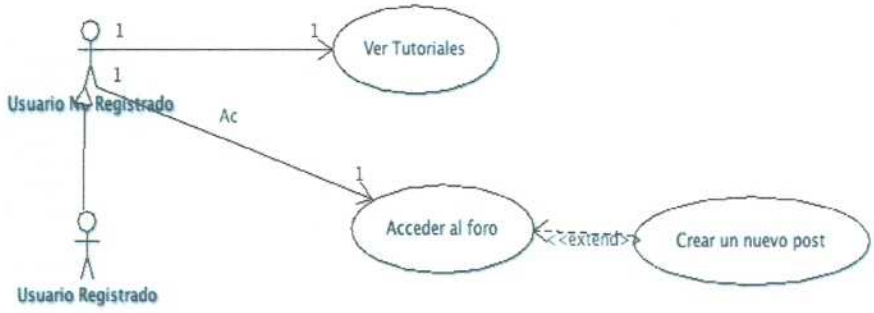
A.3.2 Administrar Servicios



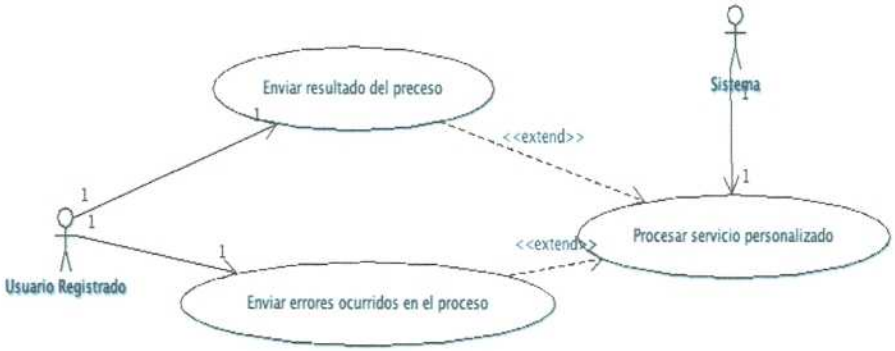
A.3.3 Explorar Servicios



A.3.4 Explorar Documentación

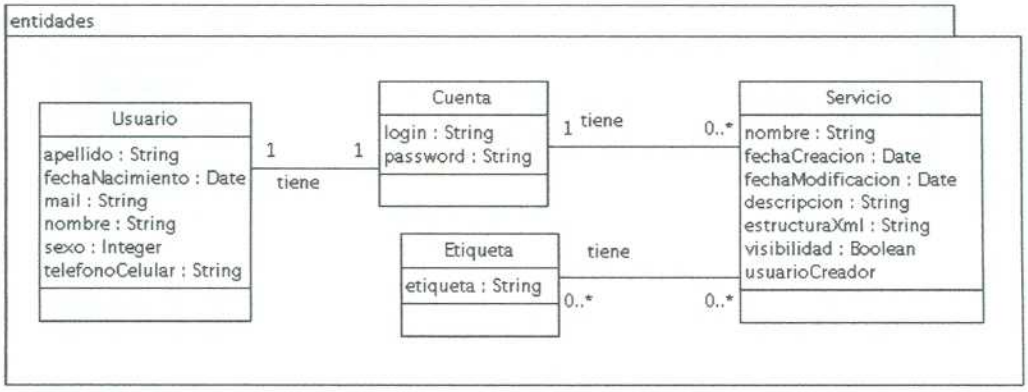


A.3.5 Consumir Servicios

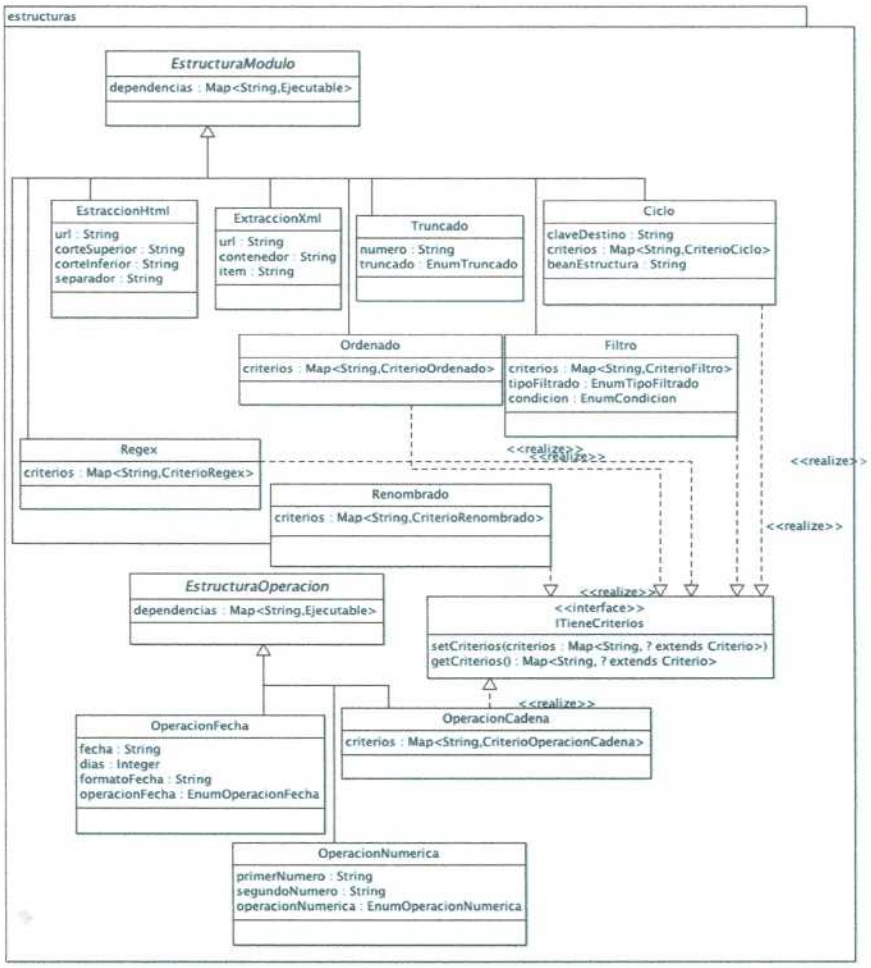


APÉNDICE "B": DIAGRAMAS DE CLASES

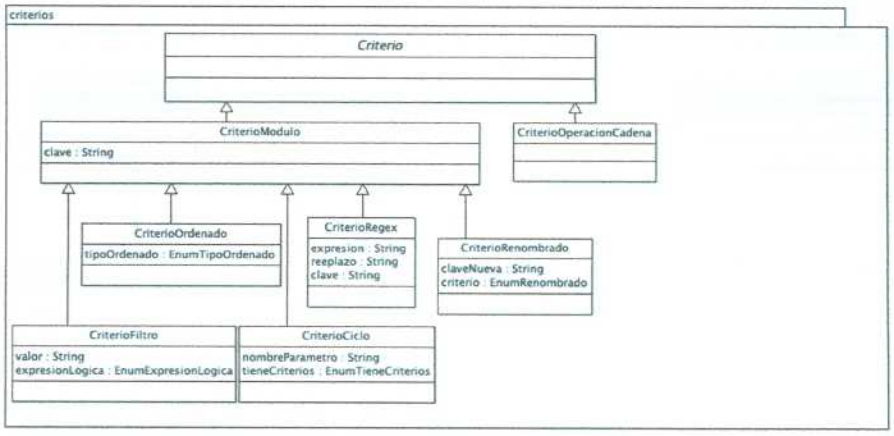
B.1 Entidades



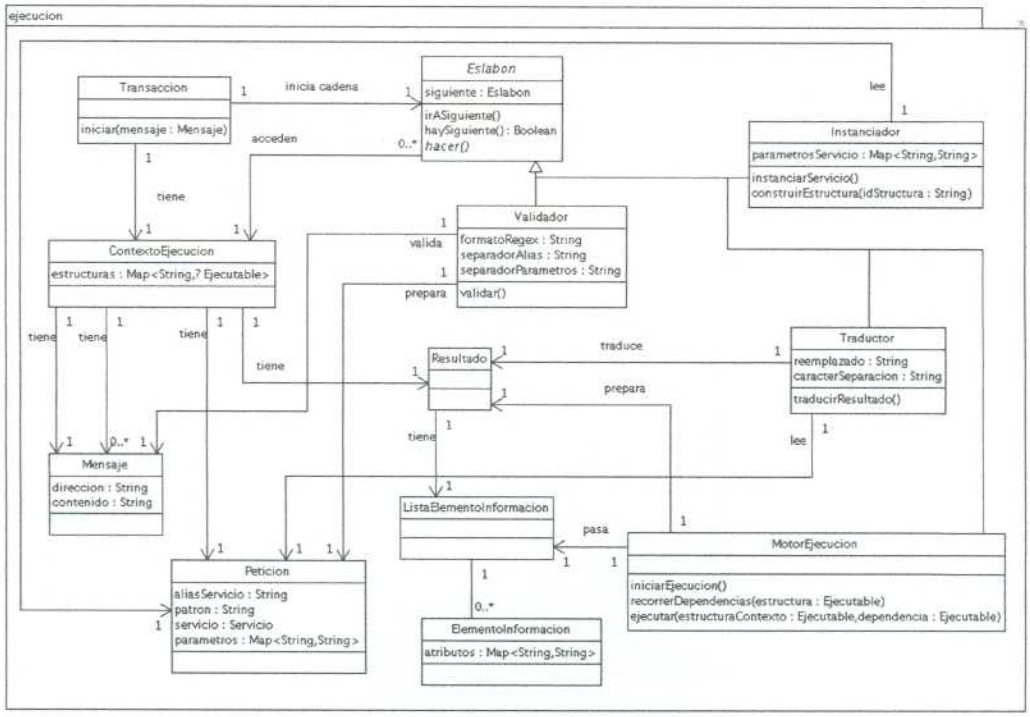
B.2 Estructuras



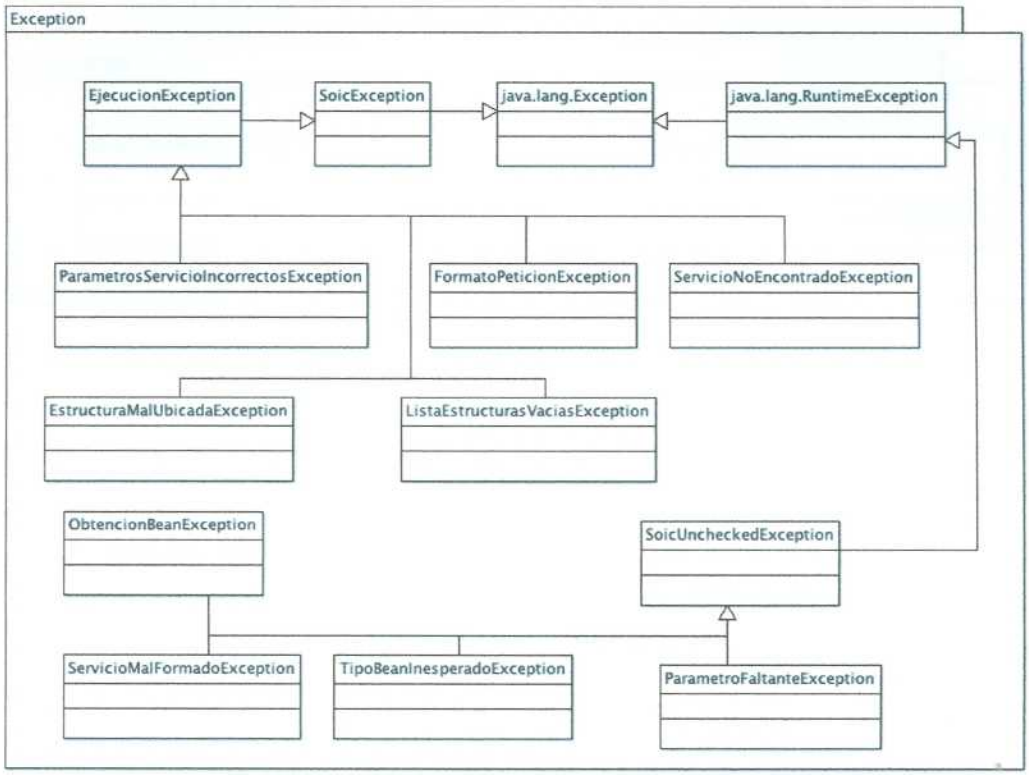
B.3 Criterios



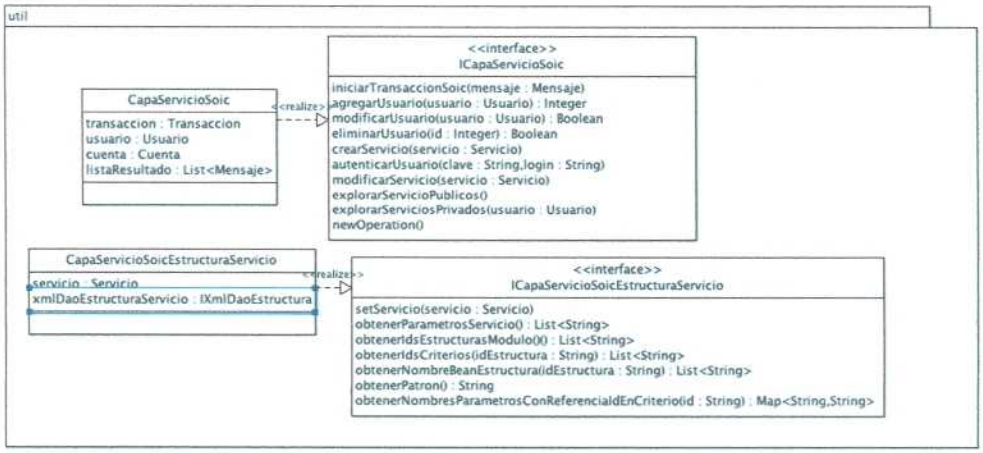
B.4 Ejecución



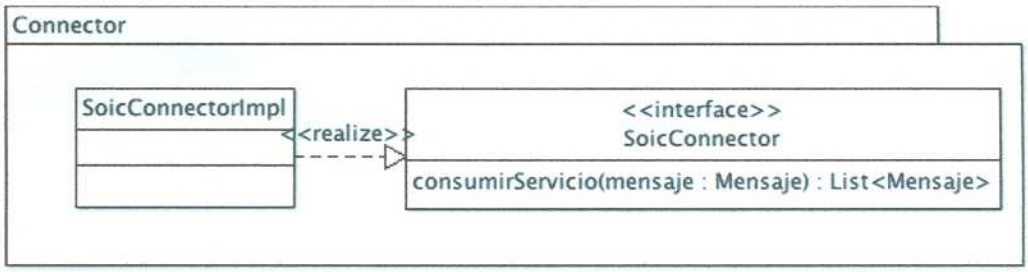
B.5 Excepciones



B.6 Capa de Servicio

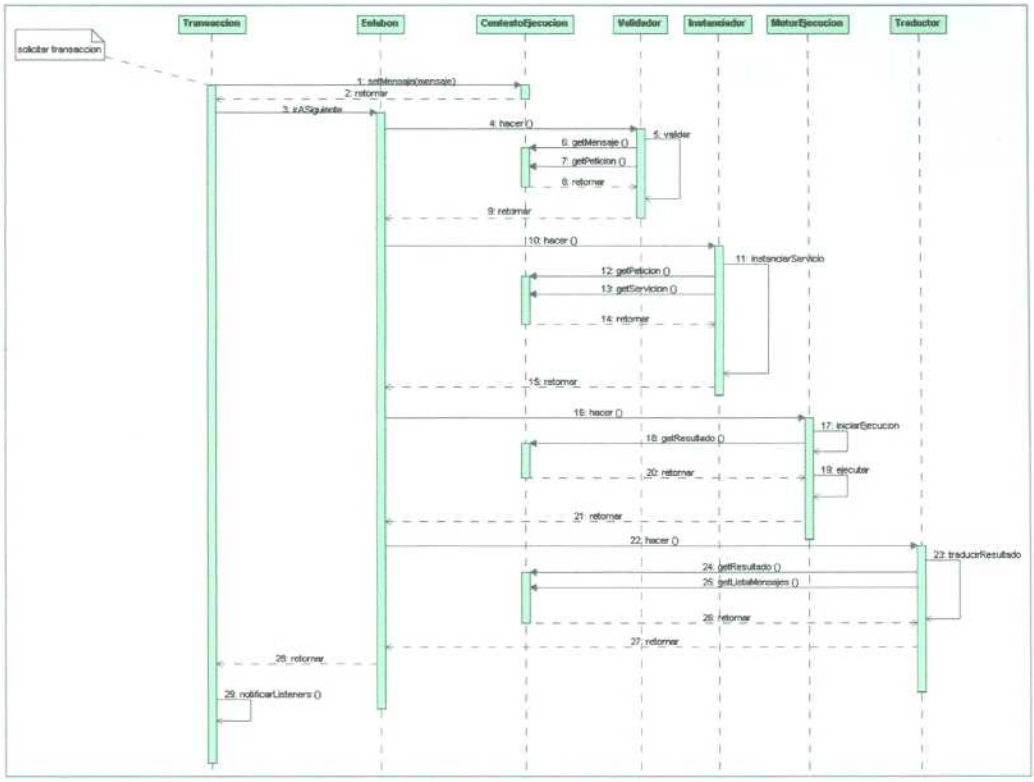


B.7 Conector

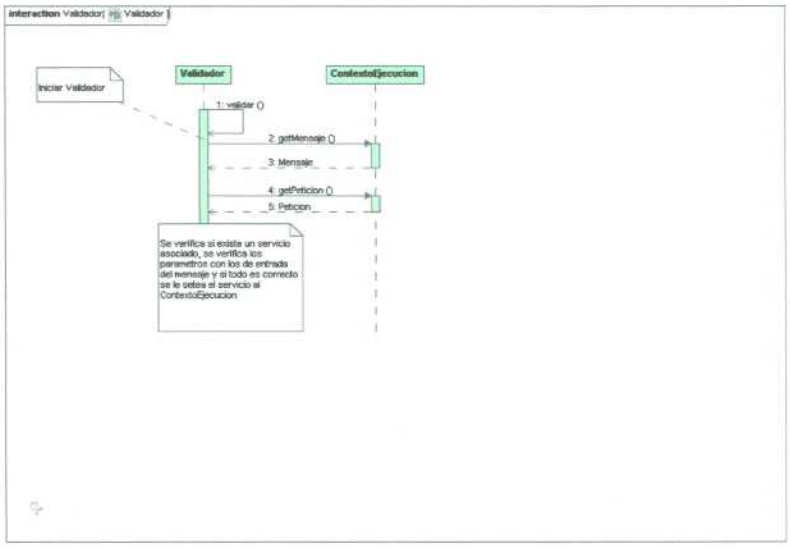

Connector
SoicConnectorImpl
<<realize>>
<<interface>>
SoicConnector
consumirServicio(mensaje : Mensaje) : List<Mensaje>

APÉNDICE "C": DIAGRAMAS DE SECUENCIAS

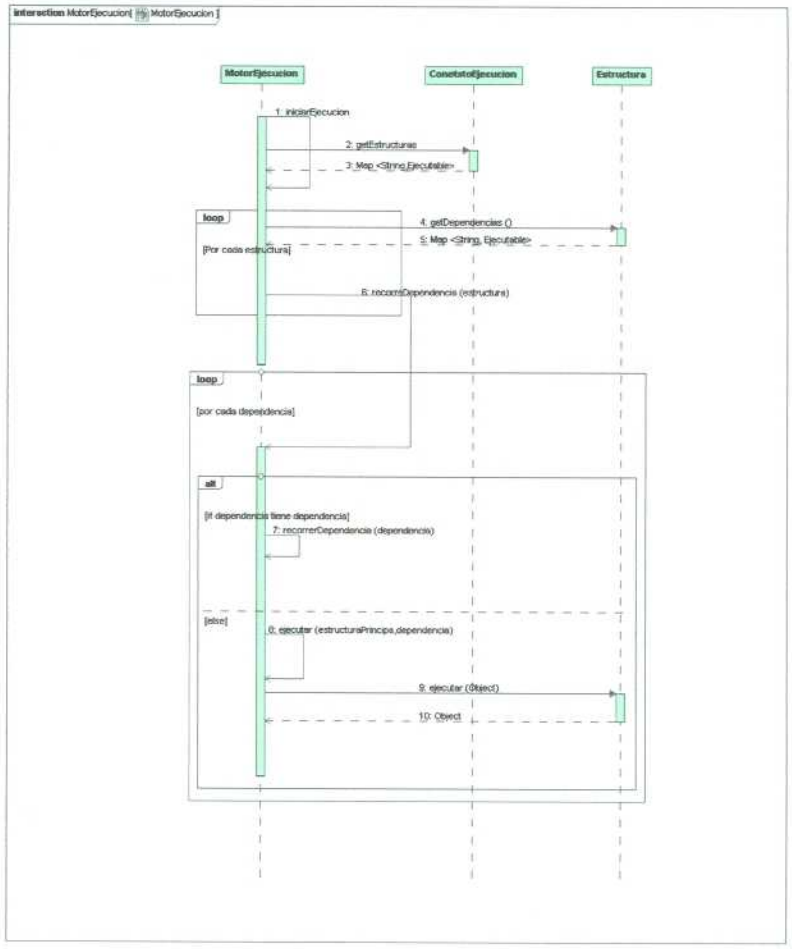
C.1 Ejecutar Servicios



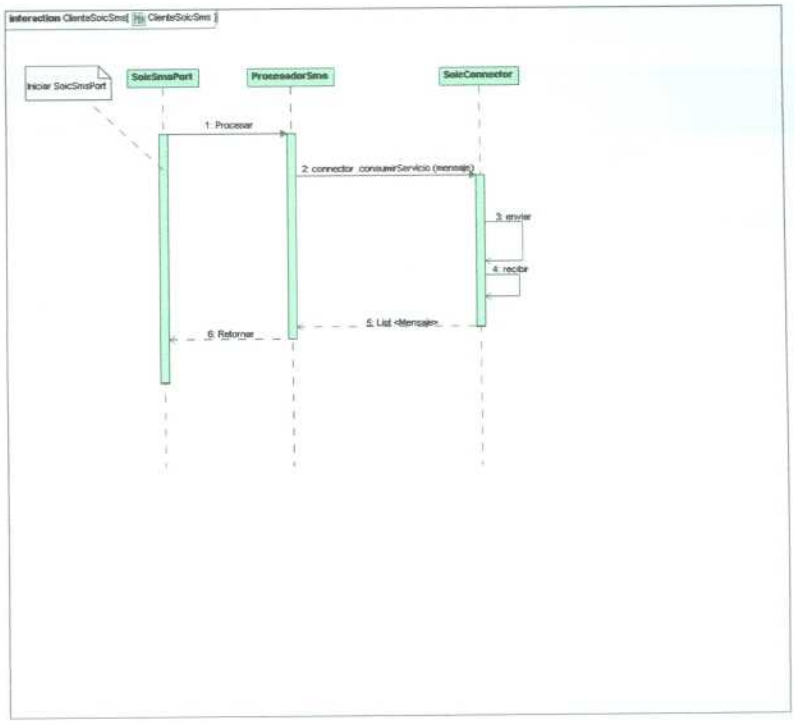
C.2 Validador



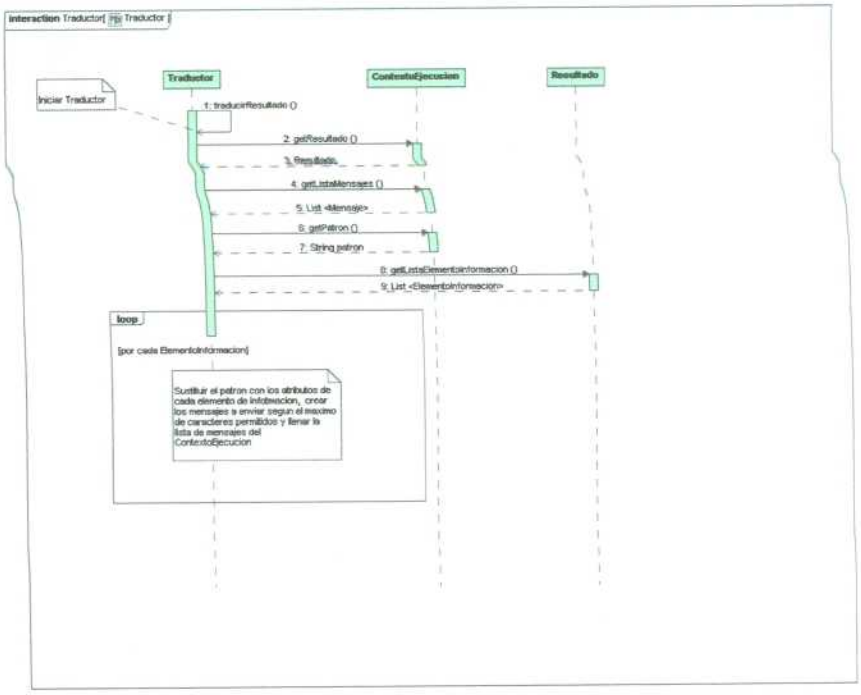
C.3 Motor de Ejecución



C.4 Traductor



C.5 Conector



APÉNDICE "D": PRUEBAS FUNCIONALES DEL SISTEMA

D.1 Explorar servicio

Introducción/visión general	
Nombre del caso de prueba	Explorar servicio
Caso de uso de referencia	CU-1
Propósito	Sirve para comprobar que la búsqueda de los servicios sea exitosa
Actividad de los casos de prueba	
Acciones	• Paso 1: seleccionar la opción explorar del menú principal. • Paso 2: escribir el termino "Orlando" en el campo de búsqueda de la pagina de explorar servicios. • Paso 3: escribir otro termino "Deportes" en el campo de búsqueda de la pagina de explorar servicios
Descripción de los datos de entrada	"Orlando" es el nombre de un usuario "Deportes" es una clasificación de los servicios
Resultados	
Resultados esperados	Para el primer caso se espera que el sistema muestre todos los servicios públicos creados por "Orlando" y en el segundo caso todos los servicios como categoría "Deportes"
Resultados reales	Correcto

D.2 Registrar usuario

Introducción/visión general	
Nombre del caso de prueba	Registrar usuario
Caso de uso de referencia	CU-2
Propósito	Sirve para comprobar que se pueden registrar usuarios en el

	sistema
Actividad de los casos de prueba	
Acciones	• Paso 1: seleccionar la opción registrar del menú principal • Paso 2: llenar el formulario con los siguientes datos: Nombre: orlando Apellido: rauseo Sexo: hombre Fecha nacimiento: 01/10/1985 Email: orlando_0110@hotmail.com Num celular: 04242222460 Login: oark0110 Clave: 123456 • Paso 3: enviar los datos presionando el botón de enviar.
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Se espera recibir una confirmación exitosa por parte del sistema.
Resultados reales	Correcto

D.3 Iniciar Sesión

Introducción/visión general	
Nombre del caso de prueba	Iniciar Sesión
Caso de uso de referencia	CU-3
Propósito	Sirve para comprobar que los usuarios pueden iniciar sesión en el sistema

Actividad de los casos de prueba	
Acciones	• Paso 1: escribir el login "oark0110" y la clave "123456" en la sección de iniciar sesión de la página principal • Paso 2: enviar los datos.
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Entrar en pantalla personal del usuario que esta intentando acceder al sistema.
Resultados reales	Correcto

D.4 Subscribirse a servicio

Introducción/visión general	
Nombre del caso de prueba	Subscribirse a servicio
Caso de uso de referencia	CU-4
Propósito	Sirve para comprobar los usuarios puedan subscribirse a cualquier servicios publico
Actividad de los casos de prueba	
Acciones	• Paso 1: iniciar sesión • Paso 2: seleccionar la opción explorar del menú principal • Paso 3: escribir el criterio "deportes" en el campo de búsqueda • Paso 4: seleccionar la opción subscribir del servicio deseado. • Paso 5: escribir el alias "clasificaciones" al servicio. • Paso 6: presionar la opción aceptar.
Descripción de los	

datos de entrada	
Resultados	
Resultados esperados	Se espera que el sistema confirme exitosamente que la suscripción del servicio.
Resultados reales	Correcto

D.5 Crear servicio

Introducción/visión general	
Nombre del caso de prueba	Crear servicio
Caso de uso de referencia	CU-5
Propósito	Sirve para comprobar que el sistema permite la creación de nuevos servicios
Actividad de los casos de prueba	
Acciones	• Paso 1: iniciar sesión. • Paso 2: seleccionar la opción crear servicio del menú de la pagina personal del usuario. • Paso 3: usar las estructuras disponibles en la paleta del editor para crear el script del servicio • Paso 4: colocar el alias del servicio, el nombre y seleccionar la opción publicar. • Paso 5: guardar el servicio.
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Se espera recibir confirmación exitosa del sistema.
Resultados reales	Correcto

D.6 Consumir servicio

Introducción/visión general	
Nombre del caso de prueba	Consumir servicio
Caso de uso de referencia	CU-6
Propósito	Sirve para comprobar el consumo de los servicios a través de los mensajes SMS y verificar la ejecución de los servicios
Actividad de los casos de prueba	
Acciones	• Paso 1: escribir un mensaje con el siguiente contenido: "clasificaciones:5,a" • Paso 2: enviar el mensaje al número designado al sistema.
Descripción de los datos de entrada	"clasificaciones" es el alias del servicio, el "5" un número a truncar, "a" una letra para filtrar exclusivamente
Resultados	
Resultados esperados	Se espera recibir dos mensajes que muestren 5 equipos de la clasificación de la liga española de fútbol ordenados alfabéticamente y que existiera la letra "a" en su nombre.
Resultados reales	Correcto

D.7 Modificar servicio

Introducción/visión general	
Nombre del caso de prueba	Modificar servicio
Caso de uso de referencia	CU-7
Propósito	Sirve para comprobar los usuarios puedan modificar los servicios

Actividad de los casos de prueba	
Acciones	• Paso 1: iniciar sesión • Paso 2: buscar el servicio en la sección personal • Paso 3: seleccionar la opción modificar • Paso 4: modificar el servicio. • Paso 5: seleccionar guardar.
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Se espera que el sistema confirme exitosamente la modificación del servicio.
Resultados reales	Correcto

D.8 Eliminar servicio

Introducción/visión general	
Nombre del caso de prueba	Eliminar servicio
Caso de uso de referencia	CU-8
Propósito	Sirve para comprobar los usuarios puedan eliminar los servicios
Actividad de los casos de prueba	
Acciones	• Paso 1: iniciar sesión • Paso 2: buscar el servicio en la sección personal • Paso 3: seleccionar la opción eliminar
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Se espera que el sistema confirme exitosamente la eliminación del servicio.
Resultados reales	Correcto

D.9 Publicar servicio

Introducción/visión general	
Nombre del caso de prueba	Publicar servicio
Caso de uso de referencia	CU-9
Propósito	Sirve para comprobar los usuarios puedan publicar servicios
Actividad de los casos de prueba	
Acciones	• Paso 1: iniciar sesión • Paso 2: buscar el servicio en la sección personal • Paso 3: seleccionar la opción publicar
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Se espera que el sistema confirme exitosamente la publicación del servicio y luego pueda ser accedido en la sección de explorar servicios públicos.
Resultados reales	Correcto

D.10 Cambiar alias del servicio

Introducción/visión general	
Nombre del caso de prueba	Cambiar alias del servicio
Caso de uso de referencia	CU-13
Propósito	Sirve para comprobar los usuarios puedan modificar el alias de los servicios
Actividad de los casos de prueba	
Acciones	• Paso 1: iniciar sesión • Paso 2: buscar el servicio en la sección personal • Paso 3: hacer clic sobre el alias actual y cambiarlo

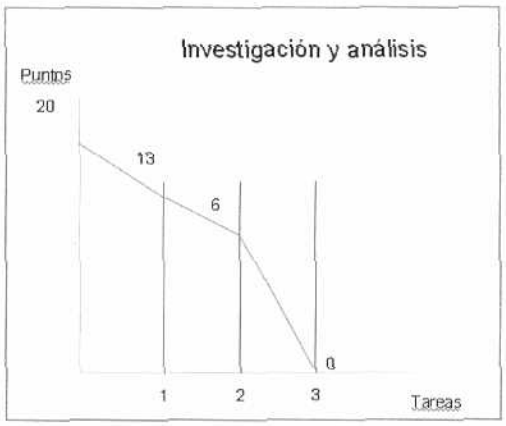
Descripción de los datos de entrada	
Resultados	
Resultados esperados	Se espera observar el nuevo alias del servicio
Resultados reales	Correcto

APÉNDICE "F": ITERACIONES

En cada iteración se asignaron 20 puntos para ser quemados, repartidos entre cada actividad.

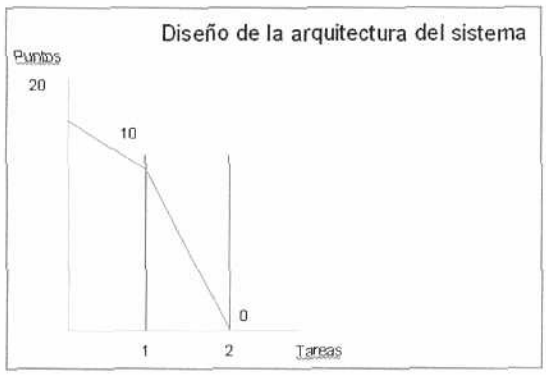
[Iteración 1]: Investigación y análisis:

- Investigar sobre *Spring, Hibernate* y *Struts.* Puntos asignados: 7.
- Investigar sobre Web 2.0 y Web *Mashups.* Puntos asignados: 7.
- Investigar sobre patrones de diseños de software y metodologías de desarrollo de software. Puntos asignados: 6.



[Iteración 2]: Diseño de la arquitectura del sistema:

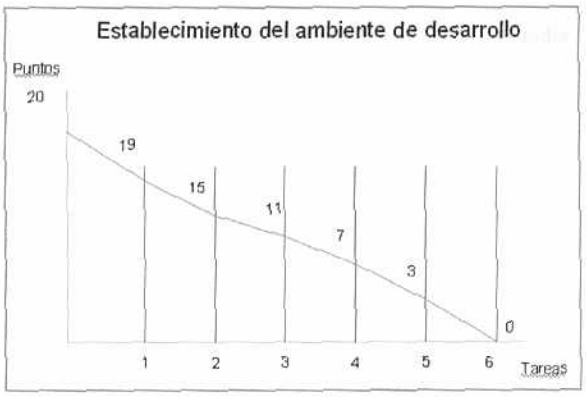
- Realizar los diagramas de casos de uso. Puntos asignados: 10.
- Realizar los diagramas de clases. Puntos asignados: 10.



[Iteración 3]: Establecimiento del ambiente de desarrollo:

- Instalar y configurar el servidor *Apache Tomcat.* Puntos asignados: 1.
- Instalar y configurar *Struts.* Puntos asignados: 4.

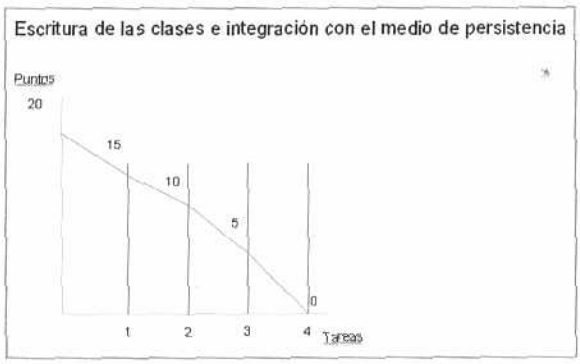
- Instalar y configurar *Spring*. Puntos asignados: 4.
- Instalar y configurar *Hibernate*. Puntos asignados: 4.



- Instalar y configurar *GWT*. Puntos asignados: 4.
- Integrar los Framework. Puntos asignados: 3.

[Iteración 4]: Escritura de las clases e integración con el medio de persistencia:

- Escribir los esqueletos de las entidades. Puntos asignados: 5.
- Escribir las Estructuras. Puntos asignados: 5.

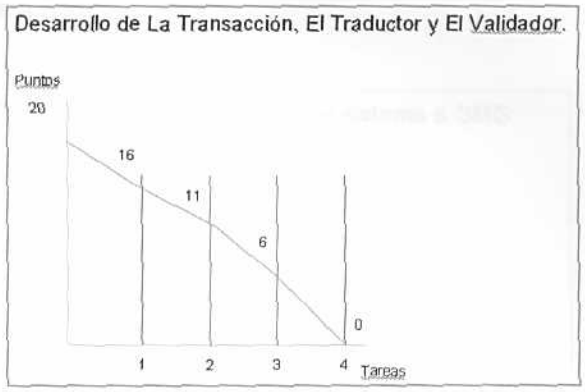


- Escribir as clases de Ejecución. Puntos asignados: 5.
- Mapear las entidades. Puntos asignados: 5.

[Iteración 5]: Desarrollo de La Transacción, El Traductor y El Validador:

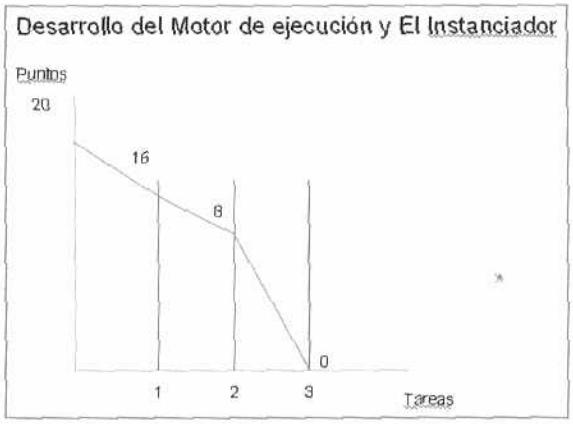
- Desarrollar las pruebas unitarias para cada uno. Puntos asignados: 4.
- Desarrollar la funcionalidad de la transacción. Puntos asignados: 5.

- Desarrollar la funcionalidad del traductor. Puntos asignados: 5.
- Desarrollar la funcionalidad del validador. Puntos asignados: 6.



[Iteración 6]: Desarrollo del Motor de ejecución y El Instanciador:

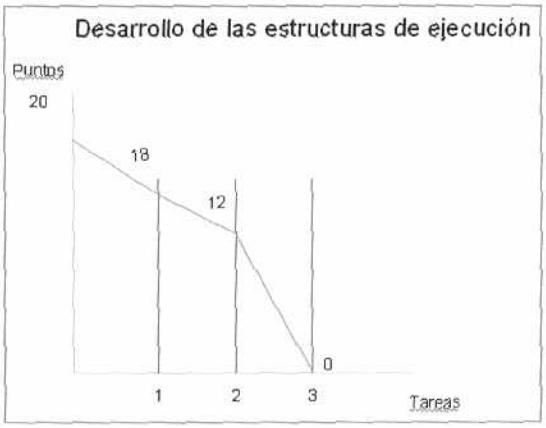
- Desarrollar las pruebas unitarias para cada uno. Puntos asignados: 4.
- Desarrollar la funcionalidad del Motor de ejecución. Puntos asignados: 8.



- Desarrollar la funcionalidad del Instanciador. Puntos asignados: 8.

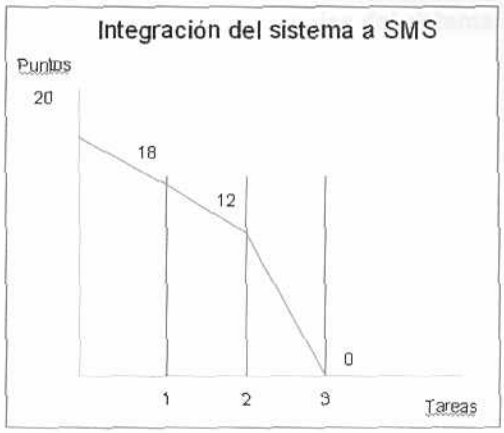
[Iteración 7]: Desarrollo de las estructuras de ejecución:

- Revisión del diseño. Puntos asignados: 2.
- Desarrollo de las pruebas unitarias. Puntos asignados: 6.
- Desarrollo de las estructuras Puntos asignados: 12.



[Iteración 8]: Integración del sistema a SMS:

- Seleccionar la librería necesaria para la integración con los módem SMS. Puntos asignados: 2.

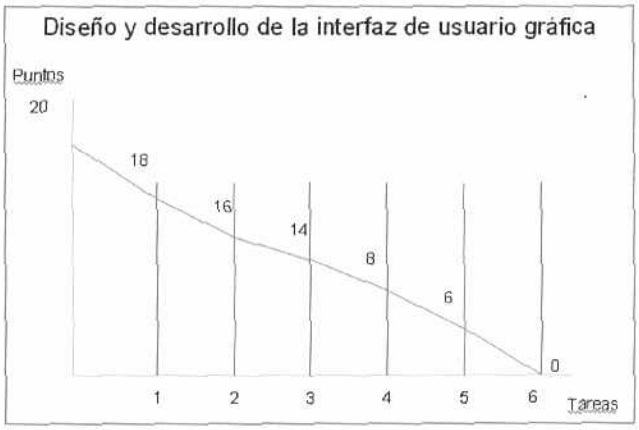


- Desarrollar la capa de conexión "SoicConnector" con el sistema. Puntos asignados: 6.

- Desarrollar el cliente para la recepción y envio de mensajes "SociSmsPort". Puntos asignados: 12.

[Iteración 9]: Diseño y desarrollo de la interfaz de usuario gráfica:

- Escoger la plantilla. Puntos asignados: 2.



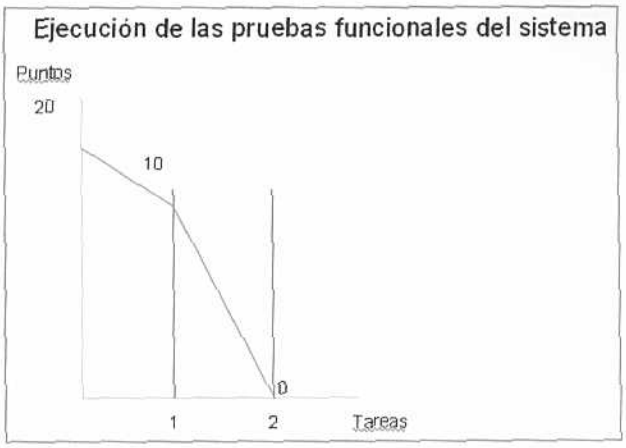
- Configurar Tíles: Puntos asignados: 2.

- Desarrollar registro de usuarios. Puntos asignados: 2.

- Desarrollar Explorar Servicios. Puntos asignados: 6.

- Desarrollar Tutoriales y Foro. Puntos asignados: 2.

- Desarrollar Sección Personal y Editor de Servicios. Puntos asignados: 6.

[Iteración 10]: Ejecución de las pruebas funcionales del sistema:

- Seleccionar datos de pruebas. Puntos asignados: 10.
- Ejecutar Pruebas: Puntos asignados: 10.