Desarrollo de módulos que apoyen procesos de marcación y consulta masiva de las novedades 155 y 032 para las AFP en la empresa Heinsohn Business Technology

Trabajo de Grado
Modalidad: Práctica Profesional

Maria Paula Aristizabal Galarza

1061803944

Asesor de la empresa: MSc. Diego Arturo Sánchez Cárdenas

Director: PhD. Sandra Milena Roa Martínez

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Programa Ingeniería de Sistemas
Grupo de investigación en Inteligencia Computacional – GICO
Línea de Investigación: Información y Tecnología

Popayán, marzo de 2024

AGRADECIMIENTOS

Agradezco principalmente a Dios y a la virgen por guiarme en este camino, por ayudarme a cultivar la perseverancia y la determinación para culminar este proyecto, a mis padres por darme su apoyo con todos aquellos actos que me formaron cada día para ser un profesional, a mi directora Sandra Milena Roa, una persona fundamental en el desarrollo de este trabajo de grado, a quien admiro y agradezco por su paciencia, dedicación y por todos los consejos recibidos en esta última etapa de mi carrera universitaria que me llevaron a formarme como una profesional.

De igual manera, quiero agradecer a mi asesor Diego Arturo Sánchez quien siempre tuvo la disposición de guiarme y corregirme en la ejecución de toda mi práctica profesional, a la empresa Heinsohn Business Technology por permitirme desarrollar esta práctica y a algunos de mis compañeros los cuales siempre tuvieron la disposición de apoyarme frente a cualquier inquietud.

Por último, quiero agradecerles a mis amigas y Brayan Collazos quienes siempre estuvieron dispuestos a escucharme y no dejarme rendir en este camino. De la misma forma, a todos aquellos compañeros que fueron parte de mi vida en toda mi carrera universitaria.

Tabla de contenido

Lista de figuras

Lista de tablas

1 Introducción

1.1 Planteamiento del problema y justificación

En 1993, Colombia estaba viviendo un cambio en el sistema general de seguridad social debido a la Ley 100, encargada de la regulación del sistema pensional en Colombia, que "busca que los trabajadores tengan una vejez digna y armoniosa. Su base fundamental es garantizar la tranquilidad económica al momento de su jubilación" [1]. En el sistema pensional de Colombia existen dos regímenes: el Régimen de Reparto o de Prima Media (RPM), administrado por Colpensiones y el Régimen de Ahorro Individual con Solidaridad (RAIS) administrado por: Colfondos, Porvenir, Protección y Skandia. *Heinsohn Business Technology* es una empresa colombiana creada el 31 de mayo de 1977, a partir de la reforma de la Ley 100 [2] que encontró una idea de negocio en el régimen privado (RAIS) orientada a la creación de un software para la administración de los fondos privados de pensiones (AFP), el cual buscaba solventar la mayoría de los problemas de información que se intercambiaban entre las AFP.

Los principales problemas que se solventaron con la creación de este software se encuentran asociados a: procesos de multiafiliación, doble aporte (afiliado en ambos regímenes) y la inconsistencia de información que existía entre fondos privados. Para la financiación, todos los fondos que estuvieran asociados al gremio Asociación Colombiana de Administradoras de Fondos de Pensiones y de Cesantías (Asofondos), debían realizar un aporte económico dependiendo del número de afiliados que cada fondo administraba [3]. El software en mención es conocido como Sistema de Información de los Afiliados a los Fondos de Pensión (SIAFP), que originalmente consistía en un sistema para el intercambio de información entre las AFP y un portal *web*, en el que los administradores de cada AFP evidenciaban los cambios, por ejemplo, actualización de nombres del afiliado, consultas de viabilidad de una vinculación a AFP y reconstrucción al historial de vinculación, entre otras [2].

En la actualidad el SIAFP funciona mediante novedades enviadas por las AFP y Colpensiones, cada una de ellas representa un proceso diferente dentro del SIAFP. En primer lugar, estas pueden recibirse a través de tres medios: portal *web*, Validador universal (procesamiento *batch*) o por *Web Services (*Procesamiento en línea*);* posteriormente, se realizan distintos procesos determinados por la novedad, los cuales pueden ser consultas o modificaciones a la base de datos. Finalizado el procesamiento se notifica la respuesta a los interesados por correo electrónico, por *Web Services* (*Alert Manager Demon*), o en algunos casos las AFP puede tomarla a través del Validador universal el cual es el "encargado de establecer una vía de comunicación para el envío y recepción de archivos entre las entidades y el SIAFP" [4].

Así, la novedad 155 denominada SIAFP Autorizador de trámite de pensión*,* tiene como objetivo permitir consultar si el afiliado cumple con los requisitos para poder

iniciar su trámite de pensión, en esta novedad existen 2 canales de envío: novedad 155 línea y consulta página *web* [5]. En el caso de la novedad 032 señalada como: Consulta de AFP actual, permite solicitar información de un afiliado o aportantes a través del envío de dicha novedad; esta solo está habilitada para Asofondos o fondos de pensiones que Asofondos autorice [6]. Dichas novedades cuentan con una alta demanda y son requeridas por las AFP de forma constante porque son claves en el proceso de otorgar el beneficio pensional al aportante.

Cabe destacar que algunas AFP, no disponen de los canales que *Heinsohn Business Technology* ha brindado a través del SIAFP para el envío de la novedad 155 puesto que, los sistemas de dichas entidades no cuentan con la implementación para hacer uso de estos canales de forma adecuada y eficiente. Esto es un impedimento para la aprobación oportuna del beneficio pensional, de igual manera en el momento que se necesite enviar la novedad por temas legales.

Por otro lado, la novedad 032 presenta un déficit de automatización debido que la Oficina de Gestión de Servicios (OGS) [7] se ve involucrada como intermediaria para obtener la información requerida por la AFP, dada la alta demanda de solicitudes que la OGS maneja, se presenta un estancamiento que conlleva a la no entrega oportuna de esta información solicitada por la AFP, lo anterior representa un problema tanto para la empresa *Heinsohn Business Technology* como para las AFP porque retrasa otros procesos para los cuales es vital dicha información.

Teniendo en cuenta las razones mencionadas anteriormente, las AFP identifican la necesidad de ampliar sus canales de envío y solicitud de dichas novedades, por tal razón, la empresa *Heinsohn Business Technology* decidió involucrar a un practicante para que lleve a cabo la implementación de estas funcionalidades que se integran en el SIAFP con el fin de solventar la problemática actual. Por lo tanto, se definió la realización de esta práctica profesional para la creación de dos (2) canales de envío masivo de la novedad, esta será ejecutada por una estudiante del programa de ingeniería de sistemas de la Universidad del Cauca.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar los módulos asociados a la marcación masiva del beneficio pensional (novedad 155) y consulta masiva de afiliados (novedad 032) Sistema de Información de los Afiliados a los Fondos de Pensión (SIAFP) soportado por el equipo de desarrollo de la empresa *Heinsohn Business Technology.*

1.2.2 Objetivos específicos

- Caracterizar el funcionamiento actual del Sistema de Información de los Afiliados a los Fondos de Pensión (SIAFP), específicamente los

procedimientos y requerimientos asociados a las novedades 155 y 032 a ser implementadas.
- Implementar las funcionalidades transversales (cargador, lanzador, notificador, validador y procesamiento) que se integraran en el desarrollo de las novedades 155 y 032.
- Implementar funcionalidades específicas adicionales del validador y de procesamiento para las novedades 155 y 032 a partir de los requerimientos definidos en la fase de caracterización.

1.3 Metodología

Para el desarrollo de esta práctica y el cumplimento de los objetivos planteados se seguirá la metodología empleada en la empresa Heinsohn Business Technology, esta consta de 3 etapas: la primera etapa es “El reconocimiento del SIAFP”, la segunda es “Desarrollo del software” y por último la etapa 3, “Documentación”; adicionalmente, en cada una de estas etapas se incluyen algunas prácticas de metodologías ágiles como son las reuniones diarias (*Daily*), retrospectivas a finalización de cada *sprint* y reuniones de estimación.

1.3.1 Etapa 1. Reconocimiento del SIAFP

Para el inicio del desarrollo de la práctica, es necesario reconocer el estado actual del proyecto sobre los procesos de las novedades 155 y 032, así mismo es importante saber las tecnologías que se están usando dentro del proyecto, las cuales serán empleadas para el desarrollo de la práctica, al igual que el entendimiento de los requerimientos ya previamente levantados por el equipo de análisis de Asofondos, los cuales dieron origen a la concepción de esta práctica.
Principales actividades:

- Reconocimiento de los procesos de negocio de la novedad 032 y 155.
- Análisis de los requisitos.
- Familiarización con las tecnologías que se emplearán.

1.3.2 Etapa 2. Desarrollo del Software

Una vez reconocido los procesos y tecnologías que se necesitan para la ejecución de la práctica se inicia el proceso de desarrollo, incluye 3 subetapas, donde en cada una se asignará el 20% del tiempo estimado para capacitación del estudiante, como una actividad contemplada por la empresa. A continuación, se describen las subetapas de esta fase:

- **Desarrollo del módulo *Framework***
  Implementación del módulo general, será utilizado durante todo el proyecto, contendrá todas las conexiones a las bases de datos y excepciones propias del SIAFP.

- **Desarrollo de módulos transversales**
    - Implementación módulo Cargador: encargado del cargue de los archivos recibidos en el portal y guardar un registro en la base de datos de negocio de *MySQL*.
    - Implementación módulo Lanzador: es un proceso asíncrono, el cual se encargará de enviar los archivos que fueron cargados y están en estado pendiente en la base de datos de negocio de *MySQL* a validar su estructura.
    - Implementación módulo Validador: encargado de generar las validaciones a cada uno de los registros de los archivos cargados, guardar en una tabla de registros si fue correcta la validación, si no almacenarlos en una tabla de error, cambiar el estado del archivo (validado) y finalmente, invocar al módulo de procesamiento. Debe ser un módulo extensible para el uso de diferentes novedades.
    - Implementación módulo Procesador: este módulo es el encargado de procesar cada registro que fue previamente validado, debe ser un módulo extensible para el uso de diferentes novedades.
    - Implementación módulo Notificador: este módulo es el encargado de hacer la notificación del archivo que se entregó por el portal.

- **Desarrollo de módulos específicos**
    - Implementación módulo específico Validador:
        - 032: Validación del tipo de documentos y números válidos.
        - 155: Validación de la longitud del registro a procesar.
    - Implementación módulo no transversal Procesamiento:
        - 032: Consulta para saber el estado general del afiliado.
        - 155: Implementación del llamado al cliente de procesamiento en línea y manejo de respuestas.

1.3.3 Etapa de documentación

Durante la práctica se llevarán a cabo reuniones con el tutor de la universidad y con el asesor de la empresa para mantener una asesoría, realizar el seguimiento para el cumplimiento de los objetivos y se entregarán informes mensuales a la universidad que permitirán evidenciar las actividades y el desarrollo de las metas durante la práctica. Finalmente, se elaborará un documento final tipo monografía de la práctica profesional desarrollada en donde se detallará todo el proceso realizado. A continuación, se presentan las principales actividades a realizar:

- Reuniones con los tutores.
- Informes mensuales del avance.
- Elaboración de monografía.

1.4 Aportes

Desde el punto de vista computacional, la presente práctica profesional ha contribuido significativamente al desarrollo de las siguientes habilidades técnicas: configuración de proyectos Maven, consumo de clientes *rest*, procesamientos *batch*, consumos de servicios mediante *API Rest*. Lo anterior, utilizando los conocimientos adquiridos a lo largo de la carrera como son: fundamentos de algoritmia, estructuras de datos, programación orientada a objetos, bases de datos relacionales, de la misma manera se ha contribuido a generar habilidades blandas como lo son: comunicación con el cliente y la adaptación al trabajo remoto.

Con respecto a la academia, la elaboración de esta práctica profesional busca enaltecer la calidad y el compromiso mostrado por los estudiantes formados en el programa de ingeniería de sistemas de la Universidad del Cauca, se pretende generar oportunidades a otros estudiantes y egresados dentro de la empresa *Heinsohn Business Technology*.

En cuanto al producto, las AFP se benefician de la siguiente manera: (i) estos nuevos canales ayudan en la entrega oportuna de la información, (ii) agiliza los procesos de ley en cuanto a la novedad 155, (iii) se elimina gran parte de la responsabilidad de los procesos a la OGS.

1.5 Organización del documento

A continuación, se describe de manera general el contenido y organización del presente informe final:
1. Introducción: Hace referencia al presente capitulo donde se describe el planteamiento del problema, los aportes generados, los objetivos a cumplir y las actividades a desarrollar.
2. Marco teórico: En este capítulo se describen los conceptos teóricos que son importantes para la realización de la práctica profesional.
3. Caracterización del aplicativo: En este capítulo se describen las características técnicas del SIAFP, como lo es la arquitectura del esquema actual de desarrollos Maven, bases de datos, adicionalmente, se caracteriza el funcionamiento actual de las dos novedades que se desarrollarán en la práctica profesional.
4. Desarrollo de funcionalidades: En este capítulo se detallan los requerimientos requeridos, las actividades realizadas y cada componente generado en cada iteración para cumplir con los objetivos de la práctica profesional.
5. Lecciones aprendidas: En este capítulo se describen las lecciones aprendidas generadas a partir del desarrollo de la práctica profesional.
6. Conclusiones: En este capítulo se describen las conclusiones generadas a partir del desarrollo de esta práctica profesional.
7. Bibliografía: En el último capítulo del presente documento se presentará toda la bibliografía usada en esta práctica profesional.

2 Marco teórico

2.1 Sistema pensional en Colombia

El sistema pensional colombiano (SGP) busca garantizar a la población, protección frente a una contingencia, invalidez, vejez y muerte mediante el reconocimiento de las pensiones y prestaciones que se determinan en la Ley 100 [8]. Su base fundamental es garantizar una estabilidad económica en el momento que esto suceda. El actual sistema reconoce las siguientes prestaciones:

- Pensión de vejez.
- Pensión por invalidez riesgo común.
- Sustitución pensional.
- Indemnización sustitutiva de pensión / devolución de saldos.
- Auxilio funerario.

El sistema pensional está compuesto por dos regímenes:

- Régimen de Reparto o de Prima Media (RPM), administrado por Colpensiones, genera beneficio pensional a sus afiliados o sus beneficiarios de los aportes que realicen durante toda la vida laboral. Este régimen se caracteriza por "ser solidario, los aportes de los afiliados y sus rendimientos constituyen un fondos común de la naturaleza publica, el estado garantiza el pago de tus beneficios, el monto de la pensión, la edad de jubilación y las semanas mínimas de cotización no cambian sin previo aviso" [9].
- Régimen de Ahorro Individual con Solidaridad (RAIS) administrado por Colfondos, Porvenir y Protección y Skandia. Este régimen esta desde 1994 donde su filosofía es que el trabajador sean dueños de su propio ahorro capital para que así tengan la opción de elegir la mejor decisión para ellos. Con este régimen se introdujo el concepto de "cuentas individuales" e "iniciando una red de seguridad en forma de pensiones mínimas y beneficios de vejez garantizado por el estado" [1].

Los Fondos de Pensiones (AFP): son entidades privadas encargadas de administrar el ahorro de cada uno de los afiliados bajo la modalidad de cuentas individuales creados para dar cumplimiento a los planes de pensiones integrados en él [10].

2.2 Sistema de Información de los Afiliados a los Fondos de Pensión (SIAFP)

Dado que los fondos de pensiones privados (RAIS) inicialmente manejaban su información a través de Excel, esto dificultaba la consistencia y la seguridad de la información, generando procesos de multiafiliaciones, doble aporte (afiliado en un fondo privado y en el público), entre otros. Para erradicar estas inconsistencias decidieron contratar a una empresa para la creación de un software, por lo tanto, cada asociado al gremio de Asofondos realiza una contribución teniendo en cuenta el número de afiliados vinculados [2].

A partir de esto, se crea el SIAFP, el cual busca “Diseñar e implantar los mecanismos estructurales, procedimentales, operativos y tecnológicos que garanticen que la información de las Administradoras de Fondos de Pensiones en Colombia sea consistente, veraz, única, oportuna, transparente y ajustada a la ley” [11]. Inicialmente era un sistema para el intercambio de información en el cual se visualizaban los cambios mediante un portal y su primer proceso fue la Consulta de viabilidad. El portal está implementado con el *framework Struts*, su arquitectura está basada en el patrón MVC clásico, conformado por: una capa de *jsp*, un *Bean* de transporte, *EJB*, *action* y unos *DAO*. Para el almacenamiento de la información y su transaccionalidad se utilizó el moto de base de datos *Oracle*, con *PL/SQL* [2].

El SIAFP ha evolucionado dadas las diferentes actualizaciones que se han presentado según las necesidades de las AFP, por esto tiene diferentes canales de comunicación como son *Web Services*, el portal y un aplicativo *standalone* (Validador universal), los cuales reciben la información que trasmiten las AFP, procesa esta información y entrega una respuesta. El SIAFP es un sistema que funciona a partir de novedades las cuales son enviadas por las AFP y Colpensiones, cada una de estas representa un proceso. Cuando son trasmitidas se realiza el procedimiento de actualización o de consulta a la base de datos, realizado por los siguientes pasos [12]:

1. Cargue de las novedades enviadas por las AFP.
2. Identificación de casos para ser consultados a entidades externas (Registraduría, Migración Colombia, Data Crédito) y generación de bloqueos.
3. Procesamiento de la novedad, generación de los archivos de respuesta y notificación de ellos [12].

A continuación, se describen los principales módulos que actualmente componen este sistema, de los cuales se enmarca el desarrollo que se realizará durante la práctica profesional propuesta:

- **Módulo de beneficios pensionales**: Tiene como objetivo “reflejar en el SIAFP la información básica de los trámites de pensiones que realizan las AFP y las entidades del Régimen de Prima Media, con el fin de prevenir el reconocimiento de dobles beneficios en el Sistema General de Pensiones y permitir la implementación de controles dentro de los diferentes procesos del sistema” [13], este módulo contiene:
    - **SIAFP Autorizador de trámite de pensión (Novedad 155):** Tiene como objetivo consultar si el afiliado cumple con los requisitos para poder iniciar su trámite de pensión. El proceso para esto es denominado Consulta de factibilidad del beneficio pensional donde se debe realizar: (i) una consulta de viabilidad, (ii) verificación de marcación de beneficios, (iii) consulta a Colpensiones, (iv) consulta a la OBP y a la Registraduría. Todo esto con la finalidad de identificar

en qué estado se encuentra el afiliado y si es viable la autorización del trámite de pensión.

Es importante tener en cuenta que: ”Si el afiliado no se encuentra en ningún trámite de pensión de acuerdo con las respuestas dadas por Colpensiones y OBP, el SIAFP le permite al usuario dar inicio al procesamiento de la novedad 155 - Actualización del estado pensional vía procesamiento en línea para notificar inmediatamente al afiliado la respuesta a este trámite” [5].

Adicionalmente, existen 3 canales para el envío de esta novedad: Consulta *Web service,* Novedad 155 Línea y Consulta página *web*.

- **Consultas masivas:** El SIAFP ofrece consultas en línea a través de la página *web* donde se pueden generar consultas masivas de información de afiliados, aportantes, historia laboral, relaciones laborales e historial de vinculaciones. Esto se hace a través del envío de novedades, las cuales cuentan con definiciones de seguridad que determinan los usuarios que podrán hacer uso de estas, al igual que ver el resultado. “Los usuarios de Asofondos son los únicos que tienen acceso a toda la información del SIAFP a través de las consultas masivas” [6]. Entre estas consultas se encuentra: la Consulta de AFP Actual (Novedad 032), siendo el objetivo de esta novedad informar a las AFP el estado actual del afiliado entregando información como lo es: datos básicos del afiliado, su historia laboral, si tiene beneficio pensional, entre otra información, donde actualmente es un proceso que lo realiza la OGS [7].

2.3 Tecnologías utilizadas en el SIAFP

Las tecnologías que en el SIAFP se usan actualmente son: ANT, Maven y *frameworks* como lo son *Struts* y *Angular*, al igual que se usan motores de bases de datos como lo son *Oracle*, seleccionado porque cumplía con las especificaciones requeridas al inicio del SIAFP, *MySQL* ya que es una herramienta *Open Source,* usada para almacenar trazas de *logs* de cada servicio o bitácoras temporales. A continuación, se detallan algunas de las tecnologías mencionadas que fueron utilizadas para la implementación de esta práctica.

2.3.1 Maven

Es una herramienta para la construcción y administración de proyectos basados en Java [14], que ayuda en la compilación y el empaquetado, ya que está basado en el Modelo de Objetos de Proyecto (POM) el cual es un archivo de Lenguaje de Marcas Extensible (XML) (pom.xml) que presenta toda la configuración y detalles del proyecto como lo son dependencias, *plugins*, versiones, este se encuentra ubicado en el directorio raíz del proyecto [15].

El objetivo principal de Maven es que el desarrollador comprenda el estado del proyecto en el menor tiempo posible gracias a las siguientes características [16]:

- Facilitar el proceso de construcción.
- Proporcionar un sistema de construcción uniforme.
- Proporcionar información de calidad sobre el proyecto.
- Fomentar mejores prácticas de desarrollo.

Maven fue creada en el 2001 para facilitar el proceso de compilación, evitar el análisis de librerías y dependencias. Debido a esto, muchos de los desarrolladores empleaban más tiempo para ejecutar esta etapa. Con el fin de erradicar esa necesidad surgió Maven como una solución *Open Source*, que proporciona y mejora las etapas del ciclo de vida del *software* desde el momento de la construcción hasta el despliegue teniendo en cuenta la etapa de pruebas [17].

Maven se caracteriza por la fácil configuración de un proyecto, la gestión de dependencias, su extensibilidad, por el control de versiones, por su extenso repositorio de librerías. Además, por tener la capacidad de compilar cualquier cantidad de proyectos teniendo como salida ejecutables tipo: .jar, .war y .ear [18].

2.3.2 Arquetipos

“Un arquetipo es un patrón o modelo sobre el que se pueden desarrollar todas aquellas tareas que son de un mismo tipo. Puede decirse que son plantillas, parametrizadas o configuradas para utilizar determinadas tecnologías, que los desarrolladores utilizan como base para escribir y organizar el código de la aplicación” [19]. Los arquetipos deben cumplir con ventajas como patrones homogéneos, reutilización de código, reducción de tiempo de implementación, entre otras. En el SIAFP existen 5 tipos de arquetipos:

- Cliente *Rest.*
- Cliente *SOAP.*
- Servicio *Rest*.
- Servicio *SOAP.*
- *Standalone*.

Estos arquetipos están implementados en Maven cumpliendo con la estructura de un archivo XML (pom.xml), ficheros donde se ubican clases y diferentes componentes como: propiedades de negocio, *scripts* para la creación de tablas de *logs*, *jobs* de base datos, propiedades de conexión a la base de datos, artefactos como archivos de configuraciones para generar el despliegue del proyecto y *templates* de informes.

2.3.3 Servicio web

Es un componente *software* con el que se comunican otras aplicaciones, describe una funcionalidad especifica de negocio comunicándose a través de mensajes codificados de XML, enviados por medio de protocolos (HTTP - Lenguajes de

Marcas para Hipertexto), estos no cuentan con interfaz de usuario [20] y permiten la comunicación sin importar el lenguaje o en que plataforma esté desarrollado. “Pueden ser diseñados para soportar la interoperabilidad máquina a máquina a través de una red. El servicio *web* tiene una interfaz descrita en un formato procesable por máquina” [21]. Los servicios *web* se caracterizan por ser accesibles a través de la *web*, el servicio debe tener una descripción de sí mismo para saber que operaciones tiene y cuál es su interfaz, esta debe ser fuertemente tipada y debe garantizar su interoperabilidad.

2.3.3.1 Servicio Web Rest (*REpresentacional State Transfer*)

Son considerados como un estilo de arquitectura para sistemas distribuidos de hipermedia, que accede a los recursos de manera sencilla y sin tener en cuenta los estados del servidor ni del cliente, es un estilo arquitectónico cliente-servidor, que expone los recursos identificados por un URI, los clientes interactúan con ellos mediante métodos de ingreso, estos la mayoría de veces procesan y generan información [21], este estilo arquitectónico utiliza 4 métodos fundamentales *GET, PUT, POST y DELETE,* estos métodos son para operaciones: recuperación, creación, actualización y borrado de recursos del servidor [22]. Este tipo de servicio se caracteriza por la escalabilidad para manejar el crecimiento de los componentes, la generalidad de interfaces dado que usa el protocolo HTTP y el poder interactuar con cualquier servidor HTTP [23].

2.3.3.2 Cliente Rest

Es una aplicación utilizada para interactuar con servicios *web* basados en la arquitectura *rest*, es cualquier herramienta que actué en representación del usuario, se comunican mediante un intercambio de representaciones de recursos donde los mensajes enviados desde el cliente se llaman peticiones y los mensajes enviados desde el servidor se llaman respuesta; el servidor nunca inicia la comunicación [24].

2.3.3.3 Servicio SOAP (Simple Object Access Protocol)

Los servicios *web* basados en SOAP incluyen estándares como WSDL, (*WSsecurity*, *WS-Transaction)* en este se definen las operaciones. Estos servicios están compuestos por un *Header* (encabezado) y un *Body* (cuerpo). La finalidad del *Header*, es proporcionar la información necesaria del mensaje, en el *Body* se especifica la carga útil, que es el motivo de la comunicación y la acción que debe realizar el destinatario. El envío de mensajes es a través de XML, utilizando el protocolo de HTTP como medio de transporte [23].

2.3.3.4 Cliente SOAP

Es un componente software que envía solicitudes y recibe utilizando el protocolo SOAP, en los mensajes se tiene una definición de encabezado (credenciales de seguridad) y cuerpo (información de la solicitud), este se formatea en un XML [25]. Para la creación del cliente se debe tener el WSDL que es el lenguaje de descripción del servicio [26].

2.3.3.5 Arquitectura Cliente-servidor

El servidor es el componente que procesa las solicitudes del cliente y retorna los servicios solicitados, es responsable de la gestión de los datos y la lógica del negocio. El cliente es el encargado de solicitar y usar los servicios proporcionados por el servidor.

Esta arquitectura se compone por clientes *web*, servidor de aplicaciones, un protocolo, un controlador, lógica de negocio y base de datos. Donde el componente del servidor de aplicaciones recibe una solicitud de un cliente por medio del servidor *web* o de aplicaciones (*wildfly*) para después procesarla. En esta capa se pueden manejar múltiples solicitudes al mismo tiempo de diferentes clientes [27].

2.3.3.6 Wildfly

Es un servidor de aplicación de Java, es *Open Source* patrocinado por Red Hat, es compatible con cualquier sistema operativo donde este instalada la máquina virtual de Java. *Wildfly* proporciona una plataforma para el desarrollo y despliegue de aplicaciones Java en un entorno de servidor, funciona por medio de un *web server* de alto rendimiento, excelente para las aplicaciones de alto nivel de tráfico de solicitudes, escalabilidad y seguridad del proyecto [28].

"*Wildfly* adopta un enfoque agresivo para la gestión de la memoria y se basa en subsistemas conectables que se pueden agregar o eliminar según sea necesario". La integración de Maven y *Wildfly* permite aprovechar el ciclo de vida de construcción (compilar, probar y desplegar) de las aplicaciones de manera eficiente facilitando su integración continua [29].

2.3.4 Procesamiento *batch* o por lotes

El procesamiento *batch* o por lotes consiste en la ejecución de un programa sin la necesidad de una supervisión humana ni la interacción de un usuario, los procesamientos *batch* no se detienen hasta su finalización o por el surgimiento de un error, tienen un orden de secuencia. "Algunos trabajos se completan en tiempo real con funciones de monitoreo e informes diarios, otros se realizan de inmediato" [30]. Generalmente son usados para la ejecución periódica y automática de trabajos repetitivos y de gran volumen, por ejemplo, son las copias de seguridad, el filtrado y la clasificación de datos [31].

Las ventajas del procesamiento por lotes son: las tareas se ejecutarán de forma eficiente, la interacción humana es casi nula, son procesos de bajo costo siendo una de las soluciones más usadas por la industria por que se ahorra dinero y tiempo, se ejecutan en momentos donde hay menor actividad, acelerando los procesos y minimizado el error humano.

2.3.5 Perfilamiento en Maven

Los perfiles ayudan en la personalización de la construcción del *software* en la compilación de un artefacto, dado que son conjuntos de valores de configuración que determinan la activación o el sobrescribir valores. Existen diferentes tipos de perfiles:

- Perfil de proyecto: Se definen dentro del pom.xml y se identifican a través de un id con el elemento *xml profiles.*
- Perfil de usuario: se definen dentro del archivo *setting.xml asociados a él %USER_HOME* dentro %/.m2/*setting.xml*.
- Perfil global: *se* definen en *setting.xml* asociado a %M2_HOME, sin embargo, puede estar ubicado en %/.config/*setting.xml*.

Entre los valores que se pueden cambiar según el perfil son las dependencias, variables, *plugin*s y módulos. Para la activación de los perfiles existe la forma explícita usando la consola con el siguiente comando: "*mvn comando -Pidperfil*", por ejemplo: *mvn clean install -Pweblogic.*
Otras formas de activación son las implícitas:

- Basada en el sistema operativo.
- Basada en el JDK.
- Basada en propiedad.
- Basada en la existencia de un fichero.

3 Caracterización del SIAFP

En la primera etapa de la metodología fue necesario reconocer aspectos del proyecto como son las tecnologías, conocimiento del negocio, requerimientos que se generaron para esta práctica profesional, al igual que metodologías de desarrollo que usa el proyecto basado en las condiciones y restricciones de la empresa *Heinsohn Business Technology* y el cliente que para el SIAFP es Asofondos.

3.1 Patrón Arquitectónico

En el SIAFP existen diferentes arquitecturas dado que es un proyecto muy extenso que lleva varios años en el mercado, por ende, se encuentra implementado en tecnologías antiguas y ha tenido diversos cambios a medida que ha pasado el tiempo. Una de estas arquitecturas es el Modelo Vista y Controlador (MVC) que se evidencia en el portal del SIAFP [32], su implementación está dada por el *framework Struts* el cual tiene esta arquitectura.

El controlador recibe las peticiones que se generaron en la vista, lo que hace que se desencadenen las acciones adecuadas en el modelo, creando una respuesta que se mostrará nuevamente en la vista [33] haciendo uso de los *jsp*, estos son una mezcla de Java y HTML, el modelo es la implementación propia de cada negocio, en este caso del SIAFP.

El portal es desplegado en el servidor de aplicaciones Java *Weblogic* donde este ya tiene la conexión a la base de datos de negocio (*Oracle*), en este servidor se encuentran otros procesos implementados en ANT, representando otra arquitectura del SIAFP, que es la arquitectura basada en componentes.

La arquitectura de software basada en componentes “se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas” [34]. Vale destacar que, un componente es cada una de las partes que cumple con un propósito del sistema donde cada uno es completamente independiente y reusable, deben ser extensibles y capaces de operar en diferentes contextos con el fin que al ser unidos se forme el sistema completo. Esta arquitectura facilita la instalación, el desarrollo y el mantenimiento junto con otros beneficios.

En el SIAFP en el esquema ANT se tienen componentes de tipo:

- APL que son todos los procesos *Standalone*.
- CLI que son los clientes para el consumo de servicios.
- NEG que son proyectos que encapsulan toda la lógica de negocio.
- WS que son proyectos para la exposición de cada servicio.
- El portal el cual es un monolito implementado en *Struts.*

En el esquema de Maven se tienen otros componentes donde la gran mayoría fueron creados por medio de los arquetipos. En el SIAFP hay componentes que son

usados por otros, ya que son procesos comunes en los proyectos como lo son conexiones a la base de datos, manejo de *logs*, manejo de propiedades, clientes (*Rest* / SOAP), *proxys* que se usan para viajar a entidades externas como Registraduría, Colpensiones o Migración Colombia, todos estos componentes se agrupan dentro de un componente que en el SIAFP se llama General, un ejemplo son los proyectos de tipo servicios (SOAP/*Rest*), dado que ellos consultan a la base de datos, generan trazabilidad (*logs*) y algunos deben consultar a entidades externas.

Un servicio de tipo *rest* en el SIAFP se conforma por tres (3) componentes:

- El componente Negocio: se aloja toda la lógica propia de cada proceso, pueden haber llamados a procesos comunes como, por ejemplo, algoritmos de comparaciones de nombres, funciones para comprimir archivos, procesos para envío de correos, entre otros.
- El componente de Persistencia: se maneja la conexión a la base de datos, se hace uso del patrón DAO para abstraer y encapsular el acceso a la base de datos. En este componente normalmente se encuentran métodos para la ejecución de: inserciones, eliminaciones, actualizaciones, consultas o llamados a procedimiento *Oracle*. Se debe agregar la dependencia de base datos del componente General.
- El componente Utilidades: se encuentran los modelos propios de cada proceso, se crean constantes, se pueden agregar dependencias comunes como son algunos clientes, los cuales son invocados en el componente de Negocio; se adicionará la dependencia al componente General para hacer uso de la trazabilidad y la obtención de propiedades, las cuales serán usadas en el componente Modelo o en sí mismo.

Adicionalmente, en la Figura 1 se pueden observar las relaciones entre los componentes: el componente Negocio tiene una dependencia al componente de Persistencia; el componente de persistencia tiene una dependencia al componente de Utilidades.

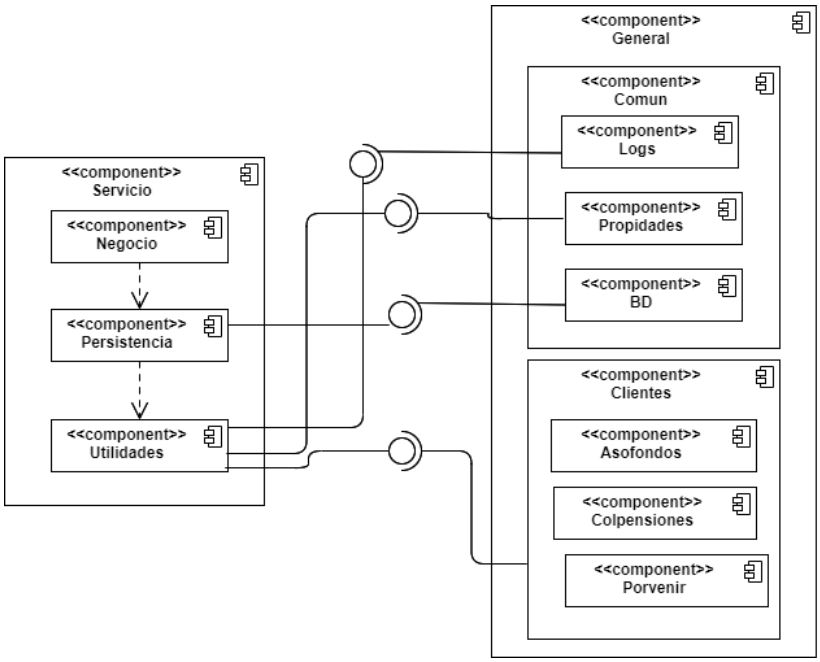


Figura 1. Abstracción de arquitectura de un servicio *rest*.

De la arquitectura de un servicio *rest* del SIAFP se tienen como ejemplo los siguientes servicios:

- Notificación de historia laboral por parte de Colpensiones.
- Consulta de viabilidad del ISS.
- Servicios para la generación de reportes del consumó por AFP a entidades externas.

Los proyectos implementados en Maven se despliegan en un servidor gratuito que es *wildfly* en una maquina *Linux* que esta contenerizada sobre *OKE,* el cual es un servicio de *kubernetes* de *Oracle*, es importante aclarar que los proyectos implementados en Maven y en ANT comparten el mismo lugar de almacenamiento de archivos de negocio (*file system* compartido), al igual que la misma fuente de datos de negocio que es la base de datos de *Oracle,* sin embargo, el punto de montaje es diferente según el proyecto, adicionalmente se utilizan tres (3) bases de datos implementadas en *MySQL:* base de datos de propiedades, base de datos *Logs*/Caché, base de datos de negocio.

3.2 Bases de datos

En el SIAFP se utilizan dos motores de bases de datos *Oracle y MySQL*. Al inicio del sistema se seleccionó *Oracle* ya que cumplía con las características necesarias como lo son seguridad, escalabilidad, entre otras, por esta razón, se implementaron bases de datos para almacenar los temas de negocio como información básica de los afiliados, información de sus vinculaciones, historial de novedades y demás.

Dado que *Oracle* no es un motor de base de datos *Open Source,* el cliente solicita migrar cierta información de bitácoras, *logs* e información de negocio a un motor de base de datos libre de pago, con el fin de reducir costos, más sin excluir algunas características ofrecidas por *Oracle*, por esta razón, se escogió *MySQL* dado que cumple con las características de: código abierto, procesar grandes volúmenes de datos, reducir tiempos y costos, con alto rendimiento y escalabilidad [35] para guardar la información indicada por el cliente.

Las bases de datos implementadas en *MySQL* son:

- Base de datos de propiedades: dedicada para almacenar las propiedades (*properties*) de los componentes, como por ejemplo URLS a externos, valores que se puedan cambiar a través de servicios de la OGS como lo son usuarios, tipos de documentos permitidos en un servicio.
- Base de datos *Logs*/Caché: dedicada para almacenar la trazabilidad de los servicios.
- Base de datos de negocio: dedicada a almacenar información de negocio como lo son bitácoras para la generación de estadísticas, información de la gestión de algún procesamiento, entre otra información según sea la necesidad.

La base de datos implementada en *Oracle* está relacionada con los datos del afiliado. Es importante aclarar que todas las bases de datos mencionadas anteriormente existen para cada ambiente (desarrollo, pruebas, preproducción, producción).

3.3 Caracterización Novedad 155 y 032

3.3.1 Novedad 155

El SIAFP funciona mediante el envío y recepción de novedades, donde estas son procesadas y notificadas a todos los interesados, esta notificación se puede realizar por diferentes medios. De la misma manera funciona la recepción, por esta razón, la novedad 155 tiene tres (3) canales de recepción que han sido desarrollados a medida de las necesidades de las administradoras (AFP) y de su tecnología.

3.3.1.1 Novedad 155 vía Consulta Web Server

El SIAFP pone a disposición un *Web Servicie* (Procesamiento en línea) a las AFP el cual les permite procesar la novedad, enviando el afiliado al cual quieren generarle el beneficio pensional. En este proceso solo se puede enviar de un afiliado a procesar, a través de la estructura Tipo 2, la cual contiene la siguiente información:

- Identificación del afiliado.
- Nombres del afiliado.
- Estado pensional actual del afiliado.
- Fecha de solicitud de la pensión.
- Tipo de pensión del afiliado.

- Lugar donde se radica la solicitud.

Este servicio se encargará de recibir esta estructura, generar las validaciones de forma y negocio como por ejemplo, validar que el tipo de documento sea válido en el SIAFP, que la longitud de la identificación este correcta según el tipo de documento enviado y asimismo se encargará de validar que el afiliado no haya recibido el beneficio pensional antes [36]. Después de aceptadas todas las validaciones, el *Web Service* deberá hacer el llamado a un procedimiento Oracle, para que se encargue de ejecutar la novedad y de generar una respuesta al *Web Service*.

La ejecución del *Web Service* es síncrona, cuando no genera la respuesta al proceso en un tiempo estimado retornará un t*oken* como identificador del proceso, con el fin de validar el resultado del procesamiento, esto también se debe a que el proyecto de procesamiento en línea no solo es usado para la novedad 155 sino para la recepción de muchas novedades del SIAFP [37].

3.3.1.2 Novedad 155 vía Portal *web*

El SIAFP dispone de un canal a través del portal para la recepción de esta novedad, el cual generará la invocación al *Web Service* de procesamiento en línea y este se encargará de validar, procesar la novedad para que el portal reciba la respuesta y se notifique al usuario [38].

3.3.1.3 Novedad 155 por *batch*

El SIAFP cuenta con un canal en que el usuario de una AFP se encargará de subir un archivo plano donde se encuentren todos los afiliados a los cuales se requiere procesar esta novedad; para autorizar su beneficio pensional cada uno de los afiliados viene representado en una estructura Tipo 2. La recepción de este archivo se realiza por medio del Validador universal, el cual es un proceso *batch* que se encarga de generar las validaciones de estructuras propias de cada novedad descritas en archivos XML, donde en ellos se tienen atributos como la posición de un campo (inicio, fin), el tipo de dato que debe cumplir, si el campo debe traer decimales y cuál es su equivalencia en la base de datos *Oracle*.

Después de esto, el Validador ubicará el archivo en la “máquina de archivos”, seguidamente un operador de la Oficina de Gestión de Servicios (OGS) lo enviará al proceso llamado despachador, el cual se encargará de hacer el llamado a los procesos PL quienes procesan las novedades, estos dos procesos se implementaron por medio de ANT.

3.3.2 Novedad 032

Para la novedad 032 se tienen dos (2) canales de recepción de los cuales uno de estos fue la razón por la cual se generó esta práctica:

3.3.2.1 Novedad 032 por servicio

En OGS, se cuenta con un servicio que solicita el procesamiento de la novedad 032, este proceso consiste en que un usuario de una AFP o de Asofondos registre el servicio en la plataforma Service Desk, para que después el operador encargado de procesar la novedad manualmente entregue la respuesta al usuario que registro el servicio, esta novedad es de consulta, lo que quiere decir que el operador encargado debe ejecutar ciertos *scripts* dentro de la base de datos *Oracle* para obtener la información que será entregada.

3.3.2.2 Novedad 032 por batch

Este canal funciona a través del Validador universal, en el cual se cargará un archivo con cada uno de los registros (tramas) que se quieren que se procesen. Las tramas de esta novedad están compuestas por dos campos que son: el tipo de documento y el número de documento. Después de recibido el archivo cada una de las tramas es evaluada, si este proceso es exitoso se mueve a la máquina de archivos, para que así un operador de la OGS lo envié al despachador, que es un proyecto que se encargará de procesar los archivos según su nomenclatura y su novedad. Posteriormente de procesado se generará un archivo de respuesta, con cada una de las respuesta del procesamiento de la tramas que se encontraban en el archivo de entrada, para que el usuario lo pueda tomar a través del Validador universal [39].

3.4 Análisis de requisitos

Para el desarrollo de esta práctica, inicialmente se realizó un análisis de los requisitos planteados por la analista de Asofondos y la tester del proyecto, realizándose las siguientes actividades:

- Lectura de ambos requerimientos.
- Validación con analista y la tester para generar el flujo de la funcionalidad.
- Validación con el asesor de la práctica para la indicación de los aspectos técnicos.

Por lo tanto, se generaron diferentes versiones del diagrama de secuencia, el cuál brinda un entendimiento del flujo general del requerimiento; en la Figura 1 se puede observar la versión final.

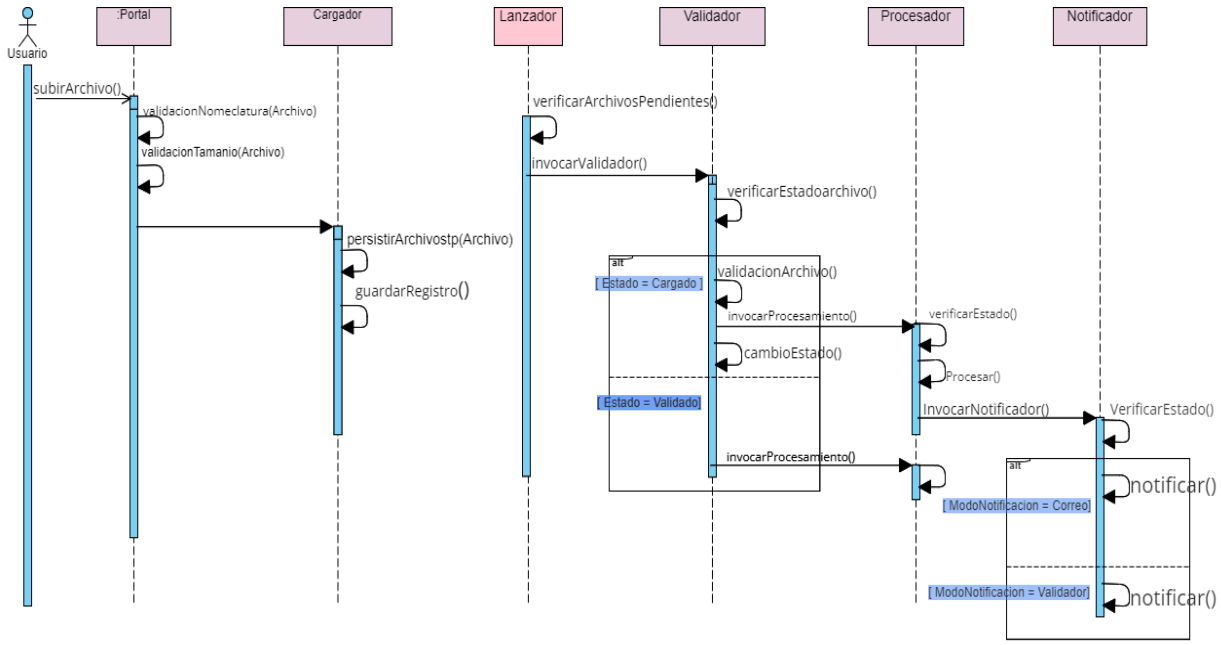


Figura 1. Diagrama de secuencia del flujo general

De la Figura 1, se observa que: primero se generará la subida del archivo por parte del usuario al portal, segundo el portal se encargará de validar nomenclatura, tamaño del archivo para posteriormente enviarlo al módulo del cargador, donde este lo persistirá y generará el registro de la entrada.

El lanzador tomará los archivos que se encuentre en estado cargado, validado o procesado según un tiempo determinado o la prioridad con lo que se deba realizar su ejecución. Los enviará al módulo de validaciones para que este valide cada uno de los registros del archivo según la novedad a la que pertenezca o si el archivo se encuentra estado validado continue con la etapa de procesamiento.

En la etapa de validación, se verificará cada uno de los registros del archivo. Si la respuesta fue exitosa se guardará en una bitácora de registros correctos y si es fallida se guardará en una bitácora de registros de error, por último, se cambiará el estado del archivo a validado e invocará a la etapa de procesamiento.

La etapa de procesamiento se encargará de procesar cada registro según sea la novedad, teniendo en cuenta que solo se procesarán aquellos registros que su validación haya sido exitosa, para que de esta forma las respuestas del procesamiento de cada uno de los registros generen dos (2) archivos CSV, uno donde se guardarán todos los registros que se procesaron con éxito y en el otro almacenará los que no. Dado que son dos archivos estos se comprimirán en un solo y seguidamente se cambiará el estado a procesado, para generar la invocación a la etapa de notificación. Si el archivo ya fue procesado solo invocara la etapa de notificación.

En la etapa de notificación, el archivo generado en la etapa anterior que contiene los archivos de error y exitoso, deberán ser notificados, para esto se definieron tres (3) medios de notificación los cuales son:

1. **Correo:** para cuando el archivo comprimido no supera el tamaño permitido para archivos adjuntos al correo o la propiedad indique que el medio es correo.
2. **Notificación por Validador universal:** para cuando la propiedad lo indique o el tamaño del archivo sobre pasa el permitido, de esta manera se ubicarán en la ruta de la máquina de archivos para que puedan ser tomado desde el Validador universal.
3. **El mixto:** para cuando la propiedad lo indica ya que este medio es la combinación de los dos (2) medios nombrados anteriormente (correo y validado universal), como se presentó en la Figura 1.

4 Construcción del Software

En este capítulo se presentará la implementación de los diferentes componentes que facilitan el proceso de marcación y consulta masiva de las novedades 155 y 032, las cuales han sido ampliamente descritas a lo largo de este documento. Este capítulo se dividirá en dos secciones, la primera donde se indicarán las funcionalidades a desarrollar para dar solución a la práctica teniendo en cuenta la arquitectura presentada por el asesor y la segunda sección donde se presentarán las iteraciones realizadas, sus respectivas actividades junto con la duración real, la duración planeada y la prioridad definida.

4.1 Desarrollo por iteraciones

Siguiendo la metodología usada dentro del SIAFP, la cual cumple con algunas prácticas de SCRUM al igual que con actividades propias de la organización se desarrolló está práctica siguiendo una metodología híbrida que permitió atender cada requerimiento asociado a cada módulo, es importante resaltar que el tiempo destinado para el desarrollo es de 180 horas por iteración, donde 160 horas son de implementación y 20 horas de escritura del documento, en caso que, por ejemplo, existan actividades que no fueron culminadas en su totalidad al cumplir el límite de tiempo, estas se desarrollan antes de iniciar la siguiente etapa.
A continuación, se presentarán los requerimientos requeridos junto a las actividades asociadas.

4.1.1 Requerimiento módulo Cargador

El sistema deberá persistir todos los archivos que fueron enviados a través del portal. Al igual que le corresponderá registrar en las bitácoras la información proporcionada de cada archivo (usuario que envía, novedad, ruta, nombre, entidad, id de la persona).

Tabla 1. Actividades generales del módulo Cargador

Módulo 1. Cargador	
Componente	**Actividades**
Generación de tablas	Creación de tablas para el registro de recepción de archivos enviados a través de portal.
Recepción de archivos	Implementación del *endpoint* encargado de recibir las peticiones enviadas desde el portal.
Persistencia de archivos	Implementación de la persistencia de los archivos en la máquina de archivos.
Registros en bitácoras	Implementación del registro de la información del archivo en la base de datos *MySQL*.
Configuración volumen de persistencia	Implementación de la configuración del volumen de persistencia de los archivos

4.1.2 Requerimiento módulo Lanzador

El sistema deberá dar inicio de manera autónoma al procesamiento de cada uno de los archivos enviados a través del portal, por esta razón se definirán dos modos: El modo priorizado el cual será ejecutado por un operador de la OGS teniendo como parámetros de entrada los identificadores de los archivos que se procesaran y el modo estándar donde se enviara a procesar archivo por archivo según el orden de llegada y el estado, de manera autónoma.

Tabla 2. Actividades generales del módulo Lanzador

Módulo 2. Lanzador	
Componente	**Actividades**
Lanzador Priorizado	Implementación del lanzador para el envío de archivos específicos que deben ser procesados antes debido a su prioridad.
Lanzador estándar	Implementación del lanzador asíncrono para el envío de archivos a procesar según su orden de llegada y su estado (cargado, validado, procesado).

4.1.3 Requerimiento módulo Validador

El sistema deberá validar cada uno de los registros entregados en el archivo, según cada novedad, al encontrar errores, estos registros se guardarán en la bitácora de errores. Los demás registros deberán almacenar en otra bitácora.

Tabla 3. Actividades generales del módulo Validador

Módulo 3. Validador	
Componente	**Actividades**
Bitácora de registros	Implementación del almacenamiento de registros que cumplieron con las validaciones de cada novedad, al igual que los registros que no cumplieron.
	Reinicio de los registros que no fueron completados con completitud.
Validador General	Implementación de la invocación de la validación especifica según la novedad.
	Construcción de respuestas.
	Invocación del servicio de procesamiento.
	Implementación de la lectura archivo.
Abstracción del validador	Implementación clase abstracta de la cual se implementarán los validadores en específicos.
Validador especifico 155	Implementación de validaciones de la línea de los archivos de la novedad 155 (Tamaño de 441).

Validador especifico 032	Implementación de validaciones de la línea de los archivos de la novedad 032 como tipos de documentos válidos, longitud de número de identificación, caracteres especiales.
Exposición servicio	Implementación del *endpoint* y el cliente *rest*.

4.1.4 Requerimiento módulo Procesamiento

El sistema deberá procesar cada uno de los registros que pasaron las validaciones según cada novedad, los cuales se encuentran almacenados en la bitácora de correctos. Después de generadas las respuestas de procesamiento, se tendrá que generar bitácoras por cada novedad para en ellas almacenar la respuesta del procesamiento, posteriormente, se crearan dos archivos: el archivo de registros errores y el archivo de registros exitosos para ser notificados.

Tabla 4. Actividades generales del módulo de procesamiento

Módulo 4. Procesamiento	
Componente	**Actividades**
Procesador general	Implementación de la invocación del procesamiento especifico según la novedad.
	Construcción de respuestas.
	Implementación de la invocación al proceso de Notificador.
Procesamiento Abstracto	Creación de métodos para la lectura de registros a procesar, procesamiento en específico y la lectura de registros ya procesados para ser implementados según cada novedad.
	Implementación de método para generación rutas del archivo.
	Implementación de generación de los archivos CSV de respuesta.
Procesamiento específico 155	Implementación del llamado al servicio de procesamiento en línea.
	Implementación del registro en la bitácora específica de la novedad la respuesta del procesamiento
Procesamiento específico 032	Implementación de las consultas de la información general del afiliado.
	Implementación del registro en la bitácora específica de la novedad la respuesta del procesamiento
Exposición servicio	Implementación del *endpoint* y el cliente rest.

4.1.5 Requerimiento módulo Notificador

El sistema deberá notificar los archivos generados en el módulo de procesamiento, esta notificación se puede generar por tres (3) medios: Uno es a través del Validador

universal, por el cual el usuario podrá tomar los archivos de respuesta, el segundo es por medio del correo electrónico, donde se adjuntan los archivos de respuesta, el ultimo es la combinación de medios anteriormente nombrados. La decisión por cual medio generar la notificación será a través de una propiedad o si el tamaño de los archivos supera el tamaño permitido en él envío de correos.

Tabla 5. Actividades generales módulo Notificador

Módulo 5. Notificador	
Componente	**Actividades**
Notificador Validador	Implementación de la ubicación de los archivos dentro de la ruta del Validador universal para que sean tomados.
Notificador Correo	Implementación del envío del archivo por correo electrónico.
Notificador	Construcción de las respuestas.

4.1.6 Requerimiento componente Framework

El sistema deberá generar la conexión a la base de datos (Oracle), al igual que a las bases de datos de propiedades y negocio (MySQL) para cada uno los perfiles (*Standalone, Wildfly y WebLogic*), igualmente deberá contar con la lectura de propiedades según tres (3) modos: el primero es la lectura de propiedades por medio de archivos punto *properties (.properties)*, el segundo es la lectura de las propiedades a través de la base de datos y el tercero es la combinación de los dos nombrados anteriormente.

Tabla 6. Actividades generales de módulo de Framework

Módulo 1. Framework	
Componente	**Actividades**
Configuración del nuevo esquema	Implementación de los pom.xml para la configuración del nuevo esquema para agregar dependencias entre hijos y padres.
Propiedades (Cargador de propiedades)	Implementación del modo cargador de propiedades por Base de datos.
	Implementación del modo cargador de propiedades por archivo.
	Implementación de modo cargador de propiedades mixto.
Excepciones	Agregar implementación del módulo de Excepciones del repositorio de Maven antiguo a este nuevo esquema.
Base de datos	Implementación del componente MySQL.
	Implementación del componente Oracle.
	Implementación del Manager.

Adicionalmente surgió la necesidad de crear un componente que contiene un conjunto de funcionalidades, las cuales son requeridas para el uso de los requerimientos anteriormente nombrados.

Tabla 7. Actividades generales del módulo Común

Componente: Común	
Componente	**Actividades**
Modelo	Creación modelo de archivo, registro, registro de error, registro de salida.
	Creación de Modelo Bitácoras de notificación de procesamiento y validaciones.
	Creación modelo del estado de archivos.
Persistencia	Implementación actualización del estado archivos.
	Implementación actualización de la información de salida de archivo.
	Implementación de consulta de archivo.

4.2 Iteraciones

Para la elaboración de esta práctica se dividió el desarrollo de cada uno de los requerimientos en actividades, a cada una de ellas se asignó una prioridad (baja, media o alta), una duración prevista y una real dentro del *Sprint*. Estas actividades se crearon basadas según las necesidades de cada módulo.

4.2.1 Primera Iteración

En la primera iteración, las actividades realizadas se basaron en una de las funcionalidades principales (Módulo *framework*), dado que este módulo es usado en todos los componentes desarrollados en esta práctica. Se recibieron diferentes capacitaciones de negocio para el entendimiento del proceso de las novedades, al igual que la explicación de la configuración del ambiente y las tecnologías con las que se implementará la solución de los requerimientos. En la Tabla 8 se evidencian las actividades realizadas para el desarrollo del módulo.

Tabla 8. Primera iteración - Actividades referentes a módulo de Framework.

Framework			
Actividades	**Prioridad**	**Duración Prevista**	**Duración real**
Capacitaciones y configuraciones			
Recibir capacitación sobre la arquitectura que se implementará en el nuevo esquema de Maven.	Media	8	8

Configuración de los pom.xml según la arquitectura dada para todo el esquema nuevo.	Alta	16	20
Excepciones			
Agregar clases del esquema viejo de Maven a al nuevo esquema el módulo Excepciones.	Baja	4	6
Cargador de propiedades			
Implementar métodos para la lectura de propiedades desde la base de datos de propiedades	Media	8	8
Implementar lectura de propiedades a través del archivo ubicado en el servidor *Wildfly.*	Baja	4	4
Implementar lectura de propiedades a través del archivo properties ubicados al mismo nivel de los *jar.*	Baja	4	4
Agregar cache en el cargador de propiedades	baja	1	2
Base de datos			
Investigación de perfilamiento en base de datos.	Alta	10	12
Creación *connection Mananger* genérico.	Alta	20	22
Creación el *connection Manager* de MySQL para el perfil *Wildfly.*	Alta	16	17
Creación el connection Manager de Oracle para el perfil Wildfly.	Alta	16	17
Creación de los archivos de propiedades perfilados.	Media	2	2
Configuración de exclusión de paquetes en el componente según cada perfil (POM.xml).	Alta	10	16

Durante la primera iteración se tomó como referencia el módulo general ya existente del esquema antiguo de *Maven* del proyecto. De igual manera se consideró la arquitectura que el asesor de la práctica proporcionó por medio de una capacitación. Se dio inició a la configuración de los pom.xml para formar lo que se llamó el "esqueleto" del nuevo esquema, igualmente se realizó la implementación del módulo *framework*, el cual está formado por un componente común y otro de integración.

En esta iteración se desarrolló el componente común conformado por el módulo de BD, el cual se encargará de la conexión a las bases de datos del SIAFP (MySQL y Oracle). A diferencia del esquema anterior, existe el perfilamiento en las bases de

datos para buscar que la conexión sea no este determinada por el punto de despliegue del proyecto, por esta razón se crearon 3 perfiles: *Wildfly, Standalone y Weblogic*, sin embargo, para esta práctica solo de implemento el perfil del *Wildfly y Standalone.*

Otro componente creado es el cargador de propiedades desarrollado con tres (3) modos de lectura de propiedades: en primer lugar, la lectura por medio de la base de datos, en segundo lugar, la lectura por archivos punto properties y por último el modo mixto que es la combinación de los dos modos nombrados anteriormente. Adicionalmente, otro componente que se generó en esta misma iteración fue el de excepciones que consiste en el manejo de los errores donde su implementación fue extraer todas las clases del esquema antiguo de *Maven*.

Otra actividad realizada en esta iteración fueron las capacitaciones de negocio para entender los procesos más importantes en el SIAFP como procesamiento de novedades, portal del SIAFP, realizar las configuraciones del ambiente, creación de bases de datos locales y lectura del esquema anterior, entre otras cosas.

4.2.2 Segunda iteración

En la segunda iteración las actividades se basaron en la creación de los módulos cargador y lanzador, para la creación de cada módulo se realizó una capacitación con el asesor de la practica donde se explicaba la arquitectura del arquetipo que se usó para su creación. El módulo cargador se realizó por medio del arquetipo servicio *rest* y el componente lanzador está implementado con un arquetipo *Standalone*. Las clasificación de las actividades realizadas para el componente cargador se presentan en la Tabla 9.

Tabla 9. Segunda iteración - Actividades referente a módulo de Cargador

Cargador			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Generación de diagrama entidad relación para la creación de bitácoras.	Alta	12	12
Generación de script de las tablas para la bitácora en la base de datosMySQL.	Alta	12	12
Generación de proyecto con el arquetipo de servicios.	Alta	4	6
Implementar DAO para registro de la información descriptiva del archivo en la base de datos de MySQL.	Bajo	14	14
Implementación para cifrado de los archivos.	Medio	6	6

Definición del pv, pvc y del yam del servicio para el manejo de volumen de persistencia dentro del contenedor.	Medio	8	10
Implementación de la persistencia del archivo según la estructura de carpetas diseñada.	Baja	6	6
Creación de *properties* para el cargador.	Media	10	10

En el desarrollo del componente lanzador se recibió una capacitación de cómo se genera el componente a partir del arquetipo y una sobre la estructura de carpetas para almacenar los archivos dictada por el CM (Administrador de configuración). Posteriormente, se realizó una investigación acerca del manejo del volumen de persistencia en los contenedores para realizar su implementación junto con el administrador *Cloud* del proyecto. Se elaboraron los *yaml* de configuración para el contenedor donde se desplego el servicio. Adicionalmente, se generó el diagrama entidad-relación para crear el *script* de las bitácoras para almacenar el procesamiento de los archivos, los registros correctos y de error. Finalmente, se establecieron validaciones para la recepción de los archivos enviados desde el portal. Las actividades generadas para el componente lanzador se observan en la Tabla 10

Tabla 10. Segunda iteración: Actividades referente a módulo de Lanzador

Lanzador			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Implementación de los modos del lanzador (priorizado y estándar).	Alta	12	16
Implementación de DAO para la obtención de archivo que debe iniciar el proceso.	Media	4	4
Implementación de la invocación al servicio del validador.	Alta	8	8
Creación de crontab para la ejecución.	Baja	6	6
Creación del *endpoint* del módulo del validador.	Alta	16	20

El componente lanzador es el encargado de iniciar el procesamiento de la novedad, enviara cada uno de los archivos que se encuentre en estado cargado, validado o procesado a la siguiente etapa de validación. En este componente se implementaron dos (2) modos: El modo priorizado y el modo estándar, el priorizado se realizó dado a que se necesite procesar un archivo con mayor prioridad antes de los que están en cola y el modo estándar para enviar a procesar archivos en un

tiempo determinado según su orden de llegada. Vale destacar que, se realizó una asesoría con el asesor para escoger el tiempo adecuado de ejecución del lanzador.

El lanzador es un proyecto tipo *Standalone*, su conexión a la base datos está dada un por un archivo .cfg donde están las credenciales y dirección de la base, por esta razón se debe implementar el perfil *Standalone* en el componente de BD debido a que es usado en el componente. En la Tabla 11 se muestran las actividades asociadas al perfilamiento de standalone.

Tabla 11. Segunda iteración: Actividades referente al perfilamiento standalone en la base de datos

Perfilamiento Base de datos			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Creación del connection Manager de *MySQL* para el perfil Standalone.	Alta	16	16
Creación del connection Manager de *Oracle* para el perfil Standalone.	Media	16	16
Configuración de exclusión de paquetes en el componente según cada perfil (POM.xml) de Standalone.	Alta	4	4
Creación de archivos .cfg para la conexión a la base de datos	Alta	2	2

En la siguiente sección se habla de las actividades que se realizaron para desarrollar el módulo validador y común, se muestra la prioridad de cada una de las actividades, el tiempo que se estimó para realizar y el tiempo real que se empleado para desarrollarla.

4.2.3 Tercera iteración

La tercera iteración se basó en la realización de dos módulos: el común y el validador teniendo en cuenta que el módulo validador debe ser extensible para diferentes novedades, en este caso para la 032 y la 155, asimismo debe reprocesar archivos si por alguna razón alguno quedo incompleto. En el componente común se agregaron funcionalidades para ser usadas en los demás componentes puesto que en todos se usan registros, bitácoras, además funcionalidades como son actualización del estado del archivo y consulta información del archivo, entre otras. Las actividades descritas en la Tabla 12 son del módulo validador.

Tabla 12. Tercera iteración Actividades referente a módulo de Validador genérico.

Validador genérico			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**

Agregar al validador la dependencia del módulo de común.	Baja	2	1
Obtención de la información del archivo.	Alta	4	2
Lectura del archivo (Con cabeceras o sin cabecera).	Alta	5	6
Implementación del cache del validador	Baja	3	3
Implementación de la invocación especifica de cada novedad según la información del archivo.	Alta	18	18
Implementación de inserción a bitácoras de error y de éxito.	Alta	20	24
Implementación de eliminación de los registros que no se terminaron de procesar.	Alta	16	16
Construcción de la respuesta.	Media	8	8
Invocación al procesamiento.	Baja	5	5
Creación del *endpoin*t procesamiento con el arquetipo de servicios.	Media	10	10

Para el desarrollo del componente común se realizó un análisis de todas las operaciones que ejecutaría cualquier archivo sin importar la novedad, teniendo en cuenta que todos los archivos tendrán información en común como es: usuario que realizó el envío del archivo, correo del usuario, ruta de persistencia del archivo de salida entre otros datos. Por esta razón, se decide elaborar una implementación del proceso general que realizará cualquier archivo, con la salvedad que su validación de los registros serán diferentes según la novedad. Con esto cada novedad extiende de este proceso general y ejecutará la implementación de su validación según sea el proceso. A continuación, en la Tabla 13 se presentan todas las actividades realizadas para la elaboración del componente común.

Tabla 13. Tercera iteración: Actividades referente a módulo de Común.

Común			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Generación modelos para el módulo de común (Archivo, Registro, Registro de error, Registro salida) y Bitácoras (Ejecución, Etapa, notificación, procesamiento y validación)	Alta	10	10

Implementación de DAO para consultar la información del archivo, Actualizar el estado, Actualizar información salida del archivo.	Alta	24	24
Implementación clase de estados de los archivos.	Baja	3	3

El componente común se creó porque se evidenció que se realizaban implementaciones de operaciones múltiples veces en diferentes partes del desarrollo y en diversos componentes. De esto modo, solamente se agregará la dependencia del módulo al componente que lo requiera para obtener funciones como: actualización de estado del archivo, consultar información de archivo y actualización de información de salida, igual que modelos como lo son Archivo, Bitácoras (Ejecución, Etapa, Notificación, Procesamiento, Validación), Registros (error y salida).

Cada novedad tiene un proceso de validación de registros diferente, por esta razón se deben generar una implementación por cada novedad para que se cumpla con la especificación de cada una de ellas, las actividades que se realizaron para implementar las validaciones de la novedad 032 se exponen la siguiente Tabla 14.

Tabla 14. Tercera iteración: Actividades referente a módulo de Validador (Nov032)

Validador Novedad 032			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Implementación método de la validación de la novedad 032.	Alta	8	12
Validación del tamaño de línea, tipo de documento, número de documento y caracteres especiales.	Media	6	6
Implementación de modelo del Registro032, que extiende de modelo de registro de módulo común.	Media	3	2

La implementación del validador de la novedad 032 es una extensión del validador genérico donde se realizaron las siguientes validaciones: primero, evaluar que la cantidad de campos no exceda el valor máximo permitido ni el valor mínimo para cada registro. Segundo, no admitir caracteres especiales. Tercero, validar que el tipo y número de documento sea el permitido en el SIAFP. Adicionalmente, se usaron métodos del validador genérico, particularmente la construcción de las respuestas, guardar en las bitácoras los registros de éxito y errores. A continuación, se describirán las actividades generadas para elaborar las validaciones de los registros de la novedad 155 como se muestra en la Tabla 15.

Tabla 15. Tercera iteración Actividades referente a módulo de Validador (Nov155)

Validador 155			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Implementación de la validación de la novedad 155.	Alta	8	10
Validación del tamaño de línea.	Media	3	3
Implementación de modelo del Registro155, que extiende de modelo de registro de módulo común.	Media	4	4

En cuanto a la validación especifica de la novedad 155, fue necesario condicionar que la longitud de la trama cumpliera según lo definido en la novedad. Además, se usaron los métodos del validador genérico.

4.2.4 Cuarta iteración

En la cuarta iteración se desarrolló el módulo de procesamiento, este al igual que el validador debe ser extensible e independiente de la novedad que deba procesar. Por esta razón, se dividió de la siguiente manera: un procesamiento genérico diseñado para gestionar las operaciones del archivo independiente a su novedad y un procesamiento específico para realizar las particularidades del proceso de cada novedad, en este caso la novedad 032 y 155. Las actividades realizadas en el procesamiento genérico se describieron en la Tabla 16 como se muestra a continuación.

Tabla 16. Cuarta iteración: Actividades referente a módulo procesamiento Genérico.

Procesamiento genérico			
Actividad	**Prioridad**	**Duración Prevista**	**Duración real**
Agregar al procesamiento la dependencia del módulo de común.	Baja	2	1
Implementación cache procesador.	Baja	2	1
Obtención de la información del archivo.	Alta	4	2
Implementación de la invocación del procesamiento específico de cada novedad.	Alta	10	12

Construcción de la respuesta del procesamiento.	Media	8	7
Invocación al módulo de Notificador.	Baja	2	2
Implementación de generación de rutas para la ubicación de los archivos procesados y los archivos de error dentro de la máquina de archivos (Índice, extensión, ubicación).	Alta	8	10
Generación de comprimido de los archivos de error y de procesamiento exitoso	Alta	12	12
Implementación de lectura de registros a procesar.	Alta	3	4
Implementación de lectura de registros de error.	Alta	5	3

El procesamiento genérico es la implementación de cada una de las etapas por las que el archivo atravesará para generar los archivos de respuesta. Por ejemplo, la generación de bitácoras de procesamiento, generación de las rutas de salida de los archivos, la construcción de archivos de error y de respuesta del procesamiento.

Adicionalmente, se tienen abstracciones que son implementadas según cada novedad, específicamente la lectura de registros válidos, el procesamiento especifico de los registros y la lectura de las respuestas obtenidas durante el proceso. Las actividades realizadas para generar la implementación de la novedad 032 son descritas a continuación en la Tabla 17.

Tabla 17. Cuarta iteración: Actividades referente a módulo procesamiento Novedad 032.

Procesamiento Novedad 032			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Creación de tablas de registros procesados para la novedad 032.	Media	2	3
Leer registros a procesar (Extensión delprocesamiento).	Alta	4	4
Creación de los modelos de la novedad 032.	Alta	3	3
Implementación de la obtención de la información del del afiliado (consulta a la base	Alta	8	12

de datos del SIAFP).			
Procesamiento de la información obtenida en la consulta de la información del afiliado.	Alta	10	6
Agregar el registro de respuesta del procesamiento en la tabla creada.	Media	4	4
Invocación del método de actualización del estado del archivo ubicado en el módulo de común.	Baja	1	1

Para la implementación del procesamiento de la novedad 032 se debió extender del procesamiento genérico, con la finalidad de realizar el método específico de la novedad 032 y obtener todos métodos para procesar un archivo independiente de la novedad, de esta manera en el procesamiento de la novedad 032 se obtendrá toda la información del afiliado que se encuentra en la base de datos de Oracle del SIAFP para guardarla en una bitácora. Como en esta práctica se generó el procesamiento para dos (2) novedades en la Tabla 18 se detallan las actividades que se realizaron para la implementación de la novedad 155.

Tabla 18. Cuarta iteración: Actividades referente a módulo procesamiento novedad 155.

Procesamiento Novedad 155			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Creación de tablas de los registros procesadospara la 155.	Media	4	4
Leer los registros a procesar (Extensión del procesamiento).	Media	8	9
Creación de modelos de la novedad 155.	Alta	3	3
Creación de cliente SOAP de procesamiento enlínea.	Alta	15	17
Invocación de procesamiento en línea por medio del cliente SOAP.	Alta	4	4
Implementación de un ciclo para obtener la respuesta de procesamiento en línea.	Media	8	12
Guardar errores generados en procesamiento en línea.	Media	8	8
Obtener estado general del afiliado.	Alta	15	17

Actualizar la tabla de registros de llamados a procesamiento en línea.	Media	4	4
Agregar el registro en la tabla creada.	Media	4	4
Invocación de actualizar estado del módulo común.	Baja	1	1

Para la implementación del procesamiento de la novedad 155 se debió crear un cliente *SOAP* para consumir el servicio de procesamiento en línea, que se encargará de procesar la novedad regresando la trama de respuesta, un token y los errores si se generaron al momento de ser procesado el registro. Seguidamente se consultó el estado general del afiliado que también hace parte de la respuesta de la novedad a través de la base de datos de los LOGS.

4.2.5 Quinta iteración

En la quinta iteración se creó el módulo de notificación, el cual es el encargado de enviar las respuestas del procesamiento de las novedades. Las respuestas se entregarán por medio de un archivo comprimido que se generó en el módulo de procesamiento, donde estarán dos archivos .CSV uno con las respuestas del procesamiento y otro archivo con las de error. Posteriormente, la notificación puede estar dada por tres (3) modos: a través del correo electrónico al usuario que realizó la subida del archivo, por medio del Validador universal, donde el usuario podrá tomar el archivo comprimido o de forma mixta que significa que la notificación se realizará por validador y por correo electrónico. La descripción de las actividades realizadas en esta iteración están explicadas en la Tabla *19*.

Tabla 19. Quinta iteración: Actividades referente al módulo notificador.

Notificador			
Actividad	**Prioridad**	**Duración prevista**	**Duración real**
Inspección del código para generar la ubicación de los archivos en Validador universal.	Alta	16	16
Implementar la ubicación de los archivos, proceso encriptación de los archivos en el Validador universal.	Alta	8	8

Agregar al notificador la dependencia del módulo de común.	Baja	2	1
Obtención de la información del archivo.	Alta	4	2
Cargar propiedad de notificación (Mixto, Cargador, correo) según cada novedad.	Alta	8	6
Implementación condicional según la propiedad.	Media	4	3
Implementación del condicional según el tamaño del archivo.	Media	4	2
Creación de los métodos para él envió del correo (Mensaje base, crear sesión, crear texto).	Media	20	25
Creación de las plantillas para el correo.	media	8	6
Agregar archivos al correo.	Media	4	4
Inspección del código para generar la ubicación de los archivos en Validador universal.	Alta	16	4
Creación rutas de ubicación del archivo en el Validador universal.	Alta	8	5
Encriptar archivo para el validador.	Baja	4	4
Agregar registros a las bitácoras del Validador universal.	Alta	8	8
definición de las propiedades para la notificación por correo y por validador.	Alta	8	8
Construcción de respuesta.	Baja	2	2
Creación de script de propiedades.	Media	4	4
Actualizar estado ha notificado.	Media	4	4

Teniendo en cuenta la información proporcionada anteriormente acerca de la duración de cada una de las iteraciones dentro de cada sprint (160 horas), fue realizado un breve análisis con el fin de identificar que tan adecuada fue la estimación con respecgto al tiempo real. En la tabla 20 se describe la información

con los tiempo estimados (en horas) y los que realmente conllevo cada iteración, al igual que el número de horas de desfases (horas positivas) u horas donde la iteración se realizó en menos tiempo de lo estimado(horas negativas), junto con su porcentaje de desfase.

Tabla 20.Tiempos reales y estimados por iteración

Iteración	Tiempo Estimado	Tiempo Real	Horas	Porcentaje de desfase
Primera	119	138	19	11,8%
Segunda	156	162	6	3,75%
Tercera	160	167	7	4,37%
Cuarta	160	168	8	5%
Quinta	132	112	- 20	0%

Se evidenció que en la primera iteración el porcentaje de desfase es más alto en comparación a las otras iteraciones, esto se debe a que se generó un nuevo esquema y las actividades del componente base de datos no estuvieron estimadas de manera adecuada, por lo tanto, su implementación tomo más tiempo del estimado. Así mismo se puede observar que a partir de la segunda iteración la diferencia es menor que la primera y fue dada por la complejidad de los componentes. Por último, en la quinta iteración no se generaron retrasos dado que es un componente pequeño y gran parte de su implementación es tomada del esquema antiguo del SIAFP, permitiendo compensar parte de la diferencia acumulada que se traía hasta la cuarta iteración.

En la siguiente sección, se realiza la explicación sobre cómo fueron implementados cada uno de los componentes de esta práctica, teniendo en cuenta que hay componentes transversales que son usados en todos los módulos e implementaciones especificadas de cada novedad.

4.3 Desarrollo de las funcionalidades

Esta sección presenta el proceso realizado para la implementación de las funcionalidades requeridas para el procesamiento de las novedades 155 y 032 por el canal de autoservicio. Está dividido en 5 módulos, los cuales describen la solución propuesta y la implementación de cada uno de ellos.

4.3.1 Módulo Framework

El módulo del *framework* es uno de los más importantes en el desarrollo, dado que será usado por los demás componentes, este contendrá funcionalidades como: las conexiones a las bases de datos de *Oracle* y *MySQL*, obtención de todas las propiedades de los componentes, algoritmos generales y propios del SIAFP usados en diferentes procesos, por último, implementación de clientes tanto propios como de servicios externos.

La representación de este módulo se realizó a través de un diagrama de componentes como se muestra en la Figura 2, donde se tienen dos componentes grandes que son: Común e Integración. El componente común está conformado por cuatro diferentes subcomponentes:

- **BD:** contiene toda la implementación de las conexiones a la base de datos.
- **Propiedades:** sirve para la lectura de la información parametrizada de cada componente.
- **Utilidades:** comprende algunos algoritmos generales y otros propios del SIAFP para ser usado por otros componentes
- **Excepciones:** usado para el manejo de errores.

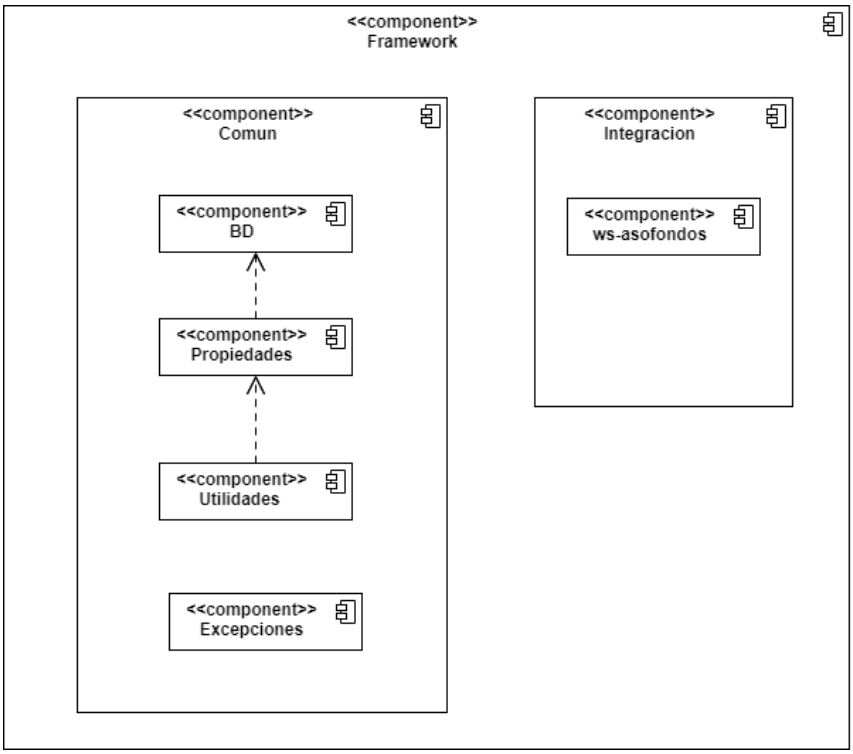


Figura 2. Diagrama de componentes módulo Framework.

En el componente integración, se ubicarán todos los clientes externos como internos del proyecto. Actualmente se encuentra ws-asofondos que es un cliente SOAP del servicio de procesamiento en línea.

4.3.1.1 Subcomponente Base de datos (BD)

El subcomponente base de datos es el encargado del manejo de todas las conexiones a las bases de datos de MySQL y *Oracle*, se implementó para ser usado por otros componentes que requieran realizar consultas o modificaciones a cualquiera de ellas. La gestión de las conexiones es determinada por contexto de ejecución que utilice el componente externo, por ejemplo, si el componente es un servicio, la conexión a la base de datos se establece mediante el pool de conexiones de los data source definidos en Wildfly. Por otro lado, si es un componente tipo

Standalone, el contexto de ejecución será en la máquina virtual de Java (JRE). La conexión se realizará por medio de la lectura de un archivo .cfg que cuenta con las propiedades de conexión a la base de datos para implementar con ellas la configuración de los del pool de conexiones a través de la librería Agroal de Java, sin embargo, esto debe ser transparente para los componentes que interactúen con la base de datos. Debido a esto se desarrolló dentro del componente la implementación del perfilamiento de las bases de datos como solución a las diferencias que existen al momento de conectarse, además para solucionar las inconsistencias que se presentan al momento de generar los puntos de montaje para los componentes. En la Figura 3 se indica la solución generada para este componente.

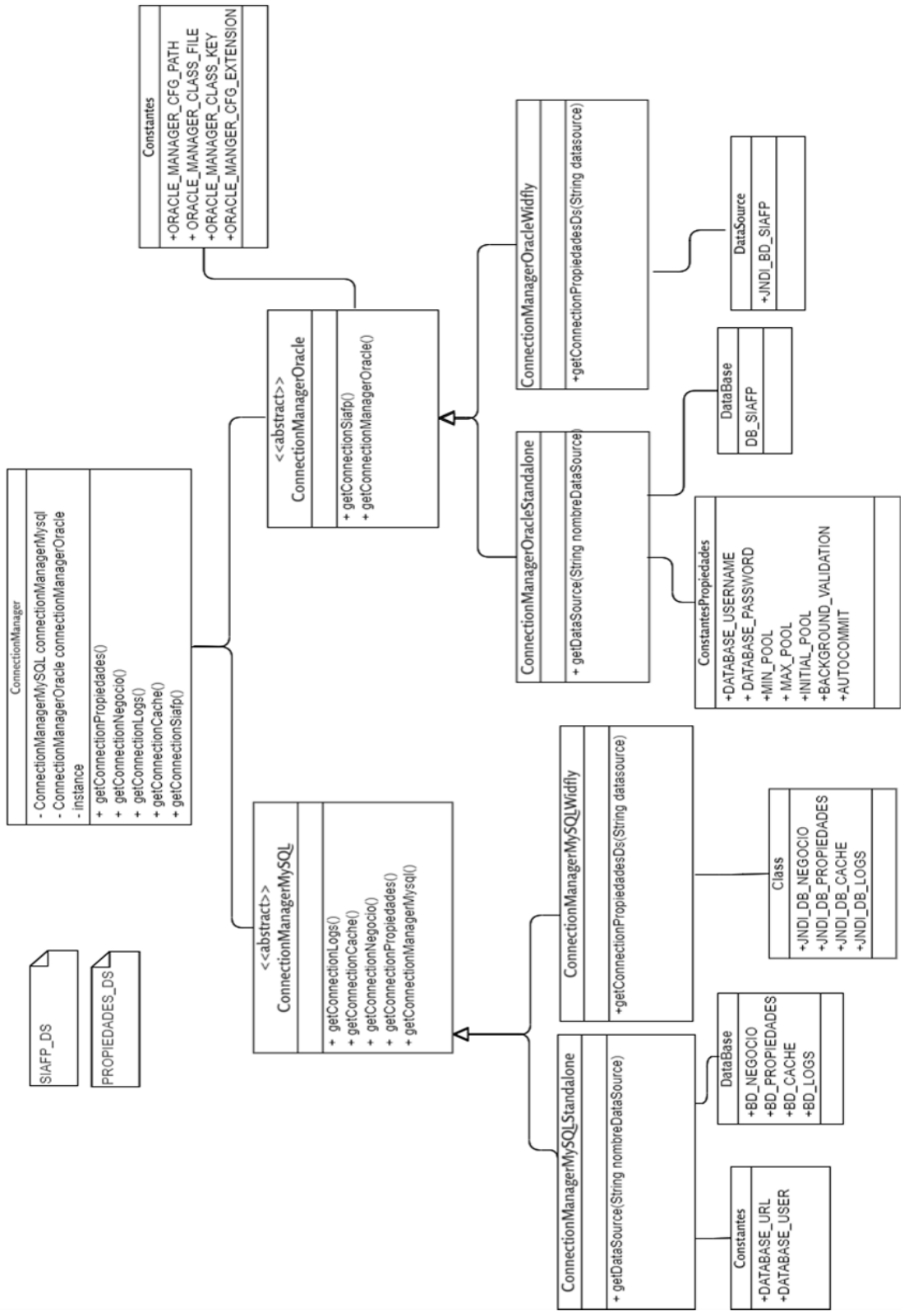


Figura 3. Diagrama de clases subcomponente BD.

Cabe resaltar que el perfilamiento debe ser aplicado a todo el esquema, es decir, cada uno de los componentes quedan con el perfil con el que se haya compilado. Por esta razón, si el componente usado es un Standalone, los componentes de tipo servicios del esquema también tendrán el perfil standalone, debido a esto se debe generar la conexión a la base de datos de los componentes de tipo servicio compilando nuevamente todo el esquema ahora con el perfil wildfly.

La implementación del perfilamiento consta en generar una exclusión de paquetes en el momento de la compilación según el perfil. Los perfiles creados fueron Wildfly, *Standalone*, se dejó indicado el de *Weblogic* debido a que en esta práctica no estuvo planeado su desarrollo. La exclusión se realizó a través de los *pom.xml* que se encuentran señalados en la Figura 4, estos se generaron dentro de los paquetes de cada base de datos (*Oracle* y MySQL), para que en ellos se decida que paquetes se deben excluir.

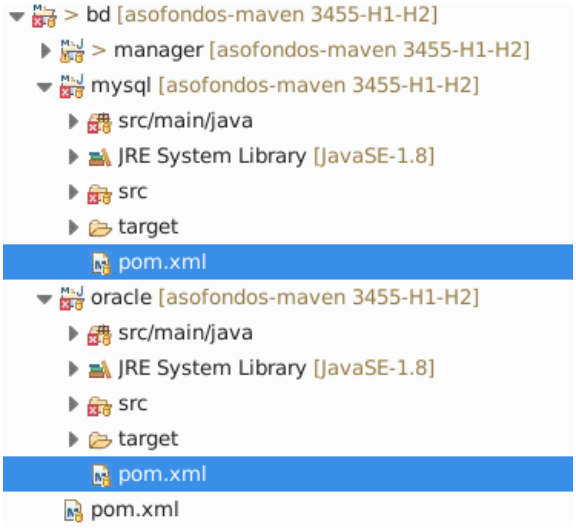


Figura 4. Pom donde se implementó la exclusión de paquetes.

Por medio del tag <*exclude*> como se muestra en la figura 5, se realiza la exclusión de paquetes según el perfil de compilacion escogido a traves del parametro enviado en el comando de compilacion. Por ejemplo, si se quiere usar el perfil de *Standalone* el comando de compilacion es el siguiente “mvn clean install -pStandalone” por ende los paquetes que se excluiran son los del perfil Wildfly y *Standalone.*

Figura 5. Implementación exclusión paquetes en archivo pom.

La comunicación de los componentes externos se hará por medio de la clase *ConnectionManager,* en esta clase se tendrá la implementación del patrón *Singleton,* dado que se necesita una única instancia para el acceso a la base de datos. Además, se tienen atributos de tipo *ConnectionManagerMySQL* y *ConnectionManagerOracle* que son clases abstractas en las que se tiene el método (*getConnectionManagerOracle* y *getConnectionManagerMysql*) para la invocación de las clases *manager* específica (*ConnectionManagerMySQLStandalone*, *ConnectionManagerMySQLWildfly, ConnectionManagerOracleStandalone, ConnectionManagerOracleWildfly*) como se puede ver en la Figura 3.

Para la invocación de las clases *manager* específicas se realizó por medio de archivos de *properties* cargados al generar la compilación del proyecto, a través del tag <directory> señalado en el recuadro rojo de la Figura 7, donde la variable env toma valor según el perfil indicado. Existen dos archivos de propiedades según cada perfil, uno para la base de datos de O*racle* y otro para la base de datos de MySQL donde la propiedad indicará la clase manager especifica según el perfil. Siguiendo el ejemplo anterior, las clases *manager* para el perfil *Standalone* serian: *ConnectionManagerMySQLStandalone* y *ConnectionManagerOracleStandalone* teniendo en cuenta que la exclusion se realiza para las dos bases de datos.

```
<build>
    <resources>
        <resource>
            <directory>src/main/resources/${env}</directory>
            <includes>
                <include>*.properties</include>
            </includes>
        </resource>
    </resources>
</build>
```

Figura 6. Implementación ruta properties.

Para obtener la instancia de una clase *manager* específica, primero se debe obtener el archivo de *properties*, después se obtiene la propiedad, por último, se crea la instancia de la clase, como se muestra en el pseudocodigo de la Figura 7. En cada una de las clases manager especificas se realizó la implementación de la obtención de la conexión a la base de datos.

```
Función getConnectionManagerMysql():
Intentar:
 propiedades = nuevo Properties()
 propiedades.cargarPropiedades(nombreArchivoPropiedades)
 ConnectionManagerMySQL claseManagerMySQL = instanciarClase(propiedades.obtenerPropiedad(claseManegerSegunPerfil))
 Devolver claseManagerMySQL
Capturar Excepciones (Excepción):
 Lanzar Excepción
```

Figura 7. Implementación ruta properties.

Para el perfil de *Standalone* se definió una lista “clave valor”, donde la clave está determinada por el nombre del *data source* y el valor por un objeto de la clase DataSource de Java. Esta lista se llena a medida que se realiza la obtención de la configuración del *pool* de conexiones, así, solamente retornan si se encuentra en la lista, en cambio si no encuentra se genera su configuración.

Para la configuración del *pool* de conexiones del *data source* se realizó a través del método *getDataSource* en el que se obtendrán las propiedades definidas en el archivo .cfg, el cual cuenta con las siguientes propiedades: *url* de la base de datos, credenciales de acceso (usuario y contraseña) y los parámetros de configuración del pool de conexiones. Posteriormente, se debió generar la configuración *pool* de conexiones a través de la librería Agroal. De esta manera, cada uno de los métodos que obtienen la conexión a la base de datos específica solo tendrán que invocar a el método *getDataSource* para obtener el pool de conexiones del *data source* y generar la conexión a la base de datos, así como se indica en la Figura *8*.

```
Funcion getConnectionSiafp retorna Connection
    Inicio
        Intentar
            Retornar getDataSource(Databases.DB_SIAFP).getConnection()
        Capturar excepcion
            Manejar la excepción de nombramiento
    Fin
```

Figura 8. Implementación obtención de la conexión cuando se tiene el data source.

En el perfil Wildfly el servidor tiene acceso a los recursos como son las bases de datos, a través del JNDI (*Java Naming and Directory Interface*) que es un servicio que permite nombrar algún objeto del servidor para después ser referenciado desde el código [40] de esta manera se traen las propiedades de configuración para regresar el pool de conexión la conexión *del data source* para posteriormente generar la conexión a la base de datos.

En la siguiente sección se indicará el desarrollo realizado en la elaboración del componente propiedades, el cual es un componente que hace parte del módulo de framework y será usado para la lectura de propiedades de los proyectos del SIAFP.

4.3.1.2 Subcomponente Propiedades

Teniendo en cuenta que las propiedades son elementos de configuración que podrán ser ajustados, de tal forma que alterarán el comportamiento de un software sin modificar su código fuente. Por esta razón se creó este componente, para obtener todas aquellas propiedades que los componentes externos necesiten. Las propiedades en el SIAFP están descritas de dos formas: la primera es a través de archivos *properties* formados por claves y valores, la segunda son filas en la tabla propiedades de la base de datos de MySQL, esta tiene las siguientes columnas: categoría para guardar el nombre del properties, llave para almacenar el nombre de la propiedad y por último la columna valor para registrar los valores de cada propiedad.

Este subcomponente de framework tiene tres (3) modos, el primero es la lectura de las propiedades a través de la base de datos (BD), el segundo es la lectura por medio de archivos *properties (Archivos)* y el último es la combinación de los modos anteriormente nombrados (Mixto), a partir de esto se generó la siguiente solución presentada en la Figura 9.

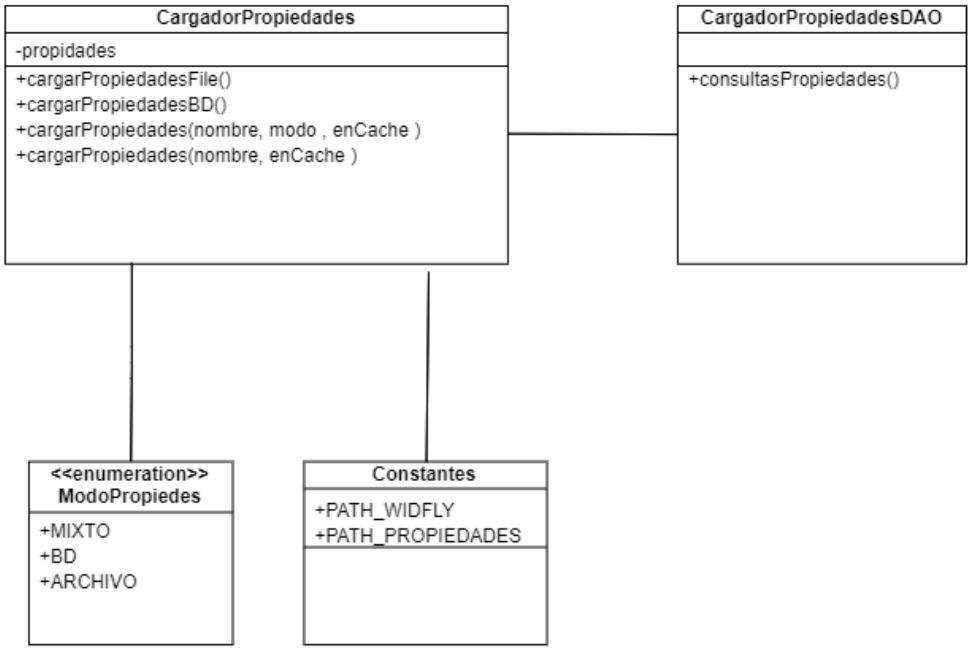


Figura 9. Diagrama de clases subcomponente Propiedades.

En el subcomponente propiedades se definió la clase CargadorPropiedades, que es el medio por el cual los componentes externos realizan la invocación para obtener las propiedades. Esto se genera a través de los métodos cargarPropiedades definidos mediante una sobrecarga de métodos. En el primer método se podrá escoger el modo por cual se obtendrán las propiedades por medio de los parámetros, en el segundo las propiedades siempre se obtendrán por el modo mixto. Ambos métodos reciben el nombre del properties y un booleano para saber si las propiedades se pueden extraer del cache o se deben volver a consultar.

La implementación del caché se generó por medio una lista “clave valor”, donde la clave es el nombre del archivo *properties* y el valor es un objeto de la clase Properties de Java. A medida que se realiza la invocación se agregarán los properties a esta lista, cabe resaltar que el objeto de la clase Properties de java tendrá todas las propiedades del properties consultado.

Ahora bien, según el modo se generó el llamado al método para obtener las propiedades, si el modo es Archivo se invocó al método cargarPropiedadesFile. Este método es el encargado de ejecutar la invocación de las propiedades ubicadas en los archivos *properties*, para la implementación de este modo se debió tener en cuenta la ubicación del archivo, puesto que, puede estar ubicado en la carpeta de propiedades del servidor *Wildlfy* o al mismo nivel del jar de ejecución, después de esto realizará el cargado de todas las propiedades del archivo.

Para el modo Base de datos BD se invocará al método cargarPropiedadesBD, este en su implementación creará una instancia de la clase CargadorPropiedadesDAO, con esta instancia se llamará al método consultarPropiedades donde este recibe

por parámetro el nombre del *properties,* dentro de la base de datos está representado con la columna categoría. La implementación de este método es una consulta a la base de datos de propiedades de MySQL obteniendo todas las propiedades del properties y regresando un objeto de la clase Properties de Java. En el modo es mixto, obtendrán primero las propiedades por el método cargarPropiedadesFile y si este no regresa información consulta por el método cargarPropiedadesBD.

Cuando un componente externo invoca al módulo de propiedades e indica la activación del cache, se realiza una búsqueda dentro de la lista para regresar el properties indicado si ha sido previamente consultado, por otra parte, si el properties no ha sido consultado se realizará la obtención de las propiedades según el modo indicado.

4.3.1.3 Subcomponente Utilidades

En este componente se realizaron las implementaciones de algoritmos de comparación de nombres, este fue extraído de un código de PL/SQL del proceso antiguo de la novedad 032 y se desarrolló en código Java, para ser usado en el procesamiento de la novedad 032. Otro algoritmo implementado fue el de comprimir archivos, este se usó en la novedad 155 para la ubicación de los archivos de procesamiento dentro del Validador universal.

4.3.1.4 Subcomponente Integración

Este componente tiene como finalidad alojar las implementaciones de las invocaciones a los servicios externos o internos (Entidad externas o SIAFP) a través de clientes (*rest* o SOAP). En esta práctica se realizó el consumo de un servicio interno del SIAFP el cual fue procesamiento en línea no seguridad, que se encarga de “establecer una vía de comunicación en línea para el envío y recepción de novedades entre las entidades y el SIAFP. Por medio del Web Service se hará la validación de la estructura de la novedad y su posterior procesamiento” [41].

Para realizar el consumo se creó un cliente SOAP con el entorno de desarrollo integrado (IDE), NetBeans, a partir del WSLD del servicio, dado que NetBeans genera todas las clases necesarias para el consumo del servicio como lo son clases *Request*, *Response* y la interfaz de definición de la invocación del servicio con cada uno de los métodos. Este cliente es usado en el componente de Procesamiento, en la implementación de la novedad 155 para realizar el procesamiento de la novedad para cada uno de los registros correctos del archivo, este cliente será el intermediario entre el servicio de procesamiento en línea y procesamiento del nuevo esquema, esta relación entre componentes como se muestra en la Figura 10.

Figura 10. Relación de componentes para invocación al servicio de procesamiento línea.

En la siguiente sección se describe la arquitectura de los arquetipos de Servicio de tipo *rest,* dado que a partir de este se generar la construcción los componentes de este tipo implementados en la práctica.

4.3.2 Arquitectura del arquetipo de Servicios

En esta práctica fueron utilizados los arquetipos del SIAFP, principalmente el arquetipo Servicio rest, con este se crearon los servicios de cargador, validador, procesamiento y notificador, este arquetipo se compone de los siguientes componentes:

4.3.2.1 Cliente-rs

El componente Cliente-rs es un cliente r*est* por el cual los procesos externos podrán consumir el servicio, por esta razón todo componente externo que requiera hacer uso del servicio lo hará por medio de este cliente, en el se agrega la dependencia *resteasy-client*, la cual ayuda en el consumo de servicios *rest* del lado del cliente, así mismo se agrega la dependencia al componente Interface-ws para que el cliente use la misma API del servicio (Interfaz de Programación de Aplicaciones).

4.3.2.2 Interface-ws

El componente Interface-ws contendrá la API del servicio que describe como se accede a las funcionalidades, por esta razón este componente tendrá una interfaz que cuenta con las declaraciones de las *urls* y los métodos HTTP, adicionalmente en este componente se crearán los *requests* y *response* necesarios para el consumo. Esta descripción se realiza mediante la dependencia de JAX-RS, que es un API que facilita la creación de los servicios *Rest*.

4.3.2.3 Utilidades

El componente utilidades guardará clases propias de los servicios, como lo son constantes, procedimientos y funciones, adicionalmente en el componente utilidades se agregarán las dependencias a: clientes externos o internos, módulo propiedades y si es necesario dependencias a componentes de otros proyectos, tiene una dependencia con el componente *Interface-ws* como se muestra en la Figura 11.

4.3.2.4 Modelo

El componente Modelo tendrá todas las definición de los modelos de negocio propios del servicio, al igual que una dependencia con el componente Utilidades.

4.3.2.5 Persistencia

El componente de Persistencia está asociado a las bases de datos por ende tendrá dependencia al componente BD del módulo Framework, realizará consultas o

modificaciones a las bases de datos a través de SQL nativo, adicionalmente el componente de persistencia tiene una relación con el componente Modelo (4.3.2.4)

4.3.2.6 Negocio

El componente negocio se encargará de toda la definición de la lógica del servicio, este se comunica con el componente de Persistencia.

4.3.2.7 WS

En el componente WS se generará la publicación del servicio, en las clases de este componente se implementan las interfaces creadas en el módulo de Interface-ws, para en ellas generar la invocación a algún método de las clases del componente Negocio que iniciara todo el proceso.

En la Figura 11 se muestra la comunicación entre componentes, recordando que Maven es una herramienta para la construcción y la gestión de proyecto, tiene la capacidad de generar diferentes tipos de ejecutables según la necesidad, en el caso del SIAFP al compilar un servicio se genera un ejecutable .ear para ser desplegado en los servidores *Wildfly.*

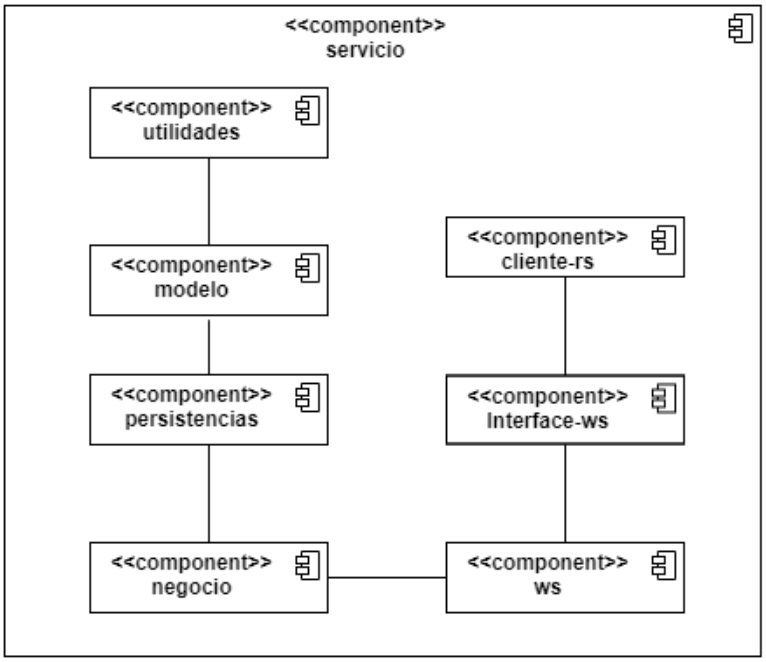


Figura 11. Diagrama de componentes Arquetipo servicio rest

Se decidió desarrollar los módulos de cargador, validador, procesamiento y notificador a través del arquetipo de servicios, porque suplen con el requisito no funcional de la concurrencia, dado que su ejecución se realiza en la capa de servidor de aplicaciones y esta capa se puede manejar múltiples solicitudes al mismo tiempo de diferentes clientes.

Para todos los servicios se debe tener en cuenta la configuración del contenedor donde se aloja la aplicación, debido a que son desechables y cada vez que se reinicie se perderá la configuración establecida, por esta razón se debe generar la configuración para el volumen de persistencia así los datos persistirán independiente del estado del contenedor, en el SIAFP se implementa esta configuración para la obtención de archivos properties, creación, eliminación, lectura y escritura de archivos. Esta configuración se realiza a través de los siguientes *yaml:*

- NF: para la publicación de las rutas de la carpeta compartida.
- PV: para la conexión a la maquina donde está ubicada la carpeta compartida.
- PVC: para solicitar el recurso de almacenamiento.
- *Yaml* servicio: para referenciar al PVC indicando que se usara el volumen de persistencia configurado.

La relación entre cada uno de los *yamls* para la configuración del volumen de persistencia de un contenedor se muestra en la Figura 12.

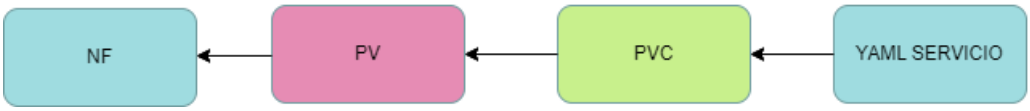


Figura 12. Relación de Yamls configuración Volumen de Persistencia.

En la siguiente sección se mostrará cómo se guarda la información de: las etapas de procesamiento del archivo, información de envió y ubicación de los archivos de entrada y de respuesta, modo de notificación, como también el almacenamiento de los registros de error y de éxito que se generen de la etapa de validación.

4.3.3 Diagrama Entidad-relación

En el diseño del diagrama entidad-relación se consideraron aspectos como: información para realizar la notificación del archivo de salida, estado del proceso para el reinicio de la etapa por presencia de algún fallo, ubicación de los archivos de entrada y de salida, bitácoras para el almacenamiento de registros exitosos y de error generados en la etapa de validación. Todo esto se representó a nivel de datos en el siguiente diagrama entidad-relación como se muestra en la Figura 13, donde cada una de las tablas ayuda a suplir cada uno de los aspectos mencionados anteriormente.

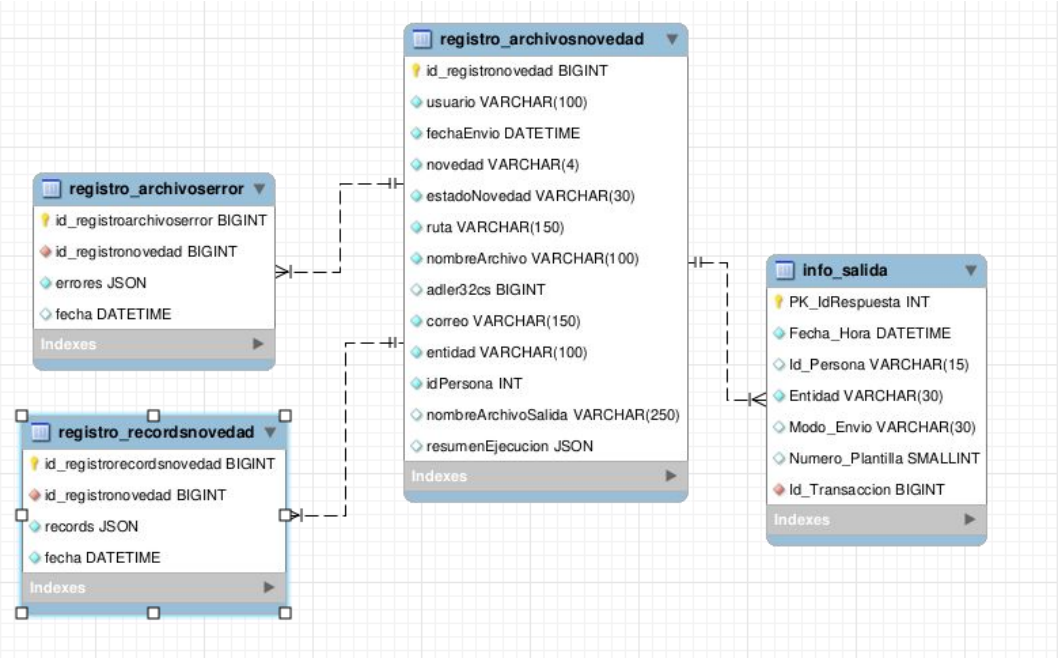


Figura 13. Diagrama Entidad Relación.

La tabla registro_archivosnovedad guarda la información del archivo de entrada junto con la información del usuario y la ruta del archivo de salida. Las equivalencia entre la información y las columnas de la tabla registro_archivosnovedad, se puede observar en la Tabla 21.

Tabla 21.Equivalencias entre la información a las columna registro_archivosnovedad

Información para guardar	Columna
Id del Archivo	Id_registronovedad
Usuario que envía	usuario
Correo del usuario que envía	correo
Entidad del usuario que envía	entidad
Número de identificación del usuario que envía.	id_persona
Estado del archivo	estado
Novedad por la que se procesara el archivo(155 o 032)	novedad
Nombre Archivo entrada	nombreArchivo
Nombre Archivo salida	nombreArchivoSalida
Ruta del archivo de entrada	ruta

El campo resumenEjecucion almacena una bitácora de cada una de las etapa por la que paso el archivo, para la etapa de validación se guarda el número de registros procesados, exitosos y de error. En la etapa de procesamiento se registra si se logró generar los archivos de respuesta o no, al igual que el número de registros procesados. Por último, la etapa de notificación guarda el modo con el que realizo la notificación. En cada una de las etapa se registra el tiempo de ejecución.

En la tabla registro_archivoserror se guarda la información de: señalado en el recuadro rojo de la Figura 14 la trama de error, señalado en el recuadro azul de la Figura 14 el índice de donde se encontró la trama dentro del archivo, adicionalmente se guarda una lista con las descripciones de los errores encontrados para esa trama, esto está representado a través de la columna "errores" de tipo *json*. Así mismo, se guarda el identificador del archivo por medio de la columna id_registroNovedad, cabe resaltar que por cada trama de error encontrada en el archivo se crea un nuevo registro en la tabla, además a partir de esta se genera el archivo de error que hace parte del archivo de respuesta del procesamiento.

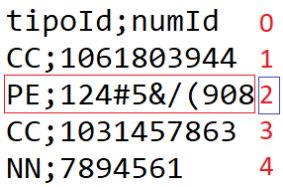


Figura 14. Trama de error e índice en el archivo

La tabla registro_recordsnovedad almacena todas aquellas tramas que no generaron errores en la etapa de validación en la columna record, en esta columna se almacena cada uno de los campos de la trama, su índice en el archivo y la novedad con el cual se procesará, esta columna es de tipo json. En la columna id_registroNovedad se guarda el identificador del archivo. A partir de esta tabla el componente de Procesamiento obtiene los registros (records) según el archivo, para así iniciar el procesamiento del registro según la novedad.

Por último, la tabla "info_salida" guarda la información del medio por el cual fue notificado el archivo (Validador universal o Correo) también la hora y fecha que se generó la notificación.

4.3.4 Componente Cargador

El componente Cargador es un servicio *rest* responsable de la recepción de los archivos, almacenarlos en la máquina de archivos y de generar su registro en la tabla registro_archivosnovedad. El podrá realizar estas actividades, si y solo si se recibe el archivo enviado a través del portal con sus respectiva información, la generación de este envío se realizó a través de un cliente *rest* implementado en el esquema antiguo del SIAFP para ser agregado dentro de las dependencias de *Struts.*

La información enviada desde el portal es: el archivo codificado en base64, el usuario que realizó el cargue, la entidad a la que pertenece, el correo y el identificador. Después de recibir esta información se inicia el proceso de cargue, en primer lugar, se agrega a los campos de entrada el campo ruta del archivo y el *checksum,* esto sirve para guardar la ubicación del archivo al momento de persistirlo y verificar la integridad de los datos. Seguidamente el proceso inspecciona que los

campos no estén vacíos exceptuando la ruta del archivo y el *checksum* que son llenados posteriormente, si los campos no están vacíos se debe validar la existencia del archivo dentro de la tabla registro_archivosnovedad, si el archivo no se encuentra registrado se debe generar la decodificación del archivo ya que está codificado en base64. A continuación se debe persistir el archivo, para generar el registro dentro de la tabla registro_archivosnovedad. Si el archivo existe o no aprueba las validaciones de entrada generar una respuesta de error, todo el proceso se muestra en la Figura 15.

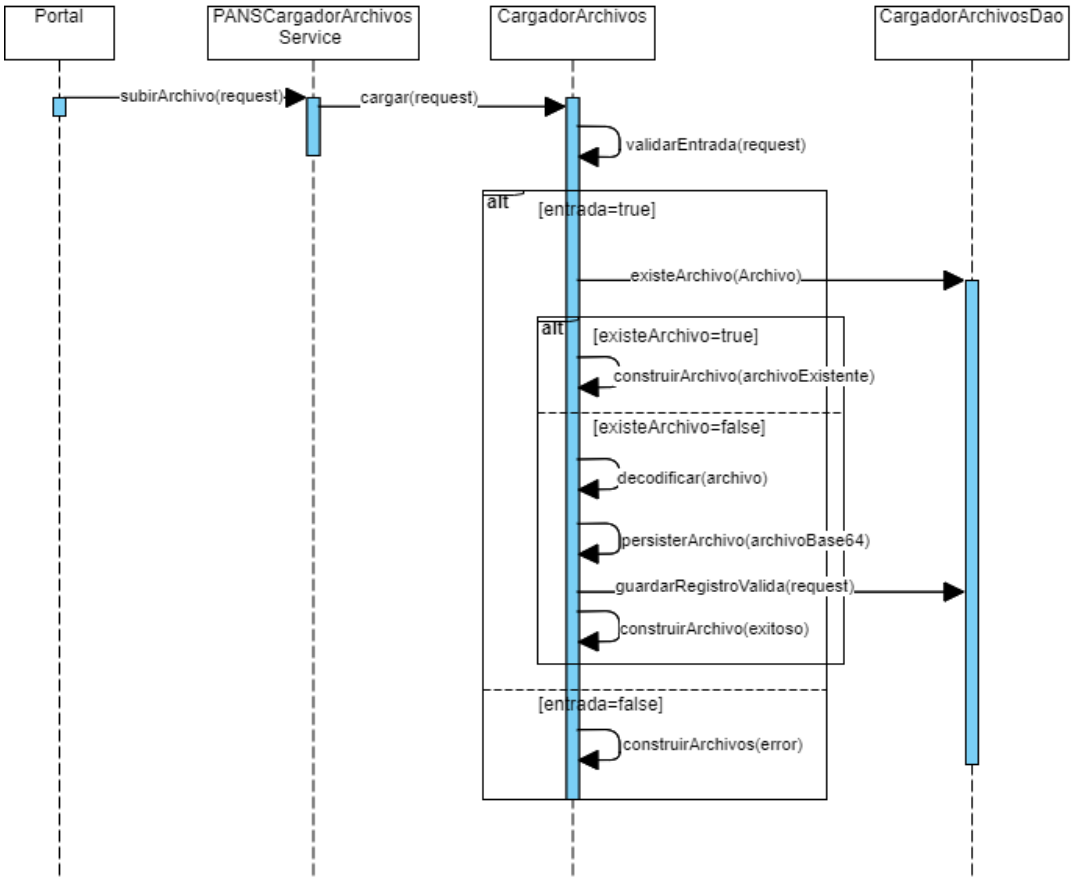


Figura 15. Diagrama secuencia de componente cargador.

El proceso de persistir los archivo tiene una serie de pasos: el primero es generar la ruta del directorio donde se alojará el archivo, segundo validar su existencia en el directorio creado, si no existe se almacena, en cambio sí existe se debe eliminar para poder almacenarlo. Después se debe verificar la integridad de los datos con el algoritmo Adler32, por último, se modifica los atributos con el *checksum* generado con el algoritmo y también la ruta del archivo como se muestra en el siguiente pseudocódigo de la Figura 16.

```
procedimiento persistirArchivo(RequestExtendido request , byte[] archivo)
        var
                Cadena: rutaDirectorio, rutaArchivo
                File: file
                fileOutputStream:outputStream
        inicio
                rutaDirectorio<-"pathBase"+"/"+request.getNovedad()
                rutaArchivo<- rutaDirectorio+"/"+request.getNombreArchivo()
                file <- rutaArchivo
                file.crearDirectorio()
                si(existeArchivo(file))
                        eliminarArchivo(file)
                fin si
                outputStream=escribirFisicamente(file)
                outputStream.escribir(archivo)
                request.setCheckSum(generarCheckSum(archivo))
                request.setRuta(rutaArchivo)
        fin_procedimiento
```

Figura 16. Pseudocodigo procedimiento de persistir un archivo .

En este método se debe tener en cuenta las propiedades del servicio, dado que ahí se encuentra la ruta de almacenamiento de los archivo, esta propiedades se obtiene a través del cargador de propiedades, está representada en en el pseudocódigo s con la variable “pathBase”.

Para la construcción de la respuesta del servicio se realizó diferentes vectores donde en la primera posición de cada vector se guardará el código de respuesta, la segunda es la descripción de la respuesta, la tercera una bandera que indica si fue exitoso o no el proceso, para que a través de ellos se forme el *response* de salida, este proceso se muestra en el siguiente pseudocódigo de la Figura 17.

```
CargadorArchivosCargarResponse función construirRespuesta(array[] respuesta)
 Inicio
  retornar construirRespuesta(respuesta[0], respuesta[1], (respuesta[2]));
fin_funcion
CargadorArchivosCargarResponse_función construirRespuesta(cadena codigo, cadena descripcion, boolean exitoso)
 var:
    CargadorArchivosCargarResponse respuesta
Inicio
    respuesta.codigoRespuesta = codigo
    respuesta.mensajeEjecucion = descripcion
    respuesta.exitoso = exitos
    retornar respuesta
fin_funcion
```

Figura 17. Construcción respuesta servicio Cargador

La siguiente sección es del arquetipo *Standalone* donde su arquitectura es muy similar a la de un servicio *rest*. La implementación del componente lanzador fue generada por medio de este arquetipo, dado que es un *Standalone* que se ejecuta cada cierto tiempo para enviar archivos a la etapa de validación.

4.3.5 Arquitectura del arquetipo Standalone

El arquetipo *Standalone* tiene la similitud con el arquetipo servicio de la sección 4.3.2 en que los componentes negocio, modelo, utilidades y persistencia cumplen con la misma función. Sin embargo, se diferencia por la ausencia de los componentes de Clientes-rs, Interface-rs ni WS. Además, se genera un *jar* que se ejecuta a nivel de la máquina virtual de java, este se debe ejecutar manualmente o por medio de un *crontab*. Ahora bien, dentro de su estructura todo inicia a través del método *main*. El manejo de propiedades debe ser por medio de archivos *propierties*.

4.3.6 Componente Lanzador

El componente lanzador es el encargado de tomar los archivos que se encuentren en estado cargado, validado o procesado y enviarlos a la etapa de validación, este componente ayuda a gestionar el reintento de las etapas de validación y procesamiento cuando se presenta alguna falla en los servicios, además de iniciar el procesamiento de un nuevo archivo.

Este componente es un proceso *Standalone* en el cual se definieron dos modos, el primero modo, es el estándar que consiste en tomar los archivos registrados en la base de datos que cumplan con los estados nombrados anteriormente. El segundo modo, es el priorizado que consiste en enviar los identificadores de los archivos que deben ser procesados antes de los que están en cola, a través de los parámetros de ejecución. Este modo es usado sobre todo por los operadores de la OGS en el momento que se genera un servicio de este proceso, debió a que las AFP o Asofondos solicita el procesamiento de un archivo con prioridad. Este proceso se ejecuta cada 5 minutos, el cual fue un tiempo sugerido por el asesor de la práctica, el cual se representó por medio de un diagrama de secuencia como se muestra en la Figura 18.

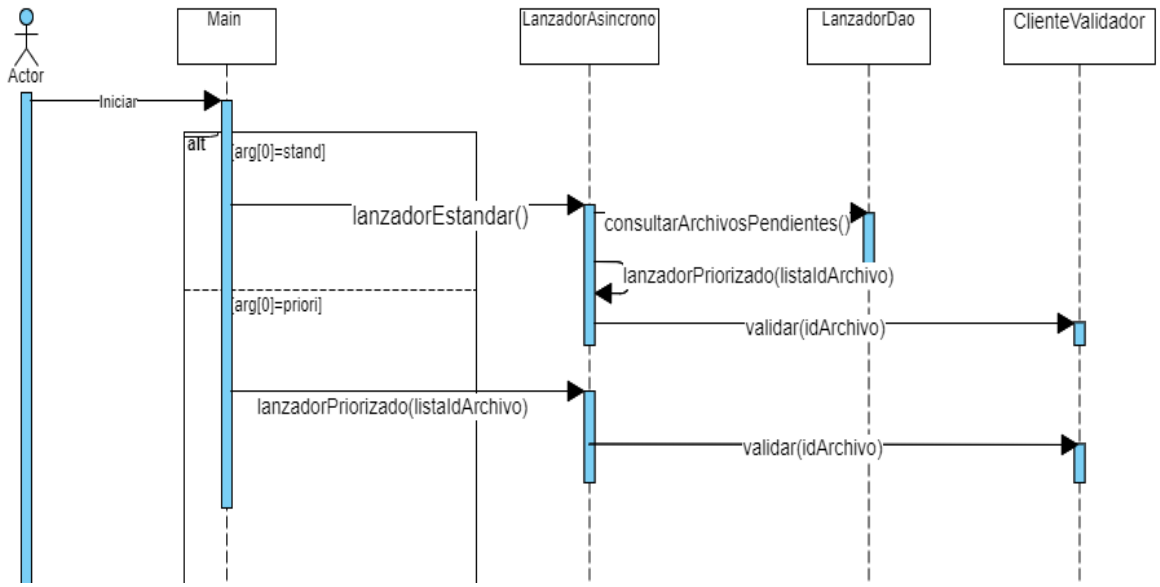


Figura 18. Diagrama secuencia Componente Lanzador.

Al iniciar el proceso se deben enviar en los argumentos del programa, el modo en el que el lanzador se ejecutará, si es modo estándar consultará los identificadores de los archivos que se encuentren en estado cargado, validado o procesado ordenándolos del más antiguo al más recientes, con estos identificadores consultados se invoca el método del modo priorizado para que el se encargue de generar la llamada al servicio Validador. Si se indica que el modo es priorizado se debe recibir en los argumentos del programa el identificador de cada archivo.

Adicionalmente se realizaron validaciones previas donde el número de argumentos sea el correcto, al igual que el modo recibido por los argumentos correspondan a los definidos en el proceso.

4.3.7 Componente común

El componente común se creó con la finalidad de generar abstracciones de todo el proceso, dado que se usan en los componentes de cargador, validador, procesamiento y notificación. Con este componente se facilita el manejo de los modelos de cada novedad, ya que se generan extensiones de abstracciones implementadas agregando campos propios de cada novedad, al igual que se podrá reutilizar modelos para la generación de bitácoras, debido a que es igual para todos los archivos, como se muestra en la Figura 19.

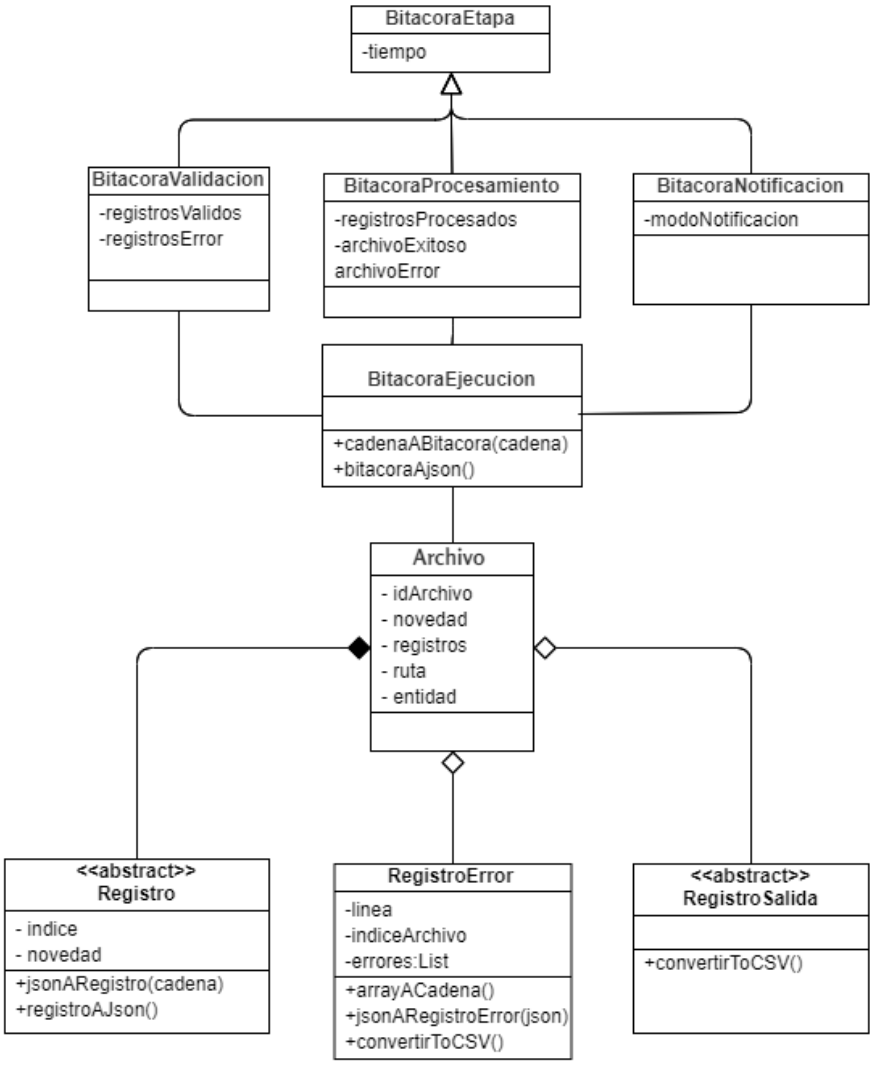


Figura 19. Diagrama de clases Componente común.

La clase Archivo tendrá toda la información suministrada al momento del cargue, al igual que todos los registros procesados, registros que no generaron error en la etapa de validación y los que sí, adicionalmente una bitácora para saber información como tiempo de ejecución, numero de registros procesados , modo de notificación entre otra.

La clase registro es una clase abstracta que indica el índice del registro en el archivo y la novedad por la cual se procesará. Los componentes tendrán que extender de ella para crear el registro especifico de cada novedad. Por ejemplo, el registro de la novedad 032 para los componentes de validador y procesador deberá extender de la clase Registro, para así obtener los método y atributos de esta clase, tendrá que implementar el método jsonARegistro el cual es encargado de crea un *json* del registro específico para que este sea guardado en la base de datos.

El componente común tiene métodos que son usados en todo el proyecto en cada uno los módulos como: obtención de la información del archivo (novedad, estado, usuario, entidad, ruta entre otros datos), actualización del estado del archivo, por último, actualización de la columna resumenEjecucion de la tabla registro_archivosnovedad después de finalizado el procesamiento del archivo. Todas estas son operaciones a la base de datos de actualización y de selección.

4.3.8 Componente validador

El componente validador se diseñó de manera extensible, permitiendo la implementación del proceso de validación para cualquier novedad. Se encarga de generar validaciones a cada uno de los registros (tramas) del archivo, teniendo en cuenta que es diferente según cada novedad porque las tramas varían en información y estructura. Por esta razón, se busca realizar una abstracción de este proceso a través de la clase “ValidorEspecifico” donde tiene un método abstracto “realizarValidacionEspecifica” como se muestra en la Figura 20, de modo que cualquier clase que extienda de ella pueda hacer su implementación de las validaciones de los registros según su necesidad o la novedad lo indique.

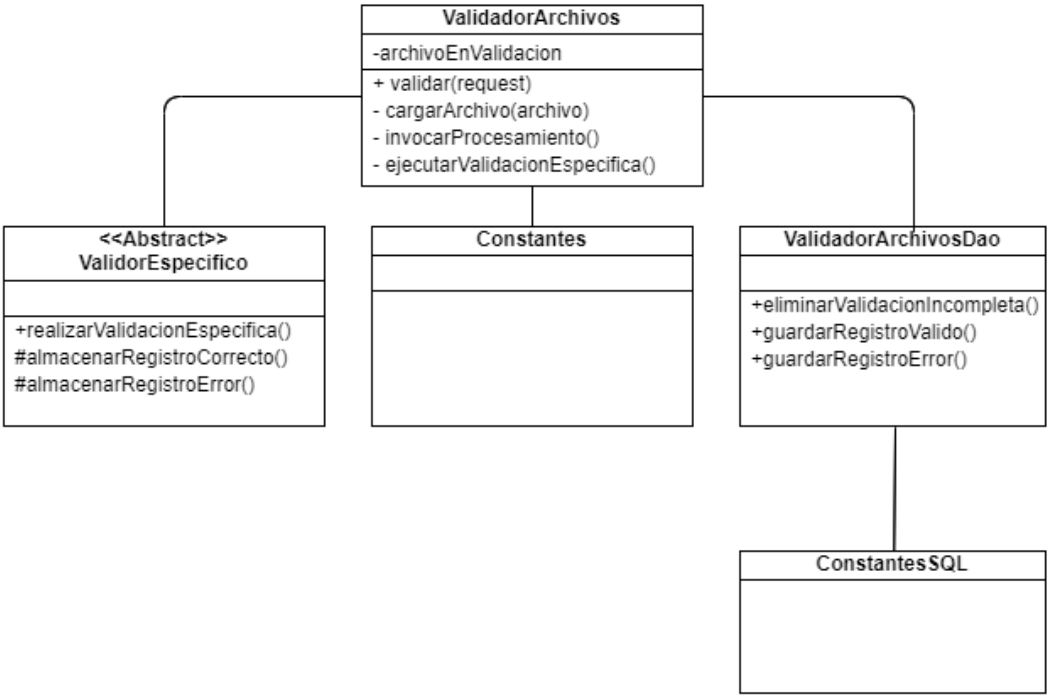


Figura 20. Diagrama de clases validador sin novedades.

El componente validador se creó a través del arquetipo de servicio rest, por esta razón al momento de recibir una petición como dato de entrada se tiene el identificador del archivo, a partir de esto se iniciará el proceso de validación a través del método “validar” de la clase “ValidarArchivo”. Lo primero que se realiza es consultar la información del archivo, esto se genera a través del componente común (sección 4.3.7) dado que es el encargado de consultar la información del archivo a

la base de datos y regresarla, esta información se guarda dentro un objeto de la clase Archivo del mismo componente, con este objeto se deben generar validaciones previas al archivo para validar cada uno de sus registros. A continuación, se presentan los pasos que realiza el proceso de validación:

1. **Verificar la existencia del archivo**: si no existe genera una respuesta de error. Si existe, continua con el siguiente paso
2. **Evaluar si el archivo se encuentra en estado cargado:**
   a. se debe cargar los registros que consiste en leer cada una de las líneas del archivo, teniendo en cuenta si tiene cabecera o no.
   b. Si el cargue fue correcto se debe generar las validaciones especificas según la novedad.
3. **Si el archivo se encuentra en estado Validado o Procesado:** se debe realizar la invocación a la etapa de procesamiento.
4. **Construcción respuesta**: en cada uno de los pasos se debe construir la respuesta según la condición, se genera de la misma manera que en el componente de cargador a través de vectores, donde en la primera posición de cada vector se guardará el código de respuesta, la segunda posición es la descripción de la respuesta, la tercera posición una bandera para indicar si se finalizó la etapa y en la cuarta posición una bandera para mostrar si se pudo enviar a la siguiente etapa para que a través de ellos se forme el *response* de salida.

Se creó un diagrama de flujo representado en la Figura 21, en el cual se indican los caminos que siga el archivo según las condiciones que se describieron con anteriormente.

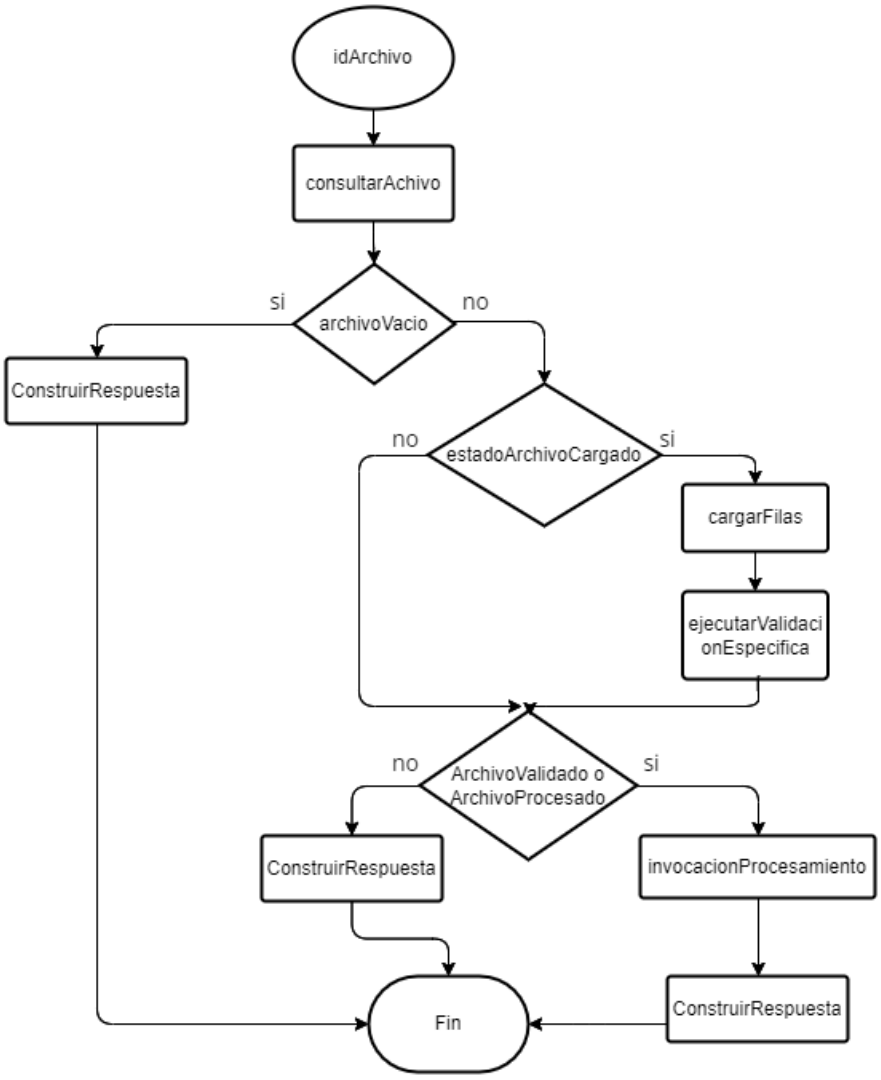


Figura 21. Diagrama de flujo validador genérico.

Para la invocación de las validaciones específicas de cada trama se construye dinámicamente el nombre de la clase según la novedad a la que pertenezca el archivo, la construcción se genera a partir de la ubicación de la clase en los paquetes y la novedad a través de la constante claseValidadorEspecifico, en ella se definió la ruta del paquete donde están las clases en específico de cada novedad concatenado con la variable wilcarNovedad que al remplazarla genera la ruta exacta de la clase en específico de la novedad del archivo que se está validando, a partir de esto se crea una instancia de la clase para invocar su validación, adicionalmente se debe tener en cuenta que el archivo no se encuentre en el caché representado con el atributo “archivosEnValidacion”, este atributo es una lista de los identificadores de archivos que se encuentran en el proceso de validación expuestos en la Figura 22.

```
Función ejecutarValidacionEspecifica(archivo):
 Si archivo.getIdArchivo() no está en archivoEnValidacion:
  validadorEspecificoName = Constantes.claseValidadorEspecifico. reemplazar(Constantes.wilcarNovedad, archivo.getNovedad())
  validadorEspecifico = instanciarClase(validadorEspecificoName)
  archivo = validadorEspecifico.realizarValidacionEspecifica(archivo)
  Remover archivo.getIdArchivo() de cache
 Devolver archivo
```

Figura 22. Pseudocódigo invocación clase especifica validación.

En la clase ValidorArchivosDao es una clase que se encarga de generar todas las transacciones a la base de datos, por esta razón tiene la implementación de la conexión a la base de datos a través del componente de BD, además la clase ValidorArchivosDao es una extensión de la clase ArchivoDao del componente Común, para así poder hacer uso de métodos como lo son consultar toda la información de archivo y actualizar el estado. De las funcionalidades generadas por esta clase son la eliminación de registros incompletos, el almacenamiento de registros de éxito y de error.

Se debe tener en cuenta que para este componente existen dos extensiones de la clase “ValidorEspecifico”, donde cada una de ellas será la forma en la que se implementarán las validaciones específicas para cada novedad, en esta práctica solo se realizó para la novedad 032 y 155, sin embargo, puede generarse para cualquier otra. En cada una de las generalizaciones se especificó, métodos como: almacenamiento de registros correctos, guardado de los registros de error, actualizaciones de bitácoras y reinicio de validaciones que quedaron inconclusas para el archivo que se está evaluando. El diagrama de clases de la Figura 23 muestra la implementación generada.

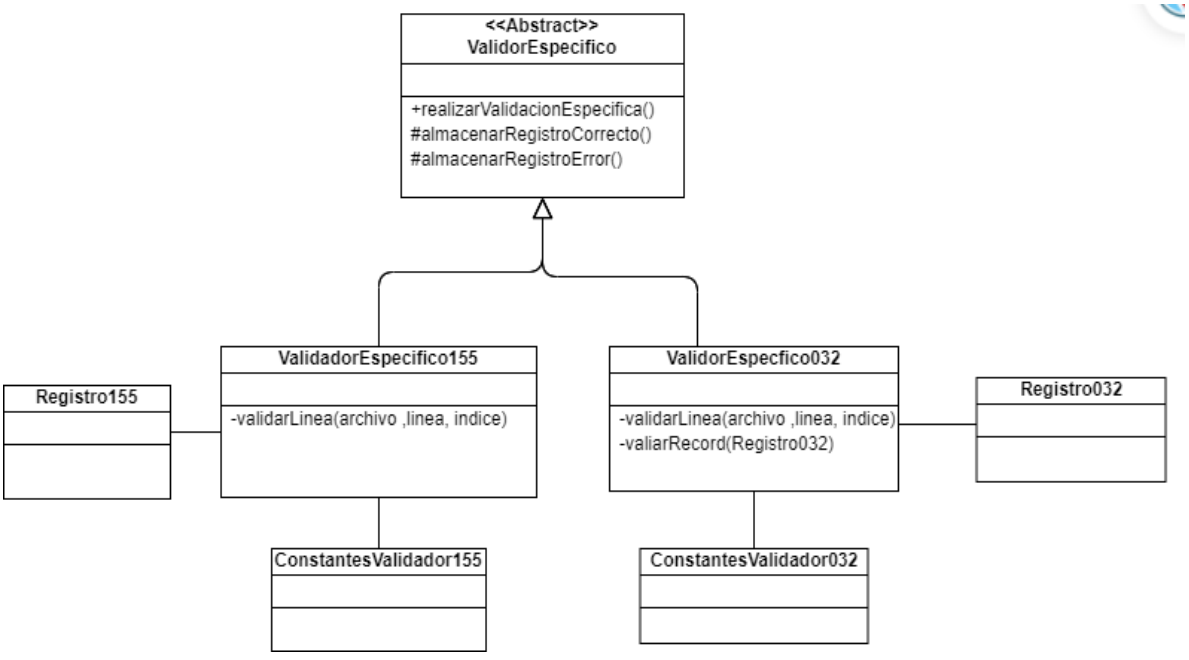


Figura 23. Diagrama de clases Validador 155 y 032.

En la siguientes dos secciones se explica el funcionamiento de la novedad 032 y otra para la novedad 155 para el proceso de validación del archivo.

4.3.8.1 Validaciones Novedad 155

En esta novedad se realizó una sola validación puesto que con la integración del proyecto de procesamiento en línea se genera las validaciones más específicas para cada trama. La validación solicitada es que la longitud de la trama cumpla con un numero de caracteres específicos, se realizó de la siguiente manera:

1. Se verifica que la etapa de validación para el archivo no se encuentre incompleta, de ser así se elimina cualquier registro del archivo de las tablas registro_archivoserror y registro_recordsnovedad, teniendo en cuenta que en ellas se almacena los registros exitosos y de error, para que así la etapa comience desde cero.
2. Se Inicia la iteración de los registros del archivo que previamente se ha cargado.
3. Para cada registro se realiza la invocación al método validarLínea, este recibe por parámetro la trama la cual se convierte en Registro155 teniendo en cuenta que esta clase extiende de la clase Registro del componente Común donde solo se agregan atributos propios de la novedad. adicionalmente se recibe el índice de donde estaba ubica la línea en el archivo.
4. En el método validarLinea genera la validación de la longitud de caracteres de la trama.
5. Si la validación es correcta guarda el registro155 y el identificador del archivo en la tabla registro_recordsnovedad.
6. Si la validación es incorrecta se crea un objeto de la clase RegistroError con índice de donde está ubicado el registro en el archivo, la trama y una lista con la descripción del error generado para guardarlo en la tabla registro_archivoserror junto con el identificador del archivo.
7. Se genera una bitácora para guardar información del total de registros validados, el número de registros que fueron exitosos, el número de registros fallidos y el tiempo que le tomo generar la etapa, además se cambia el estado del archivo a validado.

4.3.8.2 Validaciones Novedad 032

La novedad 032 es una novedad en la cual se genera muchas consultas a la base de datos, por esta razón, para obtener la información es necesario que este correctamente estructurada la trama, de tal manera la novedad 032 es una novedad a la cual se le aplican más validaciones como: longitud de la trama, tipo de dato de cada uno de los campos, longitud de cada campo, no aceptación de caracteres especiales, siendo estas las validaciones de estructura, además se generan validaciones de información propia del negocio como tipo de documentos y longitud del número de documento. Para generar estas validaciones se realizó de la siguiente manera:

1. El paso 1 y 2 de la novedad 155, se repite para esta novedad.

2. Para cada registro se realiza la invocación al método validarLínea, este recibe por parámetro la trama la cual se convierte en un Registro032 teniendo en cuenta que esta clase extiende de la clase Registro del componente común, donde solo se agregan atributos propios de la novedad, así mismo se recibe el índice de donde estaba ubica la línea en el archivo.
3. En el método validarLinea se crea una lista donde se guarda los errores y se realizan las siguientes validaciones:
    a. Validar que la trama cumpla con una longitud mínima y una máxima, si no la cumple agrega la descripción del error a la lista, si cumple sigue con la siguiente validación.
    b. Validar que el registro no tenga caracteres especiales, si los tiene agrega la descripción del error a la lista.
4. Después de superadas las validaciones se invocará el método validarRecord, este método se encarga de hacer las validaciones de negocio, al igual que guarda en una lista las descripciones de los errores generados.
5. Si cumple con las validaciones guarda el registro032 en la tabla registro_recordsnovedad junto con el índice del archivo.
6. Si no cumple con las validaciones se realiza el paso 6 la novedad 155.
7. El paso 7 es igual para ambas novedades.

4.3.9 Componente procesamiento

El componente de procesamiento es el encargado de procesar los registros exitosos de cada archivo según la novedad a la cual pertenezca, de esta manera la respuesta del procesamiento y los registros de error podrán ser notificados por la siguiente etapa a través del archivo de notificación. El componente de procesamiento tiene las mismas condiciones que el validador, debe ser lo suficientemente extensible para que pueda ser usado por cualquier novedad, además debe poder retomar el procesamiento de los registros de un archivo cuando este haya quedado incompleto por algún error. Por esta razón, se crearon abstracciones como se realizó en el componente validador nombrado en la sección anterior, donde se tendrá la clase ProcesadorEspecifico, en ella los método de realizarProcesamientoEspecifico, leerRegistroValidos, descargarArchivosProcesados como se muestra en la Figura 24. Por lo tanto, la clase que extienda de la abstracción implementará estos método según como la novedad lo describa.

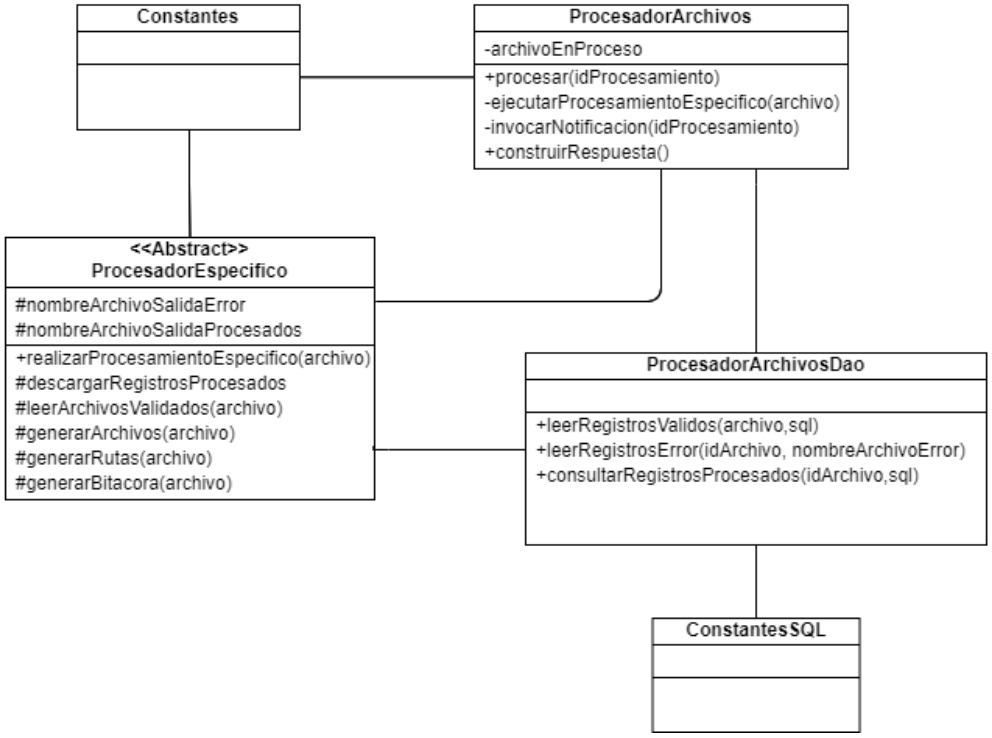


Figura 24. Diagrama de clases Procesamiento General.

En el método realizarProcesamientoEspecifico se realiza el procesamiento que debe tener un registro para dar respuesta a la novedad, la implementación esta sujeta a las condiciones de negocio de cada novedad, pueden ser, consultas a la base de datos o invocaciones a proyectos internos del SIAFP. El método leerRegistrosValidos tiene como fin leer los registros de la tabla registro_recordsnovedad, con la salvedad que no hayan sido procesados anteriormente. El método descargarRegistrosProcesados es el encargado de leer los registros que ya han sido procesados para así generar el archivo de procesamiento.

Adicionalmente, en la clase ProcesadorEspecifico se generaron funcionalidades que serán usadas por todas las subclases como: generación de archivos de error y de procesamiento, generación de rutas de ubicación de los archivos generados y por último registro en las bitácoras, estas funcionalidades son procesos en común que realizan todos los archivos sin importar la novedad.

La clase ProcesadorArchivo es la clase encargada de: generar el paso a paso en la etapa de procesamiento, generar la invocación a cada procesamiento especifico según la novedad y construir la respuesta del servicios al igual que todas las validaciones previas para genera el procesamiento de una novedad. En la clase ProcesadorArchivosDao se tendrá todas las invocaciones a la base de datos, esta clase extiende de la clase ArchivoDao del componte común, para obtener métodos como consultar información del archivo y cambiar el estado, de la misma manera

realiza funcionalidades como: obtener los registros validados con éxito teniendo en cuenta la novedad, consultar los registros de error y el número de registros procesados del archivo según su novedad. De esta clase se extienden las clases *dao* de los procesamientos en específico.

Tener en cuenta que este componente fue creado con el arquetipo de servicio *rest*, como dato de entrada del servicio es el identificador del archivo, a partir de esto se inicia el procesamiento del archivo. Como primer paso se tiene la invocación del método procesar de la clase ProcesadorArchivos generando los siguientes pasos:

1. Consultar la información del archivo a través del método del componente común.
2. Verificar si el archivo existe , si existe continuo con el proceso, si no construye la respuesta de error.
3. Verificar que el estado del archivo sea Validado, si es así continuará con el proceso, sino pregunta si el estado es Procesado para continuar el en paso número 5.
4. Comprobar si los registros validados y de error generados en el módulo validador son mayor a cero, para invocar al procesamiento especifico de cada novedad. Para generar la invocación del procesamiento especifico se realiza de la misma manera que en el componente Validador.
   Se construye dinámicamente el nombre de la clase según la novedad a la que pertenezca el archivo, a partir de los paquetes donde se encuentran las clases específicas del procesamiento y la novedad del archivo, después se carga dinámicamente la clase para crear una instancia de ella e invocar a el método especifico. Si no hubo errores en el procesamiento se construye la respuesta de éxito del servicio, si en el caso contrario si generaron errores construye la respuesta de error.
5. Verificar si el estado del archivo es procesado y la ruta de salida no es vacía, genera la invocación al servicio de notificador.

Estos pasos se representaron en el diagrama de flujo que se muestra en la Figura 25.

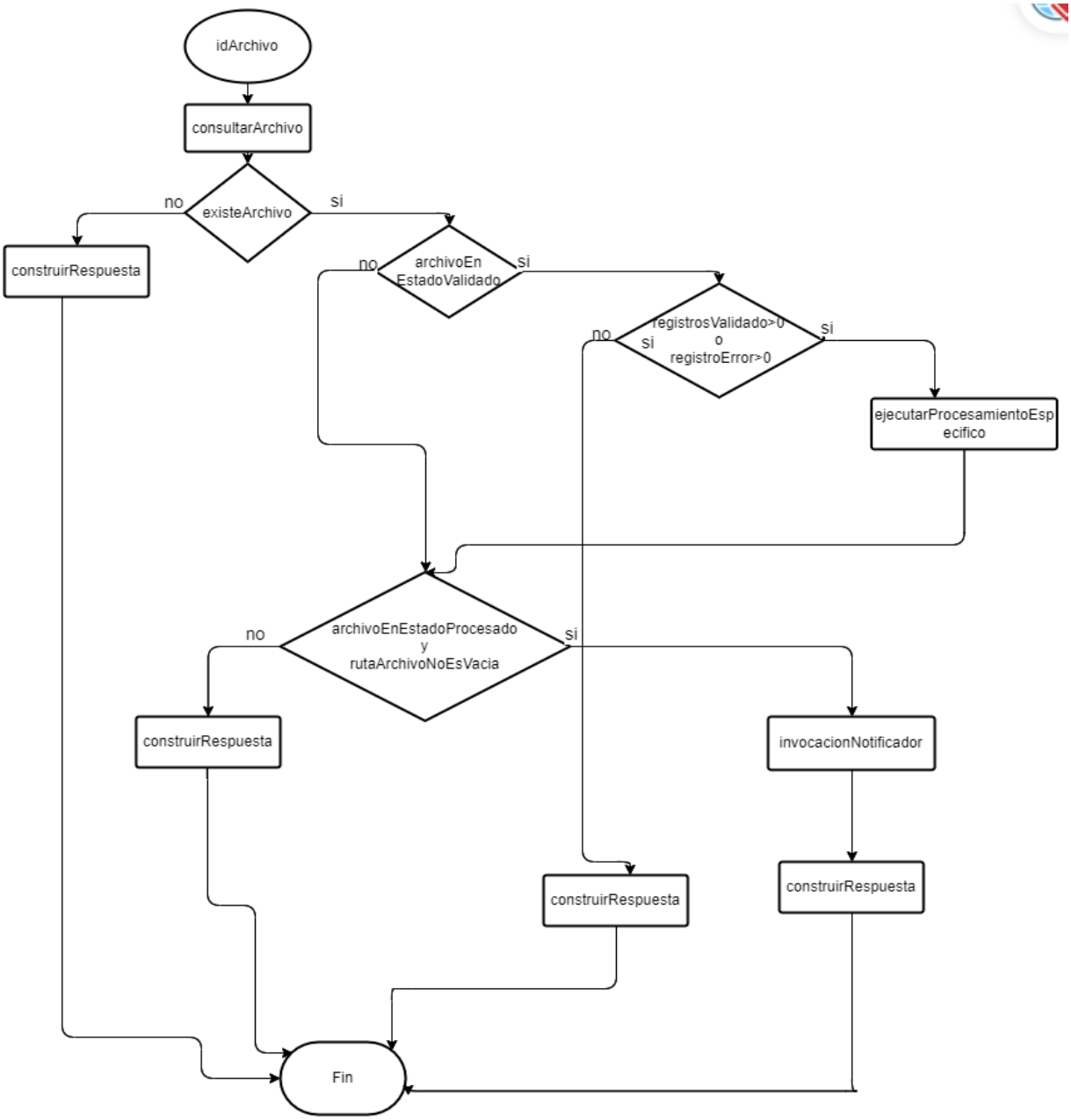


Figura 25. Diagrama de flujo del procesamiento general.

Cabe resaltar que para la construcción de la respuesta se crearon vectores donde en la primera posición de cada vector se guardará el código de respuesta, la segunda posición es la descripción de la respuesta, la tercera posición una bandera para indicar si se finalizó la etapa y en la cuarta posición una bandera para mostrar si se pudo enviar a la siguiente etapa para que a través de ellos se forme el *response* de salida.

En la práctica se realizó el procesamiento de dos (2) novedades como se ha nombrado a lo largo de este documento, la novedad 032 y 155, en el componente

de procesamiento se generó el procesamiento específico de cada una de ellas, teniendo en cuenta que la novedad 032 es una novedad de consulta del estado general del afiliado y la novedad 155 es para obtener el beneficio pensional, por esta razón cada una tiene un funcionamiento diferente.

Para la implementación de cada procesamiento en específico se extendió de la clase “ProcesadorEspecifico”, al igual que se extendió de la clase “ProcesadorArchivosdao” como se muestra en la Figura 26, dado que se tienen consultas que son propias de cada una de las novedades, al igual que consultas generales usadas en ambos procesos. La clase registros de ambas novedades son extensiones de la clase Registro del componente común y las clases registroSalida son extensiones de la clase RegistroSalida del componente común.

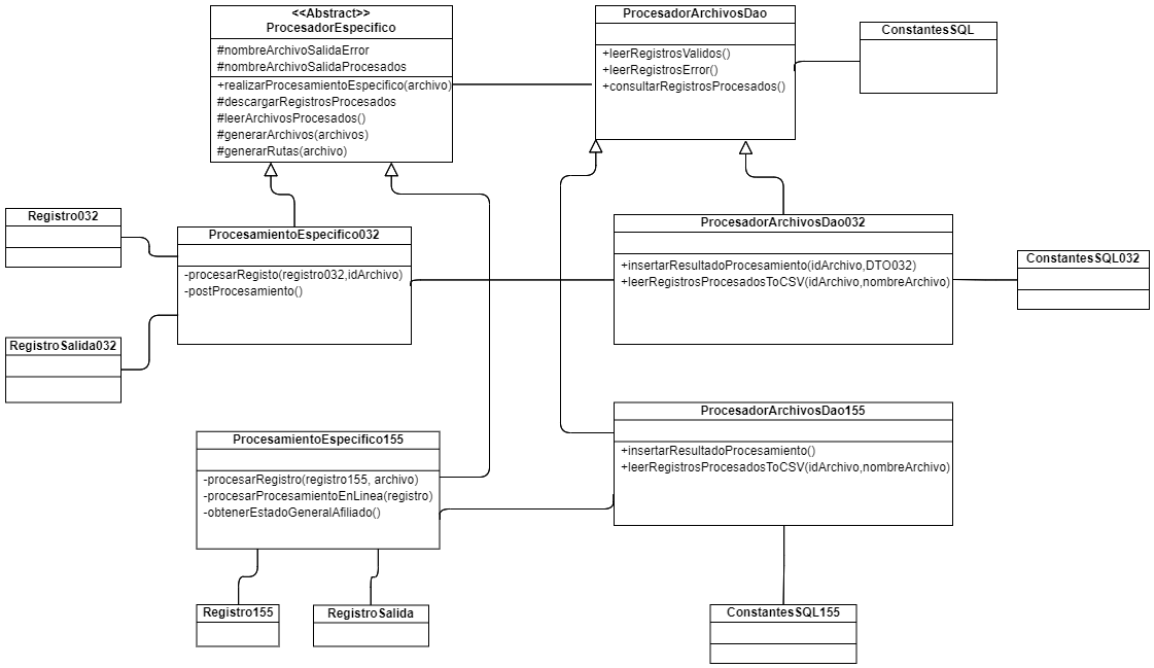


Figura 26. Diagrama de clases implementación procesamiento novedad 032 y 155.

Para el procesamiento se generaron nuevas tablas para guardar la información obtenida de cada registro. Para la novedad 032 se creó la tabla serv032_express y para la novedad 155 se creó la tabla serv155, en ellas se tienen atributos como: el identificador del archivo al que pertenece el registro, el índice del registro dentro del archivo, además cada tabla tiene atributos propios del negocio, por ejemplo, la tabla de la novedad 032 tendrá la información básica afiliado y en la de novedad 155 la respuesta del proceso para obtención del beneficio pensional, entre otra información.

4.3.9.1 Novedad 032

Todo el proceso inicia con la implementación del método “realizarProcesamientoEspecifico” siguiendo los siguientes pasos:

1. **Obtener los registros validados**: en el método leerRegistrosValidos el cual hace parte de la abstracción se obtendrán los registros que serán procesados, esto se realiza a través de los parámetros de entrada ya que por medio de ellos se enviará la consulta para obtener los registros que serán procesados, teniendo en cuenta que no estén registrados en la tabla de serv032_express, dado que si están en esta tabla es porque ya fueron procesados.

2. **Iterar los registros obtenidos en punto anterior**
   En este paso se realiza la invocación a el método procesarRegistro pasando por parámetro el registro y el identificador del archivo.

3. **Procesar registro**
   En este paso se genera el procesamiento especifico de la novedad, para la novedad 032 su procesamiento es consultar la información del afiliado como lo es nombres apellidos, es su estado pensional, qué tipo de pensión tiene entre otra información a través de la base de datos de Oracle. El resultado de estas consultas se guarda en la tabla de la novedad(serv032_express).

4. **Generación de la bitácora:**
   En este paso se guarda la información del número de registros totales a procesar, los que se procesaron correctamente y los de error.

5. **Generar Archivos:**
   Este método hace parte de la abstracción, es de los más importantes ya que en este se genera la creación de los archivos de respuesta, tanto el de error como el de procesamiento. Por esta razón se validará si el número de registros de error o de procesamiento es mayor a cero. Si es así se generan las rutas de los archivos.

   Las rutas se construyen a partir de la propiedad de la ruta de salida de los archivos, el nombre del archivo de entrada y su extensión. Para el archivo de procesamiento se agrega la extensión S, para el de error se toma el nombre del de procesamiento y se le agrega a la extensión la palabra ERROR como se muestra en la figura, después de esto se llenan los archivos según cada registros, los de error en el archivo de error, los de éxito en el archivo de éxito.

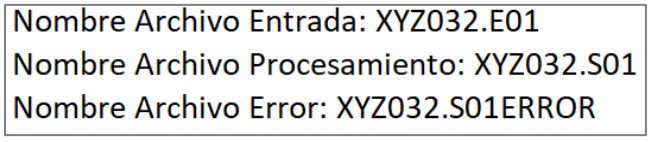


Figura 27. Nombrado archivos salida

Para obtener los registros del procesamiento, esto se realiza por medio del método descargarRegistrosProcesados, en el se invoca al método de

leerRegistrosProcesadosToCSV de la clase ProcesadorArchivos032, para cada uno de los registros consultados se debe crear un objeto de tipo RegistroSalida032, este debe tener todos los campos de la tabla serv032_express. Para los archivos de salida son CSV, se debe implementar el método de toCSVLine de la clase RegistroSalida del módulo Común en la clase RegistroSalida032, la cadena que regresa se escribe en el archivo de registros procesados.

Para construir el archivo de error lo primero que se realiza es consultar los registros de error del archivo por medio de la columna errores de la tabla registro_archivoserror. Después, con cada uno de los registros consultados se debe crear un objeto de tipo RegistroError del Componente Común, hay que tener en cuenta que la columna errores es de tipo json, por esta razón se debe convertir el json a un registro de error, esto se realiza a través del método jsonARegistroError. Seguidamente se convierte el objeto creado en una cadena mediante el método toCSVLine de la clase RegistroError.

Posteriormente, se crea un comprimido con los archivos creados (error y procesamiento) y este es el que se notificará en la etapa de notificación.

6. **Se actualiza el estado del archivo**: se debe actualizar el estado del archivo a procesado, al igual que la ruta de salida del archivo del comprimido en la tabla registro_archivosnovedad.

4.3.9.2 Novedad 155

En el procesamiento de esta novedad se genera la invocación a procesamiento en línea que es un servicio SOAP, donde el cliente usado para el consumo se expuso en la sección 4.3.1.4 , al igual que la novedad 032 está también su proceso inicia en la implementación del método “realizarProcesamientoEspecifico” siguiendo los siguientes pasos:

1. **Obtener los registros validados**:
   Este método funciona de la misma manera que la novedad 032, con la diferencia que consulta todos los registros del archivo excepto los que se encuentren dentro de la tabla serv155, Dado que si están en esta tabla es porque ya están procesados.

2. **Iterar los registros obtenidos en punto anterior:**
   Este paso funciona de la misma manera que la 032 con la diferencia que ahí se envía el archivo completo como segundo parámetro.

3. **Procesar registro:**
   En este paso es donde está la diferencia de la novedad 155 y 032, dentro de el se realizará el llamado al servicio de procesamiento en la línea, sin embargo, para esto se debe tener en cuenta lo siguiente:
   - Se debe tener la URL y la contraseña para acceder al servicio.

- Este servicio es síncrono porque puede dar respuesta en la primera invocación o devolver un token para consultar después de un tiempo.
- La respuesta del servicio es una lista de cadenas donde la longitud me identifica el tipo de respuesta
- Si el tamaño de la lista es 1 es porque respondió solamente regresando el token, por lo tanto, se debe volver a consultar para traer la respuesta. Para esto se definió un tiempo de espera y número de intentos para consultar(se definieron por medio de propiedades).
- Si el tamaño de lista es 3 quiere decir que uvo un error en el procesamiento, se deben guardar los errores para agregarlos al archivo de salida.
- Si el tamaño de la lista es 7 quiere decir que genero respuesta, aunque puede que haya regresado errores de validación de la novedad, sin embargo, esto no significa que haya sido un error de procesamiento.
- Si se generaron errores en la validación, se deben concatenar todo y regresarlos en una cadena.

Después de obtener el resultado de procesamiento en línea se debe obtener el estado general del afiliado, esto es una consulta la base de datos de logs con el token que genero el servicio de procesamiento en línea. Por último, se registrará el resultado de procesamiento en línea y de la consulta del estado general del afiliado en tabla serv155.

4. **Generación de la bitácora**
   Este paso es igual que en la novedad 032, se verificará si hay registros con error y procesados.

5. **Generar Archivos**
   Este paso también es igual por que hace parte la abstracción, la diferencia es que para obtener los registros del procesamiento se realiza por medio del método descargarRegistrosProcesados, en el se invoca al método de leerRegistrosProcesadosToCSV de la clase ProcesadorArchivos155, para cada uno de los registros consultados se debe crear un objeto de tipo RegistroSalida155, este debe tener todos los campos de la tabla serv155. Los archivos de salida son CSV, se debe implementar el método de toCSVLine de la clase RegistroSalida del módulo Común en la clase RegistroSalida032, la cadena que regresa se escribe en el archivo de registros procesados.

6. Es igual que el paso 6 de la 032.

4.3.10 Componente Notificador

El componente notificador genera la notificación de los archivos construidos en el componente anterior, cuenta con tres (3) modos de notificación los cuales son: notificación por correo, notificación por Validador universal y el último que es el modo mixto que es la combinación entre la notificación por correo electrónico y por

Validador universal. El modo de notificación es determinado por la novedad, por ejemplo, para la novedad 032 su modo de notificación es por correo, sin embargo, para la novedad 155 su modo de notificación es por Validador universal o viceversa. Teniendo en cuenta esto, se implementó el modo de notificación a través de las propiedades, así, se sabrá qué modo determino la novedad para la notificación de sus archivos procesados. Se debe tener en cuenta algunas condiciones como el peso del archivo que se adjunta al correo, ya que solo recibe archivos de tamaño máximo de 25MB.

Para que el modo de notificación este ligado a la novedad, se crea propiedades por cada una de las novedades implementadas. Estas propiedades tendrán como palabra base "modoNotificacion", seguido del número de novedad (modoNotificacion032). Para la implementación se realiza la lectura de la propiedad en la clase NotificadoArchivos la cual es encargada de realizar las llamadas a cada modo de notificación. En cuanto a la implementación de cada modo exceptuando el mixto, se realizó a través de la clase NotificadorCorreo para la notificación por correo y por medio de la clase NotificadorValidador para la notificación en modo Validador universal como se muestra en la Figura 28.

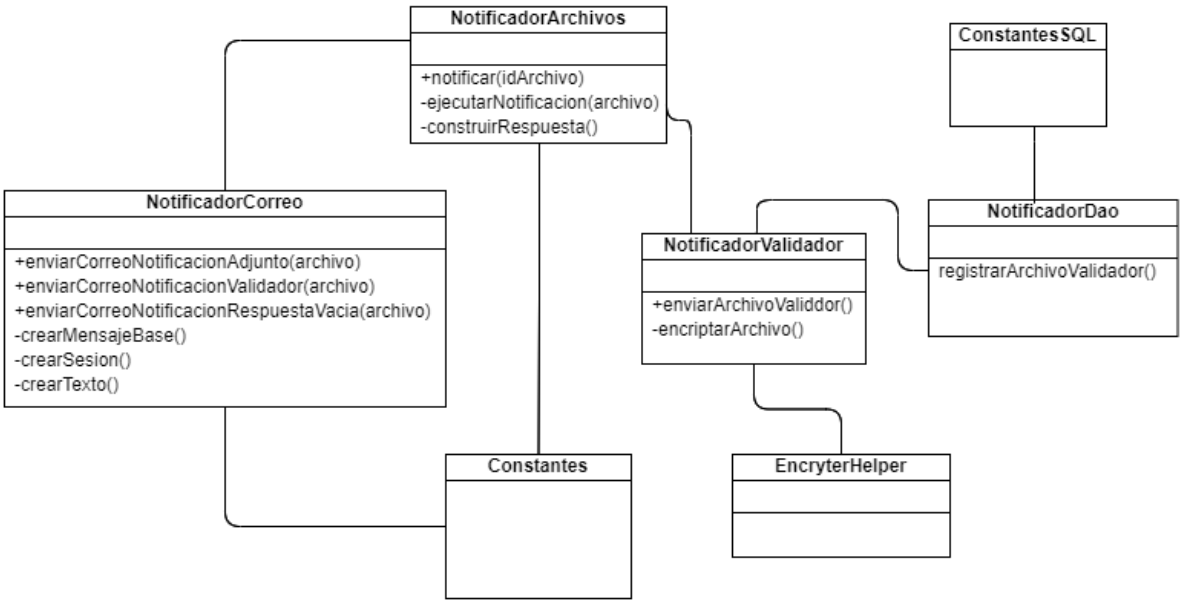


Figura 28. Diagrama de clases componente notificación.

Para realizar la notificación, lo primero que se debe hacer es obtener la información del archivo y validar que el archivo se encuentre en estado procesado, si es así se debe hacer la lectura de la propiedad y verificar el modo de notificación, si el archivo el archivo no fue encontrado en el directorio físico se debe notificar que el archivo no genero respuesta, esta notificación siempre se realizará a través del correo. Si el archivo existe se debe verificar el modo de notificación, la primera evaluación se realizó en que el modo fuera de validador, en este caso se debe ubicar el archivo en el validador y notificar por correo que el archivo se puede traer del validador,

esto es modo mixto.

Si la propiedad no es por el modo validador quiere decir que la notificación debe ser por correo para esto lo primero que se debe verificar es si el archivo cumple con el tamaño permito por el correo para ser enviado por este medio, si el archivo cumple se debe realizar él envió del correo con mensaje y el archivo adjunto de la respuesta del procesamiento. Sin embargo, si el archivo llega ser superior el tamaño permitido se debe notificar por el validador donde se le debe indicar por correo que tiene un archivo por halar por el Validador universal, para entender mejor estas condiciones se realizó un diagrama de flujo como se muestra en la Figura 29.

Si el archivo no se encuentra en estado Procesado se genera la construcción de la respuesta de error, cada una de las respuesta se construye a través de un vector donde la primera posición se guardará el código de respuesta, la segunda es la descripción de la respuesta, la tercera una bandera que indica si fue exitoso o no el proceso, para que a través de ellos se forme el *response* de salida, este proceso se muestra en el siguiente pseudocódigo

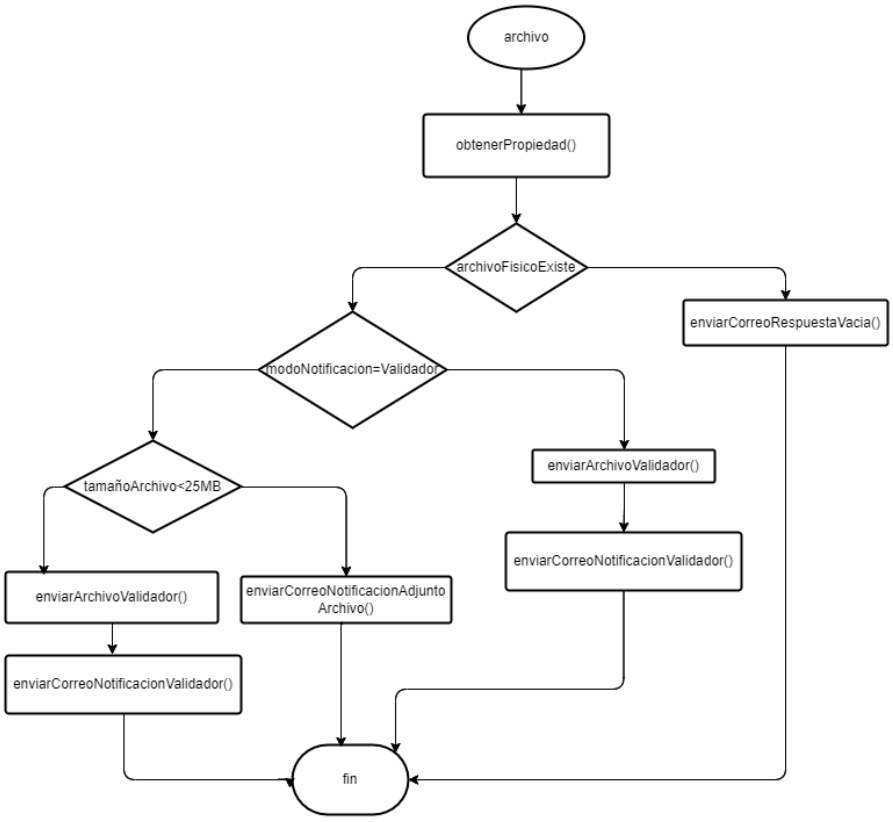


Figura 29. Diagrama de flujo notificación.

4.3.10.1 Modo Correo

El modo correo se implementó en la clase NotificadorCorreo como se muestra en la Figura 28, en esta clase se tienen diferentes métodos dado que los correos pueden

llevar archivos adjuntos o no, al igual que descripciones diferentes, por esta razón se crearon múltiples métodos con el fin de invocar el necesario como esta descrito en el diagrama de secuencia de la Figura 29.

Para el envío de los correos se deben invocar métodos que son obligatorios, uno de ellos en la “creacionSesión”, en este método se podrá establecer una sesión del correo electrónico con un servidor de correo, para crear una sesión se hace por medio de las propiedades como lo son host, puerto y bandera de autenticación, seguido de esto se debe autenticar con el correo y la contraseña, por último, siempre se retorna la sesión creada. Otro método usado es el “crearMensajebase”, en este método se recibe la sesión creada y el archivo, se crea el asunto del mensaje, el cual siempre estará formado de un mensaje base, leído de una propiedad y se remplaza el número de novedad según el archivo que se vaya a enviar, adicionalmente se agrega el destinatario que está en los atributos del archivo.

A parte del asunto se debe crear el cuerpo del correo, esto se hace con el método de crearTexto, en el cual se debe recibir por parámetro una cadena, la cual indica que plantilla se debe usar, estas plantillas se deben poder modificar, por esta razón se generaron a través de propiedades donde cada una de ellas está el mensaje que se quiere enviar como cuerpo del texto. Adicionalmente se recibe por parámetro el archivo, debido que se necesita la novedad por la cual se procesó, el usuario y el nombre del archivo para remplazarlos en la planilla. Para realizar este remplazo se definieron variables dentro de la plantilla, para que a través de ellas se pueda remplazar con la información del archivo que llegó por parámetro.

Métodos que se definieron para el envío de correos:

- envioCorreoNotificacionRespuestaVacia: Este método se usa cuando no se encontró el archivo físico en la ruta de salida. Se debe invocar los métodos obligatorios en el siguiente orden, crear sesión, crear mensaje base, crear el cuerpo del correo indicando que es un mensaje de respuesta vacía y por último se envía el mensaje.
- enviarCorreoNotificacionValidador: este método se utiliza para notificar al usuario que el archivo lo puede tomar del Validador universal, dado que el Validador universal no notifica cuando el usuario tiene archivos pendientes por tomar. Por esta razón se definió este método el cual crea una sesión, el asunto del mensaje, el cuerpo del mensaje indicando al usuario que el archivo lo puede tomar del Validador universal y por último se envia.
- enviaCorreoNotificacionAdjunto: este método es empleado cuando el modo de la notificación es correo y el archivo no supera el tamaño permitido, se invocará todos los métodos obligatorios, así mismo se debe llamar a un método que se encarga de adjuntar el archivo, para esto se debe enviar el archivo ya que en el está la ruta de ubicación del archivo de salida y por último se envía.

Después del envío se construye la respuesta de éxito y se genera la bitácora indicando por modo se realizó la notificación.

4.3.10.2 Modo validador

El modo validador se implementó en la clase NotificadorArchivo como se muestra en la Figura 28, en este solo se tiene un método en el cual se debe generar una serie de pasos para poder ubicar el archivo en la ruta del Validador universal, además se debe realizar un registro en la base de datos del archivo, para que sea posible que el usuario lo pueda tomar. Cabe aclarar que este proceso fue tomado de un proyecto el cual tiene la implementación de como agregar un archivo a la ruta del Validador universal, por esto lo que se hizo fue tomar el código y adaptarlo a nuestro componente. Los pasos son los siguientes:

1. Obtener la entidad del usuario.
2. Obtener el usuario de validador dado que no siempre el de la página corresponde al de validador.
3. Crear un archivo temporal a partir del actual con extensión .zip.
4. Comprimir el archivo creado en el paso anterior, dado que en el validador solo guarda archivo comprimidos y encriptados, para que así cuando el usuario lo tome este sin comprimir , sin embargo, para este caso se necesita que el archivo tomado sea un comprimido porque en el estarán los dos archivos el de error y el exitoso.
5. Establecer la ruta de ubicación del archivo encriptado, se forma con una ruta base del validador, la entidad del usuario, el usuario y el nombre del archivo comprimido en el paso anterior.
6. Encriptar el archivo.
7. Por último, se deben generar todos los registros en las tablas del Validador universal para que el usuario pueda tomar el archivo.

Las novedades en que a pesar de su modo de notificación este dado por correo, si el archivo a notificar pesa más de lo permitido se enviará por Validador Universal por ende tendrá que pasar por la serie de paso ya nombrados.

Después que el archivo ya está ubicado en la ruta del Validador se deberá notificar por correo que el archivo ya se puede tomar, esto se realiza por el método enviarCorreoNotificacionValidador de la clase NotificadorCorreo. Por último, se debe construir la respuesta del servicio indicando que fue correcto y generar una bitácora de por qué modo se notico.

4.3.10.3 Modo mixto

En este modo se combinan los modo de notificación por correo y por validador, este se genera cuando el archivo es enviado por validador o supera el tamaño permitido por el correo.

4.3.11 Pruebas del Software

En el desarrollo de esta práctica se realizaron dos (2) tipos de pruebas: las pruebas unitarias, que son realizadas por el desarrollador, con las que se busca probar funcionalidades pequeñas del código para validar si esta correcta o no, esto ayuda

en la modularización del código y a la detección temprana de errores. Estas pruebas se orientaron en probar cada funcionalidad de cada componente exceptuando el componente de framework, dado que este se probó a medida que se realizaban los servicios y el Standalone.

Asimismo, se realizaron pruebas de integración de cada uno de los componentes, sobre todo en los servicios en el momento de hacer la invocación a cada uno de ellos y al procesamiento de la 155 dado que en esta genera una invocación al servicio SOAP de procesamiento en línea.

El otro tipo de pruebas fueron las pruebas de validación, estas son realizadas por el equipo de calidad del proyecto, el cual se encarga de generar las pruebas de todo el desarrollo a través de un plan de pruebas que ellos crean, en este evalúan que cumpla con la funcionalidad descrita en los requerimiento, para la notificación de los errores encontrados es a través de una plataforma (mantis) donde en esta se crea un *ticket* por cada error encontrado, también se deja un documento donde está la evidencia la prueba realizada y cuál fue el resultado. Cabe resaltar que las pruebas se hicieron en ambiente de preproducción.

Las evidencias de las pruebas que se mostrarán a continuación en las figuras de la 31 a 34 son las desarrolladas por el equipo de calidad a la novedad 155, se indicará solo para esta novedad dado que todas las novedades se comportan de la misma manera, en cuanto a que deben pasar por un proceso de validación, procesamiento y notificación. Además, dentro de la planeación del cliente no se encuentran contempladas las pruebas de la novedad 032 por el equipo de calidad. Las demás evidencias no se agregaron por la cláusula de confidencialidad generada para el desarrollo de esta práctica.

En el caso de prueba número 1 que se presentó en la Figura 30, prueba que el componente de cargador no permite enviar archivos con el mismo nombre, esto se debe controlar para no sobrescribir archivos de respuesta, al igual que fue una regla de negocio descrita en el requerimiento.

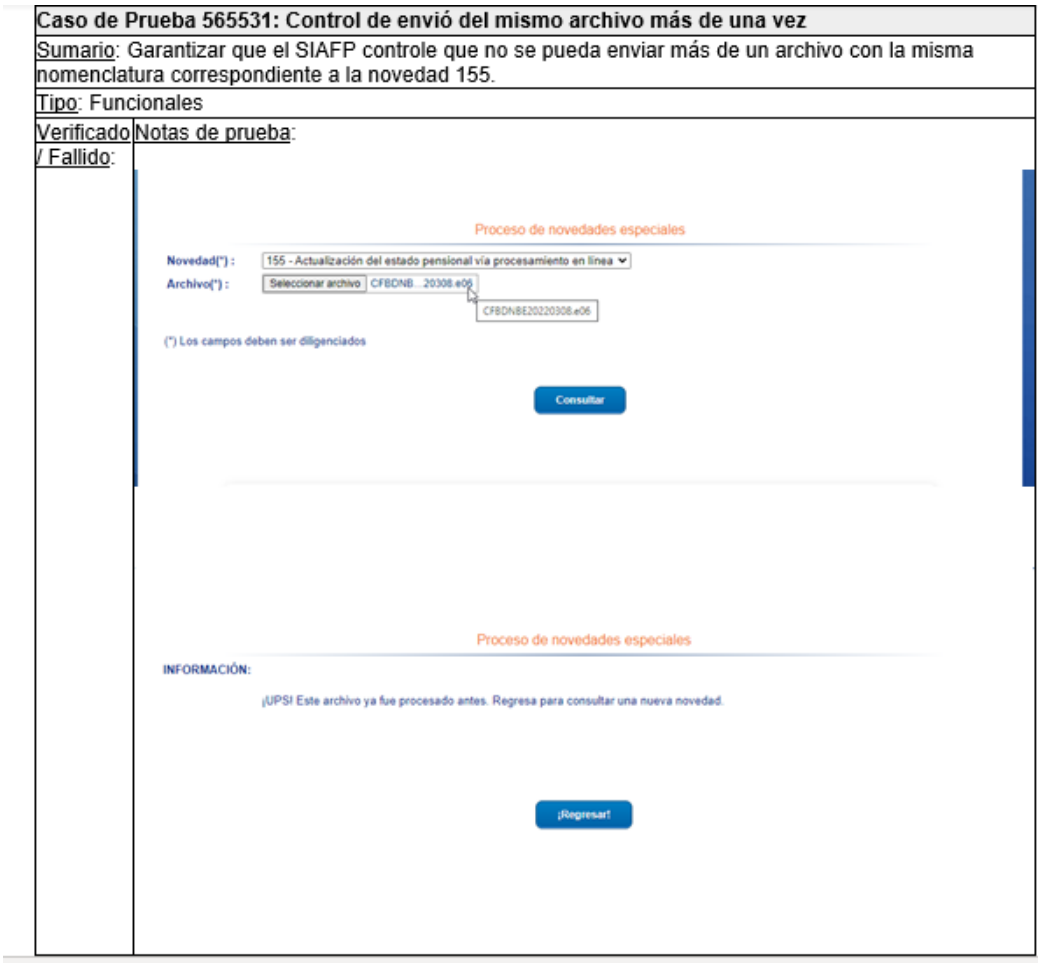


Figura 30. Caso de prueba numero 1 novedad 155.

En el caso de prueba número 2 que se muestra en la Figura 31, esta comprueba que la invocación a procesamiento en línea se está generando correctamente, dado que este servicio es el encargado de validar la obligatoriedad de cada campo sobre la trama, adicionalmente, valida que se esté generando el archivo de respuesta correctamente

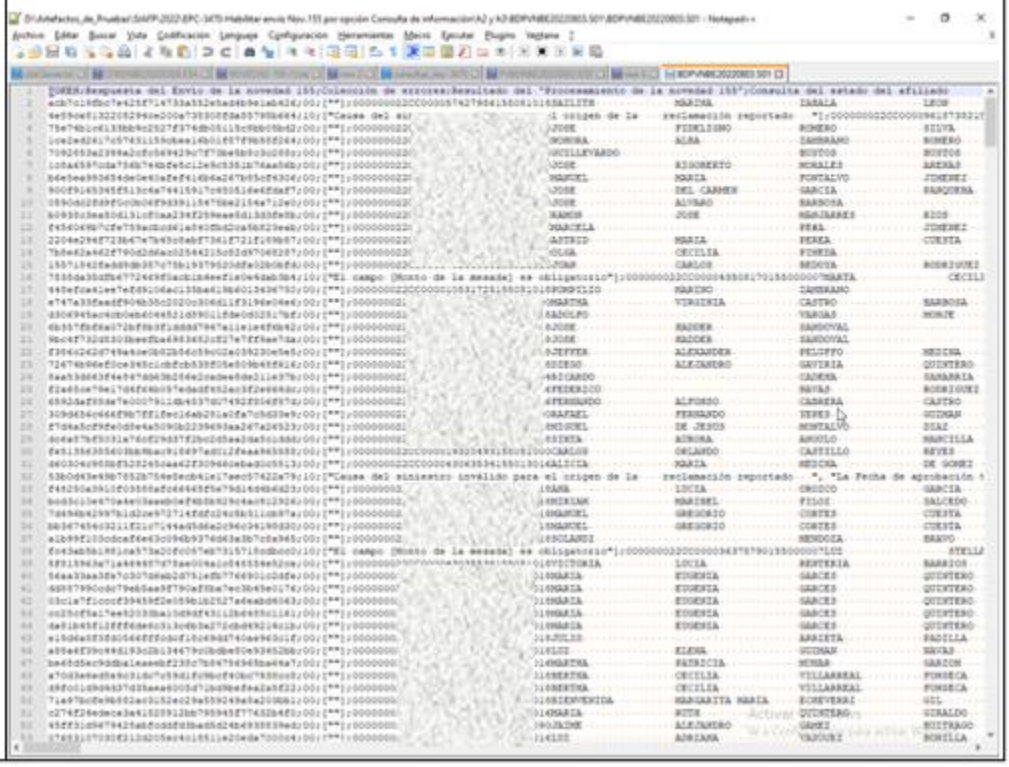
Caso de Prueba 565585: Validación exitosa de la novedad 155	
Sumario: Garantizar que el validador de fase 1 procesamiento especial por autoservicio, acepte y valide la obligatoriedad de los campos dependiendo de la información reportada.	
Tipo: Funcionales	
Verificado / Fallido:	Notas de prueba: 

Figura 31. Caso de prueba numero 2 novedad 155.

En el caso de prueba número 3 que se muestra en la Figura 32, prueba que los archivos se estaban generando con el encoding incorrecto, este proceso se corrigió y se pidió al equipo de pruebas volver a probar el caso.

.

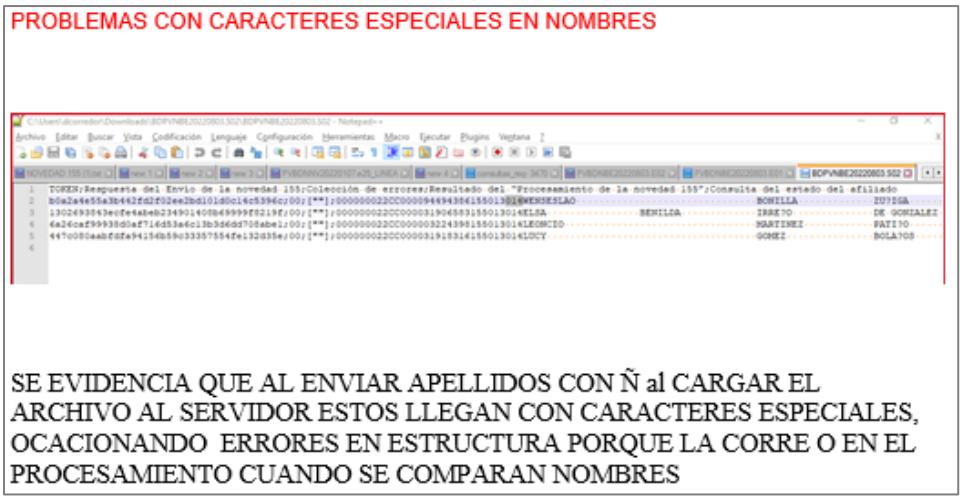


Figura 32. Caso de prueba numero 3 novedad 155.

Para el caso de prueba número 4, que se muestra en la Figura 33, se evalúa que el componente de procesamiento el cual está generando correctamente los archivos de respuesta al igual que componente de notificador realiza el envío de correos, de la misma manera al tener el archivo de error también se puede probar que los registros que no pasaron la validaciones del componente Validador.

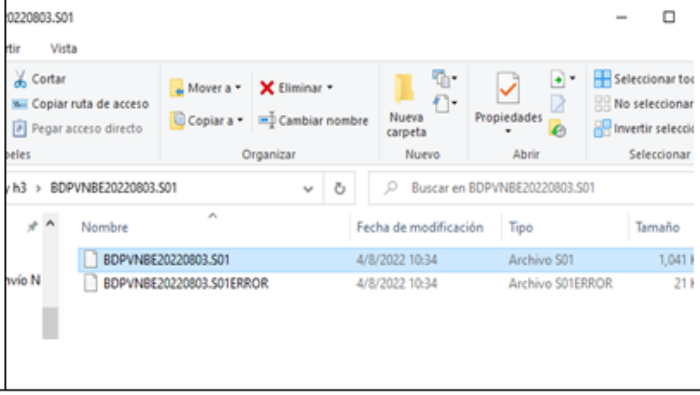
Caso de Prueba 565591: Archivo con el resultado de los afiliados enviados en archivo de novedad 155	
Sumario: Garantizar que por cada archivo de novedad 155 se genere un correo de su respectiva nomenclatura BDDDNBEAAAAMMDD.sCC, en dicha respuesta se incluyen los casos que se pudieron marcar exitosamente como los caso que presentaron problemas en las validaciones de entrada.	
Tipo: Funcionales	
Verificado / Notas de prueba: Fallido:	

Figura 33. Caso de prueba numero 4 novedad 155.

Al llevar a cabo esta pruebas para verificar el correcto funcionamiento de la novedad 155, donde muchos de los casos fueron exitosos y otros no, nos ayuda a asegurar la calidad del software al igual que el paso a producción sea limpio. Para la novedad 032 el equipo de calidad no genero pruebas, las únicas pruebas que se realizaron fueron las unitarias a partir de ellas se corrigieron errores como invocaciones a llamadas especificas a cada novedad en el validador, corrección de consultas de información básica del afiliado. A partir de esta novedad también se probó la base de datos con perfilamiento de wildfly en el servicio de validador.

5 Lecciones Aprendidas

Realizar una práctica profesional en una empresa de desarrollo de software, ayuda a desarrollar habilidades blandas como son: la comunicación asertiva, trabajo en equipo, resolución de problemas y el desarrollar confianza en tu trabajo dado que el mundo laboral es muy diferente a la academia. Adicionalmente se generan habilidades técnicas como el aprendizaje de nuevas tecnologías en este caso SOAP y MAVEN, en la cual se aplicaron muchos de los conceptos aprendidos en la academia como lo fueron: peticiones rest, arquitectura cliente servidor, diseño entidad relación, consultas SQL. Para poder desarrollar todas estas habilidades se debió pasar por un camino de errores para así aprender de ellos.

El acompañamiento del cliente es muy importante por esta razón se deben generar reuniones de acompañamiento para ir indicando el avance del aplicativo, para corroborar si se está cumpliendo con las necesidad descritas en cada uno de los requerimientos, al igual que tener una comunicación asertiva porque en la mayoría de los casos el cliente no tendrá conocimiento técnico de la aplicación. Una herramienta para esto son los diagramas de secuencia o los diagramas de flujo, ya que son una abstracción que ayuda a mostrarle al cliente el funcionamiento del aplicativo sin tener la necesidad de ir a lo técnico.

Se deben establecer planes de retoma de procesos, de esta manera se evita el re-proceso en los recursos de las máquinas, como lo son memoria, procesamiento, consultas a la base de datos. De esta manera se asegura la integridad de los datos y finalización de los procesos, esto se evidencio mediante la indexación de los registros de cada archivo de entrada permitiendo acceder rápidamente a los registros no procesados al momento que se presente una falla, ayudando al uso optimizado de los recursos.

El uso de propiedades ayuda a alterar el comportamiento de un software sin modificar su código fuente, lo cual es de gran ayuda porque de esta manera se le da flexibilidad al código de tal forma que un cambio no lo afecte, otra ventaja del uso de propiedades es que no es necesario recompilar el código para que se cambie de comportamiento, solo es necesario cambiar el valor de la propiedad.

El uso de arquetipos en los desarrollos serán de gran ayuda porque minimizan el tiempo de codificación, dado que estos patrones están compuestos por dependencias, configuraciones de despliegues, entre otros artefactos, provocando la reutilización de código.

El perfilamiento de las bases de datos ayuda a que sea transparente para los componentes la conexión a la base de datos, al igual que evita que los componentes tengan inconsistencias entre cada uno de los puntos de despliegue.

La generación de abstracciones ayuda a que los componentes sean genéricos de

tal manera que el código es reutilizable y mantenible evitando el acoplamiento, de esta manera la implementación de un nuevo proceso no deberá afectar a ninguno de los ya creados.

6 Conclusiones

Teniendo en cuenta que el beneficio pensional proporciona una seguridad financiera para la vejez, un ahorro a largo plazo, calidad de vida para todos los colombianos, la relevancia de la novedad 155 es tal, dado que a partir de ella se otorga el beneficio pensional. Igualmente, la novedad 032, que indica el estado actual de un afiliado, generando información como AFP actual a la que pertenece, su estado pensional o en qué estado esta su solicitud de pensión, historial de vinculaciones, entre otras información, es importante porque permite tomar decisiones como: si un afiliado cumple con un posible cambio de régimen o el cambio de un fondo de pensión, entre otras.

Por lo anterior, al incluir nuevos medios por los cuales se puedan procesar las novedades 155 y 032, el desarrollo de esta práctica permitió generar un nuevo medio por el cual se podrá enviar a procesar estas novedades, además apoyará a que, si alguna de las AFP no dispone de los canales que actualmente el SIAFP ofrece para generar el envío de estas novedades, pueda hacerlo por este nuevo canal.

Los requerimientos desarrollados en la práctica profesional se pudieron realizar completamente, dado que se alcanzaron todos los objetivos propuestos, crear este nuevo canal contribuye a que la oficina de la OGS no deba generar el procesamiento de la novedad 032 de forma manual, así mismo vuelve a la novedad 155 en un canal dedicado al procesamiento de la novedad 155. Cabe resaltar que este proceso no sirve para todas la novedades, solo para aquellas que cumplan con el proceso de validación procesamiento y notificación, aunque la gran mayoría en el SIAFP funcionan de esta manera.

La abstracción del procesamiento general en cada uno de los módulos ayudo a generar extensiones específicas de cada novedad, generando que el código sea extensible, adaptable y sobre todo mantenible

Durante el diseño e implementación de las funcionalidades de la aplicación se pudieron fortalecer algunas habilidades blandas necesarias para desempeñarse correctamente en el mundo empresarial, tales como: el manejo del estrés, gestión del tiempo, comunicación escrita y oral, adaptación al cambio, entre otras.

Por último, el acompañamiento del asesor de la empresa fue fundamental para poder llevar a cabo las funcionalidades requeridas por Asofondos. Al igual que la constante disponibilidad por parte de la directora del trabajo de grado y el buen ambiente laboral permitieron que la práctica profesional haya sido culminada exitosamente.

7 Bibliografía

[1] “Sistema Pensional - Asofondos.” https://asofondos.org.co/sistema-pensional/ (accessed Feb. 03, 2023).

[2] “Historia e Inicios SIAFP.mp4.” https://goo.su/aQvb9 (accessed Feb. 03, 2023).

[3] “¿Qué es Asofondos? - Asofondos.” https://asofondos.org.co/que-es-asofondos/ (accessed Dec. 06, 2022).

[4] “Validador - wikisiafp.” http://10.126.15.36/wikisiafp/index.php/Validador (accessed Dec. 06, 2022).

[5] “SIAFP Autorizador de trámite de pensión - wikisiafp.” http://10.126.15.36/wikisiafp/index.php/SIAFP_Autorizador_de_trámite_de_pensión (accessed Feb. 03, 2023).

[6] “Consultas masivas - wikisiafp.” https://siafp.heinsohn.com.co/wikisiafp/index.php/Consultas_masivas (accessed Feb. 16, 2023).

[7] “Llamada con Nestor Alejandro Salgado Arias-20230220_171108-Grabación de la reunión.mp4.” https://goo.su/a5PTdI (accessed Mar. 01, 2023).

[8] “Páginas - Sistema General de Pensiones.” https://www.minsalud.gov.co/proteccionsocial/RiesgosLaborales/Paginas/sistema-general-pensiones.aspx (accessed Feb. 03, 2023).

[9] “¿Qué es el RPM?” https://www.colpensiones.gov.co/pensiones/publicaciones/120/que-es-el-rpm/ (accessed Mar. 28, 2023).

[10] “Cuál Fondo de Pensiones en Colombia es el mejor y cómo elegirlo - Figuro.” https://figuro.la/fondo-de-pensiones-en-colombia/ (accessed Feb. 03, 2023).

[11] “Generalidades del SIAFP - wikisiafp.” http://10.126.15.36/wikisiafp/index.php/Generalidades_del_SIAFP (accessed Feb. 03, 2023).

[12] “Funcionamiento del SIAFP - wikisiafp.” http://10.126.15.36/wikisiafp/index.php/Funcionamiento_del_SIAFP (accessed Feb. 03, 2023).

[13] “Módulo de beneficios pensionales - wikisiafp.” http://10.126.15.36/wikisiafp/index.php/Módulo_de_beneficios_pensionales (accessed Feb. 03, 2023).

[14] “Experto – Introducción.” https://maven.apache.org/what-is-maven.html (accessed Jan. 23, 2023).

[15] “Maven – Introduction to the POM.” https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (accessed Mar. 09, 2023).

[16] “Maven – Introduction.” https://maven.apache.org/what-is-maven.html (accessed Feb. 04, 2024).

[17] “Simple y rápido. Entiende qué es Maven en menos de 10 min. - Javier Garzas.” https://www.javiergarzas.com/2014/06/maven-en-10-min.html (accessed Jan. 23, 2023).

[18] “Maven – Maven Features.” https://maven.apache.org/maven-features.html (accessed Mar. 10, 2023).

[19] “¿Por qué utilizar Arquetipos Maven?” https://www.enmilocalfunciona.io/por-que-utilizar-arquetipos-maven/ (accessed Jun. 12, 2023).

[20] M. C. Gallegos and V.-72 -, “FICA -EISIC Introducción a los Servicios Web,” pp. 72–97.

[21] J. P. S. H., E. L. F. C., and A. Recalde, “Análisis Comparativo entre los Estándares Orientados a Servicios Web SOAP, REST y GRAPHQL.,” *Rev. Antioqueña las Ciencias Comput.*, vol. 9, no. 2, pp. 10–22, 2019, doi: 10.5281/zenodo.3592004.

[22] S. E. N. Investigaci and E. N. Inform, “Una aproximación MDA para la conversión entre servicios web SOAP y RESTful ".

[23] C. Marcelo, S. RIVERO David, F. José, C. Daniel, C. Andrea, and V. Alejandro, “Aplicación de Servicios Web SOAP/REST para funcionalidades existentes en sistemas informáticos provinciales.”

[24] “Generalidades del protocolo HTTP - HTTP | MDN.” https://developer.mozilla.org/es/docs/Web/HTTP/Overview (accessed Jul. 31, 2023).

[25] “Simple Object Access Protocol (SOAP) 1.1.” https://www.w3.org/TR/2000/NOTE-SOAP-20000508/ (accessed Jan. 24, 2024).

[26] “XML Web Services.” https://www.w3schools.com/xml/xml_services.asp (accessed Jan. 24, 2024).

[27] “Arquitectura cliente servidor.” https://www.pispos.co/arquitectura-cliente-servidor (accessed Feb. 03, 2024).

[28] “WildFly: el servidor de aplicaciones de código abierto para desarrolladores Java EE. | ANW.” https://www.anw.es/servidores/wildfly-jboss.html (accessed Feb. 03, 2024).

[29] “About WildFly.” https://www.wildfly.org/about/ (accessed Feb. 03, 2024).

[30] “Qué es el procesamiento batch o por lotes.” https://www.profesionalreview.com/2018/11/25/que-es-el-procesamiento-batch/ (accessed Feb. 14, 2023).

[31] “¿Qué es el Procesamiento por lotes? - Guía sobre computación en la nube empresarial para principiantes - AWS.” https://aws.amazon.com/es/what-is/batch-processing/ (accessed Feb. 14, 2023).

[32] “SIAFP.” https://siafp.heinsohn.com.co/WEB-PortalAsofondos/ (accessed May 17, 2023).

[33] “Introducción a Struts: El controlador y las acciones.” http://www.jtech.ua.es/j2ee/publico/struts-2010-11/sesion01-struts-apuntes.html (accessed May 17, 2023).

[34] “Arquitectura basada en Componentes – Blog de Juan Peláez en Geeks.ms.” https://geeks.ms/jkpelaez/2009/04/18/arquitectura-basada-en-componentes/ (accessed Jun. 12, 2023).

[35] I. De Mysql, “Las 10 razones principales para usar MySQL como base de datos integrada,” 2012.

[36] “Consulta Web service - wikisiafp.” https://siafp.heinsohn.com.co/wikisiafp/index.php/Consulta_Web_service (accessed Feb. 01, 2024).

[37] “SIAFP como autorizador de pensión - Wiki-SIAFP.” http://197.0.3.19:8080/mediawiki-1.26.3/index.php/SIAFP_como_autorizador_de_pensión#PDF (accessed Feb. 01, 2024).

[38] “Novedad 155 Línea - wikisiafp.” https://siafp.heinsohn.com.co/wikisiafp/index.php/Novedad_155_Línea (accessed Feb. 01, 2024).

[39] “Caracterizacion 032 y 155.mp4.” https://hbt-https://goo.su/UZZB5Y (accessed Feb. 01, 2024).

[40] P. en castellano, “Programación en castellano. Sistema de Nombrado en Java (JNDI) [Parte I].”