UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

COMPONENTE MENSAJERO DE ANGELIA

TRABAJO INSTRUMENTAL DE GRADO

Presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR	Bastos Riccio, Hernando
TUTOR EMPRESARIAL	Magurno, Carlo
TUTOR ACADÉMICO	Blomer, Simy
FECHA	Caracas, 25 de Octubre de 2010

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Este Jurado; una vez realizado el examen del presente trabajo ha evaluado su contenido con el resultado: DIECINUEVE (19)

J U R A D O E X A M I N A D O R

Firma:	Firma:	Firma:
Nombre: Carlo Magurno	Nombre: Simy Blomer	Nombre: Oswel Sanchez

REALIZADO POR	Bastos Riccio, Hernando
TUTOR EMPRESARIAL	Magurno, Carlo
TUTOR ACADÉMICO	Blomer, Simy
FECHA	Caracas, 25 de Octubre de 2010

ÍNDICE DE CONTENIDOS

SINOPSIS 1

CAPITULO I. PROBLEMA 2

I.1 NECESIDADES DE LA EMPRESA 2

I.2 SOLUCIÓN PROPUESTA 3

I.3 OBJETIVO GENERAL 4

I.4 OBJETIVOS ESPECÍFICOS 4

I.5 ALCANCE Y LIMITACIONES 5

I.5 JUSTIFICACIÓN 6

II.1 ANGELIA 7

II.1.1 COMPONENTE VIGILANTE. 9

II.2 ODBC 10

II.2.1 COMPONENTES DE ODBC 11

II.2.2 ODBC DRIVER MANAGER 11

II.2.3 ODBC DRIVER 13

II.3 XPATH 14

II.3.1 SINTAXIS DE XPATH 14

II.4 PLANIFICACIÓN DE TAREAS 18

II.4.1 IMPORTANCIA DE LA AUTOMATIZACIÓN EN LA PLANIFICACIÓN DE TAREAS 18

II.5 FRAMEWORK QUARTZ 20

II.5.1 QUARTZ JOB 21

II.5.2 QUARTZ TRIGGER 21

II.5.2.1 QUARTZ SIMPLETRIGGER 21

II.5.2.2 QUARTZ CRONTRIGGER 22

II.6 EXPRESIONES CRON 22

II.6.1 CARACTERES ESPECIALES 23

II.6.2 EJEMPLOS DE EXPRESIONES CRON 25

II.7 INYECCIÓN DE DEPENDENCIAS 25
II.8 FRAMEWORK SPRING 27
II.8.1 COMPONENTES DE SPRING 27
CAPITULO III. MARCO METODOLÓGICO 29
III. EL PROCESO 29
III.2 ROLES 32
III.2.1 DUEÑO DEL PRODUCTO 32
III.2.2 GERENTE DE PROYECTO 32
III.2.3 EQUIPO 32
III.3 FLUJO DE TRABAJO 34
III.3.1 ACTIVIDADES DE PLANIFICACIÓN 34
III.3.1.1 CREACIÓN O ACTUALIZACIÓN DE LA LISTA DE REQUERIMIENTOS (PRODUCT BACKLOG). 34
III.3.1.2 REUNIÓN DE PLANIFICACIÓN 35
III.3.2 ACTIVIDADES DE DESARROLLO 36
III.3.2.1 REUNIÓN DIARIA 36
III.3.2.2 CREACIÓN DEL PROTOTIPO DE INTERFAZ 36
III.3.3.3 TEST DRIVEN DEVELOPMENT (TDD) 37
III.3.3.4 INTEGRACIÓN CONTINUA 38
III.3.3.5 CONTROL DE VERSIONES 39
III.3.3.4 REVISIÓN DE CÓDIGO 39
III.4 ARTEFACTOS 39
III.4.1 LISTA DE REQUERIMIENTOS (PRODUCT BACKLOG) 39
III.4.2 DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS DEL SOFTWARE 40
III.4.3 LISTA DE FUNCIONALIDADES (RELEASE BACKLOG) 40
III.4.4 LISTA DE TAREAS (SPRINT BACKLOG) 41
III.4.5 DOCUMENTO DE ARQUITECTURA DEL SOFTWARE 41
III.4.6 PROTOTIPO DE INTERFAZ 41
III.4.7 PRUEBAS 42
III.4.8 INCREMENTO FUNCIONAL (RELEASE) 42

CAPITULO IV. DESARROLLO 43

IV.1 PRIMERA REUNIÓN DE PLANIFICACIÓN 43

IV.2 PRIMERA ITERACIÓN 45

IV.3 SEGUNDA REUNIÓN DE PLANIFICACIÓN 49

IV.4 SEGUNDA ITERACIÓN 49

IV.5 TERCERA REUNIÓN DE PLANIFICACIÓN 51

IV.6 DISEÑO DEL COMPONENTE EXTRACTOR 52

IV.7 CUARTA REUNIÓN DE PLANIFICACIÓN 55

IV.8 TERCERA ITERACIÓN 56

IV.9 QUINTA REUNIÓN DE PLANIFICACIÓN 56

IV.10 CUARTA ITERACIÓN 57

IV.11 SEXTA REUNIÓN DE PLANIFICACIÓN 61

IV.12 QUINTA ITERACIÓN 61

IV.13 SÉPTIMA REUNIÓN DE PLANIFICACIÓN 65

CAPITULO V. RESULTADOS 66

CAPITULO VI CONCLUSIONES Y RECOMENDACIONES 70

CAPITULO VII BIBLIOGRAFÍA 72

APÉNDICE I - FRAGMENTO DE LA MATRIZ DE CAMBIOS REALIZADOS AL SISTEMA ANGELIA 74

APÉNDICE II - LISTA DE REQUERIMIENTOS DEL SISTEMA 79

APÉNDICE III - CAPTURAS DE PANTALLA DEL SISTEMA 82

ÍNDICE DE FIGURAS

Figura 1 Diagrama de arquitectura del Vigilante 10

Figura 2 Interacción entre los componentes ODBC 11

Figura 3 Ejemplo de un documento XML 16

Figura 4 Relación entre el beneficio de la automatización y el incremento de complejidad del trabajo 19

Figura 5 Diagrama de clases de Quartz (solo componentes de mayor importancia). 20

Figura 6 Ejemplo de dependencia 26

Figura 7 Inyección de dependencia mediante un inyector. 27

Figura 8 Componentes de Spring 28

Figura 9 Diagrama de arquitectura administrador de planificación. 45

Figura 10 Prototipo de interfaz de ingreso de opciones de planificación 47

Figura 11 Prototipo de interfaz modificación de opciones de planificación 48

Figura 12 Proceso de monitoreo de activadores 50

Figura 13 Proceso de monitoreo y actualización de activadores 51

Figura 14 Bosquejo inicial componente extractor 52

Figura 15 Diagrama inicial de clases. 53

Figura 16 Prototipo de interfaz para asociar taxonomías a la plantilla 58

Figura 17 . Proceso de relacionar en el componente extractor. 59

Figura 18 Formato archivo .angelia 62

Figura 19 Ejemplo de configuración de canales. 63

ÍNDICE DE TABLAS

Tabla 1. Relación entre URL y expresiones XPath. 15

Tabla 2 Ejemplo de sintaxis de XPath 17

Tabla 3 Campos en expresiones cron 22

Tabla 4 Ejemplos de expresiones cron 25

CAPITULO I. PROBLEMA

I.1 NECESIDADES DE LA EMPRESA

Dado el alto nivel de fraude en las transacciones electrónicas de nuestro país, el ente regulatorio del área financiera denominado SUDEBAN, se ha visto en la necesidad de exigir por ley algún mecanismo de alerta inmediata para el consumidor. A partir de esta ley los bancos se han visto en la necesidad de implementar estos mecanismos de alerta temprana al usuario.

A partir de esta resolución, los bancos y otras instituciones financieras han optado por utilizar el canal SMS como un canal barato para la notificación y envío de alertas tempranas por transacciones. En tal sentido, fue desarrollado el único sistema de gestión de mensajería multicanal en Venezuela denominado Angelia, cubriendo así las necesidades impuesta por ley de las instituciones financieras y logrando minimizar el fraude en un periodo de tiempo relativamente corto en aquellas instituciones que poseen la aplicación.

Angelia posee un roadmap o mapa de funcionalidades a desarrollar a futuro. Una de estas funcionalidades es la integración con las áreas de negocios de una corporación a través del uso de Taxonomías corporativas. Tales Taxonomías están relacionadas con los archivos de mensajería generados por las áreas de negocios corporativas. Estos archivos generados tienen la caracteriza de que su fecha de envío debe ser planeada a través de un planificador que provea varias opciones de planificación.

I.2 SOLUCIÓN PROPUESTA

Con la finalidad de alcanzar los objetivos planteados en el proyecto es necesario el diseño e implementación del componente de extracción de datos el cual debe ser capaz de poder extraer la información requerida sobre cualquier fuente de datos. Adicionalmente el componente permitirá la integración de nuevas fuentes de datos según requiera el caso (Ejemplo: Se puede agregar las hojas de cálculo como nueva fuente de datos). Sobre la fuente de datos es requerida la realización de taxonomías, las cuales categorizan los datos permitiendo crear o configurar una consulta en particular. Por lo tanto se desarrollara el componente de administración de taxonomías (CAT) el cual permitirá la creación de nuevas taxonomías (es decir nuevas clasificaciones sobre los datos) y aplicarlas para realizar una consulta, adicionalmente proveerá la funcionalidad de modificar o eliminar taxonomías.

El resultado de las consultas realizadas por el componente de administración de taxonomías permitirá crear archivos de mensajería, estos archivos se caracterizan por contener un gran número de personas a las cuales es necesario enviar mensajes. Sin embargo, como se requiere que la fecha de envió de estos archivos pueda ser planificada, se desarrollará un componente que permita planificar la ejecución de procesos permitiendo varias opciones (semanal, mensual, etc.). Este componente será el encargado de validar los procesos que deben iniciarse según la planificación asignada a dicho proceso.

Para el envió de los archivos de mensajería la empresa Trascend dispone del componente denominado Vigilante, debido a que las consultas realizadas por el CAT generan archivos de mensajería, es necesario una comunicación entre el CAT y el

componente vigilante. Para realizar dicha comunicación CAT generará un archivo de texto plano indicando los destinatarios y el contenido de los mensajes a enviar. Al componente Vigilante se le incorporará la característica de interpretar los archivos generados por el CAT y enviarlos por el canal de mensajería.

Finalmente una vez desarrollados los componentes (administración de taxonomías, planificación y el de extracción de datos) se procederá a integrarlos al sistema Angelia.

I.3 OBJETIVO GENERAL

Desarrollar un componente que dada una agenda configurable de planificación de procesos, valide los procesos que se deban iniciar, localice y extraiga de una fuente de datos información categorizada, permitiendo la relación de las categorías entre distintas fuentes de datos y el almacenamiento de los resultados para auditoria.

I.4 OBJETIVOS ESPECÍFICOS

- Desarrollar el componente de administración de Taxonomías corporativas.
- Diseñar e implementar la relación entre el componente de la administración de Taxonomías y el componente denominado Vigilante.
- Desarrollar el componente de administración de planificación de ejecución de procesos para las siguientes opciones de planificación:
    - Anual por día de semana.
    - Anual recurrente.
    - Diario recurrente.
    - Diario por el tipo de día de semana.
    - Fecha concreta fija.
    - Mensual por día de semana.

- Mensual Recurrente.
- Semanal.

- Desarrollar el componente de extracción de datos, flexible a cualquier fuente de datos (Manejador de Base de datos Relacional, Servicios Web, Servicios AS400, Archivos Planos, Archivos XML).
- Modificar el componente Vigilante para aceptar el manejo de la configuración de las Taxonomías.
- Realizar la integración de los componentes al sistema Angelia®.

I.5 ALCANCE Y LIMITACIONES

- El componente de extracción de datos permitirá consultar las siguientes fuentes de datos:
  - Base de datos relacionales.
  - Servicios Web.
  - Servicios AS400.
  - Archivos planos.
  - Archivos XML.
- El componente de administración de Taxonomías permitirá crear, modificar y eliminar las taxonomías.
- El componente de planificación proveerá de las siguientes funciones de planificación:
  - Anual por día de semana.
  - Anual recurrente.
  - Diario recurrente.

- Diario por el tipo de día de semana.
- Fecha concreta fija.
- Mensual por día de semana.
- Mensual Recurrente.
- Semanal.

- Solo se añadirá al componente Vigilante la capacidad de interpretar los archivos generados por el componente de taxonomías.

I.5 JUSTIFICACIÓN

El sistema Angelia es utilizado por las instituciones bancarias con la finalidad de mandar información a sus clientes por medio de diversos canales, sin embargo, cada empresa contiene su propio modelo de negocio. Estos modelos de negocio varían inclusive entre las empresas que persiguen un mismo fin.

Parte del modelo de negocio indica cómo se almacena la información de sus clientes en los que se puede optar por diversas maneras para realizar el dicho propósito. El sistema Angelia está obligado a ser lo suficientemente flexible como para poder adaptarse a las diferentes formas que se utilizan para guardar la información, evitando la modificación del código fuente cada vez que se requiera instalar la aplicación en alguna empresa. Con la realización del componente extractor se pretende dar dicha flexibilidad al sistema Angelia.

Adicionalmente se pretende incorporar la planificación automatizada para el envío de mensajes que requieran cierta fecha para enviarse, como por ejemplo un mensaje de felicitación a las madres en su día.

CAPITULO II. MARCO TEÓRICO

II.1 ANGELIA

En la documentación del sistema (Magurno, 2008) se define Angelia como:

"Un habilitador tecnológico que permite construir los mensajes e informar o notificar a los clientes, a través de mensajes tipo pull (Clientes→Organización) y/o push (Organización→Clientes), utilizando los distintos canales de comunicación que existan en la organización, priorizando la información a ser recibida por el cliente de acuerdo al canal utilizado. Así mismo, Angelia establece un repositorio y transporte único donde las áreas de negocio corporativas pueden definir estrategias comunicacionales sustentadas por mensajes los cuales serán utilizados para la notificación de clientes externos e internos por los distintos canales, tomando en consideración las limitaciones y restricciones de cada canal.".

Para comprender las funcionalidades de Angelia se deben conocer los siguientes términos, la definición de los mismos fueron extraídos de la documentación del sistema descrita al inicio de esta sección:

- **Origen**. Se refiere a personas o aplicaciones que dan inicio a un mensaje. Puede ser de dos tipos:
    - **Área de Negocio**. Responde a una agrupación de profesionales con fines de negocio dictados por la organización. Estas áreas definen la información que se debe enviar a los clientes. Ejemplo: el área de Mercadeo, puede requerir enviar un mensaje a sus clientes ofreciendo algún nuevo producto.

- **Habilitador.** Son una serie de aplicaciones capaces de poseer o agrupar un conjunto de mensajes sin necesidad de que exista un nivel de estrategia comunicacional.

- **Estrategia**. Es la encargada de agrupar un conjunto de mensajes con características comunes, como por ejemplo contenidos comunes, vigencias, target, etc.

- **Plantilla**. Es la estructura que contiene todo lo necesario para armar un mensaje, es decir, el contenido a enviarse, la estrategia a la que pertenece, el tipo de destinatarios a quienes va dirigido el mensaje, etc.

- **WorkFlow**. Es el proceso de aprobación que dirige a la plantilla a través de un conjunto de estatus para llevarlo, cuando esté finalmente aprobado, al motor de despacho de Angelia donde se aplican las reglas de despacho según formatos, destinatarios, prioridades y canales.

Entre los componentes de Angelia encontramos:

- **Core**: Es el componente a través del cual los usuarios del sistema manipulan la información necesaria para crear las plantillas.

- **Mensajero**: Es el componente que se encarga de verificar e identificar si existe alguna plantilla activa para que el despachador realice el envío del mensaje.

- **Armador**: Como su nombre lo indica, es el encargado de armar el mensaje final que será enviado al destinatario.

- **Integrador**: Cuando el origen del mensaje es un Habilitador, este componente se encarga de realizar la comunicación entre la aplicación que desea enviar el mensaje y Angelia.

- **Despachadores**: Es el encargado de enviar el mensaje al destinatario.

- **Vigilante**: Componente que se encarga de transportar archivos de texto que contienen mensajes, utilizando los servicios FTP o SFTP, también registra el contenido de los archivos como mensajes en la base de datos de Angelia.

En el presente trabajo uno de los objetivos consiste en la integración de los componentes desarrollados con el componente Vigilante. Por lo que a continuación se define con mayor profundidad dicho componente.

II.1.1 COMPONENTE VIGILANTE.

De acuerdo con la documentación del sistema (Magurno, 2008) el componente Vigilante se encarga de transportar archivos de texto que contienen mensajes, utilizando los servicios FTP o SFTP, también registra el contenido de los archivos como mensajes en la base de datos de Angelia. Actualmente soporta un solo tipo de archivos los .txt o .csv que son formatos de Mercurio. Mercurio es la plataforma que permite enviar mensajes cortos de texto (SMS) y mensajes multimedia (MMS) sin importar el proveedor de servicio, el tipo de contenido, ni el tipo de servicio que posea el suscriptor (Documentación sitio web Tedexis). Dicha plataforma ha sido desarrollada por Tedexis la cual es una empresa de soluciones móviles.

A continuación se muestra un diagrama de arquitectura del componente vigilante:

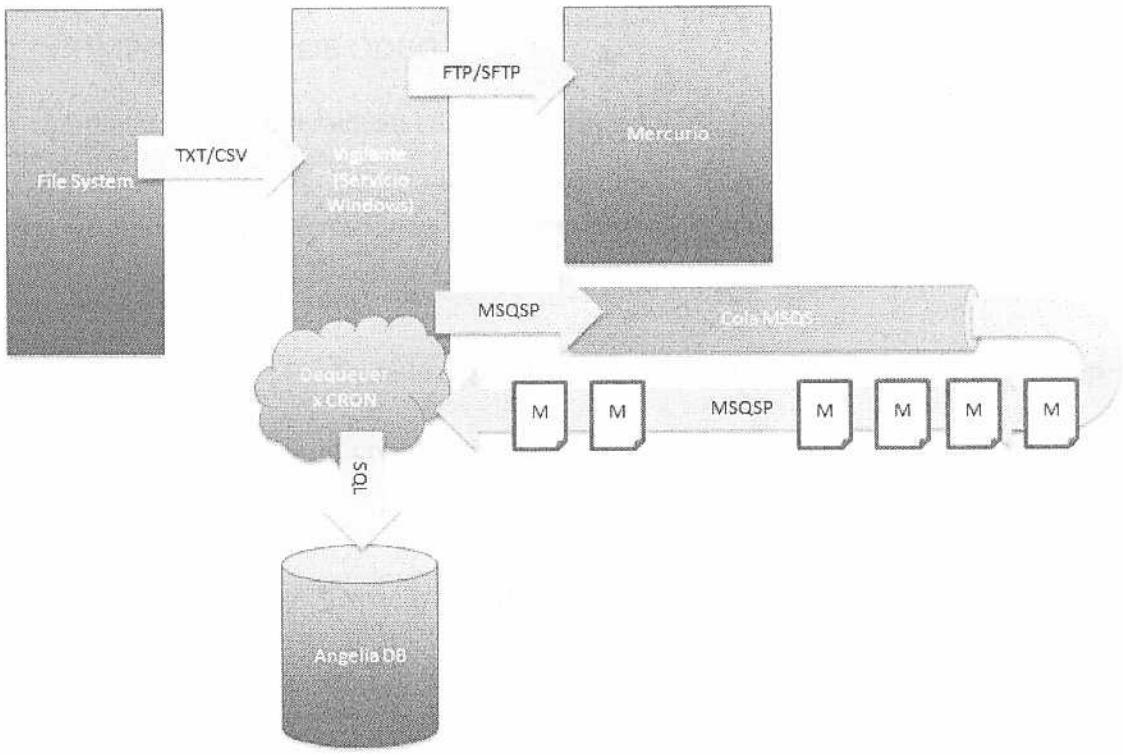

Figura 1 Diagrama de arquitectura del Vigilante

Fuente: Documentación de Angelia. (Magurno, 2008)

II.2 ODBC

Open Database Connectivity (ODBC para abreviar) es definido (Roman, 2002) como una interfaz de programación de aplicaciones o API (por sus siglas en inglés) para la conexión a bases de datos de varios tipos. Una API consiste simplemente en un conjunto de funciones, también llamado servicios, para realizar diversas tareas. Estas funciones normalmente se encuentran en una o varias librerías de enlace dinámico (DLL por sus siglas en inglés). El término de base de datos que utiliza el autor, es en sentido general,

por lo que no solo se refiere a las base de datos relacionales como Access o FoxPro, sino también a las menos tradicionales como por ejemplo hojas de cálculo de Excel.

II.2.1 COMPONENTES DE ODBC

Los componentes que interactúan cuando se utiliza ODBC, son los siguientes:

- La aplicación: Corresponde al programa que necesita extraer o modificar datos en la fuente de datos.
- Drive Manager: Gestiona la comunicación entre el ODBC driver y la aplicación.
- ODBC Driver: Es el componente que conoce como interactuar con la fuente de datos.
- Fuente de datos: Espacio en disco que contiene los datos que se desean manipular.

En la figura 2, se muestra cómo interactúan los componentes mencionados:

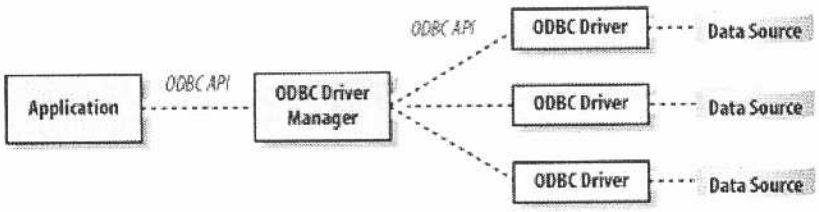

Figura 2 Interacción entre los componentes ODBC

Fuente: Access Database Design & Programming, 3rd Edition. (Roman, 2002)

II.2.2 ODBC DRIVER MANAGER

El ODBC Driver Manager (Roman, 2002) gestiona la comunicación entre la aplicación y el driver. La aplicación se comunica directamente con el Driver Manager, el cual

procesa el comando o lo envía al driver (es posible que la aplicación se comunique directamente con el driver, pero no es lo usual).

En general, el Driver Manager solo pasa las llamadas a las funciones de la API que realiza la aplicación al correspondiente driver, sin embargo, posee implementadas algunas funciones de la API y también realiza algunos chequeos básicos de errores. Particularmente es responsable de implementar las siguientes funciones que brindan información acerca del driver:

- SQLDataSources: Retorna la información sobre la fuente de datos.
- SQLDrivers: Lista de descripción de los drivers y sus atributos.
- SQLGetFunctions: Indica si un driver soporta una determinada función de ODBC.

El Driver Manager también es responsable de la gestión de conexión de un driver ODBC. Particularmente cuando la aplicación desea utilizar un driver en específico llama a uno de los siguientes métodos:

- SQLConnect: Establece la conexión con el driver y la fuente de datos.
- SQLDriverConnect: Establece la conexión usando un string de conexión.
- SQLDriverBrowse: Establece una conexión de forma iterativa.

Cada una de estas funciones debe incluir información acerca del driver en los parámetros. Utilizando esta información el Driver Manager carga el driver (si éste aún no está ya cargado) y llama a la función de conexión adecuada (SQLConnect,

SQLDriverConnect o SQLDriverBrowse) en el driver. Cuando la aplicación termina de utilizar el driver; se llama a SQLDisconnect, luego, el Driver Manager pasa la llamada al driver y éste se desconecta de la fuente.

II.2.3 ODBC DRIVER

El ODBC Driver (Roman, 2002) es un componente de código que implementa las funciones de la API de ODBC. Cada driver es específico para una base de datos en particular. Los driver deben exponer las capacidades subyacentes del sistema gestor de base de datos, pero generalmente las aumentan. La excepción principal son los driver para gestores de base de datos que no poseen motores de base de datos independientes (standalone), como lo es el caso de dBASE, XBase. En el caso del texto ASCII, se debe implementar un sistema de base de datos que soporte un mínimo de instrucciones SQL.

El driver ODBC debe implementar las siguientes tareas (entre otras):

- Conexión y desconexión de la fuente de datos.
- El envío y recuperación de datos desde el origen de datos.
- Comprobación de errores en funciones de la API que no son revisados por el Driver Manager.
- Envío de sentencias SQL al origen de datos para su ejecución. Para ello, el driver quizás tenga que modificar las instrucciones SQL de ODBC a una forma de SQL que el sistema gestor de base de datos entienda.

II.3 XPATH

Muchas empresas utilizan archivos XML (Extensible Markup Language (Lenguaje de Marcas Extensible)) para organizar y clasificar sus datos. XML se define (Dykes & Tittel, 2005) como un lenguaje de marcas que utiliza etiquetas para rotular, clasificar y organizar información de una manera específica. El marcado describe la estructura del documento (o datos) y la organización. El contenido como texto, imágenes y datos son parte del código que contienen las etiquetas.

Un software por sí solo no puede manipular un documento XML, se necesita un lenguaje que permita describir los caminos construidos dentro del documento, para indicar como seguir una ruta a través de éste. El XML Path Languague (Xpath) se encarga de realizarlo, proporcionando un lenguaje conciso, describiendo la ubicación *especifica de elementos, atributos y otros valores que se encuentran en un documento* XML.

II.3.1 SINTAXIS DE XPATH

Una expresión XPath utiliza una notación de rutas como las que se utilizan en direcciones URL, para dirigir partes de un documento XML. La expresión se evalúa para obtener un objeto del conjunto de nodos, un valor booleano, un valor numérico o un tipo de cadena. Por ejemplo, la expresión libro/autor devolverá un conjunto de nodos de los elementos <autor> incluidos en los elementos <libro>, si dichos elementos aparecen declarados en el documento XML de origen. Además, una expresión XPath puede tener predicados (expresiones de filtro) o llamadas de función. Por ejemplo, la expresión libro[@tipo="ficción"] hace referencia a los elementos <libro> cuyo atributo tipo se establece como "ficción".

En sitio web de MSDN (Sintaxis de Xpath) se obtiene la tabla comparativa entre una expresión URL y una expresión XPATH.

URL	Expresiones XPath
Jerarquía compuesta de carpetas y archivos en un sistema de archivos.	Jerarquía compuesta de elementos y otros nodos en un documento XML.
Los archivos de cada nivel tienen nombres únicos. Las direcciones URL siempre identifican un solo archivo.	Es posible que los nombres de los elementos de cada nivel no sean únicos. Las expresiones de XPath identifican un conjunto de todos los elementos coincidentes
Se evalúan en relación con una carpeta concreta, llamada la "carpeta actual".	Se evalúa en relación con un nodo concreto llamado el "contexto" para la expresión.

Tabla 1. Relación entre URL y expresiones XPath.

Fuente: http://msdn.microsoft.com/es-ve/library/ms256471.aspx

Dado el siguiente documento XML de la figura 3, se obtienen algunos ejemplos que aparecen en la tabla 2.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="myfile.xsl" ?>
<bookstore specialty="novel">
 <book style="autobiography">
  <author>
   <first-name>Joe</first-name>
   <last-name>Bob</last-name>
   <award>Trenton Literary Review Honorable Mention</award>
  </author>
  <price>12</price>
 </book>
```

```
<book style="novel" id="myfave">
  <author>
    <first-name>Toni</first-name>
    <last-name>Bob</last-name>
    <award>Pulitzer</award>
    <publication>Still in Trenton</publication>
    <publication>Trenton Forever</publication>
  </author>
  <price intl="Canada" exchange="0.7">6.50</price>
 </book>
</bookstore>
```

Figura 3 Ejemplo de un documento XML.

Fuente: http://msdn.microsoft.com/es-ve/library/ms256086.aspx

Expresión	Hace referencia a
./author	Todos los elementos <author> dentro del contexto actual.
author	Todos los elementos <author> dentro del contexto actual.
first.name	Todos los elementos <first.name> dentro del contexto actual.
/bookstore	El elemento de documento (<bookstore>) de este documento.
//author	Todos los elementos <author> en el documento.
book[/bookstore/@specialty=@style]	Todos los elementos <book> cuyo valor de atributo style es igual al valor de atributo specialty del elemento <bookstore> en la raíz del documento.
author/first-name	Todos los elementos <first-name> que son secundarios de un elemento <author>.
bookstore//title	Todos los elementos <title> uno o más niveles dentro del elemento <bookstore> (descendientes arbitrarios).

bookstore/*/title	Todos los elementos <title> que son secundarios de los elementos <bookstore>.
bookstore//book/excerpt//emph	Todos los elementos <emph> en cualquier lugar dentro de los secundarios <excerpt> de los elementos <book>, en cualquier lugar dentro del elemento <bookstore>.
.//title	Todos los elementos <title> uno o más niveles dentro del contexto actual.
author/*	Todos los elemento secundarios de los elementos <author>.
book/*/last-name	Todos los elementos <last-name> que son secundarios de los elementos <book>.
/	Todos los elementos secundarios de estos del contexto actual.
*[@specialty]	Todos los elementos con el atributo specialty.
@style	El atributo style del contexto actual.
price/@exchange	El atributo exchange en los elementos <price> dentro del contexto actual.
book[@style]	Todos los elementos <book> con atributos style del contexto actual.
book/@style	El atributo style para todos los elementos <book> del contexto actual.
@*	Todos los atributos del contexto de elemento actual.
./first-name	Todos los elementos <first-name> en el nodo de contexto actual.
first-name	Todos los elementos <first-name> en el nodo de contexto actual.
author[1]	El primer elemento <author> en el nodo de contexto actual.
author[first-name][3]	El tercer elemento <author> que tiene un secundario <first-name>.

Tabla 2 Ejemplo de sintaxis de XPath

Fuente: http://msdn.microsoft.com/es-ve/library/ms256086.aspx

II.4 PLANIFICACIÓN DE TAREAS

La planificación de trabajos (Cavaness, 2006) remonta a la época de los mainframes cuando usuarios/programadores entregaban tarjetas perforadas o cintas electromagnéticas (lo que representa un trabajo) a un operador el cual era responsable de ejecutar ese trabajo. Una vez que el trabajo se ejecutara se procedía a tomar los resultados obtenidos.

Debido a que no todos los trabajos requieren ejecución inmediata, los trabajos se programaban para ser ejecutados en el futuro. Por ejemplo el administrador del sistema puede tener una lista de tareas que debe seguir todas las noches:

- 11:00 PM: Ejecutar el reporte de ventas.
- 11:59 PM: Realizar respaldo de la base de datos.

Por lo general (Cavaness, 2006) indica que la planificación de tareas se asocia con el procesamiento por lotes (batch processing). Estos trabajos por lotes suelen ejecutarse en segundo plano y normalmente no requieren mucha interacción con el usuario.

II.4.1 IMPORTANCIA DE LA AUTOMATIZACIÓN EN LA PLANIFICACIÓN DE TAREAS

La importancia de planificar trabajos (Cavaness, 2006) radica en la afirmación "El tiempo es dinero", por lo que dedicar recursos bien pagados a tareas serviles es un desperdicio de dinero y recursos. A medida de que aumenta la complejidad del proceso incrementa el beneficio de automatizar los procesos. Esto se puede apreciar en la figura 4.

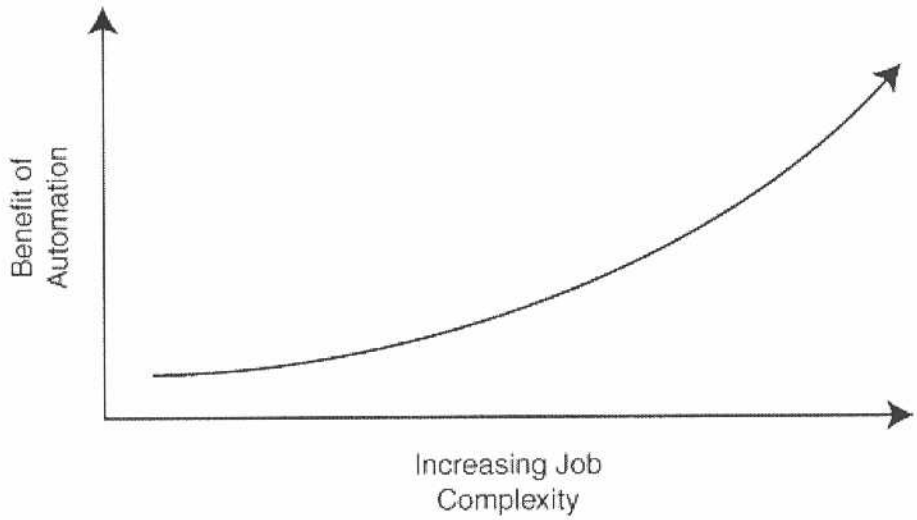

Figura 4 Relación entre el beneficio de la automatización y el incremento de complejidad del trabajo.

Fuente: Quartz Job Scheduling Framework. (Cavaness, 2006)

Los seres humanos están propensos a cometer errores con más frecuencia que un computador, por lo que al automatizar un conjunto de actividades en trabajos y establecer una planificación (con respecto al tiempo) para ejecutar el trabajo, se disminuyen las probabilidades de cometer errores.

Otra ventaja de la planificación automatizada de trabajos es la escalabilidad, un ser humano puede llegar a ser capaz de acomodar 10 o 20 trabajos por hora, pero a medida que se aumenta el número, se hace cada vez más difícil gestionar los trabajos sin introducir errores en los mismos. La planificación automatizada solo se limita con los recursos del hardware.

En resumen (Cavaness, 2006) indica las ventajas de la planificación automatizada como las siguientes:

- Eficiencia en el manejo de recursos.

- Menos errores.
- Más escalabilidad.

II.5 FRAMEWORK QUARTZ

En el sitio oficial del framework (Quartz.NET - Frequently Asked Questions), lo definen como "Un sistema de planificación de tareas que puede ser integrado o utilizado con cualquier sistema de software.". También se indica que "Quartz es bastante flexible y utiliza varios paradigmas que pueden ser utilizados separadamente o en conjunto con el fin de lograr el comportamiento deseado.". En la figura 5 se muestra un diagrama de clases donde aparecen los componentes principales del framework.

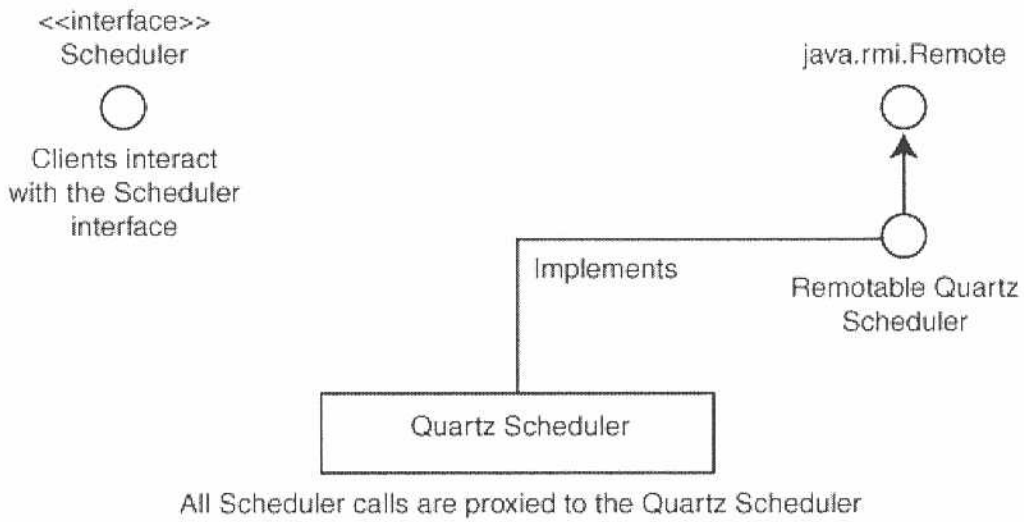

Figura 5 Diagrama de clases de Quartz (solo componentes de mayor importancia).

Fuente: Quartz Job Scheduling Framework. (Cavaness, 2006)

QuartzScheduler es el motor que impulsa todo el framework. No toda la funcionalidad se encuentra dentro de QuartzScheduler. El framework está diseñado para ser flexible y configurable, por lo que muchas de la funciones importantes se encuentran en

componentes separados y sub-frameworks. Esto significa que varios usuarios de Quartz pueden sustituir sus propias versiones en lugar de las que vienen por defecto.

II.5.1 QUARTZ JOB

Los jobs (Cavaness, 2006) son clases que realizan determinada tarea, esta tarea puede ser cualquier elemento que pueda codificarse. A continuación se mencionan algunos ejemplos:

- Obtener una sesión de Hibernate para realizar búsquedas y actualizaciones de base de datos relacionales.
- Llamadas a métodos remotos.
- Usar un cliente HTTP para invocar la URL de una aplicación web.

II.5.2 QUARTZ TRIGGER

Los jobs contienen la lógica, pero no contienen información de cómo ejecutarse en el tiempo indicado. Esta información la conoce el trigger. Existen dos tipos de trigger[1]:

- SimpleTrigger
- CronTrigger

II.5.2.1 QUARTZ SIMPLETRIGGER

SimpleTrigger es el más simple de los trigger, esta designado para jobs que requieren empezar en una fecha y hora en particular y luego repetirse N número de veces con posible tiempo de espera entre cada ejecución.

[1] En Quartz para Java se encuentra implementado el NthIncludedDayTrigger el cual no es objeto de estudio en este trabajo.

II.5.2.2 QUARTZ CRONTRIGGER

Los CronTrigger permiten ser más complejos en sus opciones de planificación. Como su nombre lo indica está basado en las expresiones cron de UNIX. En el caso de que se requiera que un job se ejecute de lunes a viernes cada 5 minutos entre las 8:00 am y las 9:00 am, utilizando expresiones cron conjuntamente con el CronTrigger se puede resolver con un solo trigger que tenga asociada la siguiente expresión cron 0 0/5 8 ? * MON-FRI. En cambio si este requerimiento se emplea utilizando SimpleTrigger quizás se necesiten de la cooperación de varios de estos.

II.6 EXPRESIONES CRON

En sitio web de Quartz (Quartz.NET - Frequently Asked Questions) se definen las expresiones cron como una cadena compuesta por seis o siete campos separados por espacios en blanco. En cada campo se permiten ciertas combinaciones, las cuales involucran caracteres especiales permitidos dentro del mismo. En la tabla 3 se muestran los campos de una expresión cron.

Nombre del campo	**Obligatoria**	**Valores permitidos**	**Caracteres especiales permitidos**
Segundos	SÍ	0-59	, - * /
Minutos	SÍ	0-59	, - * /
Hora	SÍ	0-23	, - * /
Día del mes	SÍ	1-31	, - *? / L W
Mes	SÍ	1.12 o JAN-DEC	, - * /
Día de la semana	SÍ	1.7 o SUN-SAT	, - *? / L #
Año	NO	vacío, 1970-2099	, - * /

Tabla 3 Campos en expresiones cron.

Fuente: Traducción propia del sitio http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html

II.6.1 CARACTERES ESPECIALES

Las expresiones cron permiten el uso de caracteres especiales en algunos de sus campos. A continuación se especifica el propósito de los mismos:

- Asterisco * *("Todos los valores")*: Se usa para seleccionar todos los valores dentro de un campo. Por ejemplo, "*" en el campo de los minutos significa *"cada minuto"*.

- Signo de cierre de interrogación ? *("Ningún valor específico")*: Útil cuando se tiene que especificar algo en uno de los dos campos en los que se permite el carácter, pero no el otro. Por ejemplo, si se quiere que un evento se active en un determinado día del mes (por ejemplo, el día 10), pero no importa qué día de la semana se active, se pone "10" en el campo del día del mes, y "?" en el campo del día de la semana.

- Guion -: Se usa para especificar rangos. Por ejemplo, "10-12" en el campo de la hora, significa *"la hora 10, 11 y 12"*.

- Coma ,: Se usa para especificar valores adicionales. Por ejemplo, "MON, WED, FRY" en el campo del día de la semana indica *"los días lunes, miércoles y viernes"*.

- Barra diagonal /: Se utiliza para especificar incrementos. Por ejemplo, "0/15" en el campo de los segundos indica *"los segundos 0, 15, 30 y 45"*. Y "05/15" en el campo de los segundos indica *"los segundos 5, 20, 35 y 50"*.

- Letra **L** *("último")*: Su significado varía en cada uno de los dos campos en los que está permitido. Por ejemplo, el valor "L" en el campo día del mes indica *"el último día del mes"*. Por ejemplo el día 31 de enero, el día 28 de febrero en años no bisiestos. Si se usa en el campo día de la semana, simplemente significa "7" o "SAT" (Sábado). Pero si se usa el campo día de la semana después de un valor significa *"El ultimo día xxx del mes"* - por ejemplo "6L" significa *"el último viernes del mes"*.

- Letra **W** *("Día de la semana")*: Se usa para indicar el día de la semana (de lunes a viernes) más cercano al día especificado. Por ejemplo, si se coloca "15W" como el valor para el campo día del mes, el significado es: *"el día de la semana más cercano al día 15 del mes"*. Así que si el 15 es día sábado, el evento se activa el viernes 14. Si el 15 es domingo, el evento se dispara el lunes 16. Si el 15 es un martes, entonces se activará ese mismo día. Sin embargo, si se especifica "1W" como el valor para el día de mes, y el primer día del mes es sábado, el evento se activará en el tercer día del mes, el cual es un lunes; debido a que no se puede dar un valor fuera del mes indicado, por último el carácter W solo puede ser utilizado cuando el día del mes es uno solo y no una lista de días.

- Numeral # - Se usa para especificar el "enésimo" xxx día de un mes. Por ejemplo, el valor de "6#3" en el campo del día de la semana significa *"el tercer viernes del mes"* (día 6 equivale al Viernes y "#3" = el 3er día del mes). Otros ejemplos: "2#1" equivale a "*el primer lunes del mes*" y "4#5" equivale a "el quinto miércoles del mes". Si se especifica "#5" y no hay quinto día del mes, entonces no se activará el evento.

II.6.2 EJEMPLOS DE EXPRESIONES CRON

A continuación se presenta una tabla donde se encuentran la expresión cron con su correspondiente significado.

Expresión	**Significado**
0 0 12 * *?	Se activa a las 12h (mediodía) todos los días
0 15 10? * *	Se activa a las 10:15 am todos los días
0 15 10 * *?	Se activa a las 10:15 am todos los días
0 15 10 * *? *	Se activa a las 10:15 am todos los días
0 15 10 * *? 2005	Se activa a las 10:15 am todos los días durante el año 2005
0 * 14 * *?	Se activa cada minuto a partir de las 2:00 pm y termina a las 2:59 pm, todos los días
0 0/5 14 * *?	Se activa cada 5 minutos a partir de las 2:00 pm y finaliza a las 2:55 pm, todos los días
0 0/5 * 14,18 *?	Se activa cada 5 minutos a partir de las 2:00 pm y finaliza a las 2:55 pm, luego se activa cada 5 minutos a partir de las 6:00 pm y termina a las 6:55 pm, todos los días
0 0-5 14 * *?	Se activa cada minuto a partir de las 2:00 pm y termina a las 2:05 pm, todos los días
0 10,44 14 ? 3 WED	Se activa a las 2:10 pm y a las 2:44 pm todos los miércoles en el mes de marzo.
0 15 10 ? * MON-FRI	Se activa a las 10:15 am todos los lunes, martes, miércoles, jueves y viernes
0 15 10 15 * ?	Se activa a las 10:15 am en el día 15 de cada mes
0 15 10 L * ?	Se activa a las 10:15 am en el último día de cada mes
0 15 10 ? * 6L	Se activa a las 10:15 am en el último viernes de cada mes
0 15 10 ? * 6L 2002-2005	Se activa a las 10:15 am el último viernes de cada mes durante los años 2002, 2003, 2004 y 2005
0 15 10? * 6 # 3	Se activa a las 10:15 am en el tercer viernes de cada mes

Tabla 4 Ejemplos de expresiones cron.

Fuente: Traducción propia del sitio http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html

II.7 INYECCIÓN DE DEPENDENCIAS

Antes de definir la inyección de dependencias (DI por sus siglas en inglés) primero hace falta definir el concepto de dependencia entre clases. La dependencia (Stal, 2007) entre

la clase fuente XX y la clase destino YY se da cuando cada clase XX necesita una instancia de YY, por lo que XX necesita saber cómo crear una instancia de YY.

Analizando un ejemplo más práctico consideremos el siguiente fragmento de código mostrado en la figura 6. En el ejemplo la clase A depende de crear o de obtener una instancia de B.

```
public class A {
  public B cb; // A depends on B
  public A() { cb = get_B_Object(...); }
  ...
}
public class B { ... }
public class Container {
  public A createA() {
          return new A();
  }
}
```

Figura 6 Ejemplo de dependencia

Fuente: Diapositivas (Stal, 2007) en la presentación It's Spring again - Lightweight .NET Containers

La inyección de dependencias, es un patrón de diseño, Dhanji R. (Prasanna, 2009) (2009) indica que el objetivo que persigue el patrón es de la clase cliente de conocer sus dependencias y de cómo crearlas. Por lo general la tarea de crear, ensamblar y escribir las dependencias dentro del objeto, es realizada por un framework (o librería) conocida como framework de inyección de dependencias (dependency injection framework). En la figura 7, se ilustra la idea ya mencionada. En algunas ocasiones los frameworks DI se conocen como contenedores IoC. Algunos ejemplos de estos frameworks son PicoContainer (para Java), StructureMap (para C#) y Spring (para Java y C#).

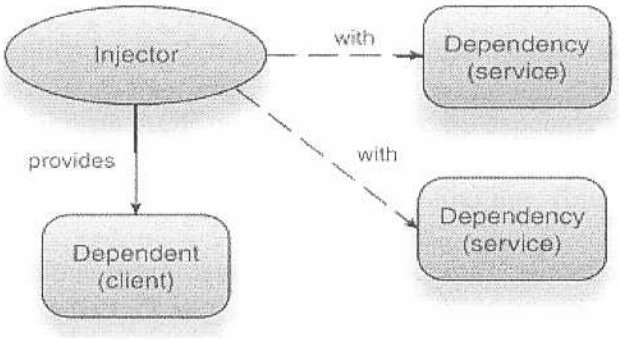

Figura 7 Inyección de dependencia mediante un inyector.

Fuente: Dependency Injection *Design Patterns Using Spring And Guice*. (Prasanna, 2009)

II.8 FRAMEWORK SPRING

Spring.NET es definido (Stal, 2007) como:

- "Un framework de aplicaciones para construir aplicaciones empresariales en la plataforma .NET (Enterprise .NET applications)".
- "Un contenedor IoC (Inversion of Control) que maneja e inyecta dependencias".

II.8.1 COMPONENTES DE SPRING

En el sitio oficial del Framework Spring para .Net (Pollack, et al., 2009) encontramos la descripción de los módulos que lo conforman, a continuación se mencionan los más destacados:

- Spring.Core: Es la parte fundamental del framework, permitiendo configurar nuestra aplicación utilizando la inyección de dependencia (Dependency Injection).

- Spring.AOP: Este módulo da soporte a la programación orientada a aspectos (AOP por sus siglas en inglés). AOP centraliza funcionalidad común que luego puede ser aplicada de forma declarativa dentro de la aplicación en forma selectiva. La librería AOP de Spring provee de aspectos predefinidos los cuales pueden ser utilizados en: transacciones, logging (bitácora), supervisión de rendimiento, almacenamiento de cache (caching), reintento de métodos (method retry) y manejo de excepciones.

- Spring.Data: Provee de la capa de acceso a datos. Adicionalmente facilita el uso de ADO.NET.

- Spring Web: Proporciona la capacidad de DI a ASP.NET.

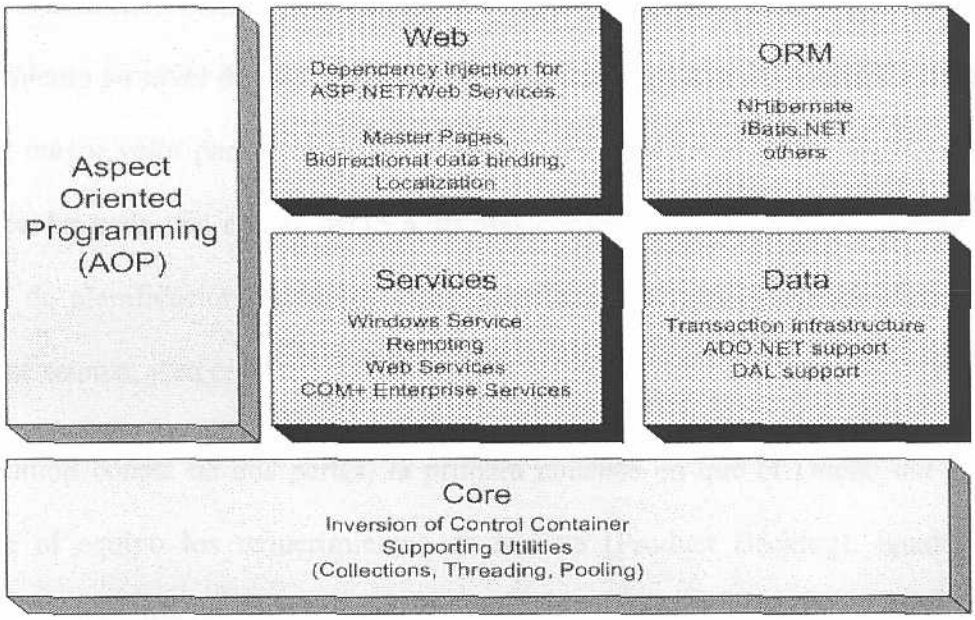

Figura 8 Componentes de Spring

Fuente: Diapositivas (Stal, 2007) en la presentación It's Spring again - Lightweight .NET Containers

CAPITULO III. MARCO METODOLÓGICO

Trascend utiliza la metodología Zen para el desarrollo de sus proyectos informáticos, por consiguiente, esta metodología será la utilizada para el desarrollo de los objetivos del proyecto. A continuación se procede a definir la metodología Zen, la fuente utilizada para definirla se encuentra en el Wiki de Trascend (Magurno, 2008)

III. EL PROCESO

El proyecto empieza con una visión, donde el Dueño del Producto es el responsable de transmitirla de una manera que maximice el Retorno de Inversión (ROI) para el cliente, formulando un plan que incluya la lista con los requerimientos que se deben cumplir (Product Backlog). Esta visión se le explica al Gerente de Proyecto y se presenta la lista mencionada anteriormente, todo esto en una primera reunión, donde se le asigna cada requerimiento su nivel de importancia y de dificultad, dándole prioridad a aquellos que generen mayor valor para la empresa. Todo el trabajo es hecho en distintas iteraciones, de las cuales cada una consta de 15 a 30 días consecutivos. Esto se da inicio con una reunión de planificación, donde el Dueño del Producto, el Gerente de Proyecto y el equipo se reúnen, y en colaboración definen lo que será hecho en la próxima iteración.

Esta reunión consta de dos partes, la primera consiste en que el Dueño del Producto presente al equipo los requerimientos de la lista (Product Backlog), igualmente se establece la idea inicial de la arquitectura del software. Posteriormente, en la segunda reunión el equipo planea la iteración junto al Gerente de Proyecto, donde se dividen los requerimientos escogidos en las tareas que se deben cumplir y asignándole una persona encargada para cada uno. Igualmente, se estima el tiempo en que se tardaran en

completarlas. Todas las tareas que compongan este plan, son colocadas en una Lista de Tareas (Sprint Backlog),

El Dueño del Producto es responsable de hacer cualquier cambio en la Lista de requerimientos (Product Backlog) actualizándolo, cambiando prioridades o reflejando nuevos requerimientos que surjan en el proceso, pero no se debe alterar la planificación de la iteración en curso. Cualquier cambio que se realice será explicado en el comienzo de la siguiente iteración. Ya planteada la Lista de Tareas (Sprint Backlog), se comienzan las actividades de desarrollo, donde se realizan todas las tareas necesarias para obtener el incremento funcional.

El equipo se reúne con el Gerente del Proyecto cada día durante 15 minutos, para sincronizar el trabajo de todos los miembros, verificando el avance de las tareas asignadas y si han tenido algún inconveniente que pueda dificultar la culminación de la tarea. En este caso se programa otra reunión para prestar la ayuda necesaria para avanzar.

En el primer o segundo día del desarrollo, el equipo deberá presentarle al Dueño del Producto, un prototipo de la interfaz que se utilizará para el incremento, con el fin de verificar que se esté tomando en consideración las funcionalidades esperadas y que sea de su agrado.

Cada integrante del equipo es responsable de escribir las pruebas, código, y documentación de las tareas que le fueron asignadas, realizando su integración a medida que complete cada una de ellas, verificando que funcionen correctamente resolviendo cualquier error que se le presente. Para el desarrollo de cada incremento funcional el

equipo utilizará Test Driven Development (TDD), por lo que cada integrante comenzará por escribir pruebas automatizadas para los métodos que luego va a implementar, con la finalidad de completar cada tarea.

El equipo realizará integración continua. A medida que se vayan completando las tareas, éstas deben ser integradas, corriendo todas las pruebas existentes hasta el momento, para verificar que no se generó ningún problema. Se utilizarán herramientas que permitan el control o manejo de varias versiones del proyecto, lo que ayudará a monitorear cualquier conflicto que se pueda generar.

Para llevar un control general del proyecto y mantener los estándares de codificación se deberá hacer revisión de código, asegurándose de que esté bien estructurado. Luego de completadas todas las tareas, se realiza una última integración, corriendo todas las pruebas que fueron diseñadas para éstas, verificando que todo funcione correctamente.

Al finalizar el Sprint se lleva a cabo una reunión de revisión, donde el equipo le presenta al Dueño del Producto lo que han desarrollado durante la iteración, incluyendo la lista con las funcionalidades del incremento, éste revisa el trabajo entregado y da su opinión, si algún requerimiento no pudo ser realizado, se puede incluir en cualquiera de las iteraciones siguientes.

Luego de la revisión, y antes de la próxima reunión de planificación, el Gerente de Proyecto sostiene una reunión de retrospectiva con el equipo, donde los alienta a revisar su proceso de desarrollo para hacerlo más efectivo y agradable para la próxima iteración.

III.2 ROLES

III.2.1 DUEÑO DEL PRODUCTO

El Dueño del Producto es el responsable de representar los intereses de todos aquellos que tengan cierto interés en el proyecto y el sistema resultante, definiendo los requerimientos iniciales del proyecto, los objetivos de retorno de inversión (ROI) y el plan de entregas. También es responsable de realizar y usar la Lista de Requerimientos (Product Backlog) para asegurar que las funcionalidades de mayor valor sean cubiertas primero.

III.2.2 GERENTE DE PROYECTO

Es responsable de manejar el proceso general, de saber sus prácticas, reglas, productos, terminología y como aplicar todo correctamente.

III.2.3 EQUIPO

El equipo está conformado por un grupo pequeño de personas, que están encargados de averiguar cómo convertir la Lista de Requerimientos (Product Backlog) en un incremento de funcionalidad en cada iteración. Por lo que deben llegar a un acuerdo con el Dueño del Producto de los requerimientos que son capaces de desarrollar durante una iteración. Se encargan de dividir los requerimientos, siguiendo las especificaciones anotadas, con la ayuda del Gerente de Proyecto, en las distintas tareas que los conforman y que deben ser cubiertas para cumplir con la funcionalidad deseada. Para cada tarea realizan estimaciones del tiempo que piensan que les tomará desarrollarla.

Dentro del equipo se pueden definir distintos roles, como por ejemplo:

- Probador: es aquel que verifica que todos los integrantes del equipo estén escribiendo pruebas antes de escribir el código funcional, igualmente se encarga de correr estas pruebas para revisar que no se haya producido algún error durante la integración. Adicionalmente, debe hacer llegar estos resultados a todos los integrantes del equipo. También se encarga de escribir pruebas funcionales, tomando en consideración las especificaciones dadas por el Dueño del Producto.
- Verificador: se encarga de revisar las estimaciones hechas por el equipo, comparándolas con el progreso real del proyecto, y mostrarlo al Gerente del Proyecto de manera que se pueda hacer seguimiento y recomendaciones para cumplir con lo planteado, o decidir si se tiene que tomar alguna otra medida. Si al finalizar una iteración las estimaciones fueron muy altas o muy bajas, debe mostrarle esto al equipo y hacer las recomendaciones pertinentes para mejorar en la siguiente iteración.
- Encargado de Interfaz: Pueden ser una o más personas, el que desempeña este rol realiza el prototipo de interfaz que se muestra al Dueño del Producto al comienzo de la iteración. Debe hacer el diseño general de la interfaz, tomando en cuenta los requerimientos que se quieren cumplir, y las necesidades planteadas durante la reunión de planificación.
- Arquitecto: es el encargado de escribir el documento de arquitectura del software, realizando los cambios correspondientes que vayan surgiendo a medida que se desarrolla el producto. Igualmente, asesora a los demás integrantes en

caso de que surja alguna duda especifica de como se integra la tarea a la idea global del proyecto.

Dependiendo de las necesidades del proyecto y del tamaño del equipo, estos roles pueden ser cambiados o se pueden agregar nuevos.

III.3 FLUJO DE TRABAJO

III.3.1 ACTIVIDADES DE PLANIFICACIÓN

III.3.1.1 CREACIÓN O ACTUALIZACIÓN DE LA LISTA DE REQUERIMIENTOS (PRODUCT BACKLOG).

Ya habiendo presentado la visión del proyecto por parte del Dueño del Producto al Gerente de Proyecto, este comenzará la elaboración de la lista de requerimientos funcionales y no funcionales (Product Backlog), con los que debe cumplir el producto. A cada requerimiento se le es dado un nivel de importancia que establece su prioridad según el valor que generen a la organización, y también se le asigna un nivel de dificultad con lo que el Gerente de Proyecto podrá ayudar al equipo a la hora de realizar las estimaciones de tiempo sobre las tareas de cada requerimiento.

Esta lista debe ser actualizada constantemente a medida que avance el proyecto, surjan nuevos requerimientos o cambien las prioridades de éstos. El Dueño del Producto podrá hacer cualquier cambio que desee, los cuales serán explicados en la siguiente reunión de planificación, siempre y cuando no afecten la iteración en curso.

III.3.1.2 REUNIÓN DE PLANIFICACIÓN

La reunión de planificación es llevada a cabo al comienzo de cada iteración y luego de que el Dueño del Producto y el Gerente de Proyecto tengan definida la Lista de Requerimientos (Product Backlog); ésta constará de tres partes, como se explica a continuación:

- Elegir requerimientos: El Dueño del Producto y Gerente de Proyecto se reúnen con el Equipo para presentarles la visión del proyecto y la Lista de Requerimientos (Product Backlog). Posteriormente, se discute cada requerimiento de la lista, se van detallando las especificaciones que deben ser cubiertas para poder cumplir con lo planteado. Luego se crea una Lista de Funcionalidades con los requerimientos escogidos, en función de la dificultad y la prioridad que tiene cada uno de estos, finalmente se define el boceto inicial de la arquitectura del proyecto, generando una idea global de cómo se va a desarrollar, y como será conformado estableciendo las relaciones entre los distintos requerimientos. Esta idea global luego será definida en la primera versión del documento de arquitectura del software.
- Creación de lista de Tareas (Sprint Backlog): Después de ser definidos los requerimientos, se desglosan cada uno de estos en las tareas que deben ser realizadas para cumplir el objetivo de la iteración. A cada tarea se le asigna un responsable, y éste realiza una estimación del tiempo que cree necesario para poderla cumplir, igualmente el Gerente de Proyecto debe tomar en cuenta la dificultad que se le asignó al requerimiento y guiar a la persona si es necesario. Al terminar las estimaciones se obtiene una lista de tareas, que contiene las

personas encargadas de realizarlas y el tiempo estimado en el cual serán completadas.

- Boceto inicial del documento de arquitectura del software: Un integrante del equipo debe generar la primera versión del documento de arquitectura del software para el proyecto. La creación de este documento debe hacerse de manera rápida, de forma que esté listo al comenzar la primera iteración. Igualmente, debe ser modificado a medida que se realiza el desarrollo o antes de comenzar cada iteración.

III.3.2 ACTIVIDADES DE DESARROLLO

III.3.2.1 REUNIÓN DIARIA

Durante el desarrollo el Gerente de Proyecto realiza reuniones diarias con el equipo de unos 15 minutos de duración, donde se informa el progreso en las tareas bajo la responsabilidad de cada miembro del equipo, de los objetivos a cumplir en el transcurso del día y los obstáculos y dificultades encontrados, todo esto con el fin del llevar el control del progreso del proyecto. Si algún miembro presenta alguna dificultad se convoca otra reunión.

III.3.2.2 CREACIÓN DEL PROTOTIPO DE INTERFAZ

En el primer o segundo día del desarrollo, el equipo debe diseñar y presentarle al Dueño del Producto un prototipo de la interfaz que se utilizará para el incremento funcional que será entregado al finalizar la iteración. Con esto, se realizará una verificación inicial de que se estén tomando en consideración las funcionalidades que se esperan para los requerimientos escogidos. Además, este prototipo debe cubrir las funcionalidades de los

requerimientos que fueron escogidos y se debe realizar en paralelo con las tareas que le fueron asignadas, de forma de no retrasar la iteración.

III.3.3.3 TEST DRIVEN DEVELOPMENT (TDD)[1]

Durante el desarrollo el equipo utilizará Test Driven Development (TDD), siendo ésta una práctica que consiste en escribir pruebas antes de escribir el código funcional, tomando en cuenta los diferentes métodos que serán implementados luego. En esta técnica de desarrollo se cumplen las siguientes etapas:

- Escribir una prueba: Los programadores escriben una prueba para la tarea que están desarrollando en ese momento.
- Compilar: Luego de que la prueba ha sido escrita, es compilada para asegurarse de que no existan errores. Debido a que no se ha escrito código del método que se está desarrollando, la compilación fallará.
- Corregir los errores de compilación escribiendo suficiente código para que la prueba compile: El programador escribe suficiente código del método que está implementando para que corra sin problemas.
- Correr la prueba y verla fallar: Luego de que se escribe suficiente código para poder compilar, se corre la prueba, debido a que sólo se escribió una pequeña parte del código con la finalidad de que compilara, la prueba fallará.
- Escribir más código para que pase la prueba: Luego de ver fallar la prueba, el programador escribe más código para completar la funcionalidad del método que está implementando.

[1] Durante el desarrollo no se utilizó el TDD al pie de la letra, ya que primero se codificó antes de realizar la prueba. Al momento de programar la prueba se siguen los pasos del TDD

- Correr la prueba y verla pasar: Al terminar de escribir el código, el programador corre la prueba, la que debería pasar sin problemas.
- Refactorizar el código: Cuando la prueba pase sin problemas, procederá a refactorizar con la finalidad de asegurarse que el código que escribió no presente repeticiones y esté lo mas simplificado posible.
- Comenzar con la siguiente prueba: Al finalizar la refactorización, comienza de nuevo en la primera etapa, escribiendo una prueba para la siguiente funcionalidad.

Las personas encargadas de escribir estas pruebas son los programadores, que deben tener claras las especificaciones y los requerimientos de las tareas que están realizando para esta iteración.

III.3.3.4 INTEGRACIÓN CONTINUA[1]

La integración continua es un proceso en el que se une el código escrito por los distintos integrantes del equipo. Este proceso permite verificar que los cambios realizados no produzcan errores. A medida que se van terminando las tareas se debe verificar que no se produzcan errores de compilación y que el código pase todas las pruebas escritas sin ningún problema.

Para controlar la integración continua, y los posibles errores que pueden surgir cuando un grupo de personas está realizando modificaciones y uniones bajo un mismo proyecto, se debe utilizar control de versiones y unas series de herramientas de software para controlar el proceso.

---

[1] La integración continua no forma parte de las actividades realizadas en este trabajo debido a que el equipo de desarrollo consistió de un solo miembro.

III.3.3.5 CONTROL DE VERSIONES

El control de versiones permite mantener un historial del código fuente, teniendo así un control sobre todos los cambios realizados. Estos sistemas permiten a más de un programador editar sobre el mismo código fuente simultáneamente.

Cada programador tendrá una copia local de la última versión del proyecto, trabajando sobre ésta, al finalizar su trabajo éste realizará "commit", subiendo así los cambios al controlador de versiones. Si se presentan conflictos entre su trabajo y los cambios que realizó otro programador sobre ese código fuente, el sistema alertará y mostrará los lugares donde se está produciendo el problema, de manera que se puedan hacer comparaciones y solucionar el conflicto.

Esta forma de manejar las distintas versiones y de alertar ante los conflictos, ayuda a los programadores durante la integración del código.

III.3.3.4 REVISIÓN DE CÓDIGO

Durante la revisión de código se revisa todo lo escrito por algún miembro del equipo, asegurando que esté bien estructurado. Para esta práctica se pueden utilizar herramientas de control, facilitando la revisión. Si se encuentra algún error, éste será corregido antes de comenzar con los objetivos planteados, evitando generar problemas futuros.

III.4 ARTEFACTOS

III.4.1 LISTA DE REQUERIMIENTOS (PRODUCT BACKLOG)

Comprende una Lista dinámica con los requerimientos funcionales y no funcionales del producto. Sus cambios deben ser administrados constantemente para identificar que necesita el producto para ser apropiado, competitivo y útil. En ésta podemos registrar el

nombre del ítem, su nivel de importancia o prioridad ante los demás requerimientos y su factor de complejidad. También se debe registrar la iteración en el cual el ítem es desarrollado.

Todos los requerimientos reflejados en la lista, son agregados por el Dueño del Producto, manteniendo siempre en cuenta la visión general del proyecto y asegurándose de que cumplan un valor para la empresa.

Para establecer la prioridad y el factor de complejidad, el Dueño del Producto es asesorado por el Gerente de Proyecto, de forma tal que se consideren tanto los aspectos importantes para la empresa como los aspectos del desarrollo en sí.

III.4.2 DOCUMENTO DE ESPECIFICACIÓN DE REQUERIMIENTOS DEL SOFTWARE

Cada requerimiento de la lista tiene especificaciones que definen lo que el Dueño del Producto realmente quiere o necesita. Estas especificaciones son discutidas durante la primera parte de la reunión de planificación. Igualmente, el Equipo anota cada una de estas especificaciones y luego las revisa para hacer la Lista de Tareas.

III.4.3 LISTA DE FUNCIONALIDADES (RELEASE BACKLOG)

Los requerimientos escogidos para la iteración son escritos en la lista de funcionalidades. Esto permite verificar que los requerimientos planteados fueron cumplidos, y será actualizada en el caso de que no se cumpla algún requerimiento. También le permite al Dueño del Producto saber que funcionalidades tiene el incremento que le fue entregado.

III.4.4 LISTA DE TAREAS (SPRINT BACKLOG)

Contiene las tareas que el equipo define para transformar los requerimientos seleccionados para la iteración en un incremento funcional y que deben ser divididas de manera tal que tome entre 4 y 16 horas terminar cada una de ellas.

Esta lista solo puede ser manipulada por los integrantes del equipo, en la primera columna se colocará las tareas que se van a realizar, en la segunda columna el nombre del encargado de esta tarea y por último una columna para cada día de la iteración. En la intersección de la tarea y el primer día de la iteración, se colocará el tiempo total estimado que se tardará en completar la tarea. Durante cada día de la iteración se irá colocando el número de horas implementadas en la tarea.

III.4.5 DOCUMENTO DE ARQUITECTURA DEL SOFTWARE

Es el que define la arquitectura del proyecto, contiene la idea global de las tareas que se deben llevar a cabo y asegurar que se cumplan todos los requerimientos. Esta arquitectura irá evolucionando a medida que se vaya desarrollando el proyecto, creciendo de manera incremental, para cubrir cualquier cambio que se haya encontrado durante el desarrollo. Todo esto debe ser escrito de una manera sencilla y fácil de leer; para simplificar el agregar cosas nuevas o realizar cambios a medida que vayan surgiendo.

III.4.6 PROTOTIPO DE INTERFAZ

Al comienzo de las actividades de desarrollo se mostrará al Dueño del Producto un prototipo de la interfaz que se utilizará para el producto. Este prototipo cumplirá con los requerimientos planteados durante la reunión de planificación, mostrando todas las

funcionalidades que serán desarrolladas y el diseño de la interfaz (colores, letra, distribución, navegación, etc.).

III.4.7 PRUEBAS

Una de las prácticas que el Equipo utilizará es el TDD, que consiste en escribir pruebas antes de escribir el código funcional. Por lo tanto, estas se escribirán para cada uno de los métodos que luego serán implementados, tomando en cuenta los requerimientos y las necesidades que se tienen que cumplir para cada tarea.

Éstas serán automatizadas, de manera que se puedan correr cada vez que se realiza integración de código y así comprobar que no se ha generado ningún problema. Con esto se verificará que el código compile y funcione de manera correcta, además se escribirán pruebas funcionales con las que se verá si los resultados son los esperados por el Dueño del Proyecto.

III.4.8 INCREMENTO FUNCIONAL (RELEASE)

Al final de cada iteración se entregará al Dueño del Producto un incremento del producto, que debe cumplir con todos los requerimientos que fueron acordados durante la reunión de planificación.

Este incremento del producto debe ser funcional, probado, bien estructurado, y con su operatividad totalmente documentada, así sea en archivos de ayuda o en documentación de usuario.

CAPITULO IV. DESARROLLO

IV.1 PRIMERA REUNIÓN DE PLANIFICACIÓN

Como indica la metodología Zen, la primera parte que se realizó fue la reunión con el dueño del producto y el gerente del proyecto, durante el transcurso de la reunión se presentó una breve introducción al sistema Angelia y se explicó la visión del proyecto que se va a desarrollar. El proyecto se puede resumir en los siguientes puntos:

- Componente de administración de las opciones de planificación (también llamado administración de activadores, debido a que las opciones de planificación también son llamadas activadores).
- Componente extractor.
- Administrador de taxonomías.
- Integración del componente extractor con el componente vigilante.

A continuación se explican de manera no exhaustiva los componentes:

- Componente de administración de las opciones de planificación: Permite programar el envío de las plantillas en un instante de tiempo. Dicho instante se define por una fecha, hora, minuto y la frecuencia (es opcional) de envío (la frecuencia puede ser: diaria, semanal, mensual o anual). También se permite la posibilidad de definir la suspensión del envío indicando opciones de desactivación. Estas opciones pueden ser por un máximo número de envío o hasta que se alcance una fecha determinada.
- Componente extractor: Es el componente encargado de extraer los datos de diversas fuentes, independientemente de cómo sea la estructura de las mismas.

La configuración de este componente se realiza durante la fase de instalación y no durante el uso del sistema.

- Administrador de taxonomías: El administrador de taxonomías indica al sistema, sobre las taxonomías almacenadas. Dichas taxonomías se pueden agregar o eliminar. Es importante destacar que el ingreso de taxonomías al sistema se realiza durante la fase de instalación del sistema y no durante su uso.
- La integración con el componente vigilante: Esto se realiza mediante la generación de un archivo de texto para luego moverlo a un directorio que se encuentra monitoreado por el vigilante.

En cuanto a los requerimientos de la primera iteración se acordó lo siguiente:

- Modificación de la interfaz que permite crear plantillas, para que se puedan ingresar las opciones de planificación.
- Modificación de la interfaz que permite modificar las plantillas, para que se puedan agregar o desactivar opciones de planificación.

El tiempo estimado para realizar la iteración es de quince (15) días hábiles.

El siguiente punto tratado fue la arquitectura que va a poseer el administrador de planificación, donde se tomó la decisión de desarrollarlo usando la librería Quartz para la plataforma de .NET y de implementarlo como un servicio del sistema operativo Windows, el componente tendrá el nombre de Mensajero. En la figura 9 se puede apreciar el diagrama inicial de arquitectura.

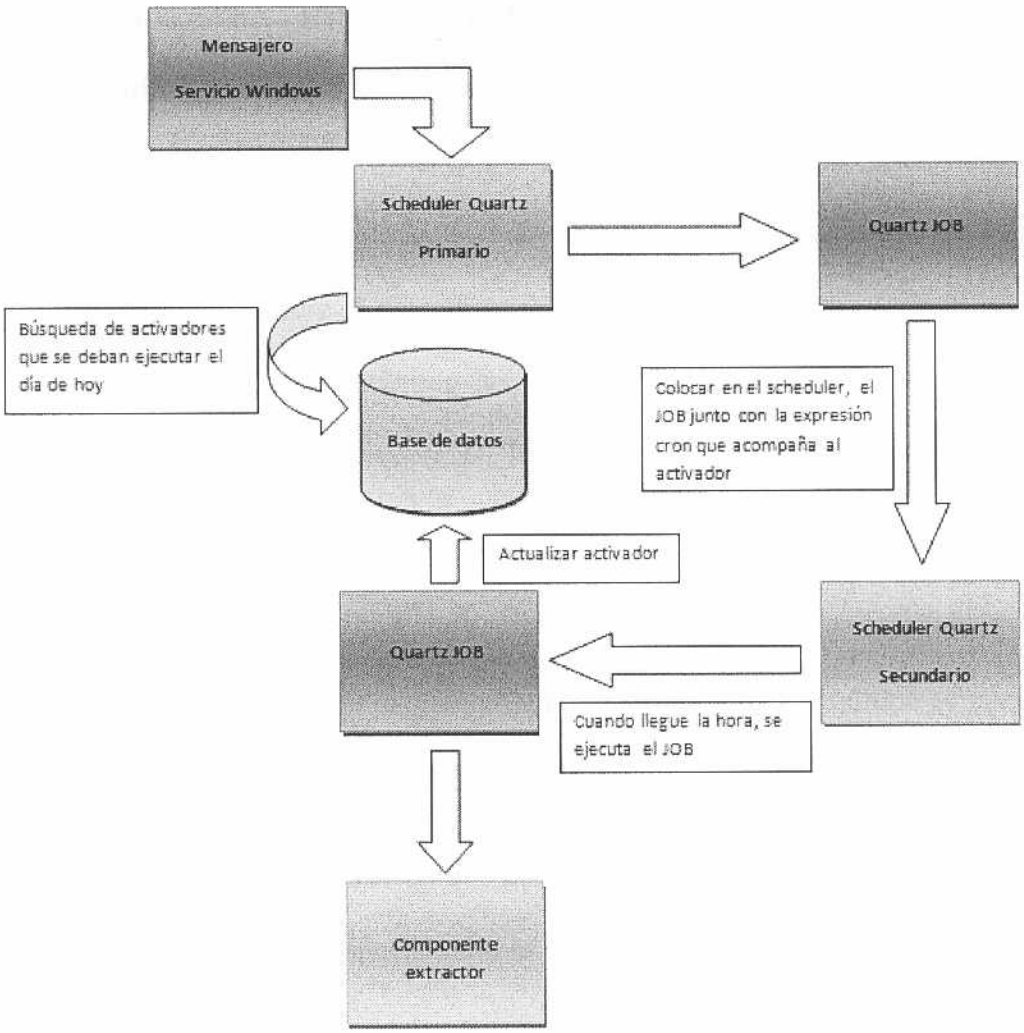

Figura 9 Diagrama de arquitectura administrador de planificación.

Fuente: Elaboración propia

IV.2 PRIMERA ITERACIÓN

Durante el transcurso de los primeros cinco (5) días hábiles, se procedió a estudiar la documentación existente sobre el sistema Angelia poniendo mayor énfasis en los comentarios presentes en el código fuente del sistema, se estudió el código fuente para poder familiarizarse con el mismo. Durante las reuniones diarias con el gerente del proyecto se aprovechó en aclarar las dudas que se generaban durante el estudio.

Finalizando la primera semana se estudiaron las expresiones cron con la razón de que son necesarias para utilizar la librería Quartz. Durante la revisión de la documentación del framework Quartz.NET y las expresiones cron, se llegó a la conclusión de que las expresiones cron no son capaces de representar por sí mismas todas las opciones de planificación que requiere el sistema Angelia. Por lo que será necesario el uso de datos adicionales que conjuntamente con las expresiones cron puedan representar todas las opciones de planificación requeridas. Este punto fue discutido durante las reuniones diarias con el gerente del proyecto.

Durante el transcurso de la segunda semana se comenzó con la creación de un prototipo de interfaz, para mostrárselo al dueño del producto. En dicho prototipo se pueden elegir los siguientes parámetros:

- Fecha de inicio: Fecha en que comenzará a enviarse la plantilla, por medio del canal que tenga especificado la misma, estos canales pueden ser e-mail y mensajería de texto.
- Hora de inicio: Se refiere a la hora exacta en que será enviada la plantilla
- Frecuencia: Se refiere a cada cuánto tiempo será enviada la plantilla, ésta puede ser diaria, semanal, mensual, anual o una fecha exacta en que se desee enviar.
- Desactivación: Se refiere a las condiciones que deben cumplirse para que se suspenda el envío de la plantilla, éstas pueden ser hasta que se alcance una fecha determinada, o hasta que se envíe un número máximo de veces. También se puede escoger la opción de nunca suspender el envío de la plantilla.

A medida de que se van eligiendo las opciones, la interfaz debe mostrar un mensaje al usuario que indique el resultado. El prototipo de la interfaz se puede apreciar en la figura 10.

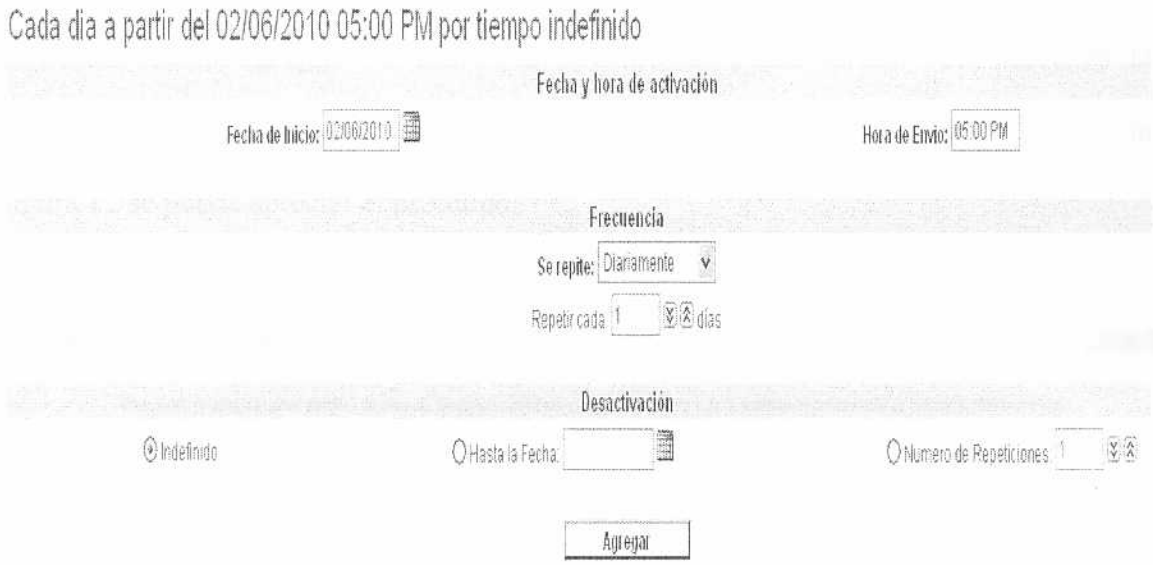

Figura 10 Prototipo de interfaz de ingreso de opciones de planificación

Fuente: Elaboración propia.

Después de mostrar el prototipo de la interfaz, la misma fue aprobada por el dueño del proyecto, por lo que se procedió a darle funcionalidad a la misma. Finalmente se procedió a probar la interfaz, durante las pruebas se realizaron las siguientes actividades:

- Validación de la entrada del usuario: Se introdujo datos inválidos, para verificar que la interfaz notifique al usuario que no son validos.
- Verificar consistencia entre lo que se elige en la interfaz con lo que se guarda en memoria: Se introdujeron valores en la interfaz para verificar la consistencia entre lo que elige el usuario con lo que se guarda en memoria. Se hizo énfasis en que las expresiones cron que genera el sistema representen lo más posible lo introdujo el usuario.

En la última semana de la iteración se procedió a crear la tabla en base de datos que guarde la información suministrada por el usuario. Adicionalmente se crearon los procedimientos encargados de ingresar y actualizar los datos en la tabla. Para finalizar se modificó la interfaz que permite modificar las plantillas proporcionándole la funcionalidad de agregar o eliminar las opciones de planificación. Para mantener la consistencia, el prototipo de esta nueva interfaz es similar al de crear plantillas. En la figura 11 se puede apreciar el prototipo.

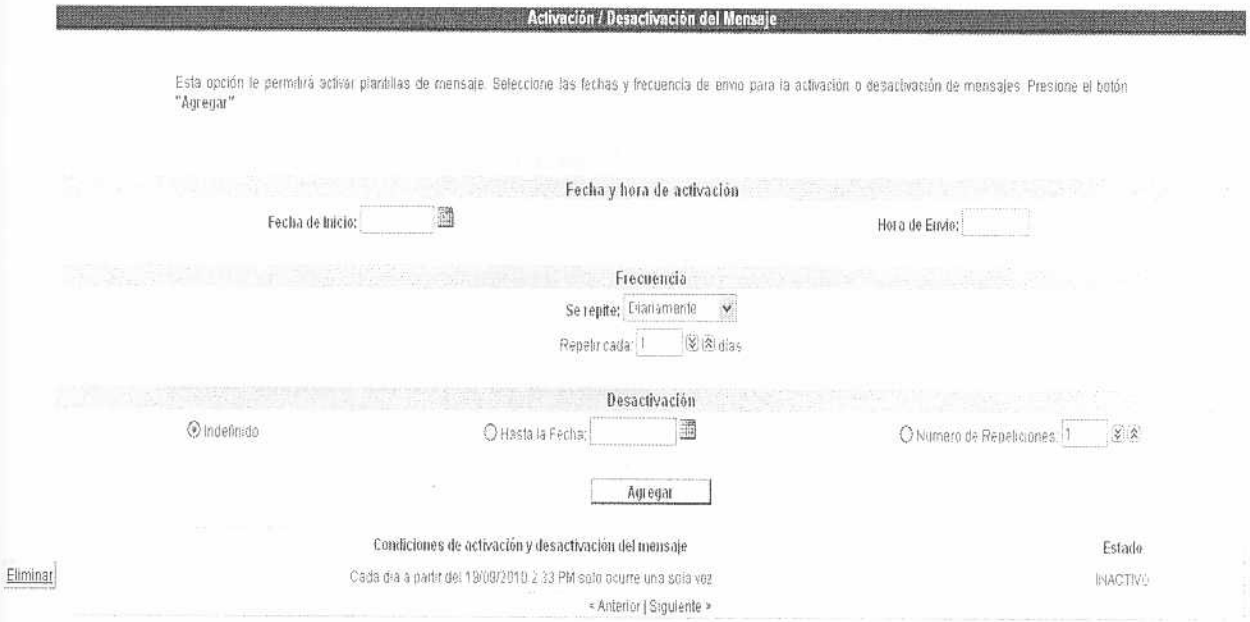

Figura 11 Prototipo de interfaz modificación de opciones de planificación

Fuente: Elaboración propia.

En la interfaz de modificar plantillas, se muestran las opciones de planificación asociadas indicando el estado de las mismas (activo, inactivo), para aquellas que presenten el estado activo podrán eliminarse. Al eliminar una opción, ésta no se elimina de la base de datos, sino que pasa de estado activo a inactivo.

Después de mostrar la interfaz al dueño del producto ésta fue aprobada por lo que se procedió a darle la funcionalidad y realizar las pruebas de la misma, dichas pruebas son similares a las que realizaron en la interfaz de crear plantillas.

IV.3 SEGUNDA REUNIÓN DE PLANIFICACIÓN

Durante la segunda reunión de planificación se mostraron los resultados obtenidos tras realizar la primera iteración. Posteriormente se procede a escoger los nuevos requerimientos que debe cumplir la siguiente iteración, los requerimientos elegidos se mencionan a continuación:

- Monitorear las fechas en las que se deban enviar las plantillas (dichas fechas se encuentran almacenadas en base de datos) y activar el envío de las plantillas cuando indique dicha fecha.
- Proveer del cálculo de la nueva fecha en la que se deba enviar la plantilla, esto se realiza después del envío de la misma.
- Suspender el envío de plantillas cuando se cumpla la condición de desactivación de la planificación.

El tiempo estimado para el desarrollo de la aplicación es de quince (15) días hábiles.

IV.4 SEGUNDA ITERACIÓN

En la segunda iteración se comenzó el desarrollo del servicio Windows el cual cuando esté activo se mantendrá monitoreando la base de datos, el monitoreo se realiza mediante la librería Quartz el cual por medio de una expresión cron se le indica el momento en que debe comenzar a monitorear. Los pasos para realizar el monitoreo se explica en la figura 12.

Activadores en BD							
ID	Fecha de Inicio	Cron	Próxima Fecha Activación	Fin	Máxima Cantidad de ocurrencia	Cantidad Restante de ocurrencias	ID de la plantilla
1	15/06/2010	xxxxxx	15/06/2010	nulo	5	5	10
2	20/06/2010	xxxxxx	20/06/2010	20/07/2010	nulo	nulo	11
3	15/06/2010	xxxxxx	15/06/2010	nulo	nulo	nulo	12

Día de hoy: 15/05/2010

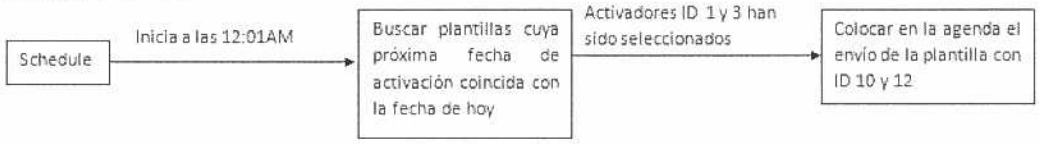

Figura 12 Proceso de monitoreo de activadores

Fuente: Elaboración propia.

El procedimiento de colocar en la agenda consiste en agregar al schedule de Quartz un JOB, el cual es una clase que hereda de la clase QuartzJobObject. Al JOB se le asigna la expresión cron que indica el momento que debe activarse el JOB, cuando éste se activa se ejecuta el método ExecuteInternal el cual debe implementar la clase. En la implementación del método se toman los datos del activador que se está ejecutando y se procede a procesar el envío de la plantilla, durante esta iteración no está previsto el envío de la plantilla por lo que sólo se procede a actualizar la información contenida en la tabla de activadores. El proceso que realiza el JOB que procesa la plantilla se describe en la figura 13.

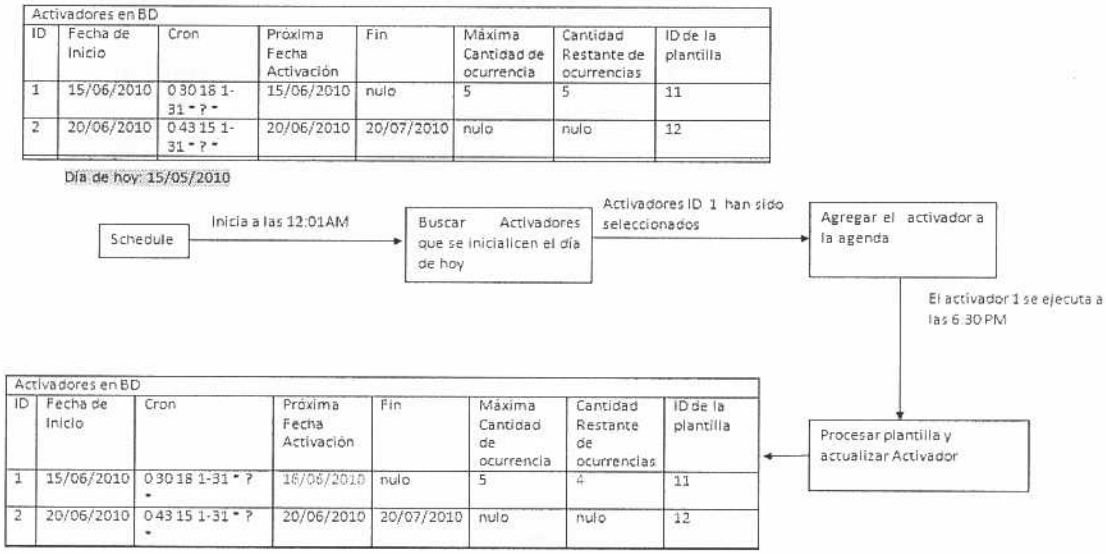

Figura 13 Proceso de monitoreo y actualización de activadores

Fuente: Elaboración propia.

Luego de finalizar el componente planificador, se realizaron las pruebas unitarias de mismo.

IV.5 TERCERA REUNIÓN DE PLANIFICACIÓN

Durante la tercera reunión se mostraron los resultados obtenidos durante la segunda iteración, seguidamente se realizó el bosquejo inicial de la arquitectura del componente extractor, dicho componente debe ser capaz de leer diferentes fuentes de datos. Se acordó otorgar un periodo de quince (15) días hábiles para lograr los siguientes objetivos:

- Diseño del componente extractor.
- Definir el lenguaje que se va a utilizar para realizar las consultas.
- Determinar el formato que deben poseer los archivos de mapeo, dichos archivos proporcionan información sobre la estructura de la fuente de datos que se va a leer.

IV.6 DISEÑO DEL COMPONENTE EXTRACTOR

Partiendo del bosquejo inicial realizado en la tercera reunión se desarrolló el diagrama de clases inicial, donde se definieron las clases que formarán parte del componente, en la figura 14 y figura 15 se muestra el bosquejo inicial conjuntamente con el diagrama de clases inicial.

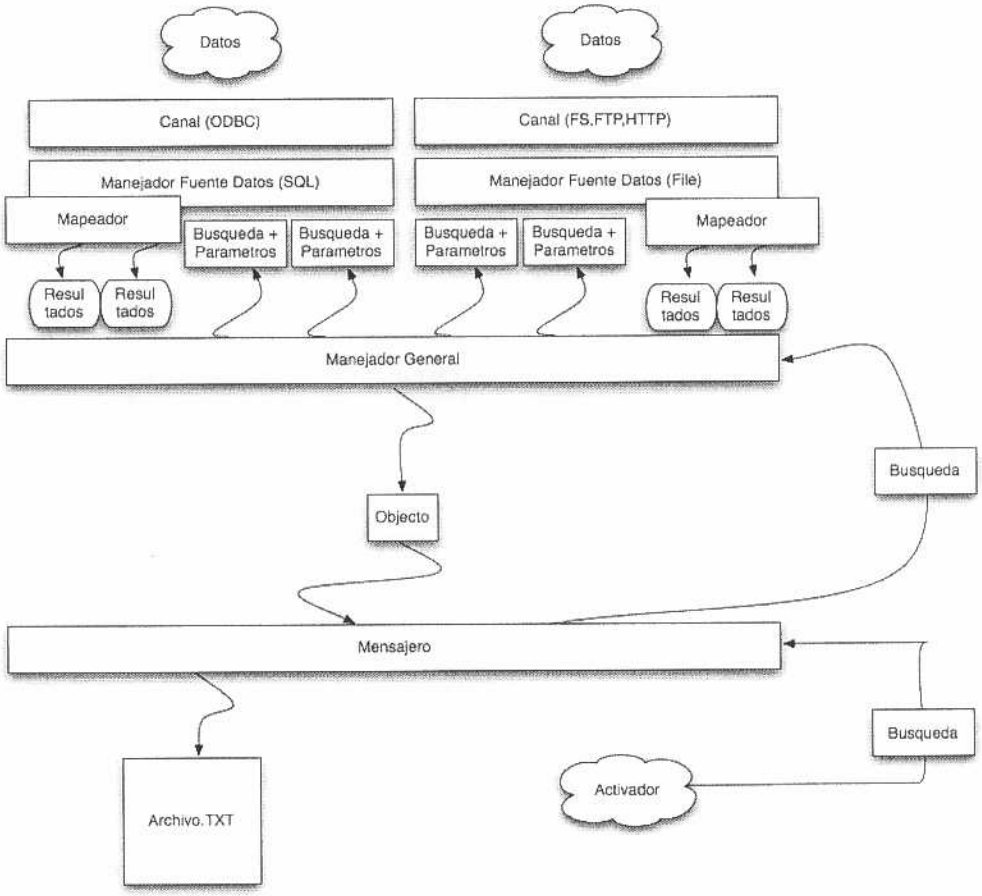

Figura 14 Bosquejo inicial componente extractor.

Fuente: Elaboración propia.

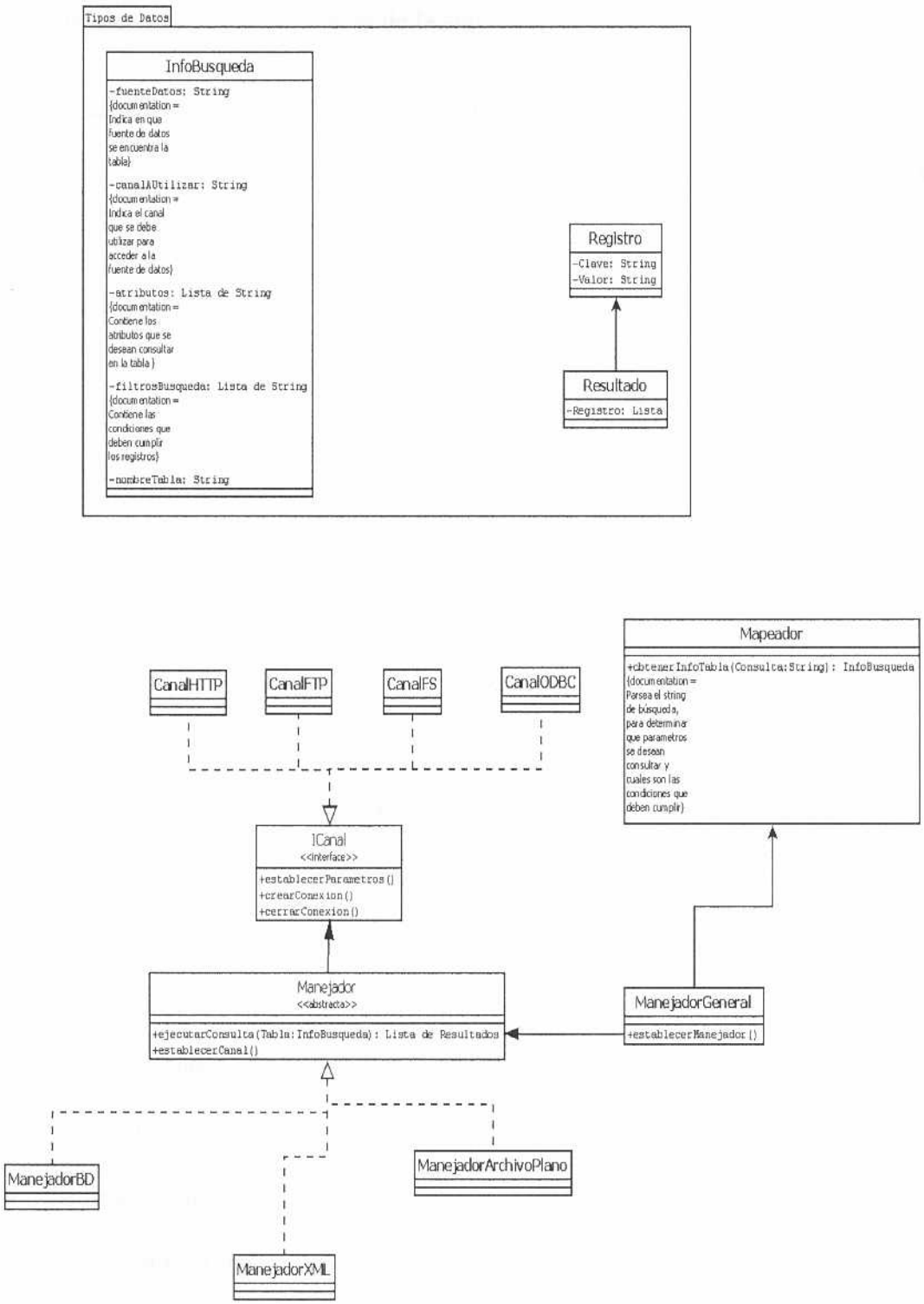

Figura 15 Diagrama inicial de clases.

Fuente: Elaboración propia.

El lenguaje de consulta se estableció de la siguiente manera:

- Cada atributo o dato que se está solicitando presentan los siguientes elementos:
    - Fuente: Indica la fuente donde se encuentra el atributo (archivo plano, archivos XML, base de datos).
    - Nombre de la fuente: Indica el nombre del archivo que se va a leer o el nombre de la base de datos.
    - Nombre de la tabla (opcional): Solo se coloca cuando se lee de base de datos, éste indica el nombre de la tabla donde se encuentra el campo.
    - Nombre del campo: Indica el nombre del campo que se está solicitando.
- El carácter que separa los elementos mencionados es '|'.
- El formato es el siguiente:
    - Fuente|nombre_fuente|campo: Este formato es utilizado cuando se consulta en archivos XML o archivos planos.
    - Fuente|nombre_fuente|nombre_tabla|campo: Este formato es utilizado cuando se consulta en bases de datos.
- Los campos solicitados se separan con el carácter coma (','). Ejemplo:
    - Fuente|nombre_fuente|campo1,Fuente|nombre_fuente|campo2,..., Fuente|nombre_fuente|campoN.
- El final de la lista de campos y el comienzo de la lista de condiciones se denota con el carácter numeral ('#'). Ejemplo:
    - Fuente|nombre_fuente|campo1,Fuente|nombre_fuente|campo2,..., Fuente|nombre_fuente|campoN # Fuente|nombre_fuente|campo1 > 41

Para efectos de alcance de este trabajo instrumental de grado (TIG), el lenguaje de consulta sólo maneja el operador AND, el cual viene representado por el carácter coma (',').

Para el mapeo se utilizaron archivos XML, en dichos archivos se guarda el nombre de la fuente, ruta para acceder a la fuente, nombre del campo, tipo de dato del campo. En casos de los archivos planos se almacena el carácter que separa los campos o la posición en que se encuentran, en archivos XML se guarda la ruta (en formato XPath) de cómo acceder a los campos.

Luego de establecer el diseño inicial se procedió a crear una base de datos conformada por archivos planos, archivos XML y tablas en manejadores de base de datos (en este caso es SQLServer 2005). Dicha base de datos servirá para realizar las pruebas a los componentes encargados de leer los datos.

IV.7 CUARTA REUNIÓN DE PLANIFICACIÓN

En esta reunión se discutieron los siguientes requerimientos que debe presentar la *siguiente iteración. Los requerimientos son los siguientes:*

- Interpretar el lenguaje de consulta para acceder a las fuentes de datos.
- Realizar las consultas en las fuentes, en esta iteración solo se realizaran las consultas en una fuente a la vez, es decir, en un solo archivo (plano o XML) o en una sola tabla de base de datos.
- Tomar en cuenta las restricciones a las que están sujetas los campos, como por ejemplo que sea mayor que cero.

El tiempo estimado para realizar la tercera iteración fue de quince (15) días hábiles.

IV.8 TERCERA ITERACIÓN

En la realización de la iteración se comenzó con el componente encargado de leer los archivos planos, para leer los archivos primero se busca la información en el archivo de mapeo y utilizando dicha información se lee el archivo plano. Durante la realización del componente se cambió la estructura en que se devuelve la información, antes del cambio los datos se almacenaban en un objeto resultado (ver diagrama de clases figura 15), ahora la nueva estructura es un DataTable. Para la implementación del componente lector de base de datos el proceso es similar con la variación de que el lenguaje de consulta utilizado por el componente extractor se cambia por su equivalencia en SQL, con la finalidad de que el manejador de base de datos ejecute la consulta y devuelva el resultado de la misma.

Para evaluar las condiciones se creó un componente capaz de inspeccionar la fila del DataTable que se forma mientras se lee la fuente para aplicar las condiciones. Las operaciones relacionales que soporta el componente son: Igualdad, mayor que, menor que y la desigualdad. Estas operaciones se realizan sobre cualquier tipo de dato. Luego de implementar los lectores se realizaron pruebas para verificar que los resultados de las consultas estén correctos.

IV.9 QUINTA REUNIÓN DE PLANIFICACIÓN

Durante la quinta reunión de planificación se mostraron los resultados de la iteración anterior, en esta ocasión no se cumplieron todos los requerimientos debido a que faltó la lectura de archivos XML, por lo que será incluido en la siguiente iteración, adicionalmente se deben lograr los siguientes requerimientos:

- Elegir taxonomías para poder asignarlas a una plantilla.
- El componente debe realizar el JOIN entre los resultados obtenidos al leer de dos o más fuentes.
- Dar la capacidad al lenguaje de consultas para poder comparar fechas, con el motivo de realizar consultas más elaboradas, como por ejemplo:
    - Clientes que cumplan años.
    - Tarjetas de crédito vencidas hace más de 3 meses.
- Las taxonomías deben guardarse en archivos XML con el formato que requiere el framework de spring, los archivos de mapeo deben cumplir con la misma condición anterior.

El tiempo que se estimó para realizar el prototipo fue de quince (15) días hábiles.

IV.10 CUARTA ITERACIÓN

Se comenzó por modificar la interfaz de crear plantilla, para permitir asociarle una taxonomía, debido a que se debe mostrar el prototipo al dueño del producto. En la figura 16 se puede apreciar el prototipo de la interfaz.

En el prototipo aparece una lista de distintas consultas o taxonomías que se pueden asociar a la plantilla, después de que el prototipo fuera aprobado por el dueño del producto, se procedió a darle funcionalidad. Los pasos que se siguieron fueron los siguientes:

- Crear el archivo de configuración de spring para guardar las taxonomías.

- Crear un servicio web que obtenga las taxonomías almacenadas en el XML y las ofrezca a la capa web en formato de arreglo. La capa web leerá el arreglo y lo mostrará al usuario.
- La relación entre la taxonomía y la plantilla se almacena en base de datos, por lo que se modificó la base de datos para agregar una tabla que almacene esta relación. Se agregó el store procedure que ingresa los registros en dicha tabla.

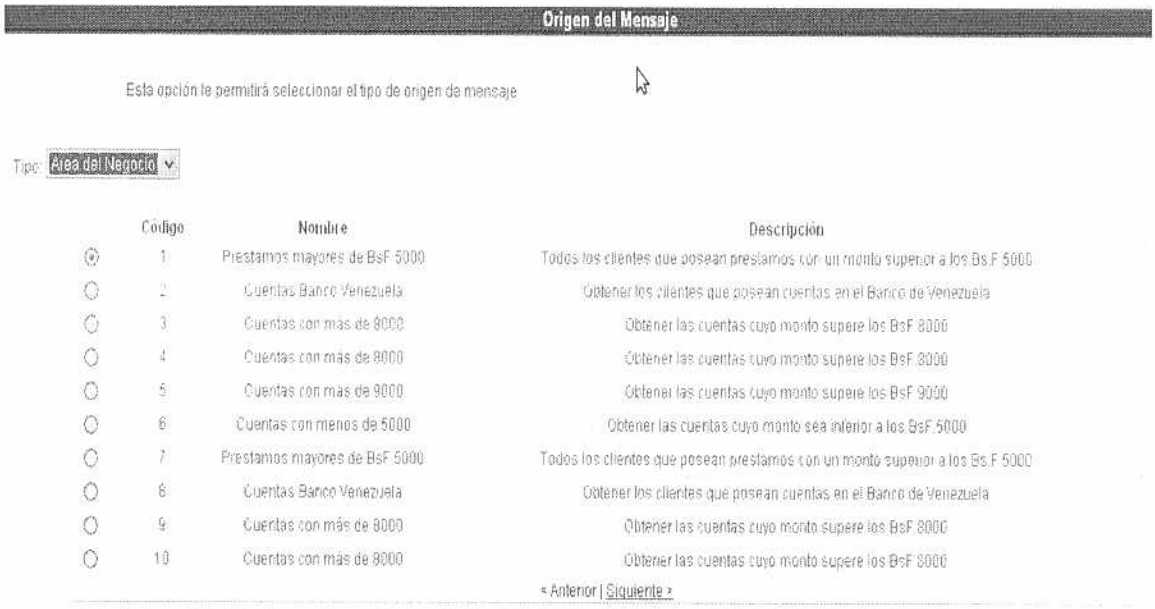

Figura 16 Prototipo de interfaz para asociar taxonomías a la plantilla

Fuente: Elaboración propia.

Luego de dar la funcionalidad a la interfaz se procede a cambiar la estructura de los archivos de mapeo para que la información sobre la estructura se guarde en un objeto, luego los manejadores instanciarán la clase para obtener información de las fuentes. Dichos objetos serán inicializados por medio de Spring. Adicionalmente se agregó la información de claves foráneas y primarias.

Se implementó el lector de archivos XML y se tuvo que modificar los otros lectores para que instancien las clases (usando Spring) y obtener la información de la fuentes. Se realizaron pruebas unitarias para verificar que los resultados de las búsquedas sean correctos.

Se desarrolló la clase que permite realizar el JOIN entre las tablas, en la figura 17 se puede apreciar la lógica que se utilizó para relacionar las tablas. Al relacionar las tablas se obtiene un archivo plano que contiene los resultados de la consulta. Dado que la generación del archivo no es un requerimiento de esta iteración, éste se pospondrá para la siguiente.

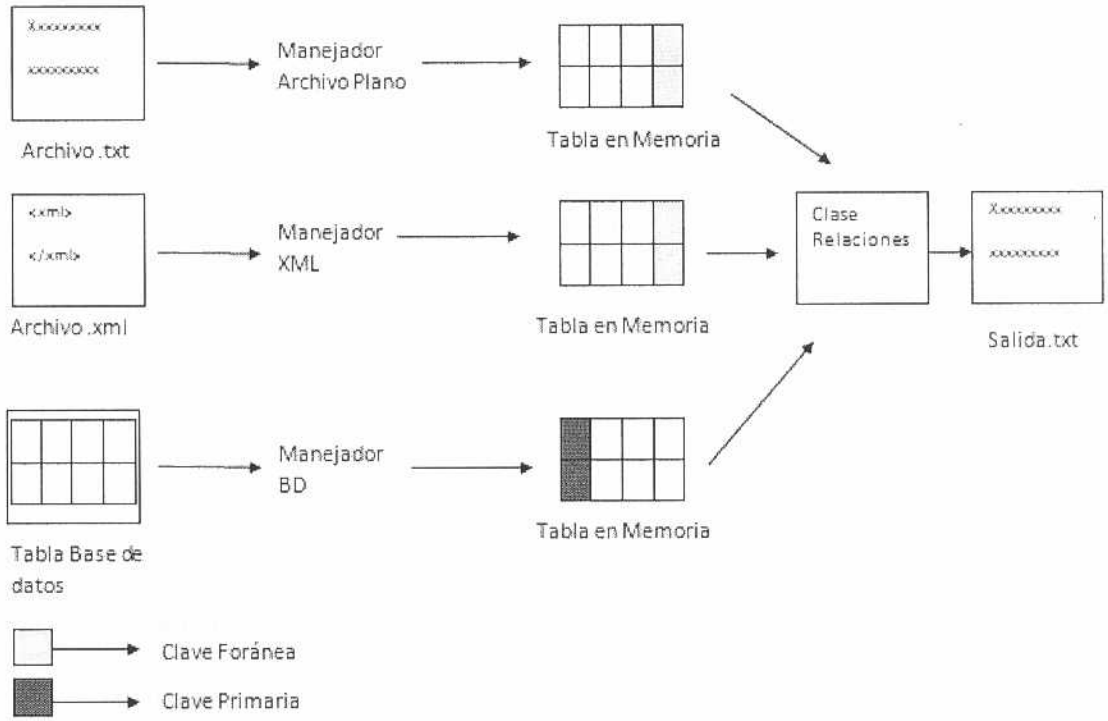

Figura 17 . Proceso de relacionar en el componente extractor.

Fuente: Elaboración propia.

La siguiente tarea es proporcionar funciones al lenguaje de consulta para la manipulación de fechas, las mejoras que se realizaron son las siguientes:

- Crear la palabra clave Fecha_Actual: Esta palabra reservada será sustituida por la fecha en que se esté generando la consulta, es decir, se remplaza por la fecha actual.
- Implementar la función CmpFecha: esta función recibe tres parámetros separados por el carácter ampersand ('&'), los primeros dos se refieren a dos fechas que se quieran comparar, por ejemplo 14/05/2010 y 15/02/2010. El tercer parámetro indica lo que se desea comparar de las fechas, es decir, el día (colocar D), el mes (colocar M), o el año (colocar A). La función devuelve tres valores,
    - Cero (0): Cuando los valores son iguales.
    - Uno negativo (-1): Cuando el primer valor es menor que el segundo.
    - Uno positivo (1): Cuando el primer valor es mayor que el segundo.

  Ejemplo:

    - CmpFecha(Fecha_Actual&plano|cliente|FechaNatal&M), ésta llamada indica que se desea comparar el mes de la fecha actual con el mes de la fecha de nacimiento del cliente.
- Implementar la función DiffFecha: Es similar a la anterior, solo se diferencia en el resultado obtenido ya que en lugar de obtener la relación entre las fechas, lo que se obtiene es el intervalo de tiempo que separa a las mismas, dicho intervalo puede ser devuelto en días (colocar D), en meses (colocar M), o en años (colocar A).

Ejemplo:

- DiffFecha(Fecha_Actual&bd|taxonomia|TarjetaCredito|Vencimiento&M) ésta llamada devolverá el intervalo de tiempo que separa las fechas medido en meses.

IV.11 SEXTA REUNIÓN DE PLANIFICACIÓN

Durante la sexta reunión de planificación se mostraron los resultados obtenidos durante la iteración, y se procedió a definir los requerimientos de la siguiente iteración, los requerimientos escogidos fueron los siguientes:

- Integrar el componente extractor con el componente vigilante.
- Acceder a las fuentes de datos por medio de los siguientes protocolos HTTP, FTP. También es permitido acceder a las fuentes por medio de la lectura del sistema de archivos.
- Para la lectura de bases de datos se debe utilizar la tecnología ODBC de Microsoft.
- Permitir modificar la taxonomía agregada a la plantilla.
- Se debe llevar una bitácora de las llamadas a los métodos que permiten el envío de la plantilla.

El tiempo estimado para realizar la iteración es de quince (15) días.

IV.12 QUINTA ITERACIÓN

Para permitir la integración con el componente vigilante, el componente extractor debe generar un archivo plano (de extensión .angelia) y colocarlo en un directorio que se encuentra monitoreado por el Vigilante. El formato del archivo que se necesita crear se

muestra en la figura 18. El archivo se crea primero con la información que generó la consulta, luego de que se termina de crear, éste se mueve al directorio que monitorea el vigilante. El vigilante procesa el archivo y lo envía vía FTP a la aplicación Mercurio de Tedexis. Mercurio es el encargado del envío de los mensajes. Es importante destacar que el envío de los mensajes no se realiza de manera inmediata ya que los mensajes masivos no requieren de un envío inmediato. El tiempo transcurrido entre la generación del archivo .angelia y el envío del mensaje al cliente es de aproximadamente 45 minutos (este tiempo es un promedio de los resultados obtenidos en las pruebas realizadas).

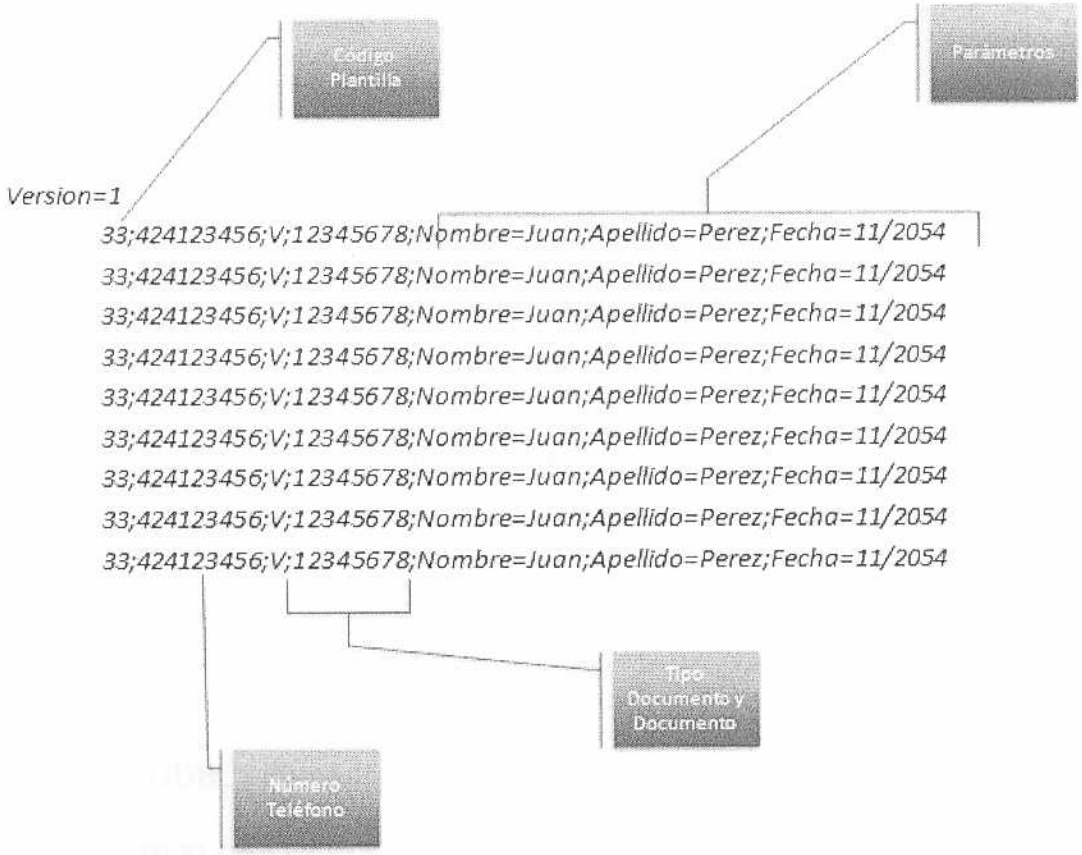

Figura 18 Formato archivo .angelia

Fuente: Modificación propia documentación Angelia.

Para permitir el acceso a las fuentes por medio de diferentes protocolos, se creó un archivo de configuración de Spring donde para cada clase que gestiona el protocolo conozca la ruta a la que tiene que acceder, se proporciona el nombre de la cuenta de usuario (login) y el password en caso de ser requeridos. Un ejemplo del archivo de configuración se muestra en la figura 19.

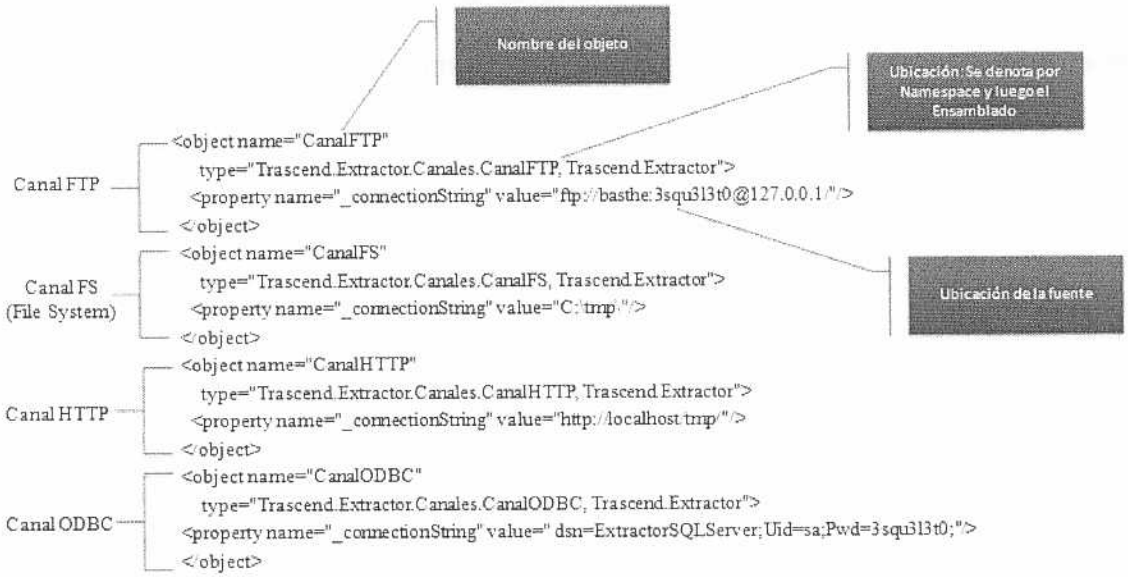

Figura 19 Ejemplo de configuración de canales.

Fuente: Elaboración propia.

Los diferentes componentes que leen fuentes podrán acceder a las mismas por medio de un solo protocolo.

Para hacer el sistema más flexible, sin importar cual sea el protocolo utilizado (a excepción de ODBC) el resultado de la lectura es un flujo de datos. La finalidad de desarrollarlo así es para no tener que modificar los lectores de las fuentes cada vez que se cambie el protocolo que deben utilizar para acceder a las mismas, ya que

independientemente del protocolo utilizado lo que siempre se espera es un flujo de datos.

Para realizar la bitácora (o logger) de la aplicación se utilizó la programación orientada a aspectos (AOP) que proporciona el framework de Spring, para interceptar la llamada a los métodos y la librería Log4Net para inicializar la bitácora de la aplicación. Log4Net se configuró mediante Spring, donde se estableció la ruta del archivo en el que se guardan las trazas, así como definir el nivel del logger. Estos niveles de acuerdo a la gravedad del error pueden ser:

- ALL.
- DEBUG.
- INFO.
- WARN.
- ERROR.
- FATAL.
- OFF.

El punto final para terminar la iteración es proporcionar la interfaz de usuario para modificar la taxonomía asignada a la plantilla, dicha interfaz es similar a la de crear plantilla por lo que no fue necesario crear un prototipo de la misma, solo se procedió a darle funcionalidad añadiendo el store procedure que permita modificar la relación entre la plantilla y la taxonomía, así como el procedimiento que permita recuperar la información para mostrársela al usuario por medio de la interfaz. Finalmente se realizaron pruebas para verificar que los datos se almacenen de manera correcta.

IV.13 SÉPTIMA REUNIÓN DE PLANIFICACIÓN.

Durante la reunión se mostraron los resultados de la iteración anterior y se presentó el sistema en su totalidad, comenzado desde la creación de plantillas con su planificación y taxonomía hasta el momento en que se genera el archivo para el componente vigilante. En este punto ya se ha culminado el desarrollo del proyecto.

CAPITULO V. RESULTADOS

A continuación se mencionan los resultados obtenidos por cada objetivo alcanzado:

Objetivo - Desarrollar el componente de administración de Taxonomías corporativas.

Resultados obtenidos:

- El archivo de configuración de Spring, donde se pueden añadir, modificar y borrar las taxonomías.
- Un servicio web que suministra las taxonomías almacenadas a la interfaz web, con la finalidad de que el usuario pueda asignar o modificar una taxonomía a una plantilla.

Objetivo - Diseñar e implementar la relación entre el componente de administración de Taxonomías y el componente denominado Vigilante.

Resultado obtenido:

- Generación de un archivo plano de extensión .angelia (que proviene del resultado de aplicar una taxonomía) dentro del directorio monitoreado por el vigilante. Por medio de este archivo se realiza la comunicación entre los componentes.

Objetivo - Modificar el componente Vigilante para aceptar el manejo de la configuración de las Taxonomías.

Resultado obtenido:

- Nuevo componente Vigilante que posee la capacidad de interpretar los archivos (.angelia) generados por el componente Mensajero.

Objetivo - Desarrollar el componente de administración de planificación de ejecución de procesos para las siguientes opciones de planificación:

- Anual por día de semana.
- Anual recurrente.
- Diario recurrente.
- Diario por el tipo de día de semana.
- Fecha concreta fija.
- Mensual por día de semana.
- Mensual Recurrente.
- Semanal.

Resultados obtenidos:

- Una interfaz de usuario que permite asignarles a las plantillas opciones de planificación.
- Un servicio Windows encargado de inspeccionar los activadores y de iniciar el envío de las plantillas cuando así lo indique el activador.
- Diagramas de arquitectura del componente administración de planificación, los cuales se realizaron en su fase de diseño.

Objetivo - Desarrollar el componente de extracción de datos, flexible a cualquier fuente de datos (Manejador de Base de datos Relacional, Servicios Web, Servicios AS400, Archivos Planos, Archivos XML).

Resultados obtenidos:

- Diagramas de clases y diagramas de arquitectura del componente extractor los cuales se realizaron en su fase de diseño.
- Se estableció el lenguaje de consulta que indica al componente extractor los datos que debe capturar asociándole restricciones a los mismos. El lenguaje posee la capacidad de poder comparar fechas o calcular el intervalo de tiempo que las separa.
- El componente extractor el cual se adapta a cualquier fuente de datos. Este componente es configurado por medio de Spring, puede acceder a las fuentes de datos utilizando diferentes canales (sistema de archivos, Open Database Connectivity (ODBC), el protocolo FTP y protocolo HTTP). Utilizando la combinación de las fuentes con los canales se pueden realizar consultas más complejas como por ejemplo Servicios Web (XML + HTTP).

Objetivo - Realizar la integración de los componentes al sistema Angelia®.

Resultados obtenidos:

- El componente Mensajero accede a la base de datos de Angelia para obtener y modificar la información sobre las plantillas y sus activadores asociados. Análogamente con las plantillas y su taxonomía asignada.
- Componente Logger el cual genera un archivo con la siguiente información:

- Fecha y hora en la que se accede a un método.
- Fecha y hora en la que se termina la llamada a un método.
- El nombre del método que se está accediendo.
- Los errores que se encontraron al momento de ejecutar el método.

Dicha información es generada con la finalidad de realizar auditorías.

CAPITULO VI CONCLUSIONES Y RECOMENDACIONES

Para el desarrollo del proyecto se utilizó el framework Quartz al momento de ejecutar acciones programadas, una de las potentes ventajas del framework es la utilización de expresiones cron para configurar la planificación de eventos como el último jueves del mes de mayo o el primer domingo del mes de mayo. El cálculo de estas fechas es transparente para el usuario del framework ya que éste es el que se encarga de calcularlas. El framework de Spring permitió realizar una aplicación más flexible permitiendo utilizar la configuración sobre la codificación, gracias a esta propiedad se pueden configurar las taxonomías sin necesidad de modificar el código de la aplicación. No solo las taxonomías sino las propiedades iniciales de los objetos pueden cambiarse sin necesidad de modificar el código fuente, debido a que Spring es utilizado para realizar la inyección de dependencias (Dependency Injection en inglés).

Gracias al uso de estas herramientas se desarrolló una aplicación capaz de adaptarse a las fuentes de datos que se van a consultar, proporcionándole la información mediante archivos XML, eliminando la posibilidad de modificar el código fuente una y otra vez cuando cambien las estructuras de las fuentes de datos.

Se dispone de una variada gama de opciones para programar el envío de una plantilla, permitiendo fechas como el día de las madres (segundo domingo del mes mayo), día del árbol (último domingo del mes de mayo), entre otras fechas que varían de año en año.

Las recomendaciones sobre el trabajo son las siguientes:

- Ampliar el lenguaje de consultas para que proporcione el operador lógico OR en las búsquedas.
- Aumentar el número de funciones que soporta el lenguaje de consultas, con la finalidad de realizar consultas más complejas.
- Al momento de realizar este trabajo, el framework Quartz para plataforma .NET, no soporta todas las expresiones cron. Por lo que se puede utilizar futuras versiones del framework para asignar más opciones de planificación a una la plantilla.
- Actualmente cada manejador (Base de datos, XML, archivo plano) accede a los datos utilizando un solo canal. Por lo que en futuras versiones se recomienda implementar la asignación de uno o más canales a un manejador en particular.
- Se recomienda la implementación de una interfaz de usuario para crear nuevas taxonomías después de la instalación del sistema, debido a que actualmente solo se pueden configurar por medio de un archivo de Spring en la fase de instalación de Angelia.

CAPITULO VII BIBLIOGRAFÍA

Cavaness, C. (2006). *Quartz job scheduling framework: building open source enterprise.* Crawfordsville, Indiana: Prentice Hall.

Documentación sitio web Tedexis. (s.f.). Recuperado el 17 de Septiembre de 2010, de Sitio web Tedexis: http://www.tedexis.com/documentos/SAM2manualFinal.pdf

Dykes, L., & Tittel, E. (2005). *XML For Dummies.* Indianapolis, Indiana: Wiley Publishing, Inc.

Magurno, C. (18 de Junio de 2008). Angelia Resumen. Caracas, Miranda, Venezuela.

Pollack, M., Evans, R., Seovic, A., Baia, B., Eichinger, E., Spinazzi, F., y otros. (17 de Diciembre de 2009). *The Spring.NET Framework Reference Documentation.* Recuperado el 10 de Septiembre de 2010, de The Spring.NET Framework: http://www.springframework.net/docs/1.3.0/reference/html/index.html

Prasanna, D. R. (2009). *Dependency Injection - Design Patterns using Spring and Guice.* Estados Unidos: Manning.

Quartz.NET - Frequently Asked Questions. (s.f.). Recuperado el 17 de Septiembre de 2010, de Quartz.NET - Enterprise Job Scheduler for .NET Platform: http://quartznet.sourceforge.net/

Roman, S. (2002). *Access Database Design & Programming, 3rd Edition.* Sebastopol, CA: O'Reilly.

Sintaxis de Xpath. (s.f.). Recuperado el 12 de Septiembre de 2010, de Microsoft MSDN Library: http://msdn.microsoft.com/es-ve/library/ms256471.aspx

Stal, M. (Octubre de 2007). *Spring.NET - Documentation.* Recuperado el Septiembre de 2010, de Spring.NET: http://www.springframework.net/documentation.html

APÉNDICE I - FRAGMENTO DE LA MATRIZ DE CAMBIOS REALIZADOS AL SISTEMA ANGELIA

(FUENTE ELABORACIÓN PROPIA)

ID	Componente	Directorio	Archivo Compilado	Archivo Fuente	Método/Procedimiento	Detalle	Autor	Tipo de Cambio
114	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx	----------	Se modifica la interfaz para elegir las opciones de planificación	Trascend	Modificación
115	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx	----------	Se modifica la interfaz para elegir las taxonomías por medio de una lista, eliminando el árbol de Areas de Negocio y las taxonomías asignadas a esa área.	Trascend	Modificación
116	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CrearArbolTaxonomiaInicial	Se comenta el método	Trascend	Modificación
117	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	SeleccionarPanelOrigen	Se modifica el método para que cargué las taxonomías, bajo el formato de un DataTable con radiobuttons, en lugar del árbol de taxonomías	Trascend	Modificación
118	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CambiarLabelPlanificacionActivacionCadaDia	Se agrega este método para que muestre en un label, el resultado de elegir la activación diaria.	Trascend	Nuevo

ID	Componente	Directorio	Archivo Compilado	Archivo Fuente	Método/Procedimiento	Detalle	Autor	Tipo de Cambio
119	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CambiarLabelPlanificacionActivacionSemanal	Se agrega este método para que muestre en un label, el resultado de elegir la activación semanal.	Trascend	Nuevo
120	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CambiarLabelPlanificacionActivacionMes	Se agrega este método para que muestre en un label, el resultado de elegir la activación mensual.	Trascend	Nuevo
121	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CambiarLabelPlanificacionActivacionAnnmo	Se agrega este método para que muestre en un label, el resultado de elegir la activación anual.	Trascend	Nuevo
122	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CambiarLabelPlanificacionActivacionFechaExacta	Se agrega este método para que muestre en un label, el resultado de elegir la activación en una fecha concreta.	Trascend	Nuevo
123	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	ControlPanelActivacion	Método que muestra el panel correspondiente según la frecuencia seleccionada	Trascend	Nuevo
124	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CargarActivacionDTDiario	Se modifica el método para que cargué en la clase ActivacionDT la expresión cron asociada a la planificación elegida por el usuario	Trascend	Modificación

ID	Componente	Directorio	Archivo Compilado	Archivo Fuente	Método/Procedimiento	Detalle	Autor	Tipo de Cambio
125	Core	Core\Interfaz\Site	Angelia.Core	CrearPlantilla.aspx.cs	CargarActivacionDTSemanal	Se modifica el método para que cargué en la clase ActivacionDT la expresión cron asociada a la planificación elegida por el usuario	Trascend	Modificación
174	Core	Core\Servicios\TipoDatos	Com.Angelia.Core.Servicios.TipoDatos	ActivacionDT.cs	----------	Se agregan el atributo que guarda expresiones cron	Trascend	Modificación
175	Core	Core\AccesoDatos\SqlServer	Com.Angelia.Core.AccesoDatos.SqlServer	DAOPlantillaAreaNegocioSQLServer	AgregarPlantillaANQ	Se comentan los métodos que ingresan las taxonomías y parámetros en la base de datos. Se agrega la lógica de guardar la relación Plantilla-Taxonomía en la tabla TaxonomiaConsulta	Trascend	Modificación
176	Core	Core\AccesoDatos\SqlServer	Com.Angelia.Core.AccesoDatos.SqlServer	DAOPlantillaAreaNegocioSQLServer	InsertarActivador	Se cambia el método para que agregue en los activadores en la tabla AC_Activador	Trascend	Modificación

ID	Componente	Directorio	Archivo Compilado	Archivo Fuente	Método/Procedimiento	Detalle	Autor	Tipo de Cambio
177	Core	Core\AccesoDatos\SqlServer	Com.Angelia.Core.AccesoDatos.SqlServer	DAOPlantillaAreaNegocioSQLServer	ModificarPlantillaANQ	Se comenta la parte relacionada con la actualización de parámetros y taxonomías en BD. Se agrega la lógica de actualizar la relación Plantilla-Taxonomía en la tabla TaxonomiaConsulta	Trascend	Modificación
178	Core	Core\AccesoDatos\SqlServer	Com.Angelia.Core.AccesoDatos.SqlServer	DAOPlantillaAreaNegocioSQLServer	ModificarActivadores	Se cambia el método para que modifique en los activadores en la tabla AC_Activador	Trascend	Modificación

APÉNDICE II - LISTA DE REQUERIMIENTOS DEL SISTEMA

(FUENTE ELABORACIÓN PROPIA)

Iteración	Requerimiento	Puntuación 1 – Más baja 10 – Más alta	Dificultad	Tiempo estimado (en días hábiles)
1	Ingreso de opciones de planificación a la plantilla	5	Media	10
	Modificación de opciones de planificación para la plantilla	5	Media	5
				15
2	Iniciar el procesamiento de plantilla	5	Media	5
	Actualizar próxima fecha de envío	5	Media	5
	Suspender el envío de plantilla cuando se cumpla la condición de parada	3	Baja	5
				15
3	Acceder a los datos mediante un lenguaje de consulta	7	Alta	7
	Realizar consultas en las fuentes	7	Media	5
	Aplicar restricciones a los campos al momento de generar la consulta	5	Media	3
				15
4	Asignar taxonomías a una plantilla	3	Baja	2
	Realizar consultas que impliquen la lectura varias fuentes	7	Alta	10
	Realizar consultas donde se pueda colocar en las condiciones un periodo de tiempo (5 meses, 30 días, etc.)	5	Media	3
				15
5	Acceso a las fuentes de datos promedio de la lectura del sistema de archivos, FTP, HTTP, ODBC.	6	Media	5

	Modificar la taxonomía asignada a una plantilla	4	Baja	3
	El sistema debe generar un log donde se indique las llamada a los métodos del sistema	5	Media	6
				15

APÉNDICE III - CAPTURAS DE PANTALLA DEL SISTEMA

(FUENTE ELABORACIÓN PROPIA)

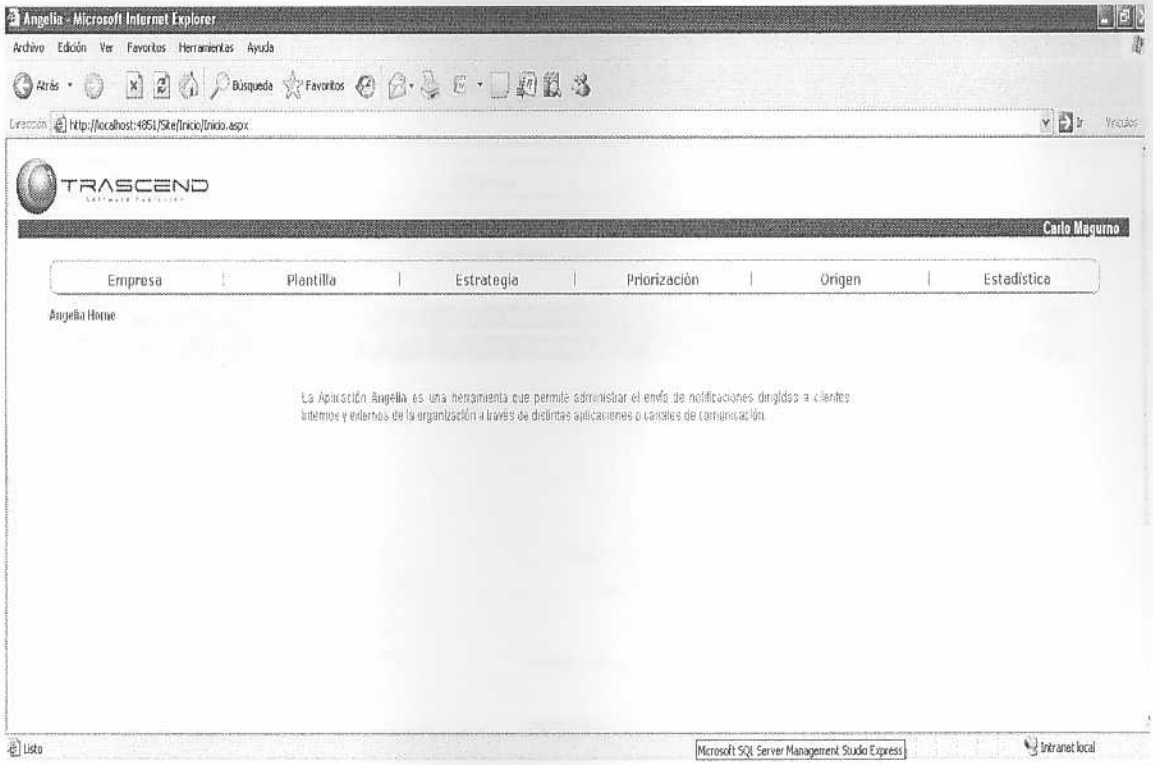

Figura 1. Pantalla inicial del administrador web de Angelia

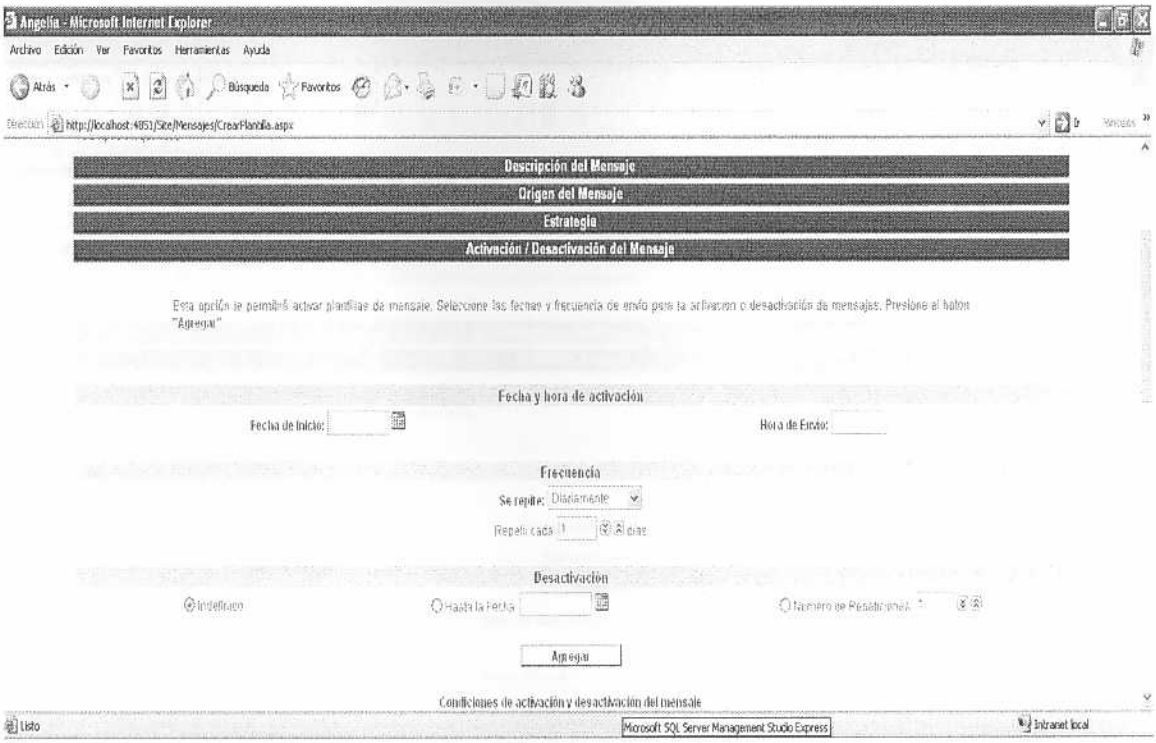

Figura 2. Interfaz para el ingreso de opciones de planificación.

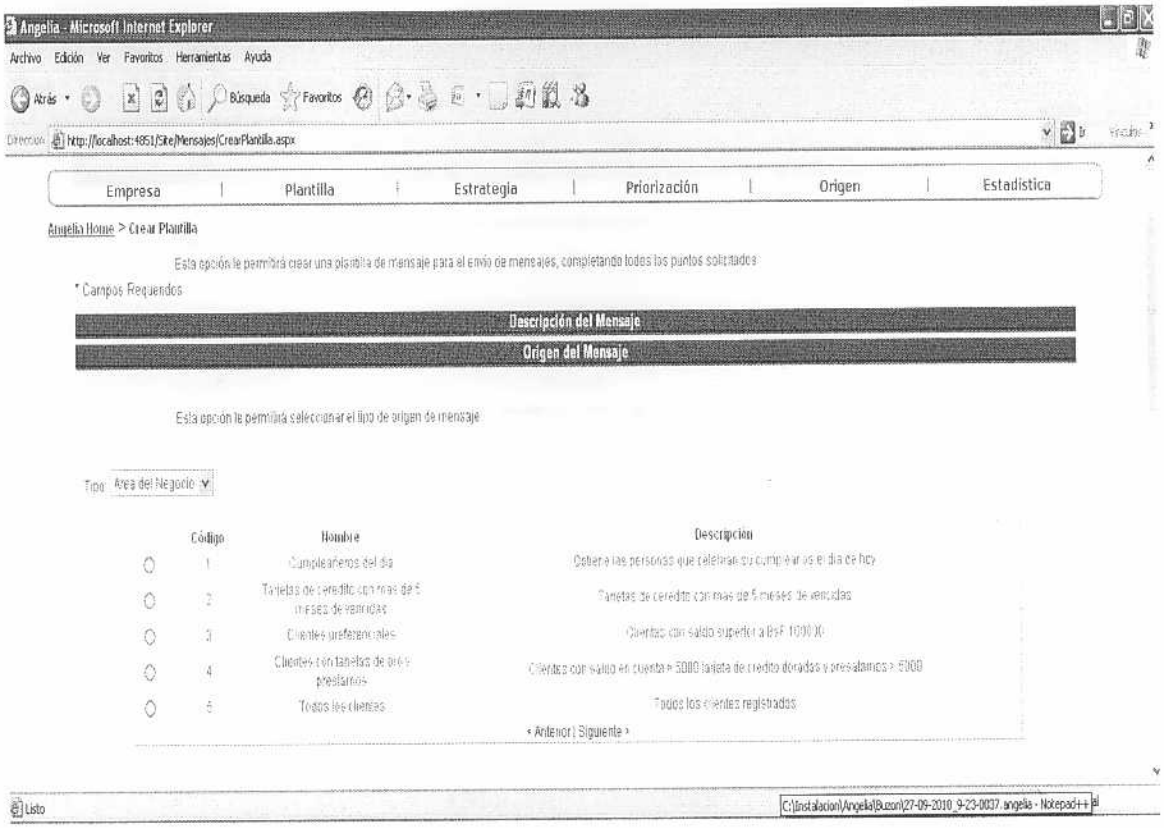

Figura 3. Interfaz para la asignación de taxonomías a las plantillas (Ingreso de plantilla).

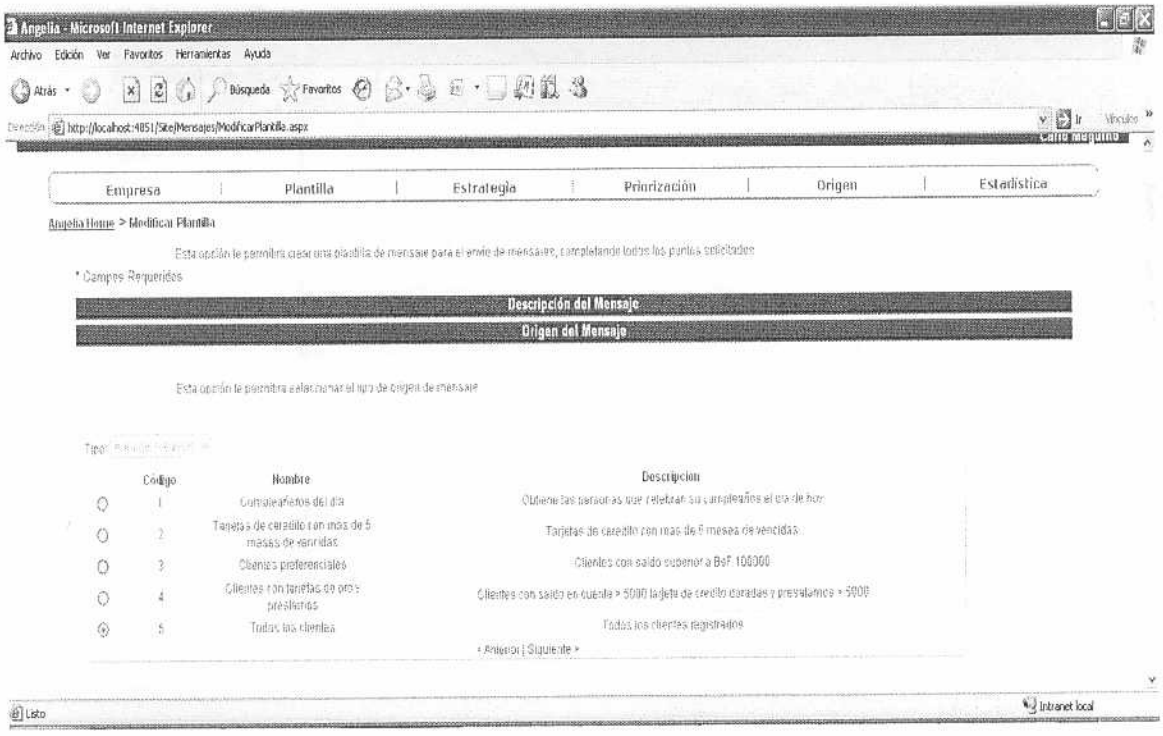

Figura 4. Interfaz para la asignación de taxonomías a las plantillas (Modificar plantilla).

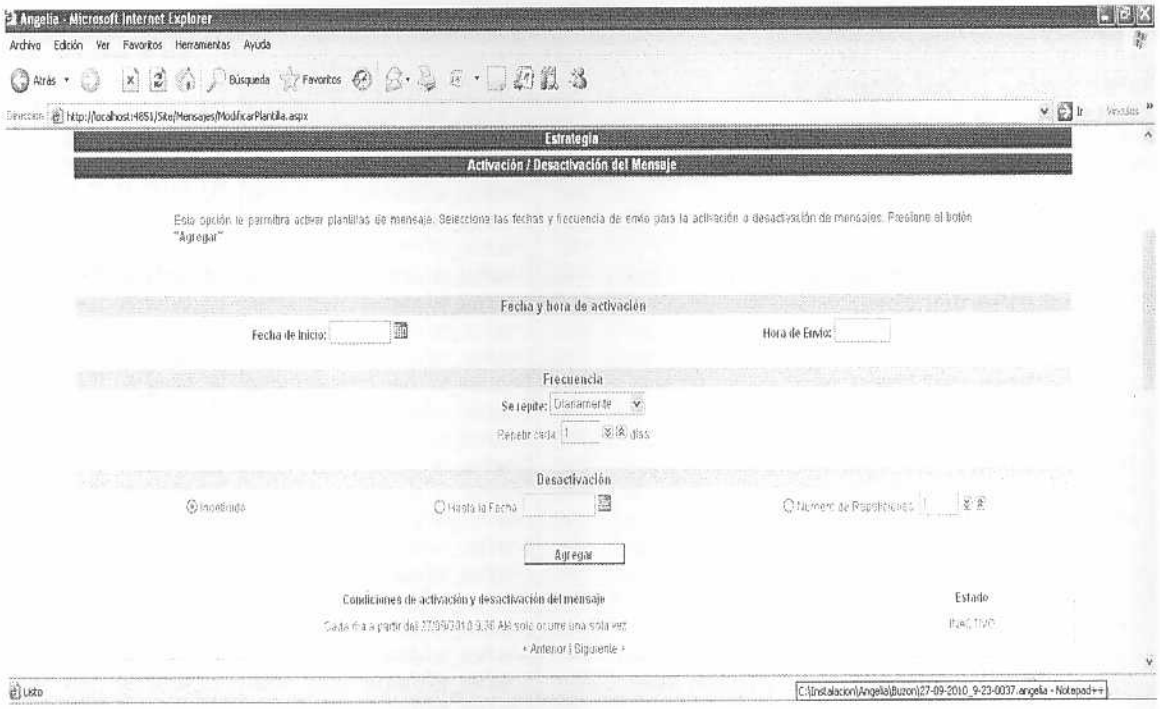

Figura 5. Interfaz para la asignación de opciones de planificación a las plantillas (Modificar plantilla).

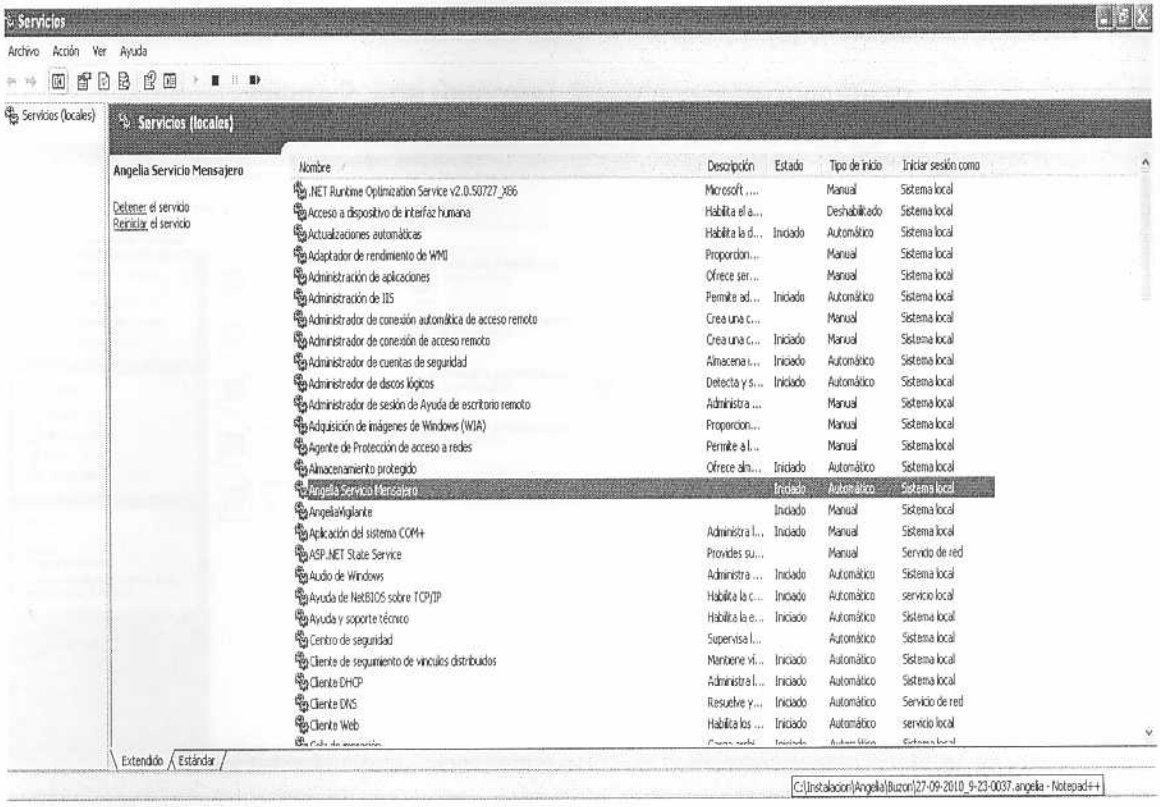

Figura 6. Componente Mensajero (Servicio Windows)

mensajero.txt - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
2010-09-27 08:50:00,036 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Schedules.AgregarAgenda [(null)] - plano|cliente|
2010-09-27 08:50:00,147 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,167 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,177 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,187 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,267 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,267 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,267 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,277 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,277 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,277 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,287 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 08:50:00,287 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,007 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Schedules.AgregarAgenda [(null)] - plano|cliente|
2010-09-27 09:23:00,017 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,017 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,017 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,017 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,027 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,037 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,037 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,037 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,037 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,037 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,047 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:23:00,047 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
2010-09-27 09:36:00,019 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Schedules.AgregarAgenda [(null)] - plano|cliente|
2010-09-27 09:36:00,019 [DefaultQuartzScheduler_Worker-1] INFO  Trascend.Mensajero.Logger.Aspecto.InstrumentacionAroundAd
```

Figura 7. Log (bitácora) del componente mensajero

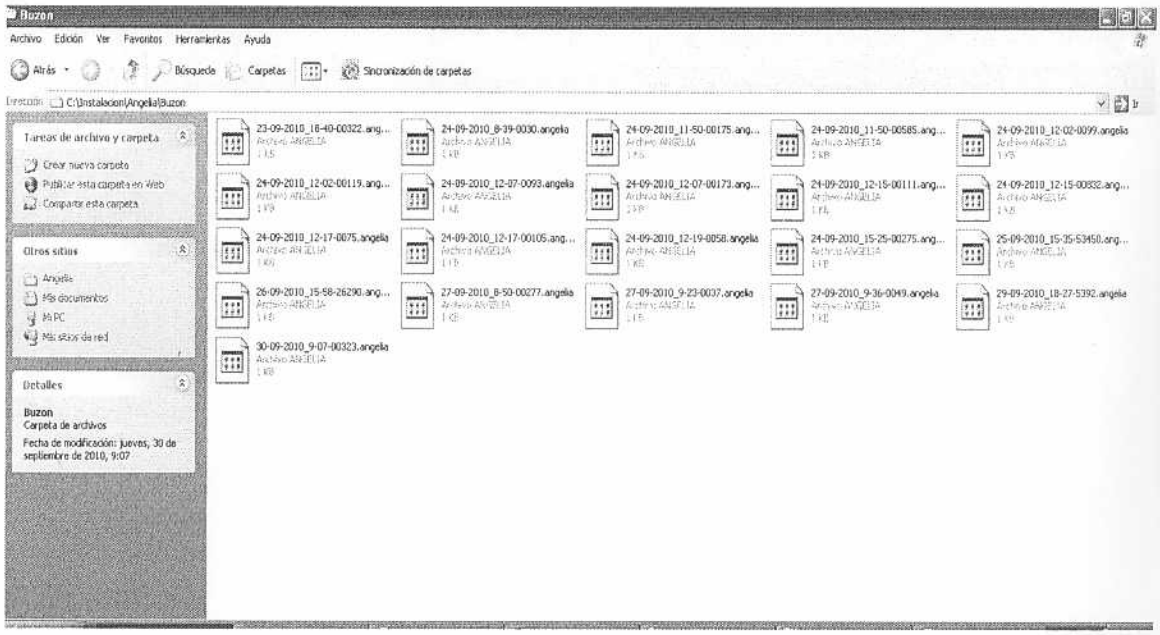

Figura 8. Archivos generados por el componente Mensajero ubicados en el directorio monitoreado por el Vigilante

24-09-2010_11-50-00175.angelia - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
Version=1
6;V;25754603;4168358853;Nombre=Irene;Apellido=Riccio;Saldo=101000,5800
6;V;87948989;4167023153;Nombre=Melissa;Apellido=Bastos;Saldo=1444719,0100
6;V;75218600;4165291304;Nombre=Hernando;Apellido=Bastos;Saldo=4145084,5900
6;V;90762990;4164115095;Nombre=Eduardo;Apellido=Bastos;Saldo=4415881,5600
```

Línea 1, columna 1

Figura 8. Archivo generados por el componente Mensajero