UNIVERSIDAD PRIVADA ANTENOR ORREGO

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA DE COMPUTACIÓN Y SISTEMAS

“DESARROLLO DE UN MOTOR JAVA DE COBRANZAS Y OFERTAS DE PRESTAMO, Y UNA APLICACIÓN WEB PARA VISUALIZAR EL DETALLE DE LAS MISMAS”

INFORME TÉCNICO PARA OBTENER EL TÍTULO PROFESIONAL DE INGENIERO DE COMPUTACIÓN Y SISTEMAS

MODALIDAD: SUFICIENCIA PROFESIONAL

FORMA: EXPERIENCIA LABORAL CALIFICADA

LINEA DE INVESTIGACION: TECNOLOGÍAS Y PROCEDIMIENTOS DE PROGRAMACIÓN JAVA PARA DESARROLLO DE SISTEMAS.

AUTOR: Br. Mario Luis Vásquez Maldonado

ASESOR: Ing. Karla Vanessa Meléndez Revilla

LIMA – PERÚ 2020

FECHA DE SUSTENTACIÓN: 13/08/2020

ACREDITACIONES

TÍTULO: “DESARROLLO DE UN MOTOR JAVA DE COBRANZAS Y OFERTAS DE PRESTAMO, Y UNA APLICACIÓN WEB PARA VISUALIZAR EL DETALLE DE LAS MISMAS”

AUTOR:

_____

Br. Mario Luis, Vásquez Maldonado

APROBADO POR:

_____

Dr. Luis Vladimir Urrelo Huiman

PRESIDENTE

Nº CIP: 88212

_____

Ing. Freddy Henrry Infantes Quiroz

SECRETARIO

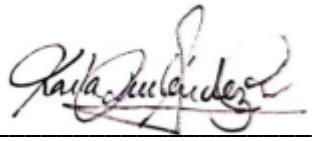
Nº CIP: 139578

_____

Ing. Jose Arturo Castañeda Saldaña

VOCAL

N° CIP: 148099



_____

Ing. Karla Vanessa Melendez Revilla

ASESOR

N° CIP: 120097

PRESENTACIÓN

Señores Miembros del Jurado:

Dando cumplimiento y conforme a las normas establecidas en el Reglamento de Grados y Títulos y Reglamento de la Facultad de Ingeniería de la Universidad Privada Antenor Orrego, para obtener el título profesional de Ingeniero de Computación y Sistemas, se pone a vuestra consideración el Informe del Trabajo de Suficiencia Profesional Titulado "DESARROLLO DE UN MOTOR JAVA DE COBRANZAS Y OFERTAS DE PRESTAMO, Y UNA APLICACIÓN WEB PARA VISUALIZAR EL DETALLE DE LAS MISMAS".

Lima, 17 de febrero de 2020.

Br. Mario Luis Vásquez Maldonado

DEDICATORIA Y AGRADECIMIENTO

Agradecemos a Dios por el regalo de la vida.

A mis padres, Mario y Selmit, por su apoyo incondicional, persistencia y por nunca dejar de creer en mí.

A cada uno de mis docentes por todos sus conocimientos brindados a lo largo de mi preparación universitaria.

Mario Luis Vásquez Maldonado

RESUMEN EJECUTIVO

Atento desde hace mucho tiempo se erige como la empresa líder en América Latina y España dentro del sector BPO (Business Process Outsourcing) y CEM (Customer Experience Management). En Perú comenzó sus operaciones desde el año 1999.

En las últimas décadas han surgido pequeñas empresas dedicas al rubro del Call Center, ingresan al mercado compitiendo directamente con ATENTO ofreciendo en líneas generales el mismo servicio a un costo mucho más bajo. ATENTO sigue siendo la empresa más grande y líder del sector pero va perdiendo terreno, es decir, algunos de sus clientes habituales –principalmente del sector bancario- con los cuales había mantenido sociedad durante muchos años, han ido confiando y mudando parcialmente o la totalidad de su ***servicio de atención al cliente vía telefónica,*** a plataformas tecnológicas de otras empresas.

ATENTO, con el objetivo de fidelizar a sus clientes del sector bancario, planea lanzar un nuevo producto que realice la gestión de cobranza preventivas. Hasta ese entonces, y según el estudio estratégico realizado por la unidad de negocios de la empresa, dicho servicio – el de gestión de cobranza preventiva- únicamente lo tenían implementado las entidades bancarias dentro de su propia unidad de tecnología, mas no como un servicio tercerizado.

Atento actualmente y desde el 2017 viene atravesando una crisis organizacional y económica que se traduce en constantes despidos de personal; particularmente la jefatura de soluciones, específicamente el área de desarrollo y mantenimiento de software, hasta el año 2018 fue reducida casi a un 30% de su capacidad operativa bajo la premisa: “ATENTO es una empresa dedicada al OUTSOURCING y EL CUSTOMER EXPERIENCE MAGENEMENT, no al desarrollo de sistemas”. En el desarrollo de este sistema, como en la mayoría de desarrollos “in house” concebidos dentro de ATENTO, se presentó un gran número de limitaciones y dudas las cuales se iban aclarando conforme avanzaba el desarrollo, todo esto a causa de un inexistente documento funcional que detalle los requerimientos y funcionalidades que el sistema debería cumplir; esto a nivel a funcional y de arquitectura. El tiempo estimado fue muy reducido así como la cantidad de desarrolladores involucrados en el proyecto; sólo un desarrollador para un sistema que se supone debía estar operativo en 6 semanas.

El informe incluye en el primer capítulo el análisis del contexto y la experiencia del autor. El marco general del proyecto se desarrolla en el segundo capítulo así como los objetivos planteados y las tecnologías utilizadas. En el tercer capítulo se presenta el desarrollo del proyecto mediante un análisis en retrospectiva evidenciando la experiencia y los resultados alcanzados.

ABSTRACT

ATENTO for a long time it stands as the leading company in Latin America and Spain within the BPO (Business Process Outsourcing) and CEM (Customer Experience Management) sectors. In Peru it began operations since 1999.

In recent decades, small clinic companies have emerged under the Call Center, entering the market directly competing with ATENTO connected in general the same service at a much lower cost. ATENTO is still the largest and leading company in the sector but it is losing ground, that is, some of its regular clients - mainly in the banking sector - with whom they had had a partnership for many years, have been trusting and moving certain or all from its customer service via telephone, to technological platforms of other companies.

ATENTO, with the aim of loyalty to its customers in the banking sector, plans to launch a new product that performs preventive collection management. Until then, and according to the strategic study conducted by the business unit of the company, said service - that of preventive collection management - the specific thing implemented by banking entities within their own technology unit, but not as a third-party service.

ATENTO now and since 2017 has been going through an organizational and economic crisis that translates into constant layoffs; specifically the solutions management, specifically the area of software development and maintenance, until 2018, almost 30% of its operational capacity was reduced under the premise: “ATENTO is a company dedicated to OUTSOURCING and EL CUSTOMER EXPERIENCE MAGENEMENT, not to systems development. "In the development of this system, as in most" home "developments conceived within ATENTO, a large number of limitations and doubts are shown, which were clarified as development progresses, all this because of a non-existent functional document that details the requirements and functionalities that the system should meet; this at the functional and architectural level. The estimated time was greatly reduced as well as the number of developers involved in the project; only a program for a system that It means having a functioning operation in 6 weeks.

The report includes in the first chapter the analysis of the context and the experience of the author. The general framework of the project is developed in the second chapter as well as the objectives raised, and the technologies used. In the third chapter the development of the project is presented through a retrospective analysis evidencing the experience and the results achieved.

Tabla de Contenido

Tabla de Ilustraciones

1. CONTEXTO Y DESCRIPCIÓN DE LA EMPRESA

El autor desarrollo el proyecto de estudio en el que se basa este informe en la empresa ATENTO PERÚ durante el periodo comprendido entre inicio y finales de marzo del 2019. En ATENTO PERÚ me desempeñe como ANALISTA DE SISTEMAS en la JEFATURA DE SOLUCIONES. Actualmente me encuentro trabajando en la empresa GLOBAL HITSS perteneciente al GRUPO AMERICA MÓVIL, me desempeño como ANALISTA PROGRAMADOR dedicado íntegramente al desarrollo de nuevas funcionalidades en el ámbito de las aplicaciones de CLARO ESTADOS UNIDOS y TELMEX ESTADOS UNIDOS.

A continuación, paso a describir mi trayectoria profesional desde mi primer trabajo luego de graduarme de la universidad hasta la actualidad:

1.1 CONSULTORÍA Y ASESORIA EN TECNOLOGÍA CONASTEC.-

Esta sección de experiencia laboral corresponde al proyecto de adaptación y modificación del módulo de ventas y producción del ERP "ODOO", de código fuente abierto, elaborado en Python e inspirado a nivel de arquitectura en JAVA SERVER FACES. El proyecto requirió aprender rápidamente Python para poder realizar las modificaciones solicitadas en los módulos señalados. La parte más complicada y que significo un duro reto para el equipo de desarrolladores fue la funcionalidad de impresión de comprobantes desde el lado del cliente (Navegador Web), para lo cual se desarrolló un Servidor WebSocket Java que corría mediante un batchero en cada una de las computadoras con una impresora instalada; el servidor websocket se mantenía a la escucha de cualquier petición originada desde el navegador web, recibía como parámetro la cadena a imprimir y el nombre de la impresora o ticketera.

- Cargo.- Analista Programador Junior
- Periodo.- Noviembre 2015 – Diciembre 2015
- Funciones.-
  - Programación y mantenimiento de módulos de OPENERP V.7 – ODOO V.8. implementado para el cliente "Corporación Antilla".
  - Desarrollo de componente de impresión a impresoras matriciales.

1.2 CONSULTORÍA Y ASESORIA EN TECNOLOGÍA CONASTEC.-

Esta sección de experiencia laboral corresponde al proyecto de facturación electrónica emprendida por la empresa CONASTEC, cuya finalidad tenía aprobar satisfactoriamente el proceso de homologación SUNAT con el objetivo de convertirse en un PROVEEDOR DE SERVICIOS ELECTRÓNICOS. El proyecto involucro tecnologías como SPRING para el control de transacciones y conexiones a base de datos, manejo de Jobs mediante SPRING SCHEDULER, gestión de beans, SPRING AOP para añadir nuevas funcionalidades como validaciones de permisos o restricciones de acceso por usuarios a ciertos métodos de la capa de servicio; HIBERNATE como ORM para un fácil acceso y modificación de los objetos de la base de datos; JAVA SERVER FACES para un desarrollo rápido de vistas así como consumo y exposición de servicios SOAP.

- Cargo.- Analista Programador Semi Senior
- Periodo.- Diciembre 2015 – Abril 2017
- Funciones.-
  - Miembro del equipo de desarrolladores del Sistema de Facturación Electrónica EBIS certificado por SUNAT, desarrollado con las tecnologías de HIBERNATE, SPRING y JSF.
  - Implementación de funcionalidades del módulo de emisión de comprobantes, y generación de UBL de comprobantes electrónicos enviados a SUNAT vía WEB SERVICE SOAP.
  - Implementación del validador XSL de SUNAT para los UBL de comprobantes electrónicos.
  - Mejora, mantenimiento y estabilización del Sistema EBIS.
  - Elaboración de documento funcional y manual de usuario.

1.3 ATENTO PERÚ.-

Esta sección de experiencia laboral corresponde al cargo de ANALISTA DE SISTEMAS, el cual consistía básicamente en atender las solicitudes de los clientes internos (un cliente interno en atento viene a ser una unidad de negocio asignada a un cliente externo, ya sea un banco, por ejemplo) e identificar los requerimientos y proponer la solución más adecuada. El corazón tecnológico usado para gestionar las llamadas telefónicas por los OPERADORES u ASESORES es el MULTIGESTIÓN, sistema desarrollador en JAVA SWING bajo la arquitectura modular OSGI.

- Cargo.- Analista de Sistemas
- Periodo.- Abril 2017 – Septiembre 2019
- Funciones.-
  - Implementación de sistemas web de envío de ofertas y realización de cobranzas vía SMS,
  - WHATSAPP, EMAIL y LLAMADAS TELEFONICAS; así como también un motor interno MULTITHREAD de envíos de promociones de ventas y cobranzas preventivas.
  - Mejora e implementación de nuevas funcionalidades del SISTEMA MULTI GESTIÓN DE ATENTO desarrollado en Java Swing.
  - Programación en TRANSACT SQL.
  - Migración de Oracle a SQLSERVER.
  - Implementación de ETL en SSIS (SQL SERVER INTEGRATION SERVICES).

1.4 GLOBAL HITSS & AMERICA MÓVIL.-

GLOBAL HITSS pertenece al grupo AMERICA MÓVIL, es una empresa que funciona como una consultora de software específicamente como una "fábrica de sotfware" cuyo esfuerzo en su totalidad está destinado a prestar servicios a empresas pertenecientes al grupo AMERICA MÓVIL, ya sea CLARO, TELMEX, EMBRATEL entre otras. Mi trabajo consiste en desarrollar los requerimientos plasmados dentro de un documento funcional previamente elaborado por un ANALISTA FUNCIONAL en ESTADOS UNIDOS; también me encargo de hacer seguimiento y dar solución a aquellas incidencias críticas que en primera instancia no pudieron ser resueltas por el equipo de soporte tecnológico.

- Cargo.- Analista Programador
- Periodo.- Septiembre 2019 - Actualidad
- Funciones.-
  - Desarrollo de nuevas funcionalidades en los aplicativos de ventas y gestión de requerimientos de CLARO USA y TELMEX USA.
  - Soporte y resolución de incidencias relaciones a los aplicativos de CLARO USA y Telmex USA.
  - Adiestramiento y capacitación en arquitectura SPRING, HIBERNATE Y JAVA SERVER FACES.

2. INFORMACIÓN GENERAL DEL PROYECTO

2.1 NOMBRE DEL PROYECTO.-

DESARROLLO DE UN MOTOR JAVA DE COBRANZAS Y OFERTAS DE PRESTAMO, Y UNA APLICACIÓN WEB PARA VISUALIZAR EL DETALLE DE LAS MISMAS.

2.2 OBJETIVO Y OBJETO DE ESTUDIO.-

Para el proyecto "DESARROLLO DE UN MOTOR JAVA DE COBRANZAS Y OFERTAS DE PRESTAMO, Y UNA APLICACIÓN WEB PARA VISUALIZAR EL DETALLE DE LAS MISMAS" –que de ahora en adelante llamaré MOTOR DE COBRANZAS, como cualquier otra implementación desarrollado en la JEFATURA DE SOLUCIONES DE ATENTO, no se ciñó a una metodología de desarrollo de software estándar, por lo que el objetivo de este informe no es la de recalcar las etapas, buenas prácticas o roles de una marco de trabajo como SCRUM –por citar alguno- aplicado al desarrollo de software.
El objetivo principal de estudio, es la de rescatar las virtudes de la programación modular OSGI aplicado al marco de desarrollo del moto de cobranza, al mismo tiempo que describiré la potente integración de tecnologías como HIBERNATE , SPRING, y JAVA SERVER FACES aplicados al marco de desarrollo de la parte WEB del motor de cobranzas; ambos marcos de trabajo respetan patrones de arquitectura de software como el MVC y estándares de codificación, aprendidos en la universidad y en mi primer empleo(una consultora de software). Como objetivo secundario, a manera de crítica constructiva, se describirá brevemente la estructura de una aplicación de gestión que representa el corazón del cuerpo de negocio de ATENTO PERÚ, la misma que fue desarrollada sin respetar patrones de arquitectura de software ni estándares básicos al momento de escribir código java, aplicación con la que tuve que lidiar casi a diario durante más de dos años.

2.3 DESCRIPCIÓN DEL PROYECTO

A nivel de estrategia de negocio, ATENTO PERÚ tiene como objetivo principal concebir un servicio para la gestión automatizada de cobranzas multicanal: vía telefónica, mensaje de texto, correo electrónico y whatsapp, que permita liberar a sus clientes del sector banca de la tarea de cobranza preventiva, logrando así mantener el rol de socio principal con sus clientes del sector bancario en cuanto a la gestión de relación con clientes.

Esta iniciativa de negocio concebida dentro de la unidad comercial dedicada a CENCOSUD CHILE, se traduce en cuanto a software en dos componentes:

2.3.1 Un pequeño SERVIDOR JAVA o llamado también MOTOR DE ENVÍOS.

Este componente permite la **administración de tareas para el envío de información** de deudas o cobranzas u promociones de ventas, vía telefónica, mensaje de texto, correo electrónico y mensajería WHATSAPP; para el caso de los canales de comunicación vía mensaje de texto, correo electrónico o

WHATSAPP, el cuerpo de la información remitida al cliente contendrá un link a una página web donde se podrá visualizar el detalle de la información.

Cada tarea a su vez se traduce en código como un hilo en java, esto quiere decir que, para el desarrollo del **MOTOR DE ENVÍOS**, se requirió un avanzado conocimiento de **MULTITHREAD en JAVA**. Así mismo, cada tipo de iteración con el cliente, ya sea: LLAMADA TELEFÓNICA, MENSAJE DE TEXTO, CORREO ELECTRÓNICO O MENSAJERÍA WHATSAPP se traduce en código como un módulo o **BUNDLE OSGI**. El proceso que alimenta de información al **MOTOR DE ENVÍOS** es un paquete **SSIS (SQL SERVER INTEGRATION SERVICES).**

2.3.2 Una aplicación web o también llamado LANDING PAGES DE ENVÍOS.

Este componente permite la visualización de la información remitida al cliente por el motor de envíos, información enviada ya sea vía mensaje de texto, correo electrónico o mensajería WHATSAPP; a su vez permite registrar al usuario una promesa de pago. La aplicación web fue desarrollada usando las tecnologías **SPRING y JAVA SERVER FACES,** siguiendo siempre el patrón de diseño RESPONSIVE para su correcta visualización en dispositivos móviles.

2.4 MARCO CONCEPTUAL QUE DA SUSTENTO AL OBJETO DE ESTUDIO.

El amplio sector de clientes potencialmente morosos de CENCOSUD, conlleva a que la información de clientes proporcionada por CENCOSUD mediante un archivo de carga Excel, sea numerosa y pesada, motivo por el cual se aprovechó la potencia y eficiencia –ampliamente utilizada y comprobada en los procesos de carga de ATENTO- que ofrecen las herramientas de **SQL SERVER INTEGRATION SERVICES** al momento de procesar e insertar gran cantidad de datos a una base de datos.

Se eligió el **aproximamiento modular que ofrece OSGI** como único FRAMEWORK base para la implementación del **MOTOR DE ENVÍOS**, porque la arquitectura de este sistema debía soportar tanto un modelo de gestión automática de cobranzas, así como también uno de ventas, y cada modelo de gestión a su vez, de acuerdo a los requerimientos y necesidades del cliente contratante, debía poder configurarse o adaptarse a una o más canales de comunicación(vía telefónica, mensaje de texto, correo electrónico o mensajería WHATSAPP) para la realización de las cobranzas o promoción de ofertas.

Así mismo, el sistema realiza una eficiente gestión de hilos en java. Pero, **¿Por qué un sistema MULTITHREAD?**, según los requerimientos solicitados por CENCONSUD CHILE, la gestión de cobranza debía iniciarse vía telefónica donde un "robot o bot" tipo **IVR** (**INTERACTIVE VOICE RESPONSE**) -diseñado por el equipo especializado de **PLATAFORMA ATENTO**-, en caso el cliente no respondiera la llamada o no se obtuviera un compromiso de pago, el sistema debía ser capaz de identificar aquellas gestiones telefónicas que no obtuvieron éxito, para luego realizar un segundo intento de cobranza mediante **MENSAJE DE TEXTO** en cuyo cuerpo se encontraría un link a una aplicación web donde el potencial moroso pudiera visualizar el detalle de sus deudas. En caso no se obtuviera respuesta mediante la cobranza por Mensaje de Texto, el sistema debía realizar un tercer intento de cobranzas mediante **MENSAJERÍA WHATSAPP**, para

finalmente pasar a la cobranza vía correo electrónico para aquellas gestiones fallidas vía mensajería WHATSAPP. Es evidente que el motor de envíos debía contar con una arquitectura MULTITHREAD que soporte uno o N pool de hilos, cada pool de hilos a su vez, estaría compuesto de uno o más canales de comunicación **(LLAMADA TELEFÓNICA**, **MENSAJE DE TEXTO**, **CORREO ELECTRÓNICO** O **MENSAJERÍA WHATSAPP)**; la cantidad de pool de hilos corriendo en el servidor seria directamente proporcional a la cantidad de clientes de ATENTO que solicitaran el servicio. Así mismo, el motor de envíos debería permitir configurar – **mediante su interfaz SWING**- el orden de inicio de los hilos o canales de comunicación y la manera cómo estos interactuarían.

Para la **aplicación web** que permite visualizar el detalle de la cobranza o promoción de venta, se eligió implementarla tomando como base los FRAMEWORKS **SPRING y JAVA SERVER FACES**; FRAMEWORKS con los cuales ya había trabajado anteriormente y a los cuales conocía muy bien. Además, dado el corto tiempo que se tenía previsto para el desarrollo del aplicativo web, y aprovechando la existencia de un **servidor web TOMCAT** propiedad de ATENTO, mi mejor elección para realizar una aplicación web de manera rápida y aprovechase la ventaja de tener un servidor java, fue definitivamente **JAVA SERVER FACES**. Finalmente debo destacar las siguientes características de SPRING y JAVA SERVER FACES, que combinadas se logra un potente, robusto y, sobre todo, rápido marco de trabajo al momento de desarrollado una aplicación web:

2.4.1 ¿QUÉ ES OSGI?

OSGi es una de las tecnologías de software distribuido que más impacto está causando últimamente. Prueba de ello es que proyectos open source tan populares como Eclipse o la framework de aplicaciones empresariales Java EE Spring han adoptado esta tecnología como base de sus productos. La razón principal para ello es que OSGi define una infraestructura extremadamente eficiente para el desarrollo de aplicaciones basadas en servicios (SOA) dentro de una máquina virtual Java (JVM). Su principal misión es reducir la complejidad de construir, mantener, desplegar y gestionar el ciclo de vida de aplicaciones en cualquier tipo de dispositivo computacional. La plataforma de servicios OSGi es una tecnología que aporta modularidad dinámica a Java y responde a la necesidad de estandarizar la integración de software. Mediante OSGi, Java se convierte en un entorno idóneo para afrontar tanto la integración como el desarrollo de software. Mientras Java proporciona la portabilidad requerida para soportar productos en diferentes plataformas, OSGi facilita las primitivas estándar para construir aplicaciones a partir de pequeños componentes reutilizables y cooperativos. Para OSGi, un componente software es una librería o aplicación que puede ser dinámicamente descubierta y usada por otros componentes.

La incorporación de la tecnología OSGi a un dispositivo en red (bien empotrado o en forma de servidor) le aporta la capacidad de gestionar el ciclo de vida de sus componentes de software y permite la composición dinámica de aplicaciones dentro de él desde cualquier otro punto de la red. Gracias a OSGi, los componentes software pueden ser instalados, actualizados o eliminados sobre la marcha, sin tener que interrumpir la operación del dispositivo o

servidor. Además, OSGi añade únicamente una ligera capa de software que es adecuada incluso para su despliegue y uso en la gestión de componentes software en dispositivos empotrados. Eso explica que este estándar no solamente prometa como una gran tecnología para mejorar los servidores de aplicaciones Java actuales, sino que sea actualmente utilizada en sectores tan dispares como la automoción, domótica o los dispositivos móviles. Los atributos de su elegante especificación lo han hecho atractivo y aplicable a diversos mercados. Por ejemplo, Nokia y Motorota han propuesto una nueva arquitectura software basada en OSGi para los teléfonos inteligentes de la próxima generación. La industria del automóvil ha adoptado también OSGi. Esta plataforma de servicios es parte integral de la plataforma de comunicaciones de BMW en la serie 5. Incluso algunos productos de electrónica como Philips iPronto, Nokia N800 y E70, la unidad de almacenamiento en red Linksys NSLU2 empiezan a incorporar OSGi

Desafortunadamente, OSGi es una tecnología que a pesar de existir desde 1999 no ha contado con gran popularidad entre los desarrolladores de sistemas distribuidos y aplicaciones web basadas en Java hasta hace poco. Sin embargo, esto está cambiando radicalmente en los últimos meses debido a que son muchos los expertos que se han dado cuenta de la idoneidad de OSGi para constituir la base de las próximas versiones de servidores de aplicaciones Java. Prueba de este interés es que frameworks Java como Eclipse y Spring ya están basadas en él, mientras que Apache e IBM ya han preparado también versiones de sus servidores de aplicaciones Java EE (Felix y WebSphere, respectivamente) siguiendo la filosofía OSGi y otros como JBoss planean hacerlo a corto plazo.

2.4.2 ¿QUÉ ES UN BUNDLE EN OSGI?

Un bundle en OSGI en un JAR y representa un módulo, plugin o unidad de implementación independiente dentro de una aplicación. Los bundles se despliegan y pueden convivir dentro de un contenedor OSGI tal como se muestra en la siguiente imagen:

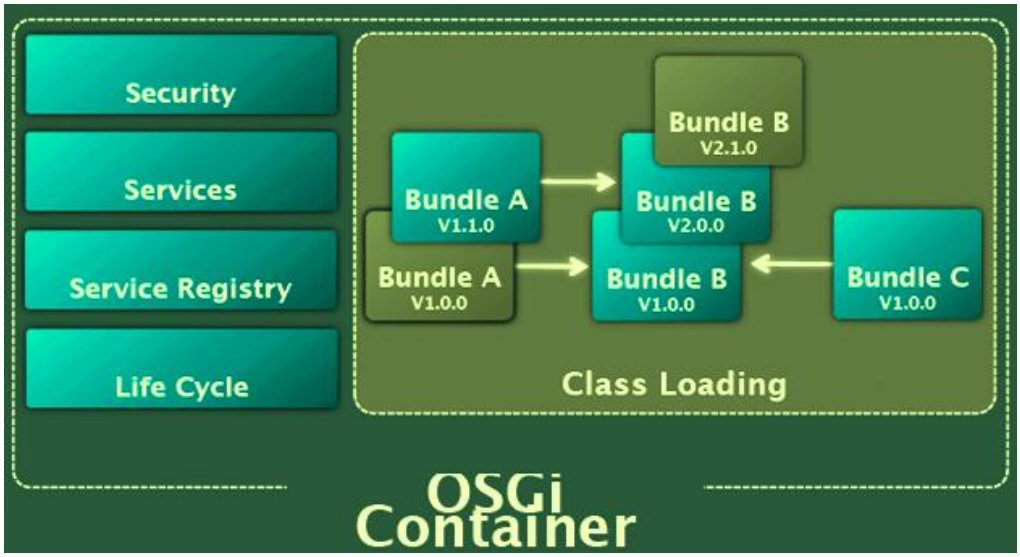


Figura 1. CONTENEDOR OSGI

Fuente: (TirthalPatel Osgi, 2014)

Un bundle contiende dentro del jar un archivo MANFIEST.MF. Este archivo MANIFIEST contiene metadatos que permiten que OSGi Framework procese los aspectos modulares del paquete. El siguiente código es un ejemplo del contenido de un archivo de manifiesto de paquete, META-INF / MANIFEST.MF:

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: MyService bundle
Bundle-SymbolicName: com.sample.myservice
Bundle-Version: 1.0.0
Bundle-Activator: com.sample.myservice.Activator
Import-Package: org.apache.commons.logging;version="1.0.4"
Export-EJB: ExampleBean
Export-Package: com.sample.myservice.api;version="1.0.0"
```

Figura 2 MANIFIEST.MF BUNDLE OSGI

Fuente: (IBM knowledgecenter, 2020)

Los metadatos en este archivo de manifiesto incluyen las siguientes propiedades clave:

- Bundle-Version.- Describe la versión del paquete y permite que varias versiones de un paquete se activen simultáneamente en la misma instancia de marco.
- Bundle-Name.- Proporciona un nombre legible para un paquete.
- Bundle-SymbolicName.- Identifica de forma exclusiva un paquete en el marco. No reemplaza la necesidad de un encabezado Bundle-Name.
- Bundle-Activator.- Recibe notificaciones del marco sobre los cambios del ciclo de vida del paquete. Esta propiedad especifica la clase que implementa la interfaz org.osgi.framework.BundleActivator.
- Import-Package.- Declara las dependencias externas del paquete que usa OSGi Framework para la resolución del paquete. Se pueden declarar versiones específicas o rangos de versiones para cada paquete. En este archivo de manifiesto de ejemplo, se requiere el paquete org.apache.commons.logging en la Versión 1.0.4 o posterior. Se usa esta propiedad para especificar los nombres de los paquetes que desea que su paquete importe desde el tiempo de ejecución. Si no especifica el paquete que su paquete necesita en esta propiedad, puede obtener un error de compilación cuando se carga el paquete.

2.4.3 CICLO DE VIDA DE UN BUNDLE.-

El contenedor OSGI gestiona el ciclo de vida de los paquetes. A medida que instala y ejecuta un paquete, pasa por varios estados.

Los posibles estados de un paquete son:

- INSTALLED.- El paquete se ha instalado, pero no se han cumplido todas las dependencias del paquete. El paquete requiere paquetes que no hayan sido exportados por ningún paquete instalado actualmente.

- RESOLVED.- El paquete está instalado y las dependencias del paquete se han cumplido, pero no se está ejecutando. Si se inicia un paquete y se cumplen todas las dependencias del paquete, el paquete omite este estado.

- STARTING.- Un estado temporal por el que pasa el paquete mientras se inicia el paquete.

- ACTIVE.- El paquete se está ejecutando.

- STOPPING.- Un estado temporal por el que pasa el paquete mientras se detiene el paquete.

- UNINSTALLED.- El paquete ya no existe en el marco.

2.4.4 SERVICIOS OSGI.-

En el marco OSGi, los BUNDLES se construyen alrededor de un conjunto de servicios cooperativos disponibles desde un registro de servicios compartidos. Tal servicio OSGi se define semánticamente por su interfaz de servicio y se implementa como un objeto de servicio.

El objeto de servicio es propiedad y se ejecuta dentro de un BUNDLE. Este BUNDLE debe registrar el objeto de servicio con el registro del servicio Framework para que la funcionalidad del servicio esté disponible para otros paquetes bajo el control de contenedor OSGI.

Las dependencias entre el BUNDLE que posee el servicio y los BUNDLE que lo usan son administradas por el contenedor. Por ejemplo, cuando se detiene un BUNDLE, todos los servicios registrados en el Framework por ese BUNDLE pasan a estado **UNREGISTERED** automáticamente.

El contenedor asigna servicios a sus objetos de servicio subyacentes y proporciona un mecanismo de consulta simple pero potente que permite que un BUNDLE solicite los servicios que necesita. El contenedor OSGI también proporciona un mecanismo de eventos para que los BUNDLE puedan recibir eventos de servicios que están registrados, modificados o no registrados.
Para publicar un servicio dentro de un bundle se puede usar:

```
BundleContext bundleContext =
FrameworkUtil.getBundle(this.getClass()).getBundleContext();
bundleContext.registerService(IMyService.class.getName(), new
MzServiceImpl(), null);
```

Figura 3 PUBLICACIÓN DE UN SERVICIO OSGI

Fuente: (Vogella Osgi Tutorial, 2016)

2.4.5 UTILIZACIÓN DE UN SERVICIO OSGI.-

Para utilizar un objeto de servicio y llamar a sus métodos, un paquete primero debe obtener un objeto ServiceReference. La interfaz BundleContext define una serie de métodos que un paquete puede llamar para obtener objetos ServiceReference del Framework:

- getServiceReference (String), getServiceReference (Class): estos métodos devuelven un objeto ServiceReference a un objeto de servicio que implementa y se registró con el nombre de la interfaz de servicio especificada. Si existen varios de estos objetos de servicio, se devuelve un objeto de referencia de servicio al objeto de servicio con el mayor SERVICIO_RANKING. Si hay un empate en la clasificación, se devuelve un objeto ServiceReference al objeto de servicio con el

SERVICE_ID más bajo (el objeto de servicio que se registró primero). Si no se registran objetos de servicio coincidentes, se debe devolver nulo.

- getServiceReferences (String, String), getServiceReferences (Class, String): estos métodos devuelven una matriz o colección, respectivamente, de objetos ServiceReference para objetos de servicio que implementan y se registraron bajo la interfaz de servicio especificada.

Esto en código se traduce como:

```
BundleContext bundleContext =
FrameworkUtil.getBundle(this.getClass()).getBundleContext();
ServiceReference<?> serviceReference =
bundleContext.getServiceReference(IMyService.class.getName());
IMyService service = (IMyService)
bundleContext.getService(serviceReference);
```

Figura 4 Consumo de un servicio OSGI

Fuente: (Vogella Osgi Tutorial, 2016)

Si no se registran objetos de servicio coincidentes, el método getServiceReferences (String, String) debe devolver nulo y el método getServiceReferences (Class, String) debe devolver una colección vacía.

La persona que llama recibe cero o más objetos ServiceReference. Estos objetos pueden usarse para recuperar propiedades del servicio subyacente, o pueden usarse para obtener el objeto de servicio real. Consulte Obtención de objetos de servicio.

Los métodos anteriores requieren que la persona que llama tenga el ServicePermission necesario [ServiceReference, GET] para obtener el objeto de servicio para la Referencia de servicio devuelta. Si la persona que llama no tiene el permiso requerido, estos métodos no deben incluir esa Referencia de servicio en el resultado.

2.4.6 ¿POR QUÉ SPRING?

SPRING es una de los frameworks JAVA más potentes en la actualidad que respeta el MODELO VISTA CONTROLADOR, muy flexible al momento de implementar diferentes tipos de arquitecturas de sistemas complejos; básicamente se puede integrar SPRING casi con cualquier tecnología JAVA.

Ventajas de SPRING utilizadas en este proyecto:

- La ya conocida destacable facilidad de SPRING al momento de la creación y administración de BEANS y la inyección de dependencias.

- Administración y manejo de conexiones y transacciones a nivel de la capa de servicios. SPRING puede ser usado para abrir conexiones a base de datos, iniciar una transacción y, en caso de producirse una excepción, realizar rollback; todo esto es posible gracias a la utilización de SPRING de la técnica de programación AOP.

- SPRING respeta el MODELO VISTA CONTROLADOR, de hecho, las tres características descritas anteriormente prácticamente obligan a implementar una aplicación bajo el patrón MVC.

- Facilidad de integración con casi cualquier framework java; entre los más importantes y polulares: Hibernate, MyBatis, JSF, JUnit, integración parcial con OSGI (aún en estudio), entre otros.

- Spring provee librerías que liberan de muchas tareas de código repetitivo al programador. Ayuda a agilizar el desarrollo a nivel de acceso a base de datos con su integración con hibernate, uso de JDBC Templates; control de transacciones automáticas; librerías que agilizan la implementación de un web service o el consumo de servicios; control de tareas o Jobs; invocación de métodos remotos mediante un fácil y sencillo uso de RMI.

2.4.7 ¿POR QUÉ JAVA SERVER FACES?

JAVA SERVER FACES es el “caballo de batalla” de la plataforma J2EE cuando se requiere de algún FRAMEWORK para la construcción de interfaces de usuario web del lado del servidor. Se eligió JAVA SERVER FACES y específicamente la implementación de ésta, PRIME FACES, por lo siguiente:

- Es un FRAMEWORK que respeta el paradigma MVC, esto es, cada vista o formulario está asociado y es manejado por un controlador definido por una clase JAVA.
- Puesto que cada vista o formulario hace “BINDING” o se enlaza a una instancia de una clase controlador, el recojo, manipulación y validación manipulación de la información de las vistas se hace muy sencillo.

- PRIME FACES ofrece una gran librería de componentes de interfaz de usuarios, enriquecidos y probados – solamente es necesario añadir la etiqueta respectiva para hacer uso de un componente-, agilizando en gran medida el diseño y programación de eventos de los diferentes componentes con los que el usuario interactúa.

- Encapsula la programación de eventos asociados a los componentes de interfaz de usuario; es decir, con JSF se puede lograr que cada evento llegue al servidor y al controlador respectivo, mediante peticiones AJAX, sin necesidad de programar directamente estas peticiones, haciendo su uso más fácil y productivo.

- JAVA SERVER FACES, gracias a sus componentes de interfaz de usuario, no se limita a la utilización un lenguaje de scripts (JAVA SCRIPT) o lenguaje de marcas (HTML) – aunque después de todo se puede incluir lenguaje nativo de scripts y HTML en las vistas JSF-, lo que permite una fácil adaptación de un desarrollador BACKEND al mundo del diseño HTML.

2.5 METODOLOGÍA

Los requerimientos de implementaciones generados por las unidades de negocios mediante la herramienta BILLET que requieren la intervención de un PMO o Project Manager Office; es decir, hacemos mención a un proyecto de desarrollo cuyo diseño, arquitectura y funcionalidad están aún por definir, en ATENTO PERÚ no son muy bien recibidos dentro de la JEFATURA DE SOLUCIONES. El campo de acción de ATENTO PERÚ es el CONTACT CENTER; no el desarrollo de software. Normalmente las implementaciones nuevas no siguen una metodología de desarrollo de software, por lo que, en mi opinión personal, no se suele identificar correctamente el problema y objetivos del proyecto, el alcance del mismo, los requerimientos de información y necesidades, y especificaciones funcionales que el sistema debe abarcar; no existe un documento funcional ni documentos técnicos de los desarrollos. El alcance del proyecto y los requerimientos funcionales pueden ir cambiando y terminan por definirse conforme el tiempo de desarrollo avanza. Por todo lo expuesto, para los ANALISTA DE SISTEMAS Y TÉCNICOS PROGRAMADORES, es un verdadero dolor de cabeza hacerse cargo de una implementación surgente que siempre está cambiando en alcance y requerimientos funcionales.
En contraste con ATENTO PERÚ, en mi experiencia laboral anterior me desempeñaba como analista programador en la consultora de software CONASTEC, recibía un documento funcional y técnico del proyecto elaborado por el líder de equipo, previa lectura de ambos documentos, procedía con la codificación de los requerimientos definidos en los documentos; se tenía un objetivo y alcance del proyecto claro y mejor definido.

En la teoría, el PMO maneja el siguiente marco de trabajo, el cual se divide en cuatro fases, tal como se muestra en la siguiente imagen:

Figura 5 Marco de trabajo PMO ATENTO PERÚ

Fuente: (Atento Perú, 2019)

En la sección 3.2 – Reconstrucción del Proyecto se abarca más a detalle cada fase del marco de trabajo y las actividades que componen cada fase del proyecto. Brevemente mencionare que, en la fase Dirección y Planificación, consistió en el levantamiento de la información en base a reuniones entre los responsables de las direcciones de negocios, el PMO y el líder técnico de la división del GRUPO CRM; la fase de Análisis y Diseño básicamente consistió de reuniones entre el líder técnico y los Analistas de Sistemas donde se dio a conocer los requisitos y funcionalidades que se espera del sistema, se comprendió lo que se esperaba del proyecto y se discutió la viabilidad y dificultad

técnica del proyecto; en la fase de diseño, realizada al mismo tiempo que el análisis(en la misma reunión), se definió la arquitectura del motor de envíos, se evaluó las opciones de soporte multithread, se definió la arquitectura para la parte web del proyecto, y se discutió aspectos de conexión a base de datos y estructuras de tablas; la fase de pruebas consistió en demostraciones de la funcionalidad del motor de envíos y del aplicativo web a los gerentes de negocio de CENCOSUD CHILE, simulando casos de envíos de cobranza; en la fase de despliegue se elaboró el pase a producción.

3. DESARROLLO DEL PROYECTO

3.2.1 INTRODUCCIÓN ECOSISTEMA TECNOLÓGICO DE ATENTO.-

Antes de iniciar con la reconstrucción del proyecto de estudio, es menester entender primero la estructura organizacional de ATENTO, y la forma en que las áreas de la organización se relacionan con el área de soluciones (área dedicada a dar soporte a los sistemas de gestión involucrados en el día a día de la interacción ASESOR TELEFÓNICO y CLIENTE).

3.1.1 ORGANIGRAMA ATENTO PERÚ.-

ATENTO lidera el servicio de CONTACT CENTER y es considerado el mayor proveedor de dicho servicio en América Latina. Actualmente ATENTO cuenta con instalaciones físicas en países como España, Argentina, Brasil, Chile, Colombia, El Salvador, Guatemala, México, Panamá, Perú, Puerto Rico, Uruguay Y Colombia.

Atento Perú pertenece a la denominada “Región Sur” conformada por todas las subsidiarias sudamericanas excepto Brasil, es decir, la región sur la conforman los países de Argentina, Chile, Colombia, Perú y Uruguay.

La estructura organizacional de ATENTO PERÚ es compleja y abarca una gran cantidad de áreas divididas según funciones. En la sección de anexos se adjunta a detalle el organigrama completo de la organización en PERÚ. En esta sección del informe me interesa enfocarme en el organigrama correspondiente a la JEFATURA DE TECNOLOGÍA O SOLUCIONES, y en el organigrama de las DIRECCIONES DE NEGOCIO; ambas unidades – la de TECNOLOGÍA y de NEGOCIOS- están en constante interacción: las direcciones de negocios constantemente solicitan a la JEFATURA DE SOLUCIONES soporte con los aplicativos de gestión, solución de incidencias e implementaciones nuevas. A continuación, se presenta los organigramas correspondientes a estas áreas de la organización.

3.1.1.1 ORGANIGRAMA JEFATURA DE TECNOLOGÍA O SOLUCIONES

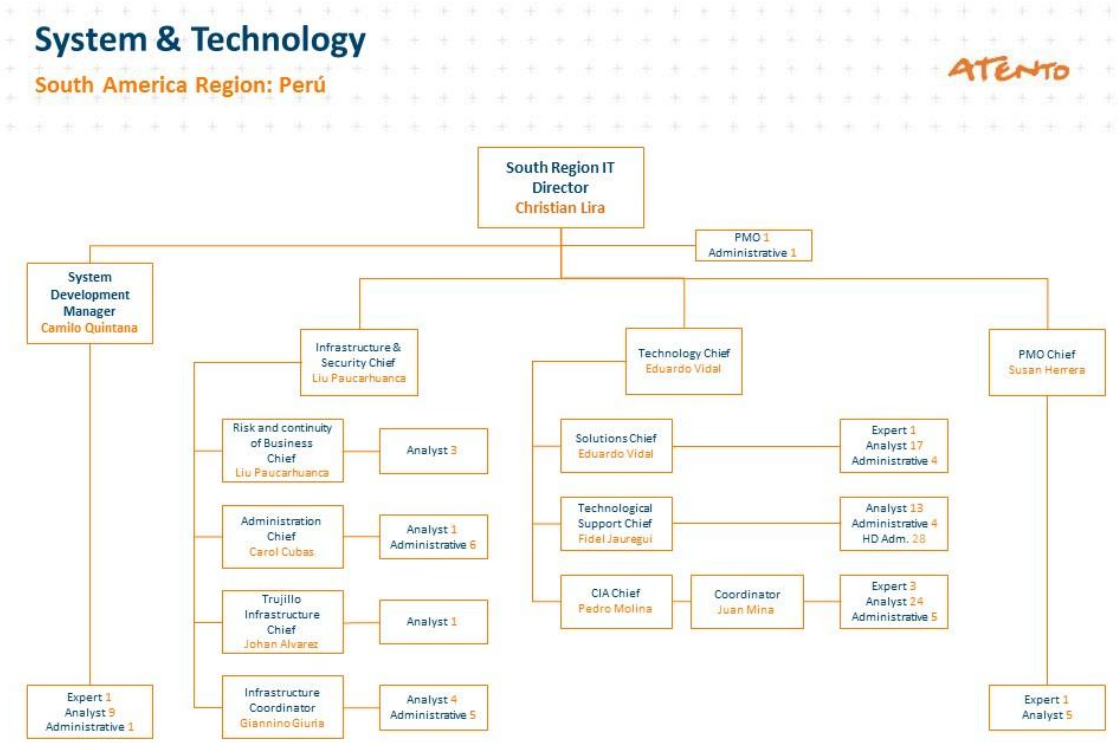


Figura 6 Organigrama Jefatura de Soluciones

Fuente: (Atento Perú, 2019)

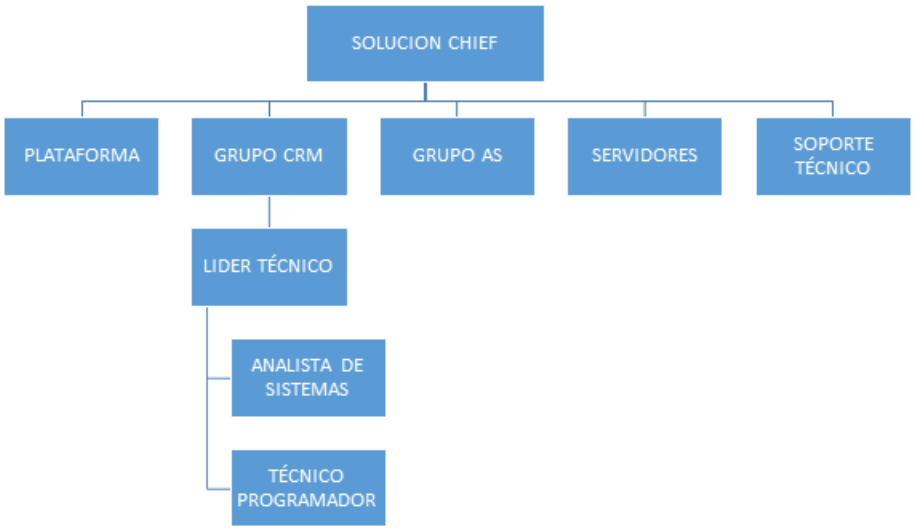


Figura 7 Organigrama detalle Jefatura de Soluciones

Fuente: (Fuente de Autoría propia, 2019)

3.1.1.1.1 PLATAFORMA.-

División tecnológica de la jefatura de soluciones encargada de administrar la infraestructura tecnológica de telefonía AVAYA. Se encarga de realizar las configuraciones necesarias para que una campaña nueva de un determinado servicio pueda iniciar operaciones de gestión telefónica.

3.1.1.1.2 GRUPO CRM.-

División tecnología de la jefatura de soluciones encargada de administrar los aplicativos de gestión de llamadas. El GRUPO CRM se encarga del desarrollo de nuevos requerimientos concernientes a los aplicativos de gestión como el MULTIGESTIÓN, SBT BBVA, CONFIGURADOR DE CAMPAÑAS, MIDDLEWARE DE CONEXIÓN A BASE DE DATOS, CROSSELLING, PROCESOS DE CARGA A TABLAS MAESTRAS mediante ETLS, entre otros; también se encarga de las integraciones del MULTIGESTIÓN con aplicaciones propiedad de terceros. El GRUPO CRM, así como las otras divisiones, está liderada por un LÍDER TÉCNICO y este a su vez tiene a cargo muchos analistas de sistemas, los analistas de sistemas pueden trabajar solo o tener como apoyo uno o varios técnicos programadores.

3.1.1.1.3 GRUPOS AS.-

División tecnológica de la jefatura de soluciones encargada de recibir las incidencias de gestión, de ser posible solucionarlas, o derivarlas al área correspondiente para su solución. También tienen en custodia aplicativos java como el motor disparador de llamadas “CALLER”, el aplicativo de descarga de grabaciones “GIGA”, el sistema de gestión de correos electrónicos, el aplicativo BARRA ATENTO o BARRA AVAYA y el aplicativo GRABADOR IP.

3.1.1.1.4 SERVIDORES.-

División tecnológica de la jefatura de soluciones encargada de la administración de base de datos, administración de servidores WINDOWS y Linux, administración de redes, y la ejecución de los pases a producción.

3.1.1.1.5 SOPORTE TÉCNICO.-

División tecnológica de la jefatura de soluciones encargada del mantenimiento de los puestos o computadores de los asesores, instalación y configuración de software, configuración de perfiles de usuarios Windows y resolución de incidencias menores.

3.1.1.2 ORGANIGRAMA DIRECCIÓN DE NEGOCIO TELEFÓNICA Y MULTISECTOR

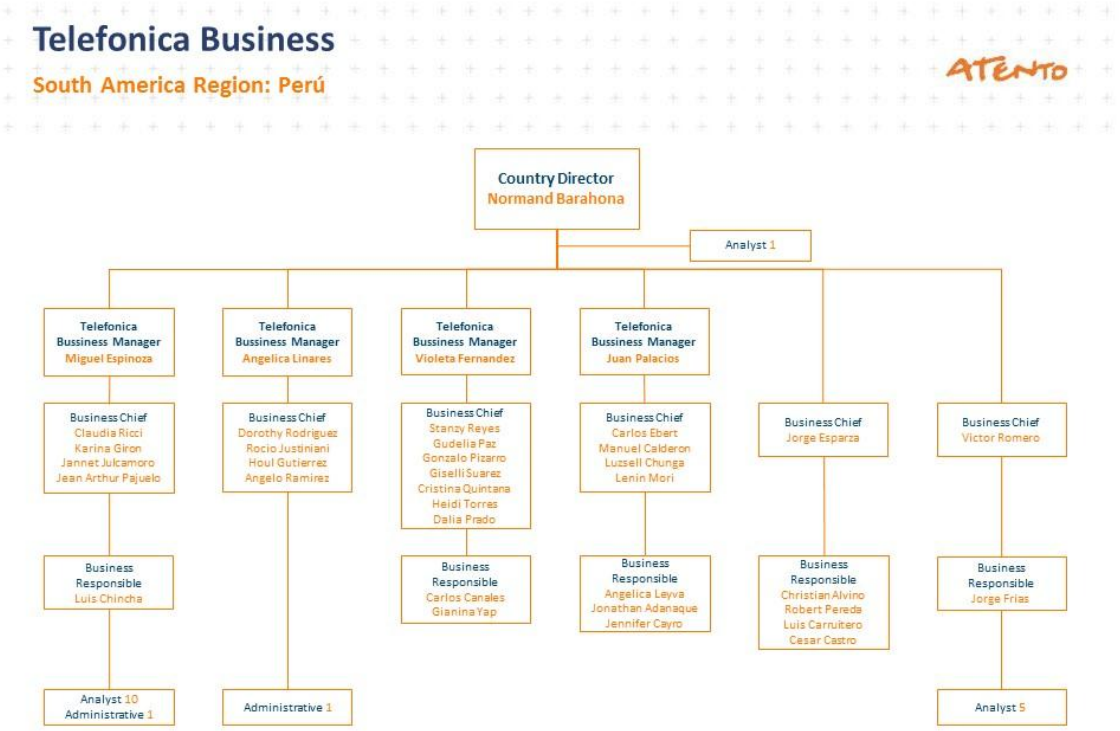


Figura 8 Organigrama Telefónica

Fuente: (Atento Perú, 2019)

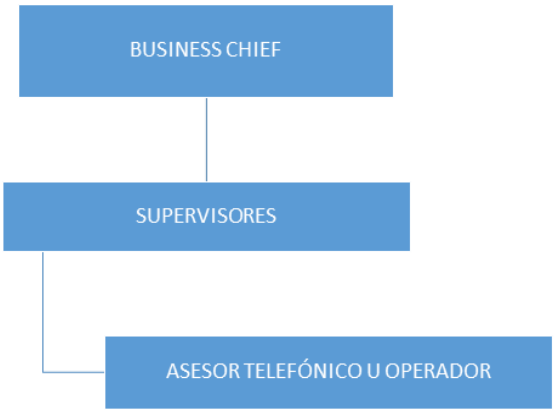


Figura 9 Detalle Organigrama Multisector

Fuente: (Fuente de Autoría propia, 2019)

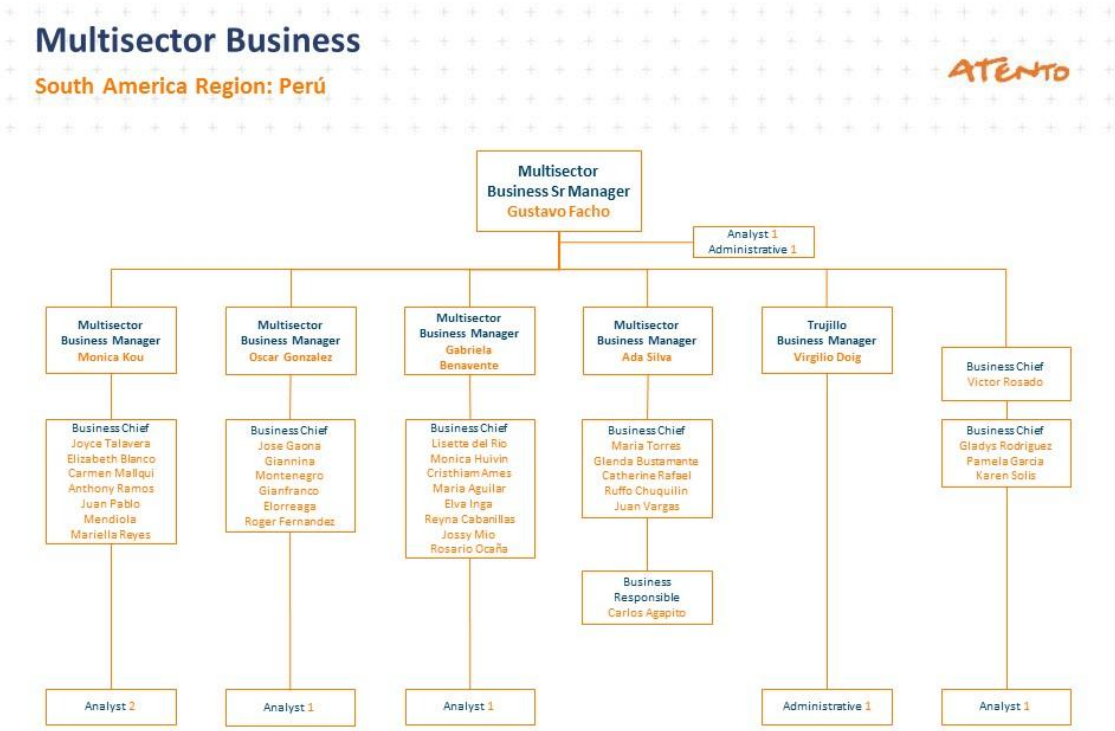


Figura 10 Organigrama Multisector

Fuente: (Atento Perú, 2019)

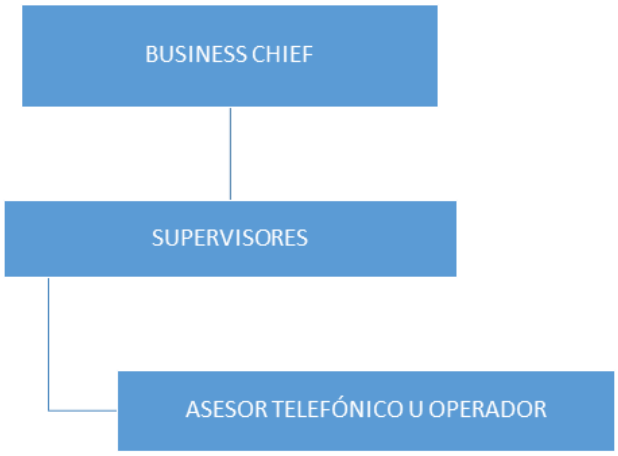


Figura 11 Detalle Organigrama Multisector

Fuente: (Fuente de Autoría propia, 2019)

3.1.1.2.1 DESCRIPCIÓN ORGANIGRAMA MULTISECTOR Y TELEFÓNICA.-

Cuando un nuevo cliente contrata los servicios de atento, éste se asigna a una de las direcciones de negocio, ya sea a MULTISECTOR o TELEFONICA; un cliente del sector bancario por lo general se suele asignar a la dirección MULTISECTOR. Dentro del contrato de servicios firmado por ATENTO PERÚ y el CLIENTE, se acuerda un número de PUESTOS DE ASESOR o COMPUTADORES a utilizar para la gestión. En cada puesto un ASESOR TELEFÓNICO iniciará sus labores. La responsabilidad de cierto número de ASESORES TELEFÓNICOS recae sobre los SUPERVISORES. Los supervisores a su vez responden al BUSINESS CHIEF o JEFE DE OPERACIONES asignado al nuevo cliente. Los BUSINESS CHIEF y los BUSINESS MANAGER tienen comunicación directa con el cliente final, siempre están atentos a las inquietudes o iniciativas de negocios que su cliente solicite o, en su defecto, proponen a su cliente nuevas ideas de negocio que en muchas ocasiones implican IMPLEMENTACIONES NUEVAS o IMPLEMENTACIONES DE RUTINA derivadas a la jefatura de soluciones.

3.1.2 IMPLEMENTACIONES DE RUTINA.-

Las implementaciones de rutina solicitada por las direcciones de negocio a la jefatura de soluciones se originan cuando un nuevo cliente contrata los servicios de atento. Para la puesta en producción del inicio de gestión para un nuevo cliente, se requiere gestionar una serie de configuraciones e implementaciones a las divisiones de PLATAFORMA y GRUPO CRM.
Como su nombre lo dice, son implementaciones con un procedimiento ya establecido y no implican desarrollos nuevos con alta complejidad.

Los requerimientos llegan a la JEFATURA DE SOLUCIONES mediante la herramienta de gestión de requerimientos e incidencias BILLET. Cuando un BUSINESS CHIEF o SUPERVISOR genera un "TICKET" en el sistema BILLET, adjunta una plantilla conocida como MRT (Modelo de Requerimiento de Trabajo) donde especifica el detalle del requerimiento solicitado.

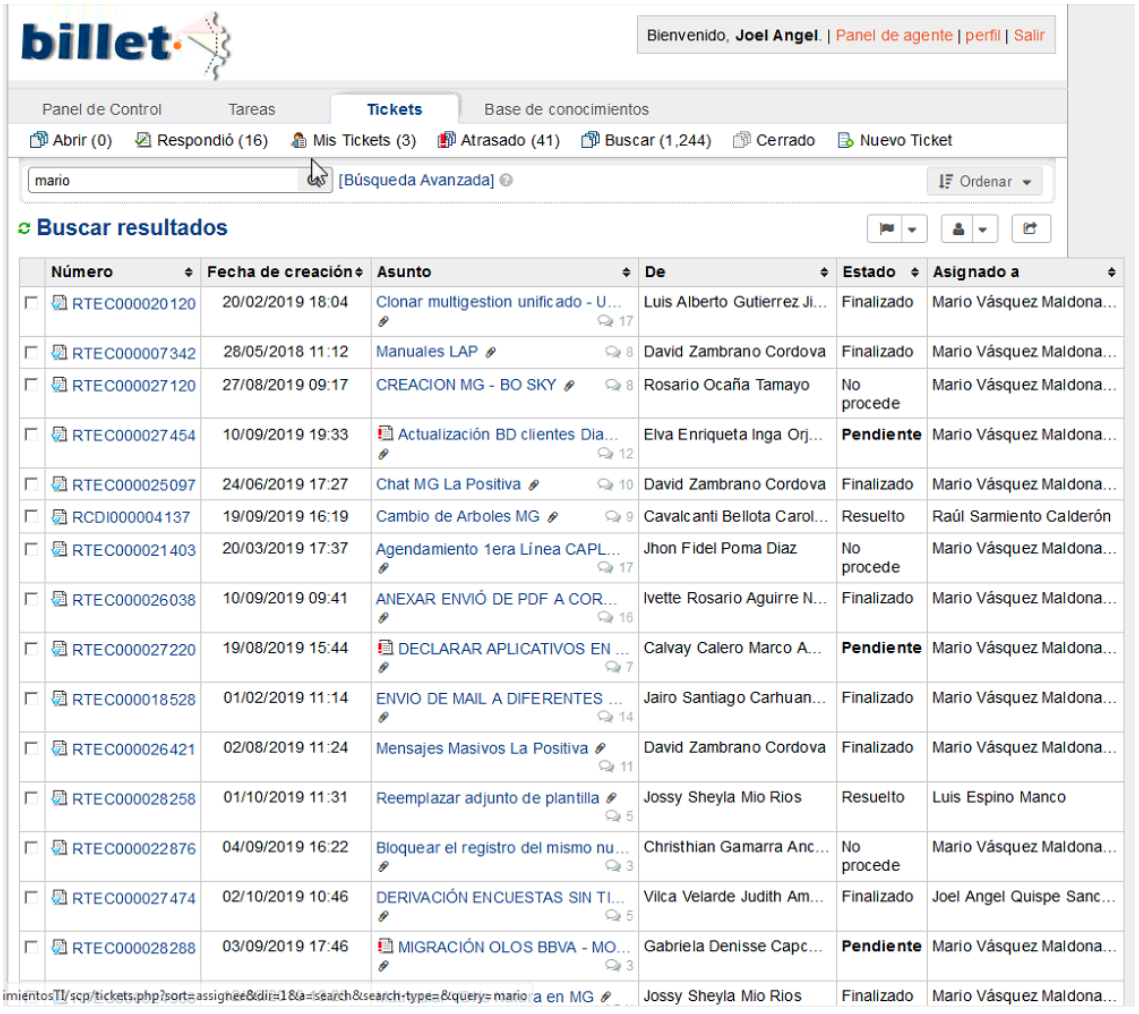
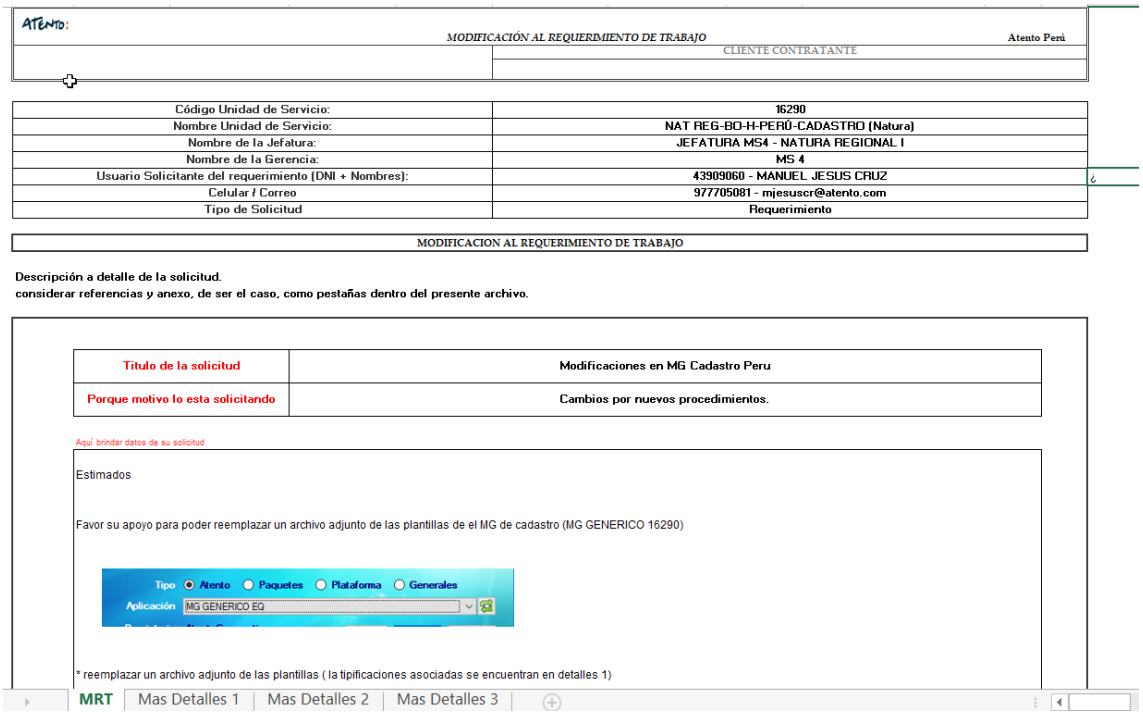


Figura 12 SISTEMA BILLET

Fuente: (Atento Perú, 2019)



ATENTO:

MODIFICACIÓN AL REQUERIMIENTO DE TRABAJO Atento Perú

CLIENTE CONTRATANTE

Código Unidad de Servicio:	16290
Nombre Unidad de Servicio:	NAT REG-BO-H-PERÚ-CADASTRO (Natura)
Nombre de la Jefatura:	JEFATURA MS4 - NATURA REGIONAL I
Nombre de la Gerencia:	MS 4
Usuario Solicitante del requerimiento (DNI + Nombres):	43909060 - MANUEL JESUS CRUZ
Celular / Correo	977705081 - mjesuscr@atento.com
Tipo de Solicitud	Requerimiento

MODIFICACION AL REQUERIMIENTO DE TRABAJO

Descripción a detalle de la solicitud.
considerar referencias y anexo, de ser el caso, como pestañas dentro del presente archivo.

Titulo de la solicitud	Modificaciones en MG Cadastro Peru
Porque motivo lo esta solicitando	Cambios por nuevos procedimientos.

Aquí brindar datos de su solicitud

Estimados

Favor su apoyo para poder reemplazar un archivo adjunto de las plantillas de el MG de cadastro (MG GENERICO 16290)

Tipo Atento Paquetes Plataforma Generales
Aplicación MG GENERICO EQ

* reemplazar un archivo adjunto de las plantillas (la tipificaciones asociadas se encuentran en detalles 1)

MRT | Mas Detalles 1 | Mas Detalles 2 | Mas Detalles 3

Figura 13 EJEMPLO DE MRT

Fuente: (Atento Perú, 2019)

Las implementaciones de rutinas requieren la intervención de las divisiones de PLATAFORMA, GRUPO CRM, GRUPO AS, y CIA para la creación de reportes de ventas y de facturación.

NOTA.- En las siguientes secciones se hará uso de algunos términos técnicos los cuales se pasaran a describir en la sección **"3.1.4 ARQUITECURA PARA EL PROCESO DE LLAMADAS DE ENTRADA Y SALIDA"**.

3.1.2.1 CAMPAÑAS ATENTO.-

En atento una CAMPAÑA se conoce como una gestión operativa que involucra un grupo de asesores telefónicos, supervisores y Jefes de Operaciones, asociadas a un cliente especifico (BBVA, NATURA, CENCOSUD CHILE, etc.). Así, por ejemplo, el cliente BBVA puede tener muchas CAMPAÑAS operativas: Campañas de cobranza, venta de tarjetas, préstamos, etc. Cada campaña para la puesta en marcha de su gestión, requiere una serie de configuraciones técnicas a nivel de aplicativos que involucran a áreas como PLATAFORMA, GRUPO CRM y GRUPO AS. Cada campaña tiene asignada una unidad de servicio - un número de 4 dígitos que identifica a la campaña de manera única- la misma que sirve para agrupar e identificar todas las configuraciones asociadas a la campaña.

3.1.2.2 IMPLEMENTACIONES DE RUTINA – PLATAFORMA.-

Algunas de las implementaciones de rutina para la división de plataforma son:

- Creación de VDNS AVAYA para el tráfico de llamadas telefónicas de la campaña.
- Administración de troncales AVAYA para la salida y recibimiento de llamadas de la campaña.
- Creación de claves de llamada AVAYA, importante al momento de realizar llamadas telefónicas de salida.
- Diseño y creación de IVR'S (Interactive Voice Response).
- Configuración de colas de esperas para VDNS.
- Configuración de tonos de esperas.
- Configuración de derivaciones a encuestas.
- Configuraciones de tráficos de trunks o troncales.

3.1.2.3 IMPLEMENTACION DE RUTINA – GRUPO CRM.-

Algunas de las implementaciones de rutinas son:

- Implementaciones o desarrollos nuevos para agregar nuevas funcionalidades a los aplicativos o modificar las ya existentes.
- Creación de procesos ETL para poblar las tablas maestras de las campañas.
- Registro de las claves de llamadas generadas por PLATAFORMA en las tablas de configuración del MULTIGESTIÓN.
- Registro de VDN generados por plataforma en las tablas de configuración del MULTIGESTIÓN.
- Configuración de ficha de cliente a visualizar en el MULTIGESTIÓN.
- Configuración de ficha de gestión a visualizar en el MULTIGESTIÓN.
- Configuración de módulos de MULTIGESTIÓN a usar por la campaña.
- Configuración de productos a usar por la campaña.
- Configuración de árboles de tipificación a usar por la campaña.
- Configuración de resultados de gestión.

3.1.2.4 IMPLEMENTACION DE RUTINA – GRUPO AS.-

- Implementaciones o desarrollos nuevos para agregar nuevas funcionalidades a los aplicativos o modificar las ya existentes.
- Creación de ETL para cargar tablas maestras del motor caller para el disparo de llamadas automáticas.
- Administración de repositorios de grabaciones donde se almacenará las grabaciones de llamadas de las campañas.
- Configuración de usuarios para aplicativos GIGA con la finalidad de permitir a la campaña descargar el reporte de sus grabaciones.

3.1.3 IMPLEMENTACIONES NUEVAS.-

Son requerimientos que implican una solución nueva para la JEFATURA DE SOLUCIONES, por lo que es necesario que las direcciones de negocio soliciten la intervención de un PMO o Project Manager Office para el levantamiento de la información, elaboración de cronogramas y actas de reuniones en conjunto con los BUSINESS CHIEF y el CLIENTE FINAL. En algunos casos, cuando los requerimientos tienen una complejidad alta, el PMO solicita la ayuda de los líderes técnicos para un mejor levantamiento de información.
Cuando la etapa de proceso de levantamiento de información culmina, el PMO consulta la viabilidad del requerimiento al líder técnico y a los analistas de sistemas y, en caso el requerimiento resulte viable, consulta por los tiempos de desarrollo. Luego el PMO genera un nuevo TICKET de requerimiento en la herramienta BILLET, el LIDER TÉCNICO asigna en la herramienta BILLET el o los desarrolladores que intervendrán en el requerimiento. El Ticket generado entra a una cola de atención o, en algunos casos, dependiendo de la importancia y urgencia, es escalado hasta el TECHNOLOGY CHIEF para una pronta atención.

3.1.4 ARQUITECURA PARA EL PROCESO DE LLAMADAS DE ENTRADA Y SALIDA.-

A continuación se intentara describir a grandes rasgos cómo se realiza el proceso de recibir una llamada telefónica, sus componentes más importantes y como éstos interactúan.

La imagen 6 ilustra la arquitectura para el proceso de llamadas mediante troncales y un PBX AVAYA; ampliamente usados en ATENTO PERÚ. Para comprender la imagen 6, es necesario tener una idea de los siguientes conceptos:

- Red de telefonía pública.- Una red de telefonía pública consta de uno o varios números fijos públicos proveídos por compañías telefónicas. Estos números públicos son utilizados por los clientes para contactarse con una organización en particular.
- PBX.- siglas en inglés de Private Branch Exchange que traducido al español seria Ramal Privado de Conmutación, es una central telefónica interna dentro de una empresa, dedica a administrar las llamadas

internas, las llamadas entrantes y de salida. Este servicio lo proveen empresas especializadas en telecomunicaciones como AVAYA, CISCO, GENESYS, entre otras.

- AVAYA.- Empresa de telecomunicaciones ampliamente usado en ATENTO PERÚ como proveedor de PBX.
- LÍNEAS TRONCALES.- La línea troncal es el encargado de comunicar simultáneamente múltiples llamadas desde la RED DE TELEFONÍA PUBLICA hasta la PBX.
- VDN.- Siglas en ingles de Vector Directory Number, es un componente de software fundamental dentro de una PBX, para ser más preciso, se trata de un número de extensión de software (no es una extensión física). Es el encargado de administrar las colas de llamadas; transfiere la llamada de acuerdo a los skills a una extensión física o AGENTE AVAYA disponible. Un vdn puede estar asociados a uno o varios skills.
- Agentes Avaya.- El agente Avaya es un software que se instala en cada uno de los puestos o computadores donde se supone iniciará labores un asesor telefónico. Este software convierte al computador en una extensión telefónica física. Un agente avaya tiene asignado un skill, de esta forma el VDN puede identificar aquellos AGENTES AVAYAS que cumplan con un determinado SKILL para luego poder transferir llamadas.

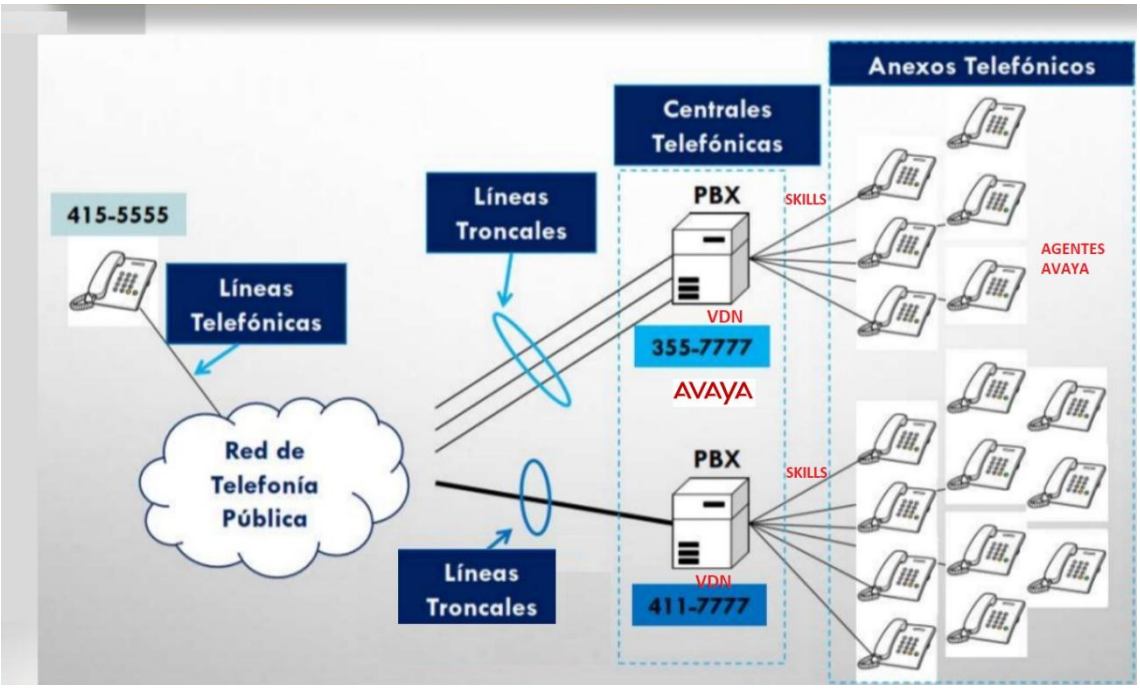


Figura 14 ARQUITECTURA PBX ATENTO PERÚ

Fuente: (Fuente de Autoría propia, 2019)

3.1.5 ¿CÓMO SE GESTIONA LAS LLAMADAS EN LOS DIFERENTES SERVICIOS DE ATENCIÓN?

En la sección anterior se describió de manera general la arquitectura tecnológica empleada para recibir llamadas telefónicas y transferir las mismas a los puestos de asesor. Pero, eso no acaba ahí cuando el asesor telefónico responde una llamada telefónica; no es suficiente con levantar el handset telefónico y responder las inquietudes del cliente. Se necesita de un sistema de gestión –instalado en cada uno de los puestos de asesor- que identifique a los clientes por el número de teléfono usado por el cliente para llamar, o quizás identifique al cliente por el número de identificación ingresado momentos antes de la transferencia de llamada; se necesita de un sistema que mantenga actualizada la base de datos con la información histórica de las llamadas.

El sistema MULTIGESTION concebido en la jefatura de soluciones de ATENTO PERÚ, es el sistema usado en cada uno de los puestos de asesor de cada una de las plataformas de servicios de ATENTO PERÚ y ATENTO COLOMBIA. Hablar del MULTIGESTIÓN es hablar del aplicativo CORE usado para gestionar las llamadas telefónicas de SALIDA (cuando el OPERADOR realiza una llamada) y de ENTRADA (cuando ingresa una llamada al puesto del OPERADOR).

Figura 15 PLATAFORMA DE SERVICIO NÚMERO 6, SEDE ATE

Fuente: (Atento Perú, 2019)

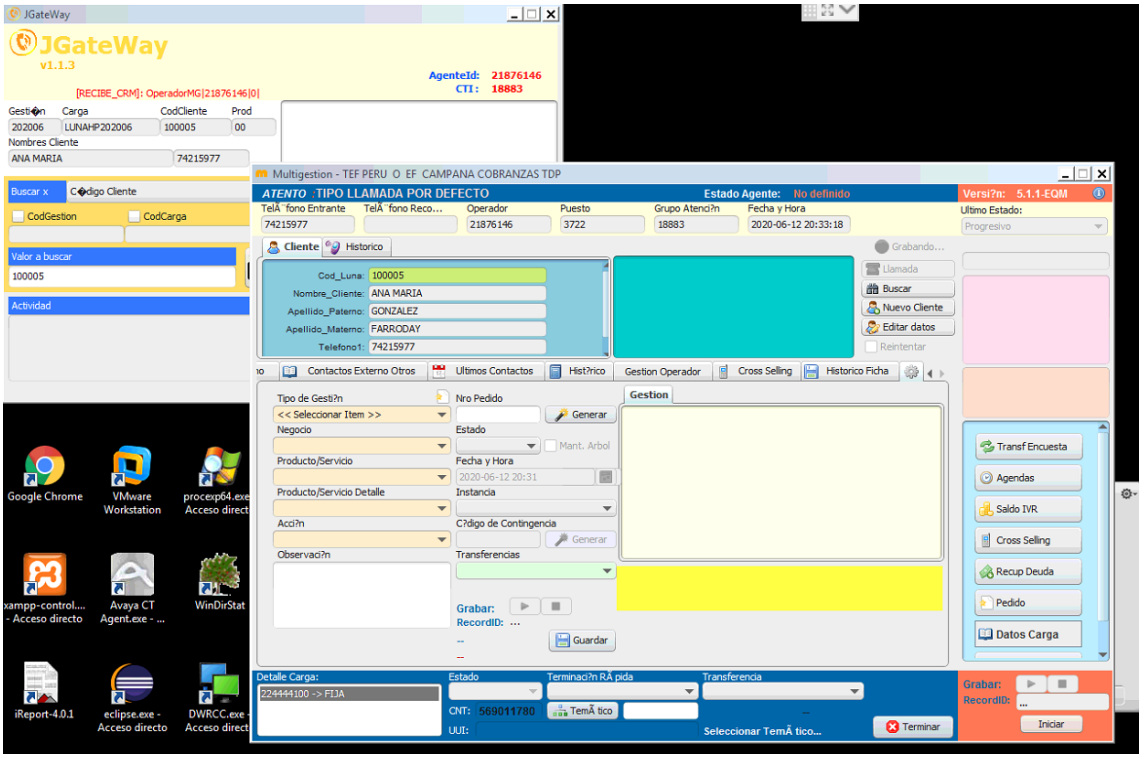


Figura 16 APLICATIVO MULTIGESTIÓN

Fuente: (Atento Perú, 2019)

Los datos que el MULTIGESTION persiste en las bases de datos transaccionales conciernen a datos de la llamada, del cliente, identificador de la grabación, datos de la gestión según tipificación configurada a las campañas del servicio, entre otros datos. Como se puede comprender, el MULTIGESTIÓN es la herramienta que genera y persiste los datos de gestión en base de datos, información vital que alimenta a los reportes de ventas, indicadores de gestión, cubos y datamarts. Cabe mencionar que dentro de ATENTO PERÚ existe un área especializada en reportes conocida como CIA (CENTRO DE INVESTIGACIÓN Y ANÁLISIS) y también un área de Business Analytics; ambas áreas se valen de los datos proporcionados por el MULTIGESTION para generar información confiable y oportuna para la toma de decisiones gerenciales, y para las facturaciones a los socios comerciales de ATENTO (CLIENTES).

Si bien es cierto, sin el aplicativo MULTIGESTIÓN la información vital e imprescindible de las llamadas telefónicas de entrada y de salida no sería persistida en las diferentes tablas de base de datos transaccionales, para que un operador puede realizar su trabajo de gestión (básicamente contestar o realizar llamadas telefónicas), en un puesto de asesor aparte del MULTIGESTION, se necesita tener instalado los siguientes aplicativos:

3.1.5.1 AVAYA ONE X COMMUNICATOR.-

El software de AVAYA instalado en un puesto de asesor convierte al computador en un anexo físico. AVAYA ONE X COMMUNICATOR permite la opción de configurar puertos UDP en el computador a donde se enviarán mensajes para notificar cuando se reciba una llamada, se corte la llamada, se realiza una llamada, el agente se ponga en estado no disponible u ocupado, entre otros eventos de telefónicos. Este software no es propiedad de atento, es un software licenciado por la empresa AVAYA.

Figura 17 AVAYA ONE X COMMUNICATOR

Fuente: (Atento Perú, 2019)

3.1.5.2 AGENTE AVAYA.-

El agente AVAYA permite que un asesor telefónico inicie sesión en la PBX de ATENTO mediante un ID AGENTE, un número de extensión y contraseña.

Figura 18 AGENTE AVAYA

Fuente: (Atento Perú, 2019)

3.1.5.3 BARRA ATENTO.-

También conocido como BARRA AVAYA, es un aplicativo desarrollado en el lenguaje C por el GRUPO AS (ver organigrama). Este aplicativo constantemente interactúa con el los aplicativos AVAYA anteriormente descritos, la comunicación se produce mediante mensajes UDP. La BARRA ATENTO constantemente está a la escucha del puerto configurado por AVAYA para identificar los diferentes eventos telefónicos. La BARRA ATENTO también es capaz de ordenar a AVAYA que inicie ciertos eventos como un: INCOMING CALL, DROP CALL, MAKE CALL, etc. Otra función principal de la BARRA DE ATENTO es la de notificar al MULTIGESTIÓN mediante mensajes UDP cuando se inicie ciertos eventos AVAYA y también notificar eventos propios de la BARRA DE ATENTO como lo son el: START RECORDING, STOP RECORDING.

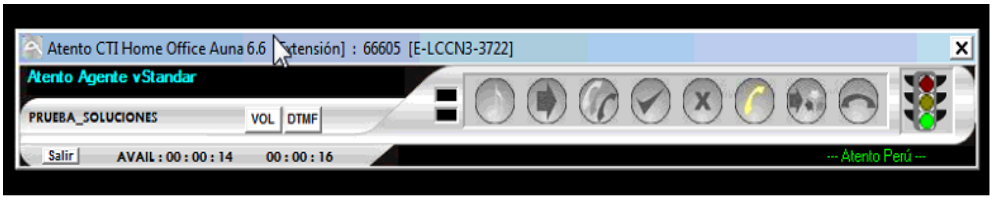


Figura 19 BARRA ATENTO O BARRA AVAYA

Fuente: (Atento Perú, 2019)

3.1.5.4 GRABADOR IP.-

Aplicativo desarrollado en el lenguaje C. Este aplicativo se encarga de realizar las grabaciones de llamadas. Interactúa constantemente con la BARRA ATENTO mediante mensajes UDP para saber cuándo iniciar o detener una grabación.

3.1.5.5 MULTIGESTIÓN.-

Aplicativo modular desarrollado en JAVA siguiendo el enfoque de modularidad de OSGI. Constantemente esta interactuando con la BARRA ATENTO. Cuando el MULTIGESTIÓN se inicia en un puesto de asesor, internamente configurar un SERVIDOR SOCKET UDP para recibir las notificaciones de la BARRA ATENTO; el MULTIGESTIÓN también conoce el puerto de escucha de la BARRA ATENTO, de esta manera puede ordenarle iniciar ciertos eventos AVAYA.

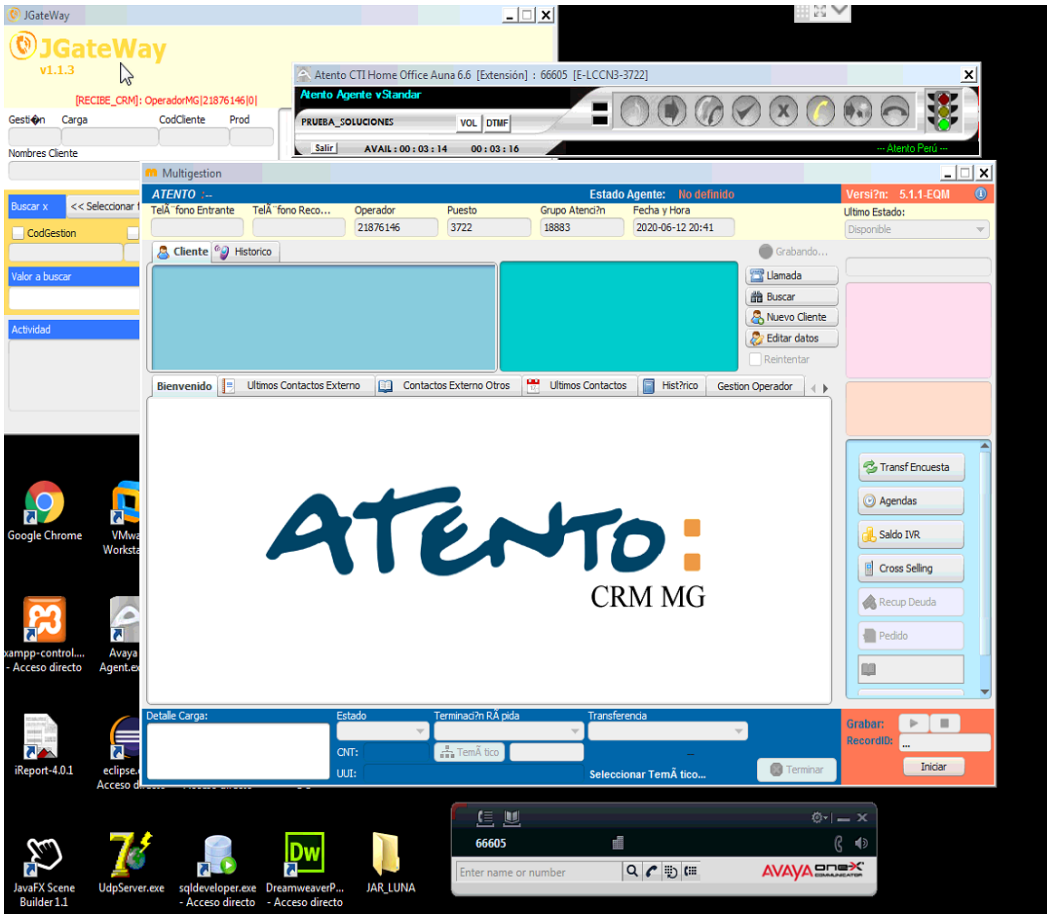


Figura 20 MULTIGESTION

Fuente: (Atento Perú, 2019)

El MULTIGESTIÓN se comparta diferente dependiendo de los eventos que recibe. Por ejemplo, si recibe una trama UDP de tipo “INCOMING CALL” – esta trama contiene información como VDN, teléfono de llamada, DNI de cliente, etc.- el MULTIGESTIÓN realiza lo siguiente:

a. Registro en base de datos del inicio de gestión de llamada.
b. Mediante el VDN identifica al servicio de atención y su configuración asociada al MULTIGESTION.
c. Por el teléfono o DNI identifica al cliente y muestra información de éste en el aplicativo.

3.2.2 RECONSTRUCCIÓN DE LA EXPERIENCIA LABORAL – LÍNEA DEL TIEMPO

El MOTOR DE ENVÍOS o MOTOR DE COBRANZAS surgió como una iniciativa dentro de la dirección de negocio correspondiente a CENCOSUD CHILE. Necesito la intervención del PMO LUIS HUMBERTO PEREZ URTEAGA para el levantamiento de información, la definición de objetivos, control y seguimiento del cronograma de actividades; requirió la intervención del líder técnico del grupo CRM, LUIS ESPINO, para el análisis y diseño del sistema en conjunto con los analistas de sistemas (incluido yo); la responsabilidad del desarrollo y codificación del proyecto estuvo bajo mi responsabilidad, así como la elaboración del pase a producción.

Tal como se hace mención en la sección **2.5 de Metodología**, el proyecto del Motor de Cobranza o Envíos se dividió – en teoría- en cuatro etapas: Dirección y planificación, Análisis y Diseño, Construcción, Pruebas y Pase a producción. Las tareas o actividades correspondientes al marco de trabajo de este proyecto, el PMO Luis Humberto las organizó y distribuyo de la siguiente manera:

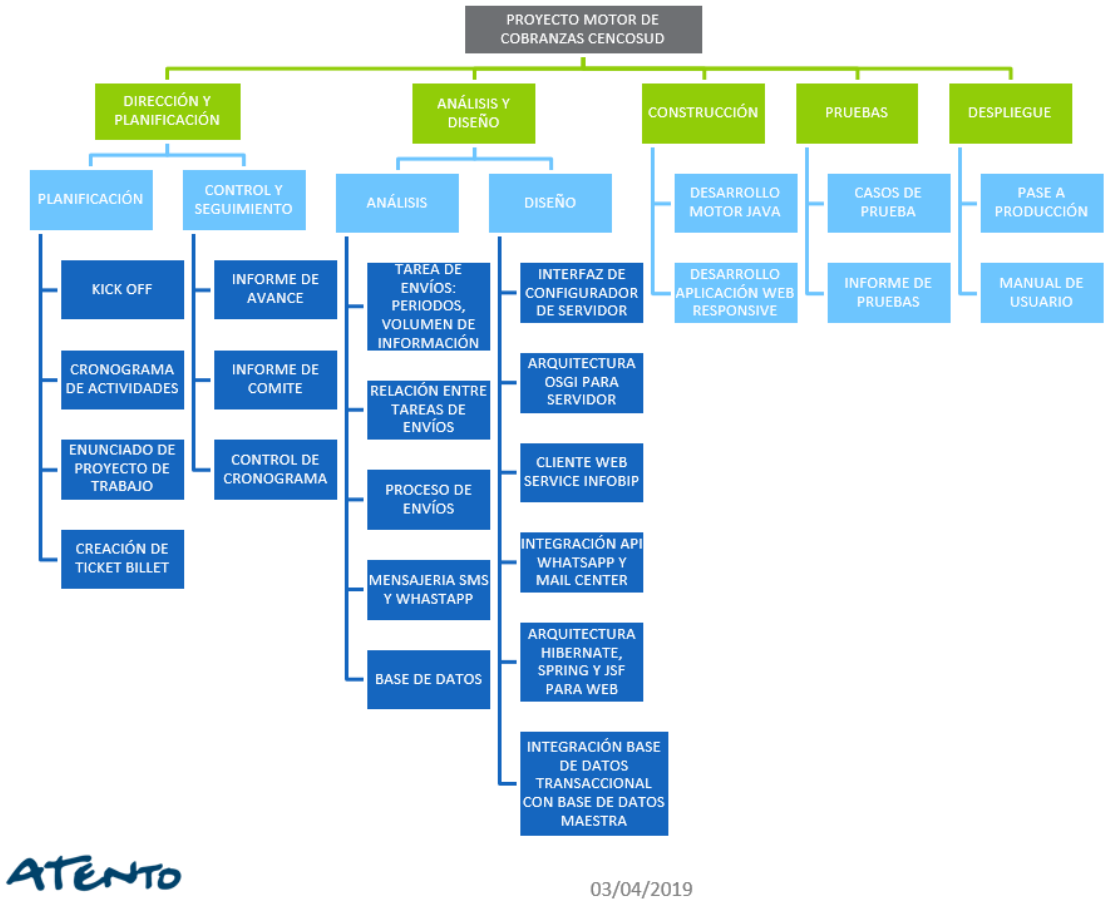


Figura 21 FASES Y ACTIVIDADES DEL PROYECTO. ELABORACIÓN LUIS PÉREZ

Fuente: (Atento Perú, 2019)

3.2.1.1 ETAPAS DEL PROYECTO EN LAS CUALES EL AUTOR PARTICIPÓ.-

A continuación, se ilustra las etapas del proyecto en las cuales el autor de este informe, como analista de sistemas, se involucró ya sea como responsable o participante:

ETAPA	ACTIVIDAD	NIVEL DE PARTICIPACIÓN
Análisis	Tara de Envíos: Periodos, volumen de información	PARTICIPANTE
Análisis	Relación entre tareas de envíos	RESPONSABLE
Análisis	Proceso de envíos	PARTICIPANTE
Análisis	Mensajería SMS y Whastsapp	RESPONSABLE
Análisis	BASE DE DATOS	PARTICIPANTE
Diseño	Interfaz de configuración del servidor	RESPONSABLE
Diseño	Arquitectura OSGI para servidor	RESPONSABLE
Diseño	Cliente Web Service Infobip	RESPONSABLE
Diseño	Integración API WHATSAPP y Mail Center	RESPONSABLE
Diseño	Arquitectura Hibernate, Spring y JSJ.	RESPONSABLE
Diseño	Integración base de datos transaccional con base de datos maestra	RESPONSABLE
Construcción	Desarrollo Motor Java	RESPONSABLE
Construcción	Desarrollo aplicación Web	RESPONSABLE
Pruebas	Pruebas con cliente	PARTICIPANTE
Despliegue	Elaboración de pase a producción	RESPONSABLE

En las siguientes secciones pasare a detallar dichas etapas prestando más énfasis y atención a la parte técnica de las etapas de Diseño y Construcción del sistema.

3.2.1 DIRECCIÓN Y PLANIFICACIÓN.-

En esta fase del proyecto no tuve participación alguna, por lo que me limitare solamente a narrar desde la perspectiva de un DESARROLLADOR la forma en que se realiza dicha etapa en los proyectos de implementaciones de software en ATENTO PERÚ.

3.2.1.1 RESPONSABLE.-

La responsabilidad de esta etapa del proyecto recae sobre el o los PMO involucrados en el proyecto, para este proyecto la responsabilidad recayó sobre LUIS HUMBERTO PEREZ URTEAGA.

3.2.1.2 PARTICIPANTES.-

Para esta etapa del proyecto se vieron involucrados tantos representantes de las unidades de negocio comercial dedicado a CENCOSUD así como el líder técnico del GRUPO CRM para el apoyo al levantamiento de información y opinión técnica.

3.2.1.3 ¿CÓMO SE REALIZÓ EL ANÁLISIS DE LA INFORMACIÓN?

Las actividades de esta fase del proyecto estuvieron bajo la responsabilidad del PMO asignado al proyecto. Normalmente los ANALISTAS DE SISTEMAS solemos enterarnos de la puesta en marcha oficial de un proyecto cuando el requerimiento se nos asigna mediante un TICKET BILLET generado por la herramienta de gestión de incidencias y requerimientos BILLET; es decir, la fase de DIRECCIÓN Y PLANIFICACIÓN puede llevarse a cabo mucho antes de que los desarrolladores tengan conocimiento de la definición de objetivos y alcance del proyecto, lo que, en definitiva, suele causar molestias entre los analistas y desarrolladores.

La actividad más importante de esta fase del proyecto es el KICK OFF. Esta actividad consiste en una o más reuniones que el PMO organiza, cuenta con la presencia de los Jefes de Operaciones o Business Chief, los Business Manager y representantes del cliente final (CENCOSUD CHILE para este caso), también cuenta con la presencia del líder técnico del GRUPO CRM. La finalidad principal del KICK OFF es la de “aterrizar el proyecto” definiendo las necesidades del mismo, objetivos y alcance, y fechas críticas; la función del líder técnico en las reuniones es la de desestimar o garantizar la factibilidad de lo solicitado desde su perspectiva técnica. Luego, una vez “aterrizado” el proyecto, el PMO acuerda una reunión por separado con el líder técnico del GRUPO CRM, con la finalidad de definir -“coloquialmente”- las funcionalidades y requerimientos del sistema y las fechas tentativas para las pruebas y pase a producción; todo esto se formaliza mediante un TICKET BILLET dirigido al GRUPO CRM y el correspondiente MRT (Modelo de Requerimiento de Trabajo).

Durante las reuniones llevadas a cabo en esta fase del proyecto, el líder técnico del GRUPO CRM pudo concluir que lo solicitado era viable en cuanto a implementación. Básicamente lo que la dirección de negocio de CENCOSUD CHILE requería es la concepción de un sistema que realice la gestión de cobranza automatizada valiéndose de diferentes canales de comunicación y que dicho sistema sea capaz de integrar y mantener relacionados estos canales de comunicación. Por canales de comunicación nos referimos a medios para interactuar con el cliente para hacer de su conocimiento la deuda a vencer u oferta de préstamo vigente.

3.2.1.4 ENUNCIADO DEL PROYECTO.-

Las reuniones llevadas a cabo en esta fase del proyecto, específicamente en las reuniones de KICK OFF que sirvió a su vez para realizar el levantamiento de información, permitieron enunciar el proyecto en palabras del PMO que dirigió el proyecto, al momento de informar a los analistas sobre el proyecto que se avecinaba, de la siguiente manera:

"Actualmente, como es de su conocimiento y para aquellos que aún no lo saben, CENCOSUD CHILE hace uso de nuestro canal de comunicación IVR (INTERACTIVE VOICE RESPONSE) para la cobranza preventiva. Mediante el motor "CALLER" desarrollado por la división de tecnología "GRUPO AS", se puede realizar la ejecución de llamadas telefónicas de manera automática, para ello el motor "CALLER" consulta su base de datos de carga la cual le indica a quien llamar, cuando el cliente responde la llamada se inicia un proceso de interacción entre el IVR y el cliente, el IVR es capaz de reproducir grabaciones de voz que contienen preguntas e interpretar las respuestas cortas del cliente. En muchas ocasiones el cliente no responde la llamada, corta la misma, también hay escenarios en los cuales el IVR no es capaz de interpretar correctamente las respuestas de voz del cliente o simplemente el cliente se niega a responder las preguntas. Para estos escenarios, la dirección de negocio de CENCOSUD CHILE desea contar con un sistema que le permita seguir con el procedimiento de cobranza mediante otros canales de comunicación los cuales se identificaron como: EMAIL, MENSAJERÍA DE TEXTO SMS Y WHATSAPP. Así, por ejemplo, el canal de comunicación SMS debe ser capaz de identificar aquellos intentos fallidos de cobranza realizadas mediante el IVR y enviar un mensaje de texto conteniendo un link de redirección hacia una página web que permita registrar al cliente un compromiso de pago. Al igual que el procedimiento de cobranza mediante el IVR, existe la posibilidad que el intento de cobranza vía SMS fracase, ya sea porque el cliente ignore el SMS no abriendo el link o no se comprometa mediante la página web a realiza el pago por vencer, para estos casos se requiere de un tercer intento de cobranza mediante el canal de comunicación WHATSAPP y también de un cuarto intento de cobranza mediante el canal de comunicación EMAIL para aquellos intentos de cobranza vía WHATSAPP fallidos. La manera en que interactúan los diferentes canales de comunicación debe ser configurable; es decir, el orden de ejecución de los procedimientos de cobranza mediante los diferentes canales de comunicación puede variar de acuerdo a las necesidades del cliente.

Adicionalmente – y esto es importante- el EQUIPO DE PLATAFORMA desarrollará un nuevo diseño de IVR en caso se desee que el cliente que contesta la llamada interactúe con un BOT programado.”.

3.2.1.5 EVIDENCIAS SOBRE LAS EXPERIENCIA.-

3.2.1.5.1 TICKET BILLET.-

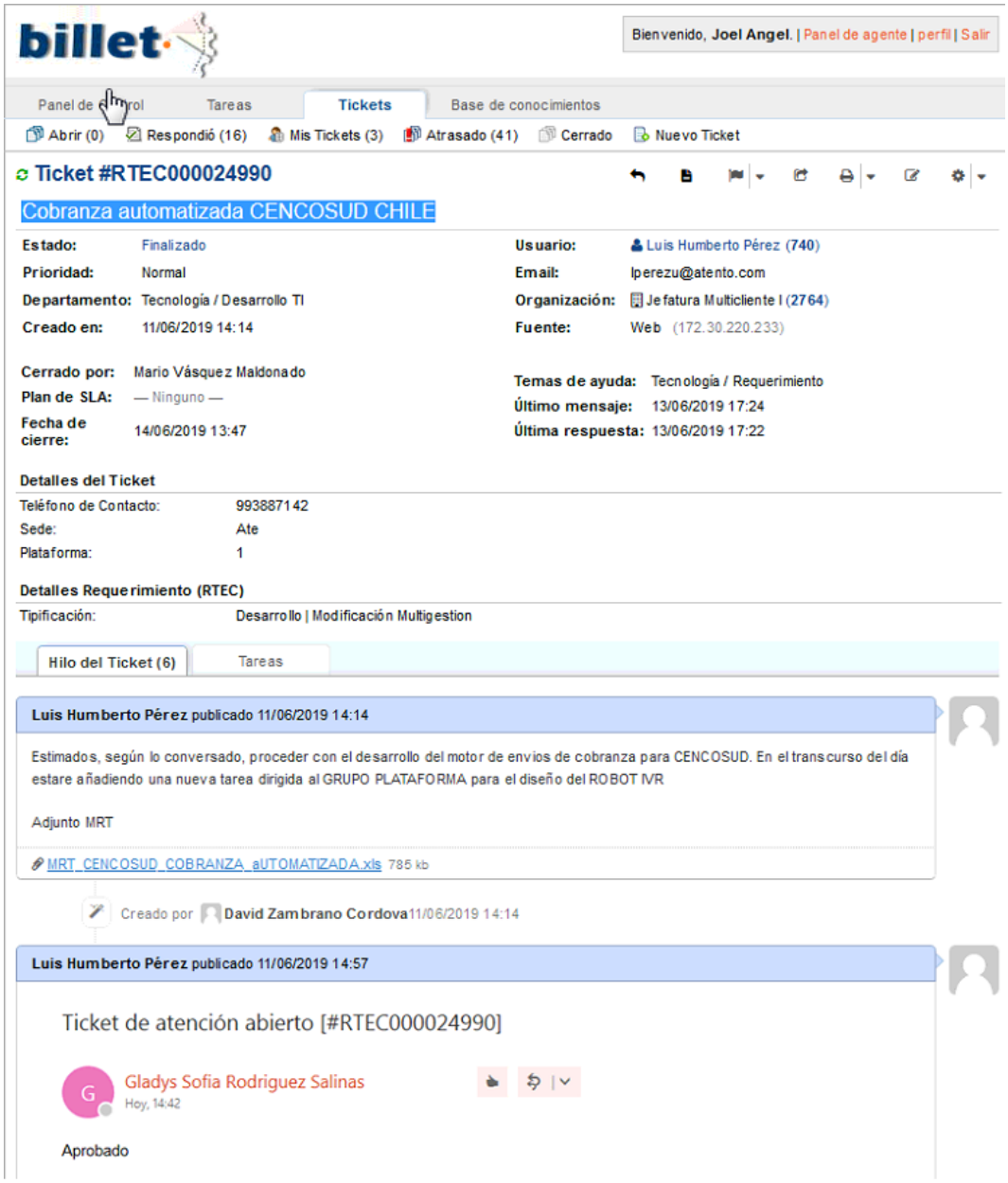


Figura 22 TICKET BILLET DEL PROYECTO: ELABORADO POR LUIS PEREZ

Fuente: (Atento Perú, 2019)

3.2.1.5.2 MRT.-

ATENTO:

MODIFICACIÓN AL REQUERIMIENTO DE TRABAJO Atento Perú

CLIENTE CONTRATANTE

Código Unidad de Servicio:	18342
Nombre Unidad de Servicio:	CENCOSUD CHILE COBRANZA}
Nombre de la Jefatura:	JEFATURA MULTICLIENTE
Nombre de la Gerencia:	MS
Usuario Solicitante del requerimiento (DNI + Nombres):	PMO LUIS HUMBERTO PEREZ
Celular / Correo	LPEREZU@atento.com
Tipo de Solicitud	Requerimiento

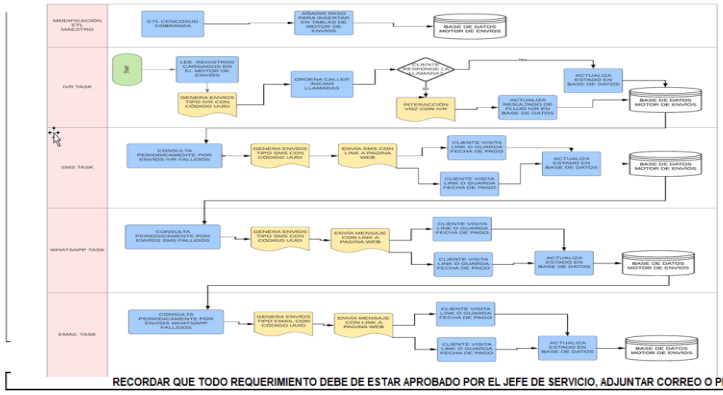
MODIFICACION AL REQUERIMIENTO DE TRABAJO

Descripción a detalle de la solicitud.
considerar referencias y anexo, de ser el caso, como pestañas dentro del presente archivo.

Titulo de la solicitud	Implementación Motor de cobranza para CENCOSUD CHILE
Porque motivo lo esta solicitando	Cambios por nuevos procedimientos.

Aquí brindar datos de su solicitud

Estimados
Por favor proceder con urgencia el desarrollo del Motor de Cobranza para CENCOSUD CHILE
De lo conversado en la última reunión que tuvimos, adjunto diagrama de procesos resumen.



RECORDAR QUE TODO REQUERIMIENTO DEBE DE ESTAR APROBADO POR EL JEFE DE SERVICIO, ADJUNTAR CORREO O PRINT EN LA WEB

JEFE (Nombres + EMail): Gladys Rodriguez - groriguezs@atento.com	GERENTE: (Gerente de Negocios)
12-Jun	

Figura 23 MRT DEL PROYECTO: ELABORADO POR LUIS PEREZ

Fuente: (Atento Perú, 2019)

3.2.1.5.3 CRONOGRAMA DE ACTIVIDADES.-

	Nombre	Duracion	Inicio	Terminado	Predecesores	Nombres del Recurso
34	**Motor Cobranza Automatizado**	**24,143 day...**	**18/05/2019 08:00**	**5/07/2019 09:08**		
2	**PLANIFICACION**	**5 day**	**18/05/2019 08:00**	**23/05/2019 17:00**		
3	Kick Off	3 day	18/05/2019 08:00	21/05/2019 17:00		PMO- Luis Pérez
4	Levantamiento de Información	2 day	18/05/2019 08:00	21/05/2019 17:00	3	PMO- Rosa Bada
5	**Grupo CRM**	**29 days**	**3/06/2019 08:00**	**4/07/2019 17:00**		
6	Modificación proceso de carga	2 days	**3/06/2019 08:00**	**5/06/2019 08:00**		Joel Quispe
7	Modificación de configurador/implementación	4 days	5/06/2019 08:00	9/06/2019 08:00		Soluciones - Mario Vásquez
8	Cliente web service INFOBIP	2 days	9/06/2019 08:00	11/06/2019 17:00		Soluciones - Mario Vásquez
9	Implementación Twilio Api	5 days	11/02/2019 17:00	16/06/2019 17:00		Soluciones - Mario Vásquez
10	Desarrollo de motor de cobranza	7 days	16/06/2019 17:00	23/06/2019 17:00		Soluciones - Mario Vásquez
11	Desarrollo de landing page	5 days	23/06/2019 17:00	4/07/2019 17:00		Soluciones - Mario Vásquez
12	Integración con caller	4 days	23/06/2019 17:00	4/07/2019 17:00		Soluciones - Mario Vásquez
13	**PLATAFORMA**	**13 days**	**8/07/2019 08:00**	**21/07/2019 17:00**		
14	Diseño Robot IVR	5 days	8/07/2019 08:00	13/07/2019 17:00		Plataforma - Carlos Gonzales
15	Integración Motor de cobranza	3 day	13/07/2019 17:00	16/07/2019 17:00		Plataforma - Carlos Gonzales
16	Integración caller	3 days	16/07/2019 17:00	19/07/2019 17:00		Plataforma - Carlos Gonzales
17	Configuración AVAYA	2 days	19/07/2019 17:00	21/07/2019 17:00		Plataforma - Carlos Gonzales
18	**BILLET**	**1 day**	**3/06/2019 08:00**	**3/06/2019 08:00**	**5;11**	
18	**BILLET**	**1 day**	**3/06/2019 08:00**	**3/06/2019 08:00**	**5;11**	
19	Registro de billet	1 day	3/06/2019 08:00	3/06/2019 08:00		PMO- Luis Pérez
20	**CRM 1RA LINEA**	**2 day**	**8/07/2019 08:00**	**10/07/2019 17:00**	**16**	
21	Configuración de temáticos	1 day	8/07/2019 08:00	8/07/2019 08:00		Soluciones - Luis Espino
22	Configuración de agendas	1 day	9/07/2019 08:00	9/07/2019 08:00		Soluciones - Luis Espino
23	**CREACION DE CAMPAÑA**	**2 days**	**8/07/2019 08:00**	**10/07/2019 17:00**	**16**	
24	Configuración de proceso de carga caller	1 days	8/07/2019 08:00	8/07/2019 08:00		Grupo AS - Hugo Balabarca
25	Configuracion de campaña caller	1 days	8/07/2019 08:00	8/07/2019 08:00		Grupo AS - David Barreto
26	Creación de Usuarios	1 days	9/07/2019 08:00	9/07/2019 08:00		Grupo AS - David Barreto
27	**Grupo AS**	**2 days**	**10/07/2019 08:00**	**12/07/2019 17:00**	**24**	
28	Configuracion de la campaña en CALLER	2 days	10/07/2019 08:00	12/07/2019 17:00		Grupo AS- Aaron Guerrero
29	**PRUEBAS**	**2 days**	**13/07/2019 08:00**	**15/07/2019 08:00**	**16;18;24;28**	
30	Validación de Proceso de Carga	1 day	13/07/2019 08:00	13/07/2019 08:00		Operaciones - Luis Gutierrez
31	Demo con gerencia CENCOSUD CHILE	2 day	13/07/2019 08:00	14/07/2019 08:00		Soluciones - Luis Espino, Mario
32	**REPORTE**	**5 days?**	**15/07/2019 08:00**	**21/07/2019 17:00**	**30**	CIA - Felix Hernandez

CENCOSUD AUTOMATIZACIÓN DE COBRANZA

	Nombre	Duracion	Inicio	Terminado	Predecesores	Nombres del Recurso
33	**VALIDACION**	**1 day**	**21/07/2019 08:00**	**21/07/2019 08:00**	**34**	
34	Validación de reporte	1 day	21/07/2019 08:00	21/07/2019 08:00		Operaciones - Luis Gutierrez
40	**PRODUCCION**	**0,143 days**	**17/07/2019 08:00**	**17/07/2019 08:00**		
41	Salida a Produccion	0,143 days	17/07/2019 08:00	17/07/2019 08:00		PMO- Rosa Bada ;Soluciones - Lui...

Figura 24 CRONOGRAMA DE ACTIVIDADES DEL PROYECTO: ELABORADO POR LUIS PEREZ

Fuente: (Atento Perú, 2019)

3.2.1.5.4 DISEÑO ROBOT IVR.-

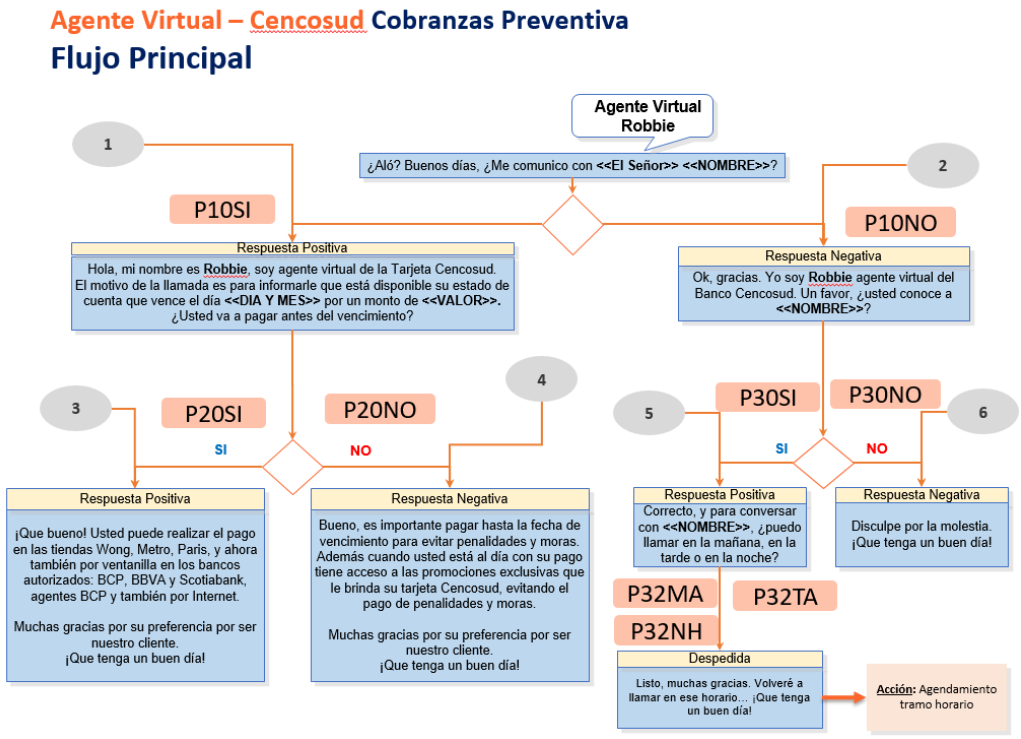


Figura 25 DISEÑO ROBOT IVR: ELABORADO POR EQUIPO DE PLATAFORMA

Fuente: (Atento Perú, 2019)

3.2.2 ANÁLISIS Y DISEÑO.-

En mi trabajo anterior a ATENTO PERÚ, es decir, en la consultora de software CONASTEC, la fase de análisis y diseño comprendía la elaboración de un documento funcional a cargo del analista funcional, y que abarcaba la narración de flujos del negocio, requerimientos funcionales, modelo, especificación y diagrama de casos de uso, diagrama de actores y de paquetes, diagrama de clases de Análisis, entre otros; además, adicionalmente y no menos importante, siempre era requerido la elaboración de un documento técnico a cargo del líder técnico, en este documento se detallaba la estructura y organización del código java con respecto a las diferentes capas de la aplicación: capa DAO o de persistencia, que comprendía la definición de las interfaces y clases que implementaban las mismas para la interacción con la base de datos; capa de SERVICIO que comprendía la definición de interfaces y sus clases de implementación utilizadas para definir en código la lógica del negocio, también comprendía la documentación de la capa de controlador y vista; en caso el desarrollo comprendía la implementación de un webservice o cliente webservice, en el documento técnico se detallaba las URL del servicio, tipo de webservice(SOAP o REST), métodos expuestos, parámetros y tipo de parámetros de los métodos expuestos, respuestas esperadas, entre otros.

En contraste con ATENTO PERÚ, la fase de ANÁLISIS Y DISEÑO para un desarrollo de software es inexistente en cuanto a documento funcional se refiere, por tanto el alcance del sistema y los requerimientos a nivel funcional pueden ir variando durante la etapa de construcción, lo que quiere decir que el análisis y diseño del sistema puede incluso extenderse hasta la FASE DE CONSTRUCCIÓN DEL SOTFWARE y llevarse a cabo en paralelo.

La fase de ANÁLISIS Y DISEÑO en ATENTO PERÚ, tiene su punto de inicio justo después de que el correspondiente TICKET BILLET generado por el PMO responsable, se deriva al GRUPO CRM y se asigna un analista para su implementación, luego se lleva a cabo una reunión entre el líder de equipo y el analista a cargo del desarrollo, el líder de equipo expone la definición del requerimiento y el alcance, en conjunto con el analista de sistemas se define, por los general, los siguientes puntos técnicos:

a. La arquitectura del sistema, es decir, si el desarrollo seguirá el estándar de arquitectura modular OSGI o si en cambio se materializara mediante una arquitectura web.
b. Las tablas de bases de datos transaccionales involucrados, si se crearan nuevas tablas, los STORE PROCEDURES a modificar o a crear.
c. Definición de procesos de cargas SSIS que alimentaran las tablas transaccionales.
d. Forma de interactuar con la base de datos ya sea por intermedio de un middleware o mediante conexión JDBC directa.
e. En caso se requiera implementar un cliente webservice se discute y define donde se desarrollará el cliente webservice: en base de datos –si, en ATENTO PERÚ aún se suelen consumir webservices desde base de datos Oracle o SQL SERVER- o aplicación java, entre otros aspectos técnicos.

3.2.2.1 ¿CÓMO SE REALIZÓ EL ANÁLISIS Y DISEÑO DEL SISTEMA?

La fase de ANÁLISIS Y DISEÑO en ATENTO PERÚ, tiene su punto de inicio justo después de que el correspondiente TICKET BILLET generado por el PMO responsable, se deriva al GRUPO CRM y se asigna un analista para su implementación, luego se lleva a cabo una reunión entre el líder de equipo y el analista a cargo del desarrollo, el líder de equipo expone la definición del requerimiento y el alcance, en conjunto con el analista de sistemas se define, por los general, los siguientes puntos técnicos:

a. La arquitectura del sistema, es decir, si el desarrollo seguirá el estándar de arquitectura modular OSGI o si en cambio se materializara mediante una arquitectura web.
b. Las tablas de bases de datos transaccionales involucrados, si se crearan nuevas tablas, los STORE PROCEDURES a modificar o a crear.
c. Definición de procesos de cargas SSIS que alimentaran las tablas transaccionales.

d. Forma de interactuar con la base de datos ya sea por intermedio de un middleware o mediante conexión JDBC directa.

e. En caso se requiera implementar un cliente webservice se discute y define donde se desarrollará el cliente webservice: en base de datos –si, en ATENTO PERÚ aún se suelen consumir webservices desde base de datos Oracle o SQL SERVER- o aplicación java, entre otros aspectos técnicos.

3.2.2.1.1 BOSQUEJO DE ARQUITECTURA DEL SISTEMA.-

Antes de comenzar a describir los aspectos técnicos y de diseño discutidos durante esta fase de análisis y diseño del sistema que definieron la arquitectura del sistema, es menester presentar el siguiente bosquejo de la arquitectura del sistema –de autoría propia- el cual trata de ilustrar el enunciado del proyecto definido en la sección “3.3.4 Enunciado del proyecto”:

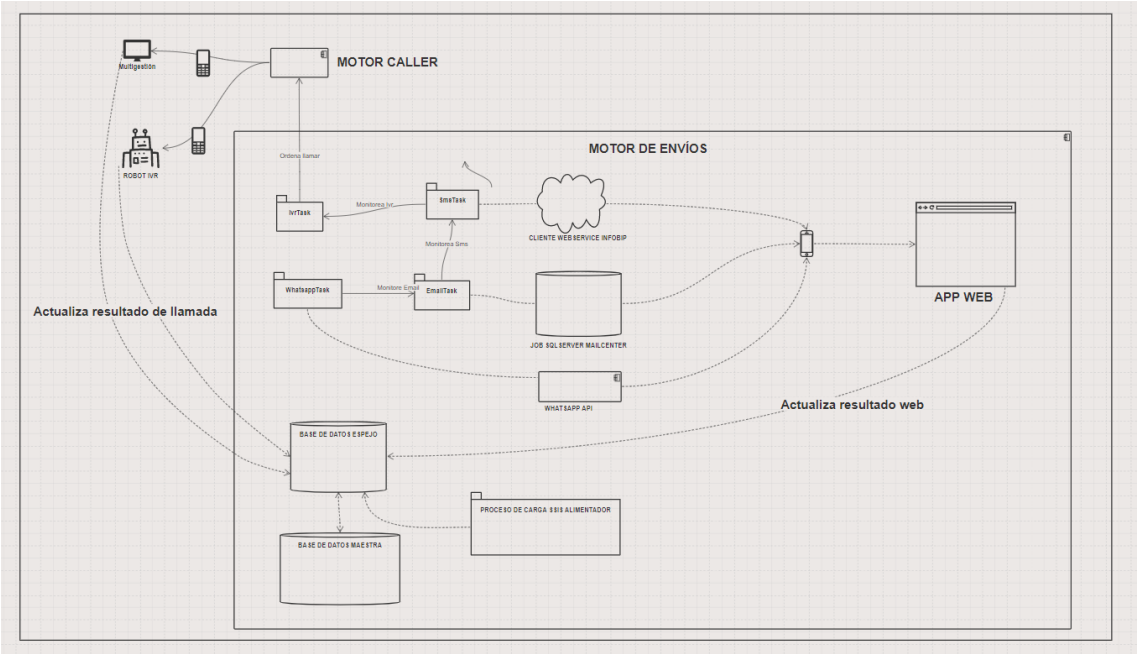


Figura 26 BOSQUEJO DE ARQUITECURA DEL SISTEMA

Fuente: (Fuente de Autoría propia, 2020)

El MOTOR DE ENVÍOS realiza el proceso de cobranza o envío de ofertas mediante cuatro canales de comunicación: Llamadas telefónicas, mensajes de texto SMS, envío de correos electrónicos y mensajería Whatsapp. Estos canales de comunicación conviven e interactúan entre sí. Dependiendo de la necesidad del cliente contratante, el proceso de cobranza podría utilizar uno o más canales de comunicación –no necesariamente la cobranza debe utilizar todos los canales de comunicación- y el orden de interacción entre los canales puede variar dependiendo del cliente, esto es, por ejemplo, cierto cliente de ATENTO PERÚ podría requerir utilizar únicamente los canales de mensajes de texto SMS y mensajería Whatsapp para la cobranza, iniciando este proceso enviando SMS mientras que el canal de comunicación Whatsapp, en segundo plano, monitorea todos aquellos envíos de SMS que no hayan obtenido el resultado esperado para un reintento de cobranza vía Whatsapp.

En el bosquejo de arquitectura de la imagen x, se ilustra un proceso de cobranza completo que incluye los cuatro canales de comunicación. El proceso de cobranza inicia con el "disparo de llamadas" ordenado desde el canal de comunicación de ***Llamadas telefónicas*** (***de ahora en adelante llamado IVRTAKS***). En paralelo se encuentra en funcionamiento el canal de comunicación de ***Mensajería SMS (de ahora en adelante llamado SMS TAKS)***, realizando el monitoreo del canal de comunicación IVR TASK identificando todas aquellas llamadas que no hayan obtenido un resultado exitoso para realizar el reintento de cobranza por **SMS;** los mensajes de texto contienen un link hacia una aplicación web que permite visualizar deudas y registrar fechas de pago. También, en paralelo, se encuentra en funcionamiento el canal de comunicación de ***"Correos Electrónicos" (de ahora en adelante llamado EMAILTASK)***, que se encuentra monitoreando el canal de comunicación SMSTASK para los reintentos de cobranza de todos aquellos ***envíos de correos electrónicos*** que no hayan obtenido un resultado exitoso. Por último, tenemos en funcionamiento al canal de comunicación ***"Mensajería Whatsapp" (de ahora en adelante llamado WHATSAPPTASK)*** el cual se encuentra monitoreando el canal de comunicación ***EMAILTASK***.

3.2.2.1.2 ELECCIÓN DE ARQUITECTURA PARA EL MOTOR DE ENVÍOS.-

La arquitectura modular OSGI permite dividir un sistema en módulos totalmente independientes que pueden interactuar entre sí en tiempo de ejecución; esto se logra gracias a que OSGI proporciona a cada módulo su propio classpath separado del resto de classpath de los demás módulos. La instalación, arranque, parada, actualización y desinstalación de cada BUNDLE (módulo) se realizan dinámicamente en tiempo de ejecución sin tener que detener por completo la plataforma. Para el MOTOR DE ENVÍOS, cada canal de comunicación es independiente y, dependiendo de la necesidad del cliente contratante, pueden entrar en acción uno o más canales de comunicación; evidentemente la arquitectura OSGI encaja perfecto para la implementación del MOTOR DE ENVÍOS.

En reunión con el líder de equipo, se acordó basar el desarrollo del MOTOR DE ENVÍOS sobre la ya conocida – y ampliamente usada en los desarrollos- arquitectura OSGI. Se definieron los siguientes BUNDLES o módulos:

a. Bundle SCHEDULERTASK
b. Bundle IVRTASK
c. Bundle SMSTASK
d. Bundle EMAILTASK
e. Bundle WHATSAPPTASK

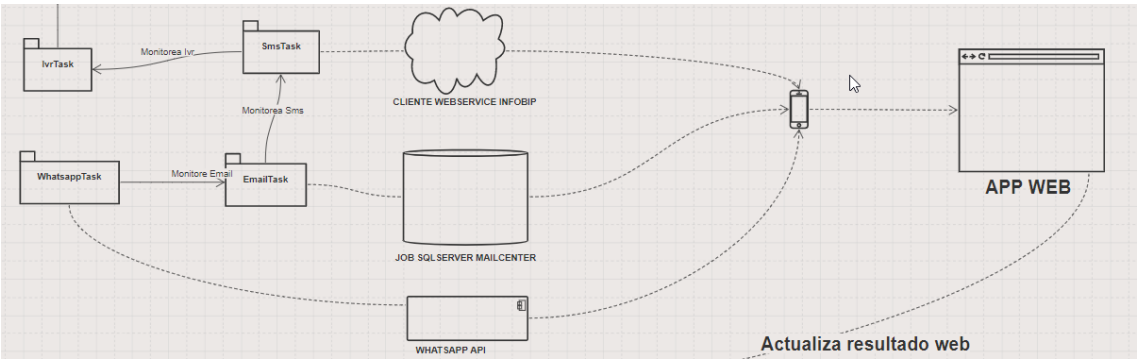


Figura 27 BUNDLES

Fuente: (Fuente de Autoría propia, 2019)

3.2.2.1.3 BUNDLE SCHEDULER.-

El bundle SCHEDULER será el encargado de leer las tablas de configuración para identificar los BUNDLES requeridos para un cierto cliente e instalarlos a demanda. También albergará la interfaz de usuario que permitirá monitorear el progreso de cobranza de cada canal de comunicación.

Este bundle, luego de leer las tablas de configuración- deberá ser el encargado de crear un nuevo Thread por cada canal de comunicación configurado. Además, este BUNDLE iniciara Threads adicionales para la actualización automática de nuevos canales de configuración y destrucción de Thread inactivos.

3.2.2.1.4 BUNDLE IVRTASK.-

Este bundle será el encargado de ordenar al **Motor Caller** proceder con las llamadas de cobranza, para ello será necesario invocar un procedimiento almacenado que active el barrido de las tablas del Motor Caller, generando llamadas telefónicas derivándolas al aplicativo MULTIGESTIÓN o al ROBOT IVR. Tanto si la llamada es derivada al MULTIGESTIÓN o al ROBOT IVR, se tendrá que proporcionar un procedimiento almacenado que actualice las tablas del motor de envíos con el resultado de las llamadas.

3.2.2.1.5 BUNDLE SMSTASK.-

Este bundle es el encargado del envío de SMS, para ello es necesario consumir el WebService Rest de INFOBIP para él envió de mensaje de textos. El cuerpo del mensaje contendrá un link hacia una aplicación web que actualizará las tablas del motor de envíos con el resultado de la cobranza.

3.2.2.1.6 BUNDLE EMAILTASK.-

Encargado de realizar el envío de correos electrónicos, para lo cual se hará uso de las tablas de colas del MAILCENTER para la inserción en base de datos de los correos electrónicos a enviar, delegando al JOB SQLSERVER del MAILCENTER el envió de los correos. El cuerpo del mensaje contendrá un link hacia una aplicación web que actualizará las tablas del motor de envíos con el resultado de la cobranza.

3.2.2.1.7 BUNDLE WHATSAPPTASK.-

Para el envió de mensajería WHATSAPP se hará uso del api TWILIO para el envío de mensajes vía WHATSAPP. El cuerpo del mensaje contendrá un link hacia una aplicación web que actualizará las tablas del motor de envíos con el resultado de la cobranza.

3.2.2.1.8 SOBRE LA APLICACIÓN WEB.-

La gran cantidad de sistemas dentro del ecosistema tecnológico de ATENTO PERÚ son aplicaciones JAVA DE ESCRITORIO basados en Java Swing; los desarrollos webs son pocos.

Valiéndome de mi experiencia en desarrollo web bajo la arquitectura Java Server Faces, Spring e Hibernate adquirida durante mi permanencia laboral en la consultora de software CONASTEC, fue inevitable aplicar estas mismas tecnologías en el desarrollo de la Aplicación Web que permitiera visualizar las deudas de un cliente y realizar el registro de las fechas de pago de las mismas.

Brevemente adelantaré – y esto será detallado en la sección de “Construcción” de este informe- que el autor de este informe implemento una librería java que permite hacer el binding de cada componente definido en las vistas o páginas “.xhtml” con objetos java que se asemejan a los componentes de interfaz de usuario Java Swing pero en versión Web, permitiendo tener la capa “Vista/Controlador” del sistema más organizada, entendible y fácil de mantener. Cabe mencionar que el contenedor de aplicaciones web a utilizar será APACHE TOMCAT 8.0

3.2.2.2 SOBRE LA ESTRUCTURA DE TABLAS DEL MOTOR DE ENVÍOS.-

La estructura de tablas del motor de envíos son una réplica o “espejo” de las tablas maestras transaccionales usadas por el aplicativo MULTIGESTIÓN, pero con una estructura más simplificada en cuanto al número de columnas (las tablas maestras pueden llegar a tener más de 100 columnas) y relaciones de tablas, esto con la finalidad de almacenar información relevante que será usada por los diferentes canales de comunicación: Nombre del cliente, número de teléfono, monto de deuda, correo electrónico, entre otros.

Las tablas maestras transaccionales constantemente están sometidas a estrés pues cada minuto están siendo consultadas por el aplicativo MULTIGESTIÓN el cual a su vez se encuentra en funcionamiento es más de diez mil (10, 000) puestos de asesor. Por este motivo, se eligió crear una estructura de tablas “Espejo” simplificada para evitar aumentar el estrés de las tablas maestras.

3.2.2.3 SOBRE LA FORMA DE ESTABLECER CONEXIÓN A LA BASE DE DATOS.-

El aplicativo MULTIGESTIÓN, así como la gran mayoría de aplicativos de gestión de ATENTO PERÚ, establecen conexión indirecta a base de datos por intermedio de un MIDDLEWARE. El MIDDLEWARE de ATENTO PERÚ en un WEBSERVICE SOAP QUE permite ejecutar sentencias SQL Nativas e invocar procedimientos almacenado, en su interior mantiene activo un determinado número de conexiones a las diferentes bases de datos, las conexiones abiertas a las bases de datos son reutilizadas, se mantienen activas evitando así el coste de abrir y cerrar conexiones; todo esto se logra gracias al bien conocido Pool de Conexiones Java Proxool. Sin Proxool cada uno de los 10,000 puestos de asesor donde el MULTIGESTIÓN es utilizado tendría que abrir 10,000 conexiones de base de datos, ¿Una locura, cierto?, bueno esto se evita utilizando un Pool de Conexiones como Proxool.
La conexión a base de datos por intermedio de un MIDDLEWARE WEBSERVICE permite, si bien es cierto, liberar a las bases de datos del estrés de abrir y cerrar conexiones, la resolución de una consulta se hace más lenta: se debe sumar el tiempo que demore la base de datos en resolver la consulta más el tiempo de respuesta del WEBSERVICE.

Para aquellos aplicativos de alta transaccionalidad que requieren una respuesta inmediata de la base de datos, se habilita el permiso de conexión directa a las bases de datos. Para el MOTOR DE ENVÍOS, se acordó establecer conexión a la base de datos de manera directa, esto es mediante JDBC.

3.2.2.4 SOBRE EL PROCESO SSIS DE ALIMENTACIÓN.-

Será necesario realizar la modificación del proceso de carga existente de CENCOSUD que alimenta las tablas maestras, agregando un “task” adicional que alimente las tablas simplificadas del motor de envíos.

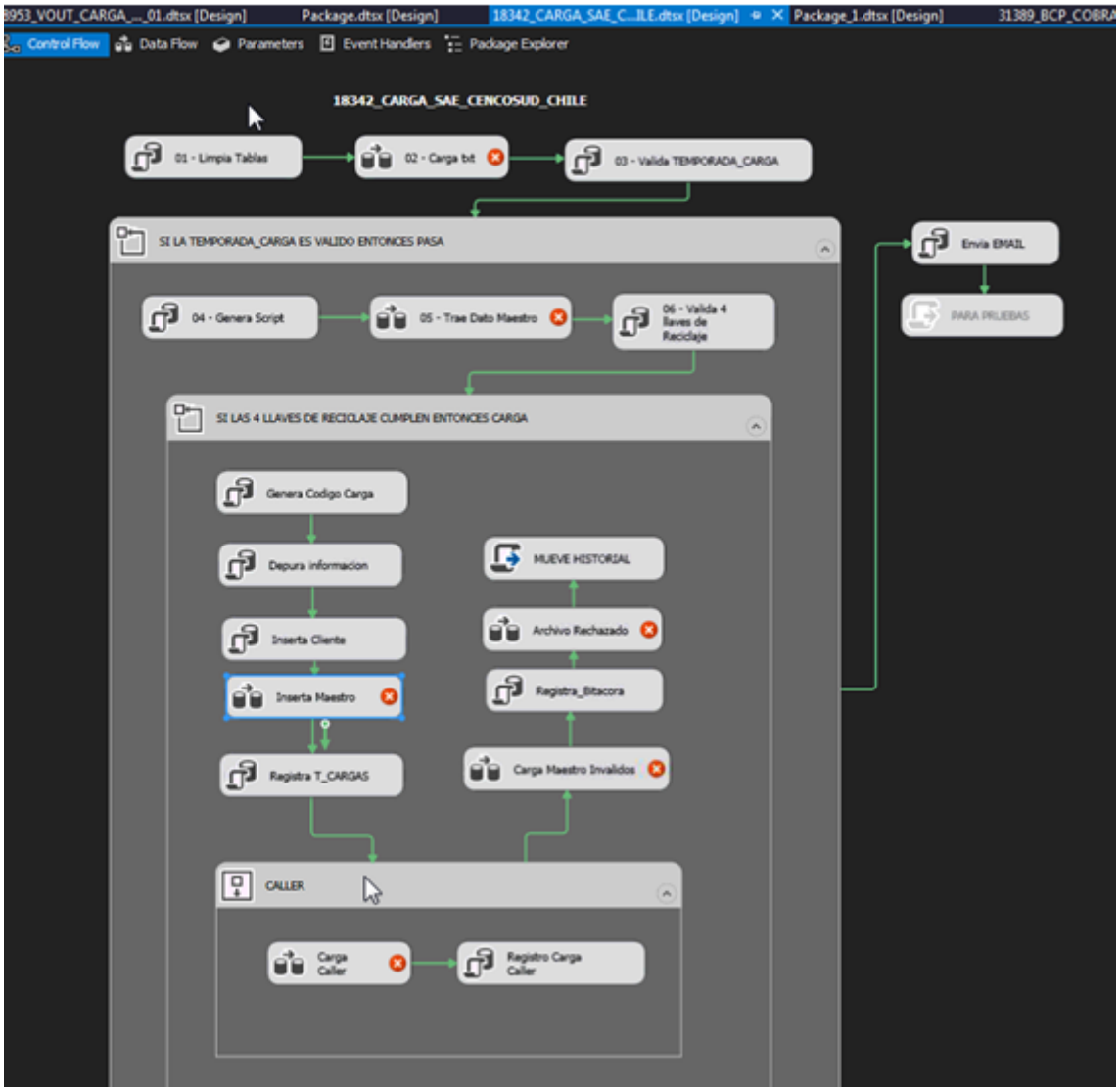


Figura 28 PROCESO DE CARGA ACTUAL

Fuente: (Atento Perú, 2019)

3.2.2.5 SOBRE CONFIGURADOR DEL MOTOR DE ENVÍOS.-

Sera indispensable añadir al CONFIGURADOR del MULTIGESTIÓN existente, una nueva opción de menú que permita, para una campaña o cliente, realizar el registro de los canales de comunicación a utilizar, definir la forma como estos interactuaran, definir el volumen de información a consumir por cada canal de comunicación, los tiempos de expiración para cada envío de cobranza entre otros aspectos técnicos (esto se explicara más a detalle en la sección de 3.5 - CONSTRUCCIÓN).

3.2.2.6 EVIDENCIAS SOBRE LA EXPERIENCIA.-

Figura 29 CONFIGURADOR ACTUAL

Fuente: (Atento Perú, 2019)

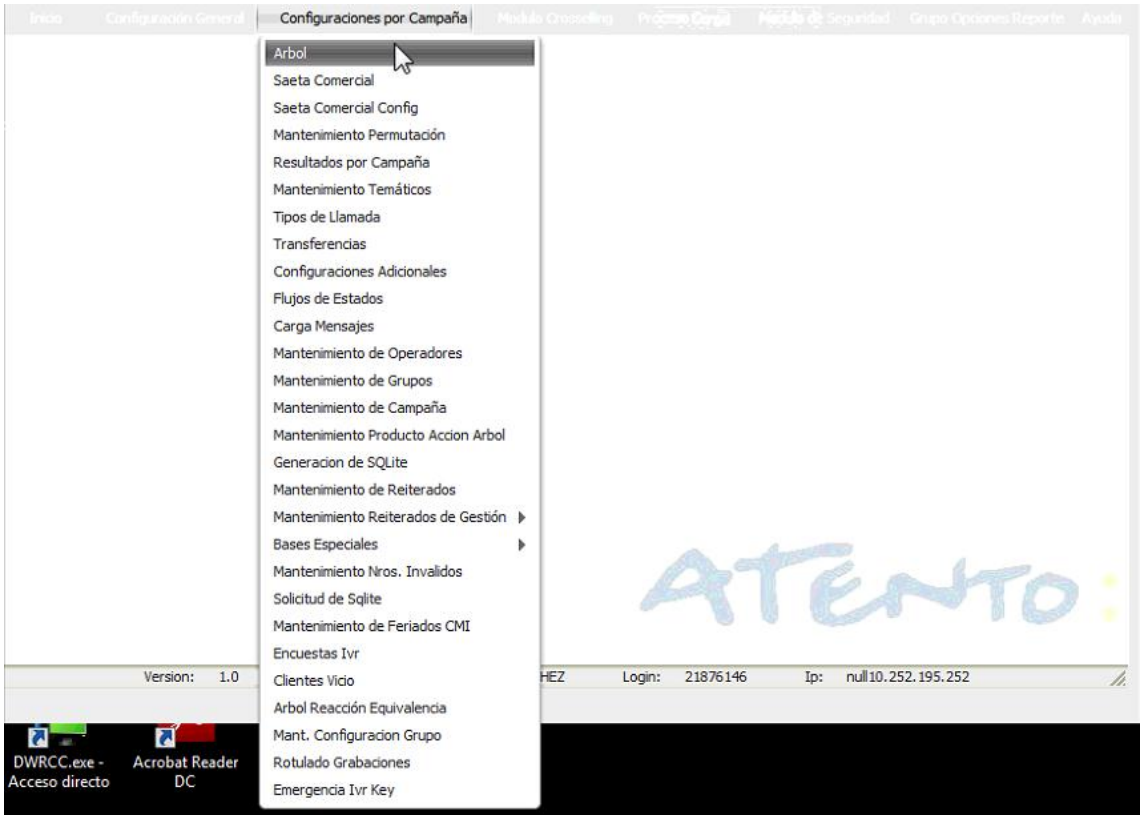


Figura 30 CONFIGURADOR ACTUAL

Fuente: (Atento Perú, 2019)

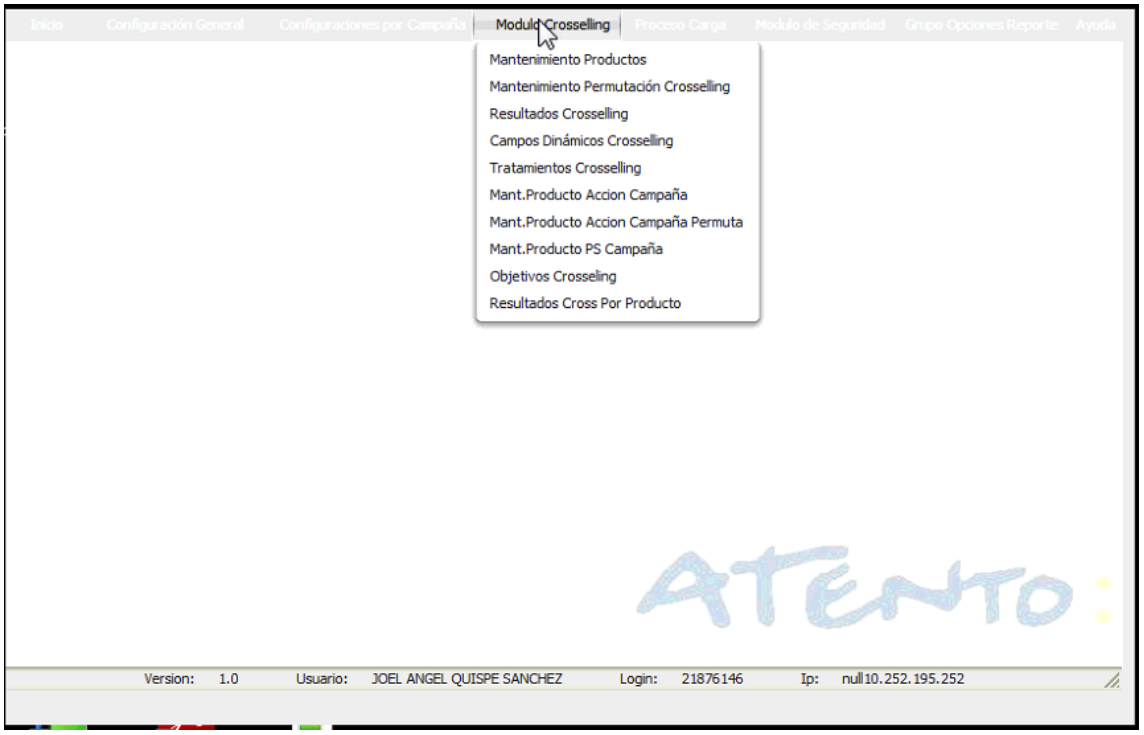


Figura 31 CONFIGURADOR ACTUAL

Fuente: (Atento Perú, 2019)

3.2.3 CONSTRUCCIÓN, “¡MANOS A LA OBRA, A PROGRAMAR!”. -

3.2.3.1 PROCESO DE CARGA MODIFICADO.-

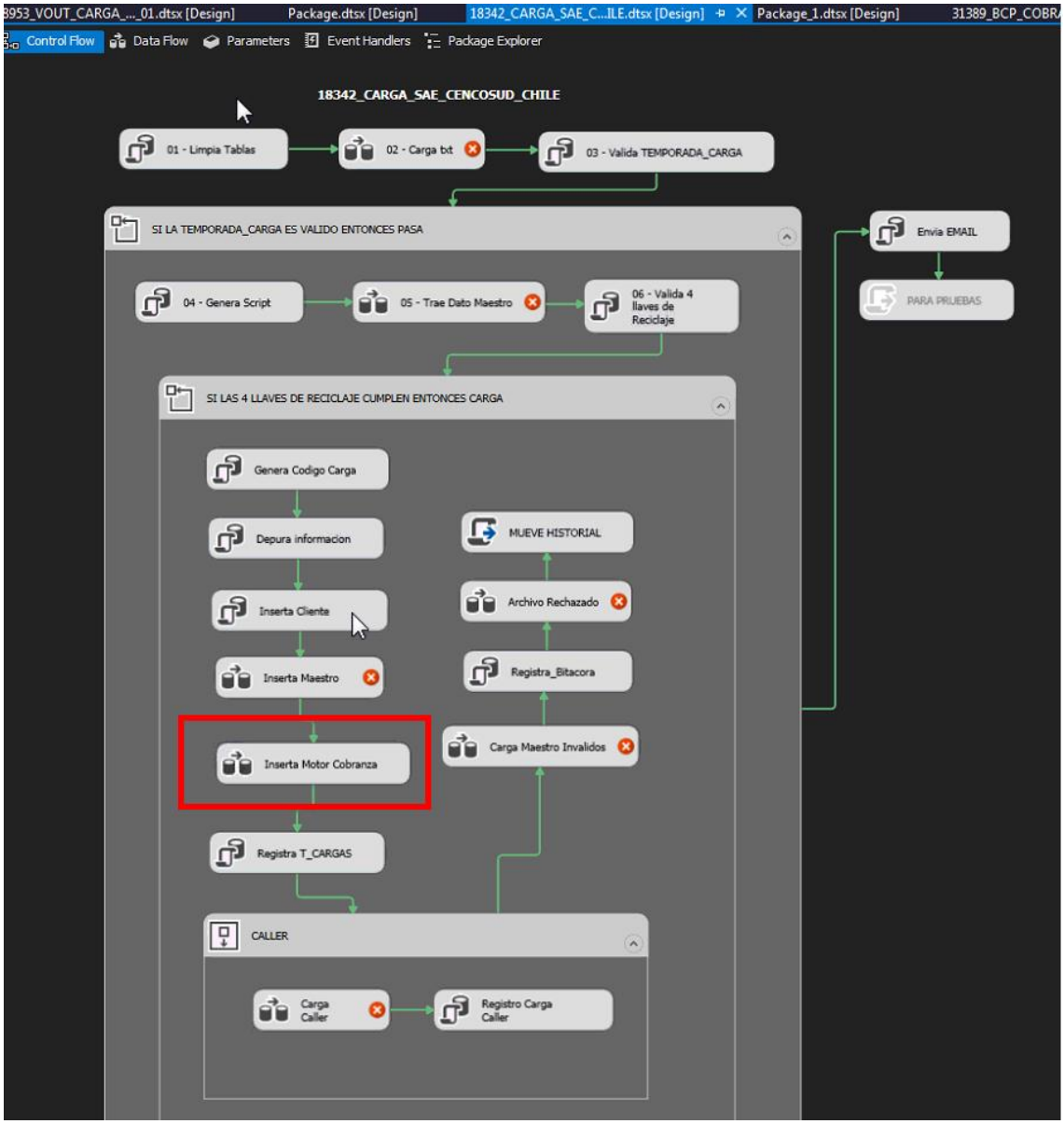


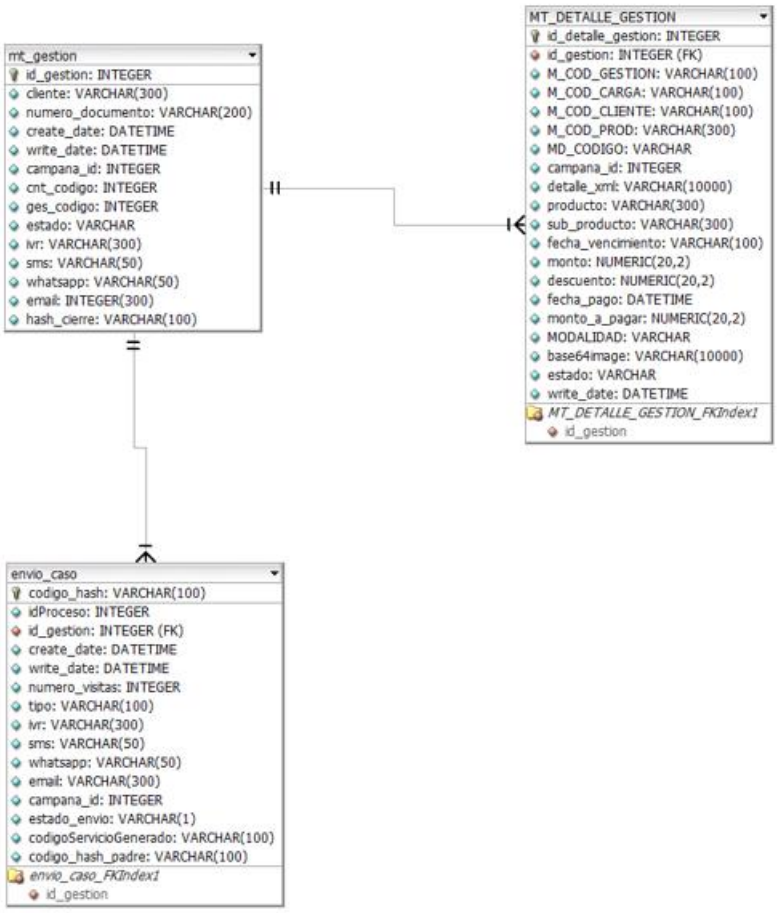
Figura 32 PROCESO DE CARGA MODIFICADO

Fuente: (Atento Perú, 2019)

3.2.3.2 ESTRUCTURA DE TABLAS.-

POBLACIÓN DE BASE DE CARGA DE MOTOR DE ENVIOS

1. **ESTRUCTURA DE TABLA.** – Los campos sombreados de color NARANJA son obligatorios al momento de invocar los procedimientos almacenados que insertan en las tablas del motor.



Estructura de tablas

Figura 33 ESTRUCTURA DE TABLAS Y DESCRIPCIÓN

Fuente: (Atento Perú, 2019)

1.1 TABLA "MT_GESTION". - Representa la gestión: Venta o Cobranza.

CAMPO	DESCRIPCIÓN
Id_gestion	Llave primaria de tabla
Cliente	Nombre de cliente
Numero_documento	Nombre de cliente
Create_date	Fecha creación de registro. Por defecto toma la fecha actual.
Write_date	Fecha de actualización de registro
Campana_id	id de campaña
Cnt_codigo	cnt código
Ges_codigo	Código de gestión
Estado	Estado de registro. Para que el registro sea tomado por el motor, enviar NULL
Ivr	Número telefónico de cliente a usar por el IVR
Sms	Número telefónico de cliente para enviar sms
Whatsapp	Número telefónico de cliente para enviar whatsapp
Email	Email de cliente
Hash_cierre	Código hash que cerro el envio.

Figura 34 ESTRUCTURA DE TABLAS Y DESCRIPCIÓN

Fuente: (Atento Perú, 2019)

1.2 TABLA "MT_DETALLE_GESTION". Representa los detallas de la venta: Productos o Deudas.

CAMPO	DESCRIPCIÓN
id_detalle_gestion	Llave primaria de tabla
id_gestion	Llave foránea(mt_gestion)
M_COD_GESTION	Código gestión maestro
M_COD_CARGA	Código carga maestro
M_COD_CLIENTE	Código cliente maestro
M_COD_PROD	Código producto maestro
MD_CODIGO	Md Código maestro
campana_id	Campaña id
detalle_xml	XML conteniendo todos los campos del maestro. Ver Ejemplo detalle XML.
producto	Nombre del producto a ofrecer o nombre de deuda.
sub_producto	Nombre del sub producto
fecha_vencimiento	Fecha vencimiento de DEUDA. Aplica solo para COBRANZA. Formato de fecha YYYY-MM-DD

Figura 35 - ESTRUCTURA DE TABLAS Y DESCRIPCIÓN

Fuente: (Atento Perú, 2019)

3.2.3.3 CONSTRUCCIÓN DE CONFIGURADOR PARA EL MOTOR DE ENVÍOS.-

El objetivo de esta etapa de la construcción, fue crear un nuevo formulario de escritorio que permita realizar el registro y mantenimiento de los canales de comunicación a usar por una determinada campaña. Este nuevo formulario seria añadido como nueva opción de menú en el CONFIGURADOR DEL MULTIGESTIÓN.

3.2.3.3.1 ESTRUCTURA DEL CÓDIGO FUENTE ACTUAL DEL CONFIGURADOR MULTIGESTIÓN.-

La estructura del código fuente del CONFIGURADOR DEL MULTIGESTIÓN y de cualquier aplicativo JAVA desarrollado en ATENTO PERÚ, no respeta el patrón de diseño MVC. Así, se pueden identificar dos capas principales en el código fuente: una CAPA DAO para el acceso a base de datos y una "CAPA FORM" que alberga clases que cumplen el papel de vista y controlador al mismo tiempo; el uso de INTERFACES JAVA es inexistente. La capa de servicio que se supone debe contener la lógica del negocio es inexistente, por lo que la lógica del negocio se encuentra dispersa en casi cualquier parte del código.

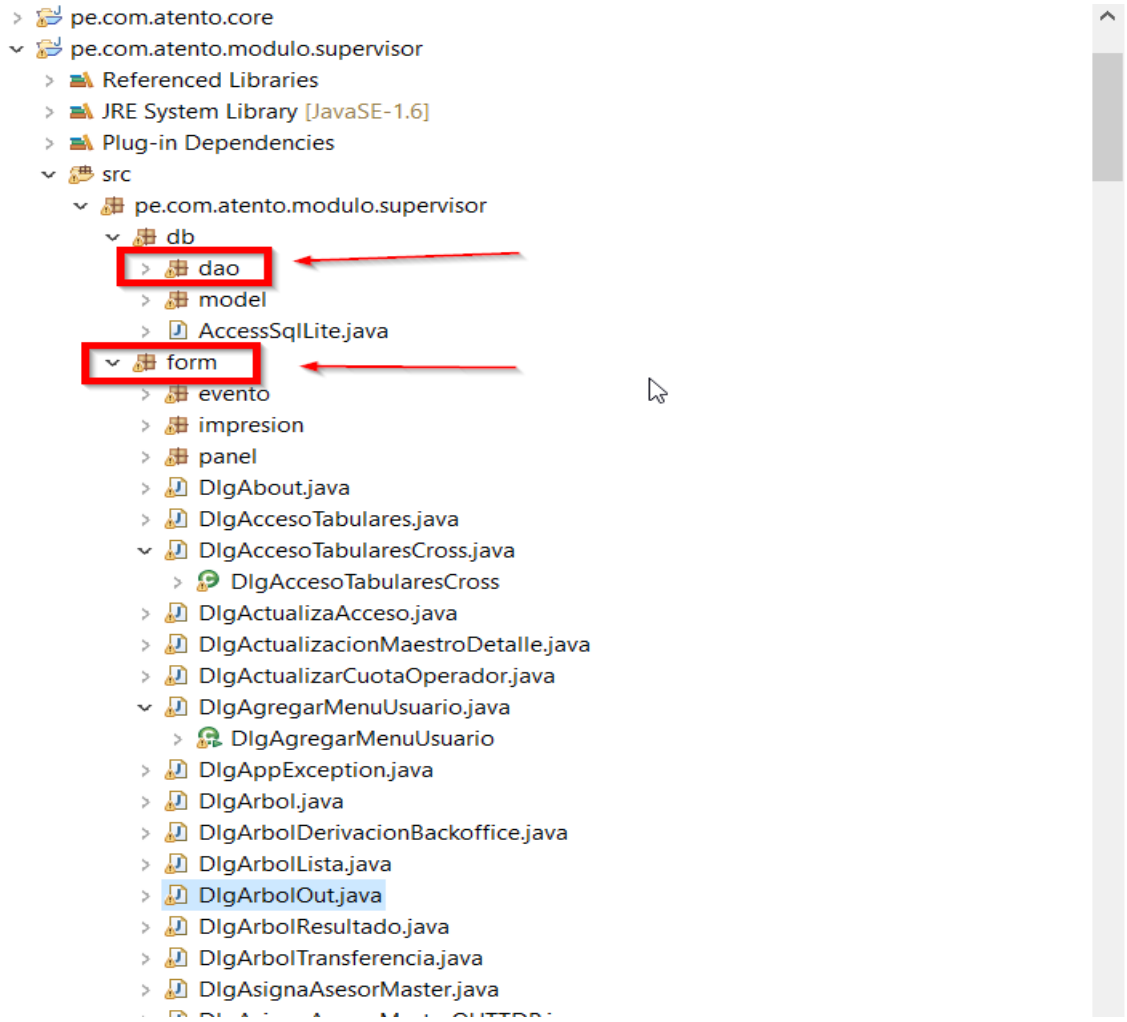


Figura 36 ESTRUCTURA CÓDIGO ACTUAL CONFIGURADOR MULTIGESTIÓN

Fuente: (Subversion SVN Atento Perú, 2019)

En la "CAPA FORM", las clases cumple el rol de vista, controlador y alberga lógico del negocio. En una misma clase se hacen las definiciones de los componentes de interfaz de usuario, se programan los eventos de estos componentes, se hacen llamados a la CAPA DAO y al mismo tiempo se mezcla lógica del negocio; es fácil imaginar lo caótico que puede llegar a ser una clase que manejo todos estos aspectos.

A continuación – y solo a manera de exponer malas técnicas de programación- se presenta la **FrmPrincipal.java** perteneciente a la "CAPA FORM", esta clase tiene nada menos que 26336 líneas de código (y sigue creciendo), en ella se ha implementado código para definir interfaces de usuario, manejo de eventos de interfaces de usuario, lógica del negocio y llamadas a bases de datos; todo en una sola clase, la frase "divide y vencerás" parece no tener significado alguno.

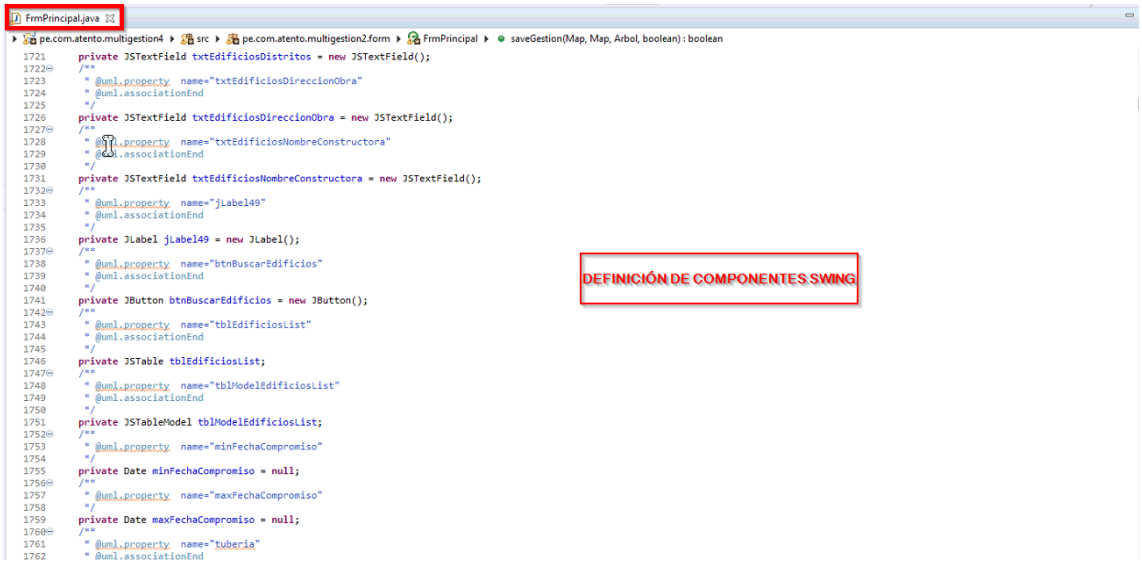


Figura 37 CLASE "FrmPrincipal" que concentra casi 80% del código del sistema en una sola clase.

Fuente: (Subversion SVN Atento Perú, 2019)

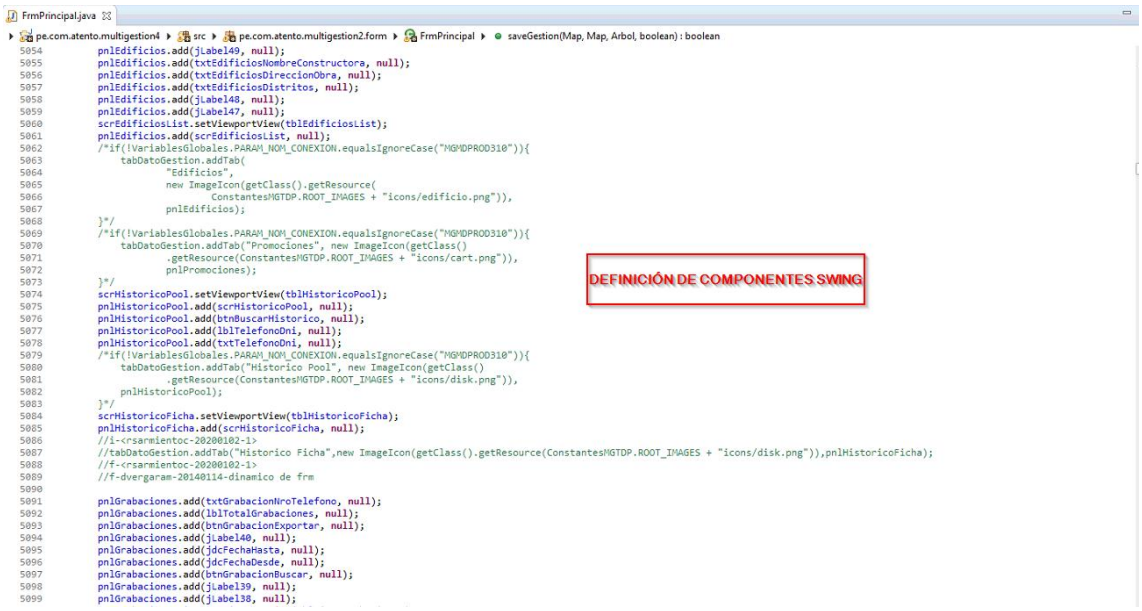


Figura 38 CLASE "FrmPrincipal" que concentra casi 80% del código del sistema en una sola clase.

Fuente: (Subversion SVN Atento Perú, 2019)

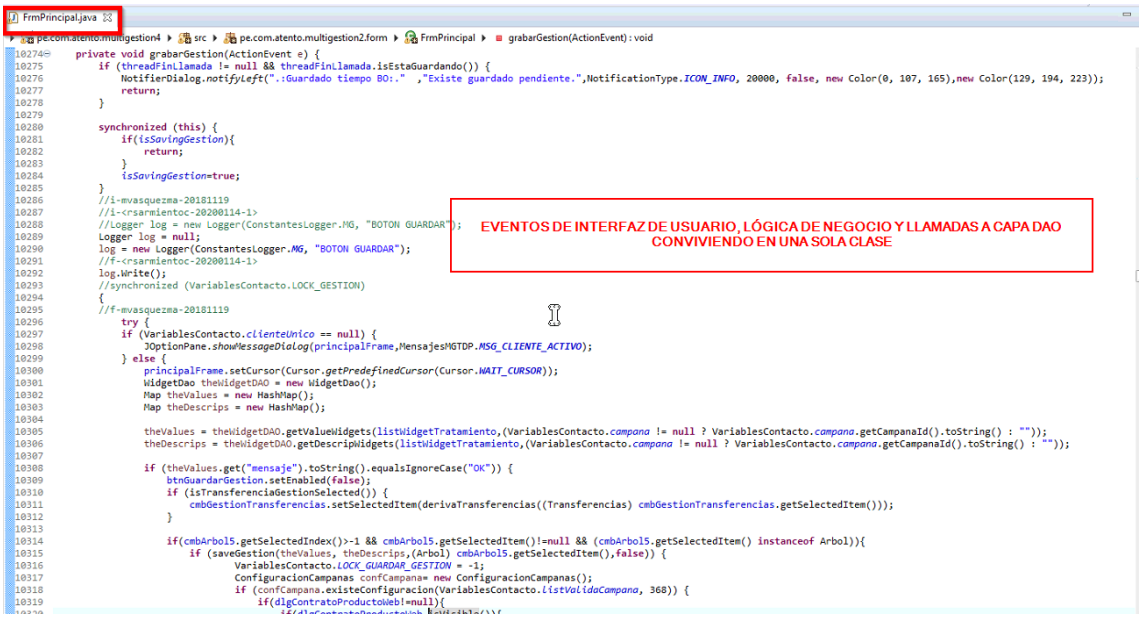


Figura 39 CLASE "FrmPrincipal" que concentra casi 80% del código del sistema en una sola clase.

Fuente: (Subversion SVN Atento Perú, 2019)

Figura 40 CLASE "FrmPrincipal" que concentra casi 80% del código del sistema en una sola clase.

Fuente: (Subversion SVN Atento Perú, 2019)

Figura 41 CLASE "FrmPrincipal" algunas muestras de malas prácticas.

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.3.2 Estructura del código fuente para el configurador del MOTOR DE ENVÍOS.-

Para implementar la funcionalidad del CONFIGURADOR DEL MOTOR DE ENVÍOS, se agregó dentro de la CAPA FORM un nuevo paquete llamado "webmotor". En este paquete se define una clase java que cumple el rol de VISTA, donde se define los componentes de interfaz de usuario; en una segunda clase java que cumple el rol de CONTROLADOR, se definen los eventos asociados a los componentes de interfaz de usuario definidos en la VISTA. Esta distribución del código, al menos para el requerimiento del CONFIGURADOR DEL MOTOR DE ENVÍOS, hace el mantenimiento del código un poco más fácil debido a la división de responsabilidad, sin embargo, la estructuración del código en general sigue siendo caótica.

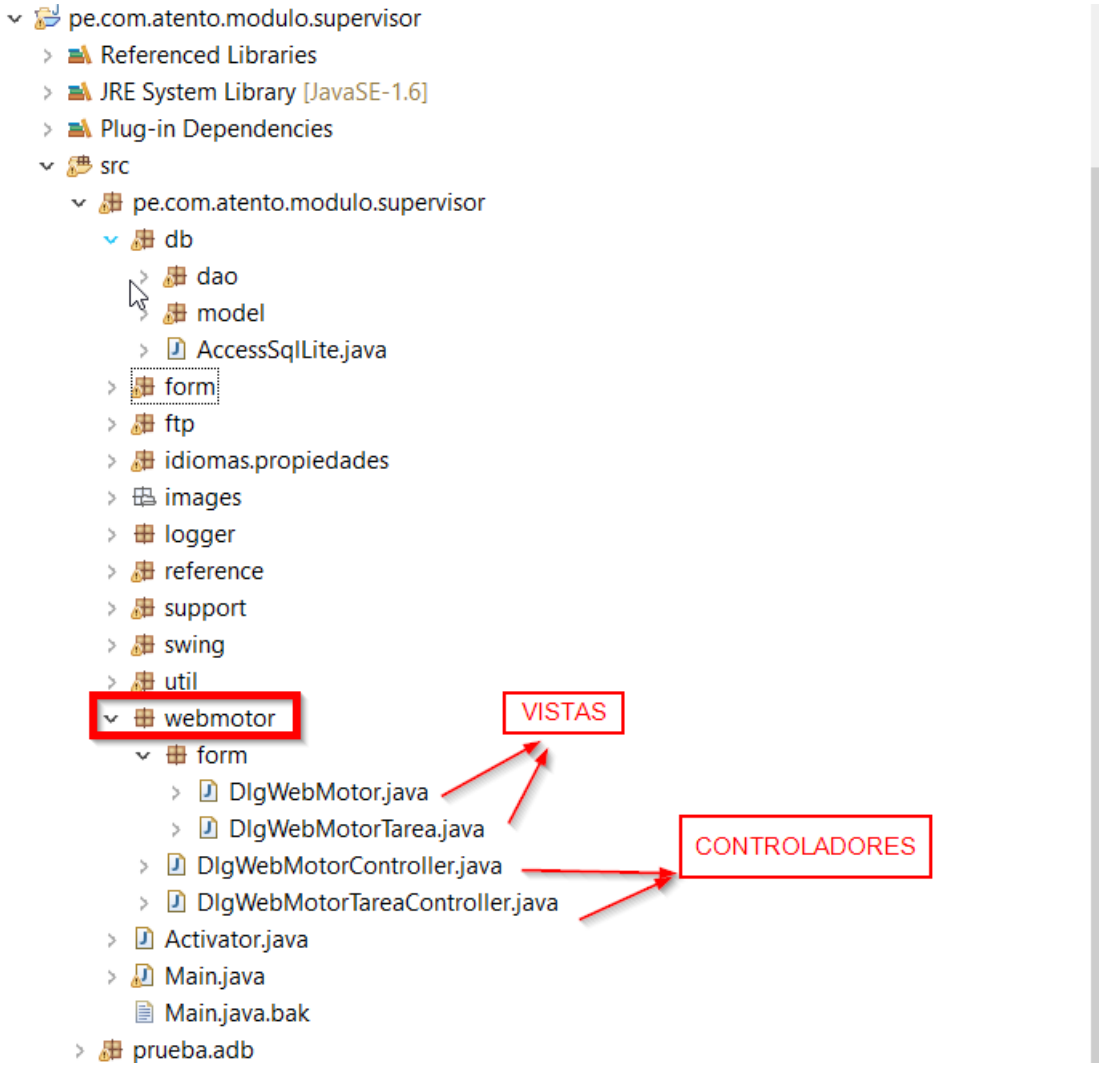


Figura 42 VISTA Y CONTROLADORES AÑADIDOS AL CÓDIGO EXISTENTE.

Fuente: (Subversion SVN Atento Perú, 2019)

DlgWebMotorTarea.java

```java
package pe.com.atento.modulo.supervisor.webmotor.form;

import javax.swing.JDialog;

@SuppressWarnings({ "rawtypes", "serial" })
public class DlgWebMotorTarea extends JDialog{
    private JTextField txtIdConfiguracionEmail;
    private JTextField txtIdPlantilla;
    private JTextField txtIdConfiguracionSms;
    private JSpinner spnPeriodoHilos;
    private JSpinner spnProcesamientoHilo;
    private JSpinner spnNumeroHilos;
    private JSpinner spnTiempoReenvio;
    private JComboBox cmbTareaMonitorear;
    private JComboBox cmbModoInicio;
    private JSpinner spnOrdenInicio;
    private JSpinner spnPeriodoTarea;
    private JCheckBox chkEstado;
    private JCheckBox chkPuntoPartida;
    private JComboBox cmbTipoTarea;
    private JButton btnAceptar;
    private JButton btnCancelar;
    private JPanel pnlConfigSms;
    private JPanel pnlConfigEmail;
    private JTextField txtIdOperadorMovil;

    public DlgWebMotorTarea() {
        initGui();
    }

    public DlgWebMotorTarea(DlgWebMotor padre) {
        super(padre, "Dialogo Tarea", true);
        initGui();
    }

```

Figura 43 VISTA "DLGWEBMOTORTAREA" UTILIZADO EXCLUSIVAMENTE PARA DEFINIR LOS COMPONENTES DE LA INTERFACE DE USUARIO.

Fuente: (Subversion SVN Atento Perú, 2019)

DlgWebMotorTareaController.java

```java

@SuppressWarnings("unchecked")
public class DlgWebMotorTareaController {
    private DlgWebMotorTarea form;
    private Tarea model;
    private List<Tarea> currentTareas = new ArrayList<Tarea>();
    private FormAction action = null;

    private static final String EMAIL_TASK = "EMAIL_TASK";
    private static final String IVR_TASK = "IVR_TASK";
    private static final String SMS_TASK = "SMS_TASK";
    private static final String WHATSAPP_TASK = "WHATSAPP_TASK";

    public void show(DlgWebMotor padre, Object model, FormAction action) {
        this.model = (Tarea)model;
        this.action = action;
        form = new DlgWebMotorTarea(padre);
        form.getCmbTareaMonitorear().addItem(ConstantesCore.SELECCIONAR);
        initListener();
        loadTipoTareas();
        loadModel();
        form.setLocationRelativeTo(null);
        form.setVisible(true);
    }

    private void initListener() {
        form.getCmbTipoTarea().addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                cmbTipoTareaActionListener();
            }
        });

        form.getChkPuntoPartida().addActionListener(new ActionListener() {
            @Override
```

Figura 44 CLASE "DLGWEBMOTORTAREACONTROLLER" UTILIZADO EXCLUSIVAMENTE PARA DEFINIR LOS EVENTOS O LISTENER DE LOS COMPONENTES DE INTERFACE DE USUARIO.

Fuente: (Subversion SVN Atento Perú, 2019)

DlgWebMotorTareaController.java

pe.com.atento.modulo.supervisor/src/pe/com/atento/modulo/supervisor/webmotor/DlgWebMotorTareaController.java

```
164     private void chkPuntoPartidaActionListener() {
165         boolean selected = form.getChkPuntoPartida().isSelected();
166         if(selected) {
167             form.getCmbTareaMonitorear().setSelectedIndex(0);
168             form.getCmbTareaMonitorear().setEnabled(false);
169         }
170         else {
171             form.getCmbTareaMonitorear().setEnabled(true);
172         }
173     }
174
175     private void cmbTipoTareaActionListener() {
176         String selectedTarea = (String)form.getCmbTipoTarea().getSelectedItem();
177         form.getContentPane().remove(form.getPnlConfigEmail());
178         form.getContentPane().remove(form.getPnlConfigSms());
179
180         if(IVR_TASK.equals(selectedTarea)) {
181             loadModoInicio(true);
182             loadTareasAMonitorear(selectedTarea);
183             form.getSpnTiempoReenvio().setEnabled(false);
184         }
185         else {
186             loadModoInicio(false);
187             if(EMAIL_TASK.equals(selectedTarea)) {
188                 form.getContentPane().add(form.getPnlConfigEmail());
189             }
190             else if(SMS_TASK.equals(selectedTarea)) {
191                 form.getContentPane().add(form.getPnlConfigSms());
192             }
193             loadTareasAMonitorear(selectedTarea);
194         }
195         form.repaint();
196     }
197
```

Figura 45 - CLASE "DLGWEBMOTORTAREACONTROLLER" UTILIZADO EXCLUSIVAMENTE PARA DEFINIR LOS EVENTOS O LISTENER DE LOS COMPONENTES DE INTERFACE DE USUARIO.

Fuente: (Subversion SVN Atento Perú, 2019)

En las etapas de construcción siguientes se tiene la oportunidad de realizar el desarrollo de una aplicación partiendo desde cero, por lo que es posible implementar y distribuir el código respetando la arquitectura MVC, patrones y estándares de programación Java.

3.2.3.3.3 INTERFACES DE USUARIO CONFIGURADOR DE ENVÍO.-

Las interfaces de usuario mostrados de este informe se explicarán a detalle en la demostración del aplicativo funcional preparada para este informe. A continuación, se ilustran los formularios:

a. Formulario DlgWebMtor.-

Presenta un listado de todos los canales de comunicación registrados para una campaña. Permite crear, editar o borrar canales de comunicación.

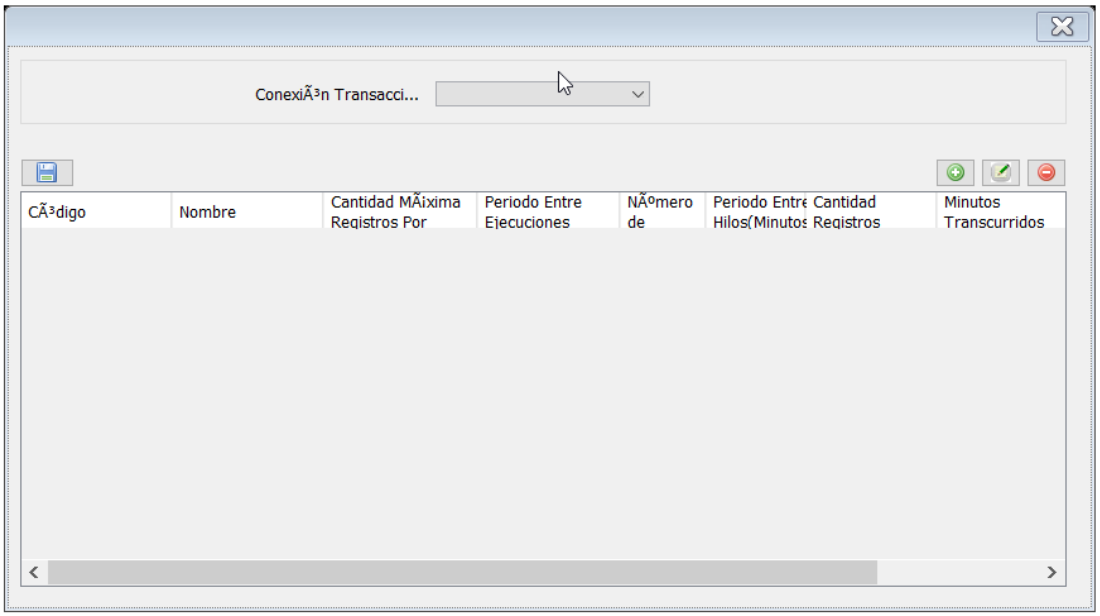


Figura 46 LISTADO DE CANALES DE COMUNICACIÓN CONFIGURADOS

Fuente: (Subversion SVN Atento Perú, 2019)

b. *Formulario DlgWebMotorTarea.-*

Este formulario permite configurar un canal de comunicación. Permite definir el tipo de tarea o también llamado canal de comunicación, el tipo de tarea a monitorea, asignar una configuración SMS y de correos, el numeró de thread o hilos que apertura la tarea, la cantidad de procesamiento por cada thread, entre otros detalles.

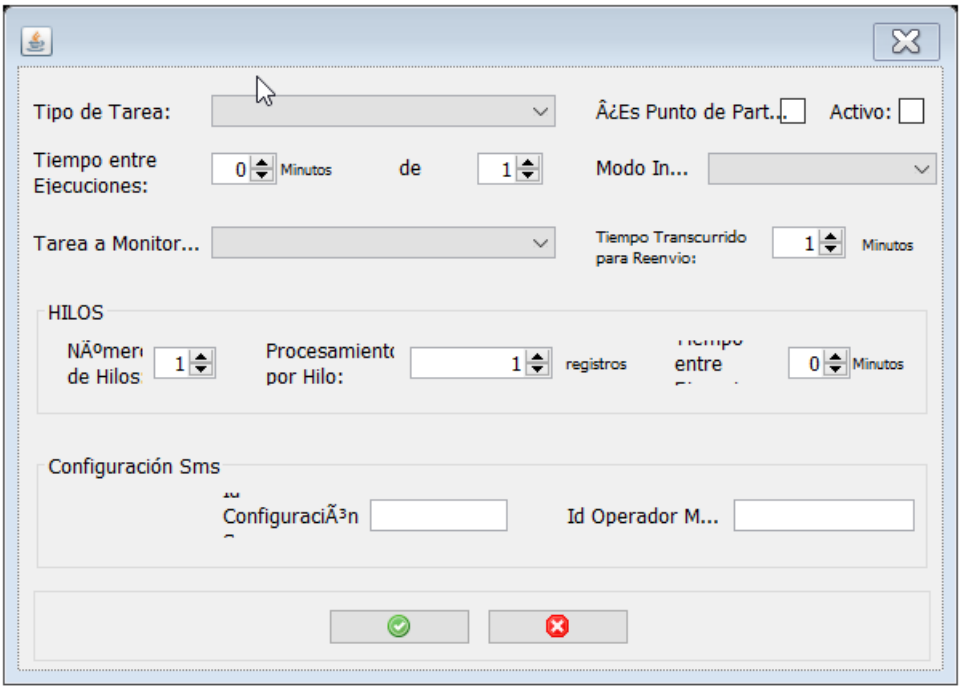


Figura 47 FORMULARIO DE CREACIÓN DE CANAL DE COMUNICACIÓN

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4 CONSTRUCCIÓN DEL MOTOR DE ENVÍOS.-

3.2.3.4.1 ¿QUÉ CONVIERTE A UN JAR EN UN BUNDLE O MÓDULO OSGI?

Para que un JAR común y corriente se transforme en un Bundle o Módulo OSGI, este necesita definir la METADATA propia de OSGI en su archivo de manifiesto **MANIFEST.MF**. En el archivo de manifiesto se define la versión del Bundle, el identificador, nombre del Bundle, las librerías que conformaran parte del classpath del Bundle, los paquetes que expondrá el Bundle y que podrán ser usados por otros Bundles y, y esto es muy importante, se define la CLASE ACTIVATOR que será invocada cuando el Bundle se instale correctamente para desencadenar toda su funcionalidad.

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Core
Bundle-SymbolicName: pe.com.atento.core;singleton:=true
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: pe.com.atento.core.Activator
Bundle-RequiredExecutionEnvironment: JavaSE-1.6
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ClassPath: .,
 lib/commons-beanutils-1.9.1.jar,
 lib/axis2.jar,
 lib/basicplayer3.0.jar,
 lib/cobra.jar,
 lib/commons-compress-1.0.jar,
 lib/commons-io-1.3.2.jar,
 lib/commons-logging-1.1.1.jar,
 lib/commons-net-3.0.1.jar,
 lib/commons.jar,
 lib/GrabadorTDM.jar,
 lib/jbrowser-1.9.jar,
 lib/commons-lang3-3.4.jar,
 lib/commons-collections-3.2.1.jar,
 lib/jcommon-1.0.17.jar,
 lib/jl.jar,
 lib/jlg.jar,
 lib/jna-3.2.2.jar,
```

Figura 48 EJEMPLO DE ARCHIVO DE MANIFIESTO

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.2 PATRÓN MVC Y ESTANDARES JAVA EN DESARROLLO DE BUNDLES.-

En cada uno de los bundles desarrollados en esta etapa de construcción, se aplicó el patrón MVC, por lo que se podrá identificar claramente la CAPA DAO o de PERSISTENCIA, la CAPA DE SERVICIO que contiene la lógica del negocio, la CAPA DE VISTA y los CONTROLADORES de las mismas. Así mismo, en cada capa de la aplicación se hace uso de INTERFACES JAVA para definir métodos y se definen las clases que implementas estas INTERFACES; se hace uso del patrón "SINGLETON", uso de constantes o atributos "finales estáticos"; manejo correcto de excepciones; se respeta el estándar de nomenclatura java para nombrar atributos, clases, métodos, constantes y más; uso de espacios en blanco para una mejor legibilidad del código; correcta visibilidad de atributo e instancias de clases; división de código en métodos más pequeños.

3.2.3.4.3 BUNDLE "SCHEDULER".-

3.2.3.4.3.1 ARCHIVO MANIFIESTO.-

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: Scheduler
Bundle-SymbolicName: Scheduler
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: scheduler.Activator
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ClassPath: .,
 lib/jackson-annotations-2.5.3.jar,
 lib/jackson-core-2.5.3.jar,
 lib/jackson-databind-2.5.3.jar,
 lib/log4j-1.2.14.jar,
 lib/mysql-connector-java-5.0.4.jar,
 lib/swingx-0.9.1.jar,
 lib/synthetica.jar,
 lib/syntheticaSimple2D.jar,
 lib/c3p0-0.9.1.2.jar
Export-Package: pe.com.atento.scheduler.commons,
 pe.com.atento.scheduler.domain,
 pe.com.atento.scheduler.domain.impl,
 pe.com.atento.scheduler.service,
 pe.com.atento.scheduler.service.impl,
 pe.com.atento.scheduler.taskcontroller,
 pe.com.atento.scheduler.util,
 pe.com.atento.scheduler.util.databasepool,
 pe.com.atento.scheduler.util.exception,
 pe.com.atento.scheduler.util.swing,
 pe.com.atento.scheduler.view,
 pe.com.atento.scheduler.view.form,
 scheduler,
 timertask
Require-Bundle: pe.com.atento.core;bundle-version="1.0.0"
```

Figura 49 MANIFIEST.MF SCHEDULER

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.2 CLASE ACTIVATOR.-

```
1  package scheduler;
2
3  import org.osgi.framework.BundleContext;
5
6  public class Activator implements BundleActivator {
7
8      private static BundleContext context;
9
10     static BundleContext getContext() {
11         return context;
12     }
13
14     public void start(BundleContext bundleContext) throws Exception {
15         Activator.context = bundleContext;
16         String arg = context.getProperty("args");
17         if(arg == null){
18             arg = "";
19         }
20         String[] args = {arg};
21         Launcher.main(args);
22     }
23
24     public void stop(BundleContext bundleContext) throws Exception {
25         Activator.context = null;
26     }
27 }
28
```

Figura 50 ACTIVATOR SCHEDULER

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.3 ESTRUCTURA DE CÓDIGO.-

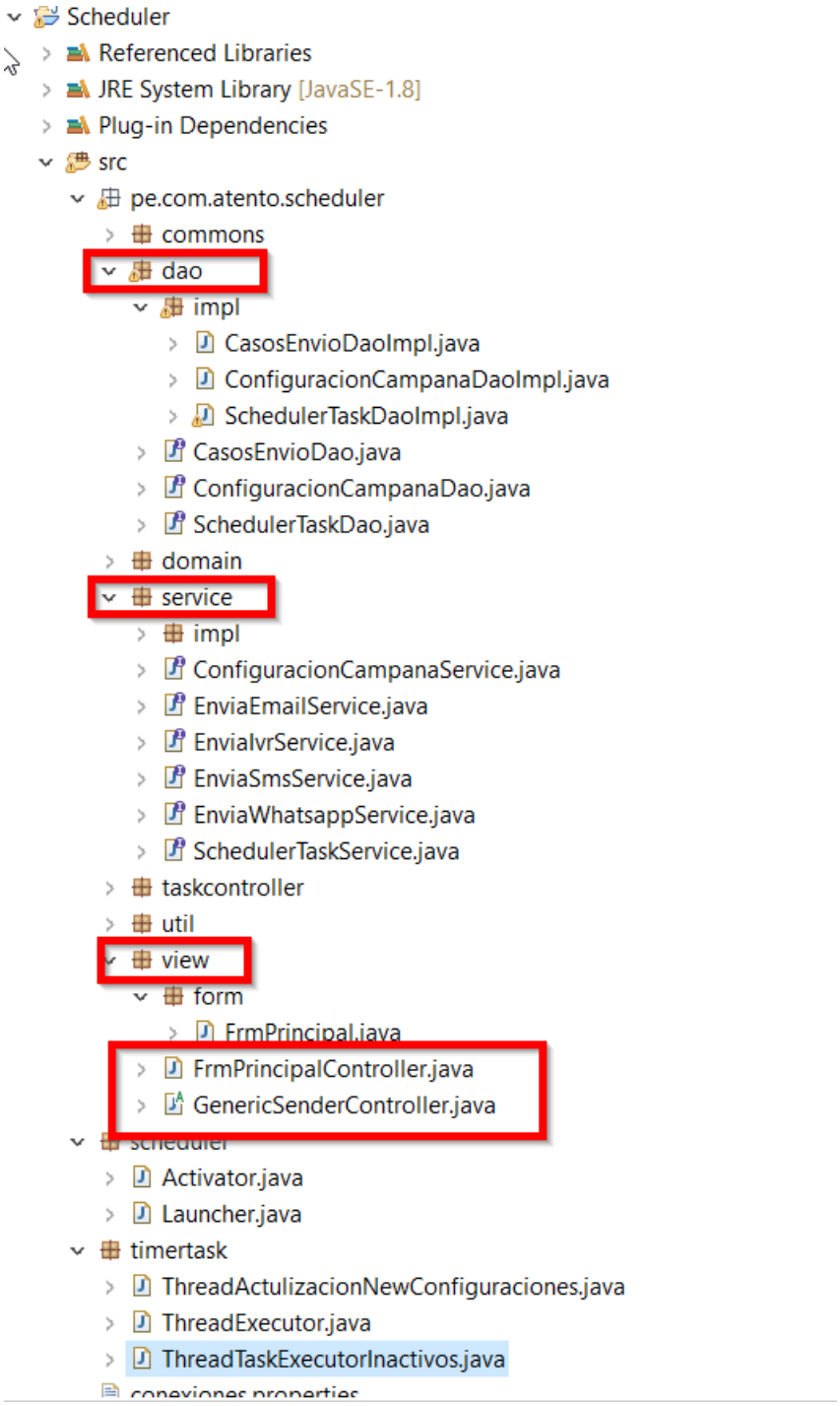


Figura 51 ESTRUCTURA CÓDIGO SHCHEDULER

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.4 CAPA DAO.-

3.2.3.4.3.4.1 INTERFACES.-

```java
package pe.com.atento.scheduler.dao;

import java.util.List;

public interface CasosEnvioDao {
    public List<CasoEnvio> getListCasosAntesesores(TareaImpl tarea) throws Exception;
    public List<CasoEnvio> getListCasosEnvioPropios(TareaImpl tarea) throws Exception;
    public void actualizaEstadoCaso(String codigoHash, Object codigoServicioGenerado,
            String estado, Campana campana) throws Exception;
}
```

Figura 52 INTERFACE "CasosEnvioDao"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.dao;

import pe.com.atento.scheduler.domain.Campana;

public interface ConfiguracionCampanaDao {
    public ConfiguracionValidaCampana getConfiguracionPorCampanaId(Integer campanaId,
            Integer configId, String tipoConexion) throws Exception;
    public void getUsAndDescripcion(Campana campana) throws Exception;
}
```

Figura 53 INTERFACE "ConfiguracionCampanaDao"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.dao;
import java.util.LinkedList;

public interface SchedulerTaskDao {

    public LinkedList<TareaImpl> obtieneListaTareasPorEstadoYCodigo(String estado,
            String codigo) throws Exception;

    public TareaImpl getTareaPorCampanaYCodigo(Integer campanaId, String codigoTarea,
            String estado) throws Exception;
}
```

Figura 54 INTERFACE "SchedulerTaskDao"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.4.2 IMPLEMENTACIÓN DE INTERFACES.-

```java
package pe.com.atento.scheduler.dao.impl;

import java.math.BigDecimal;

public class CasosEnvioDaoImpl implements CasosEnvioDao{

    @Override
    public List<CasoEnvio> getListCasosAntesesores(TareaImpl tarea) throws Exception{
        //********ESTO FUNCIONALIDAD CON EL MOTOR SIN MAESTRO; AÚN NO SE DEFINIO EL COMPARTAMIENTO CON LA NUEVA
        //FUNCIONALIDAD ADAPTADADA AL MAESTRO
        List<CasoEnvio> list = new ArrayList<CasoEnvio>();
        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection cn = null;
        try {
            cn = VariablesGlobales.getPoolConnectionMaestroPorTipo(tarea.getCampana().getTipoConexionTransaccional()).getConnection();
            ps = cn.prepareStatement("{call obtieneListaCasosAtensesoresAEnviar (?, ?, ?, ?, ?)}");

            int idCampana = tarea.getIdCampana();
            int cantidadRegistros = tarea.getCantidadRegistros();
            Long tiempoTranscurrido = tarea.getTiempoTranscurrido();
            String codigoTareaAtensesora = tarea.getCodigoTareaAtensesora();
            String codigo = tarea.getCodigo();

            PrintLogUtil.printInfo(getClass(), "getListCasosAntesesores " + idCampana + ", " + cantidadRegistros + ", "
                    + tiempoTranscurrido + ", " + codigoTareaAtensesora + ", " + codigo);

            ps.setInt(1, idCampana);
            ps.setInt(2, cantidadRegistros);
            ps.setLong(3, tiempoTranscurrido);
            ps.setString(4, codigoTareaAtensesora);
            ps.setString(5, codigo);

            rs = ps.executeQuery();
            if(rs != null){
                while(rs.next()){
```

Figura 55 CLASE "CasosEnvioDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.dao.impl;

import java.sql.Connection;

public class ConfiguracionCampanaDaoImpl implements ConfiguracionCampanaDao{

    @Override
    public ConfiguracionValidaCampana getConfiguracionPorCampanaId(Integer campanaId, Integer configId, String tipoConexion) throws Exception {

        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection cn =  null;
        try {
            PrintLogUtil.printInfo(getClass(), "getConfiguracionPorCampanaId " + campanaId + ", " + configId + ", "
                    + tipoConexion);

            cn =  VariablesGlobales.getPoolConnectionPorTipo(tipoConexion).getConnection();
            ps = cn.prepareStatement("{call SP_VALIDA_CAMPANAS_POR_CAMPANA_ID(?, ?, ?)}");

            ps.setInt(1, campanaId);
            ps.setInt(2, configId);
            ps.setString(3, Constantes.KEY_MESSAGE.ACTIVO);
            rs = ps.executeQuery();
            if(rs != null && rs.next()){
                Integer campanaConfigId = ObjectUtil.getInteger(rs.getString(1));
                Integer _campanaId = ObjectUtil.getInteger(rs.getString(2));
                String campanaDescrip = rs.getString(3);
                String campanaResultId = rs.getString(4);
                String campanaTemaId = rs.getString(5);
                String campanaVdn = rs.getString(6);


                ConfiguracionValidaCampana conf = new ConfiguracionValidaCampana();
                conf.setCampanaConfigId(campanaConfigId);
                conf.setCampanaId(_campanaId);
                conf.setCampanaDescrip(campanaDescrip);
                conf.setCampanaResultId(campanaResultId);
```

Figura 56 CLASE "ConfiguracionCampanaDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.dao.impl;

import java.sql.Connection;

public class SchedulerTaskDaoImpl implements SchedulerTaskDao{

    @Override
    public LinkedList<TareaImpl> obtieneListaTareasPorEstadoYCodigo(String estado, String codigo) throws Exception {
        PreparedStatement ps = null;
        ResultSet rs = null;
        LinkedList<TareaImpl> list = new LinkedList<TareaImpl>();
        Connection cn = null;
        try {
            PrintLogUtil.printInfo(getClass(), "obtieneListaTareasPorEstadoYCodigo " + estado + ", " + codigo);

            cn = VariablesGlobales.POOL_CONEXION_PRINCIPAL.getConnection();
            ps = cn.prepareStatement("{call scheduler_obtieneListaTareasPorEstadoCodigo (?, ?)}");

            ps.setString(1, Constantes.KEY_MESSAGE.ACTIVO);
            ps.setString(2, codigo);
            rs = ps.executeQuery();
            if(rs != null){
                Campana campana = null;
                while(rs.next()){
                    String nombre = rs.getString(1);
                    String descripcion = rs.getString(2);
                    Integer idCampana = rs.getInt(3);
                    String _codigo = rs.getString(4);
                    Timestamp inicio = rs.getTimestamp(5);
                    Long periodoTarea = rs.getLong(6);
                    Long periodoHilo = rs.getLong(7);
                    Integer hilos = rs.getInt(8);
                    Integer cantidadRegistros = rs.getInt(9);
                    String _estado = rs.getString(10);
                    Integer orden = rs.getInt(11);
                    Integer idConfig = rs.getInt(12);
                    String idPlantilla = rs.getString(13);
                    String codigoTareaAntesesora = rs.getString(14);
```

Figura 57 CLASE "SchedulerTaskDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.4.3 POOL DE CONEXIONES C3P0.-

```java
package pe.com.atento.scheduler.util.databasepool;

import java.sql.Connection;
import com.mchange.v2.c3p0.ComboPooledDataSource;

public class DataSource {
    private ComboPooledDataSource comboPooledDataSource;

    public DataSource(String url, String driver) throws Exception {
        comboPooledDataSource = new ComboPooledDataSource();
        comboPooledDataSource.setDriverClass(driver);
        comboPooledDataSource.setJdbcUrl(url);
        comboPooledDataSource.setMinPoolSize(5);
        comboPooledDataSource.setMaxPoolSize(20);
        comboPooledDataSource.setAcquireIncrement(5);
        comboPooledDataSource.setCheckoutTimeout(10000);
    }

    public DataSource(String user, String pass, String url, String driver) throws Exception {
        comboPooledDataSource = new ComboPooledDataSource();
        comboPooledDataSource.setDriverClass(driver);
        comboPooledDataSource.setJdbcUrl(url);
        comboPooledDataSource.setUser(user);
        comboPooledDataSource.setPassword(pass);
        comboPooledDataSource.setMinPoolSize(5);
        comboPooledDataSource.setMaxPoolSize(20);
        comboPooledDataSource.setAcquireIncrement(5);
        comboPooledDataSource.setCheckoutTimeout(10000);
    }

    public Connection getConnection() throws Exception {
        return comboPooledDataSource.getConnection();
    }
}
```

Figura 58 CLASE "DataSource o Pool de Conexiones c3p0"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.4.4 DEFINICIOS DE THREADS.-

```java
package timertask;

import java.util.TimerTask;

public class ThreadActulizacionNewConfiguraciones extends TimerTask{
    private SchedulerTaskService schedulerTaskService = (SchedulerTaskService)SingletonFactory
            .obtieneInstancia(SchedulerTaskServiceImpl.class);

    @Override
    public void run() {
        try {
            PrintLogUtil.printInfo(getClass(), "Inicio de thread de actualización de configuraciones");
            schedulerTaskService.instalaNuevasTareas();
            PrintLogUtil.printInfo(getClass(), "Fin de thread de actualización de configuraciones");
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), e);
        }
    }
}
```

Figura 59 Thread "ThreadActulizacionNewConfiguraciones": Se crea una única instancia de este thread para actualización de canales de comunicación.

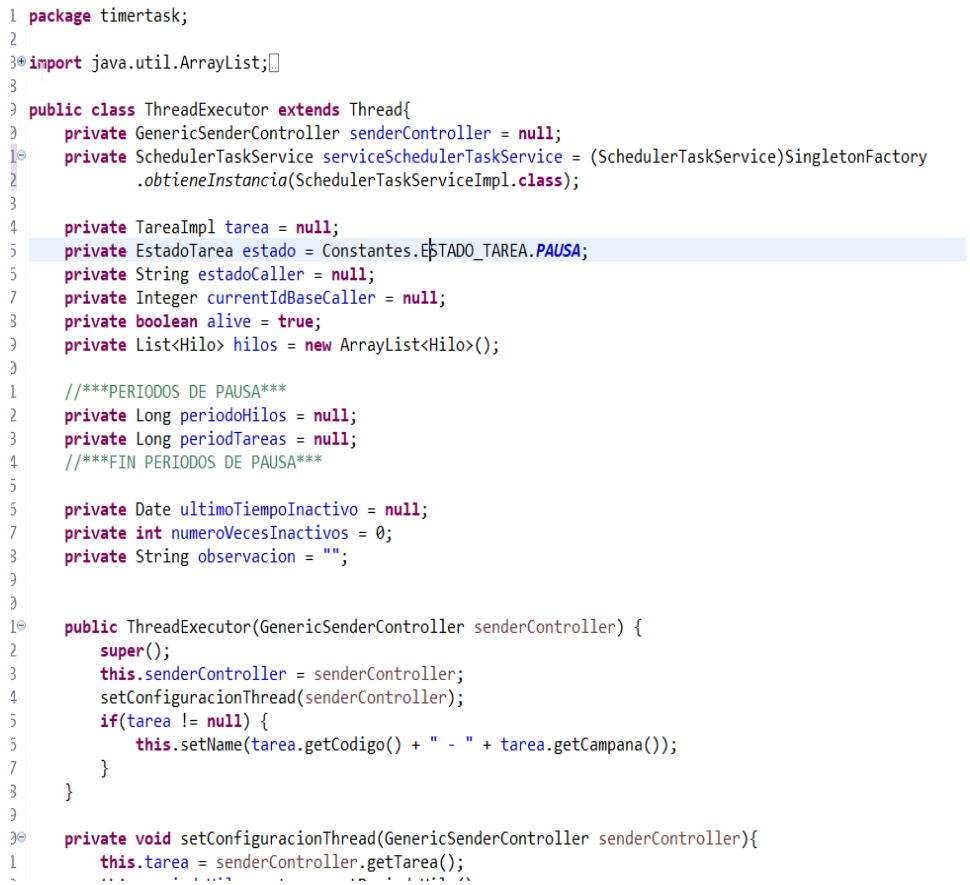
Fuente: (Subversion SVN Atento Perú, 2019)

```java
public class ThreadTaskExecutorInactivos extends TimerTask{

    @Override
    public void run() {
        PrintLogUtil.printInfo(getClass(), "Inicio de thread de destrucción de tareas inactivas");
        Long now = new Date().getTime();
        for (Campana c : VariablesGlobales.LIST_CAMPANAS) {
            for (TareaImpl t : c.getListTareas()) {
                ThreadExecutor te = t.getThreadExecutor();
                if(te != null) {
                    if(!Constantes.ESTADO_TAREA.PARADO.equals(te.getEstado())) {
                        Date ultimoTiempoInactivo = te.getUltimoTiempoInactivo();
                        int numeroVecesInactivos = te.getNumeroVecesInactivos();
                        if(ultimoTiempoInactivo != null || numeroVecesInactivos > 0) {
                            if(((now - ultimoTiempoInactivo.getTime()) >= Constantes.TIEMPO_INACTIVO)
                                    || numeroVecesInactivos > Constantes.VECES_INACTIVO) {
                                try {
                                    PrintLogUtil.printInfo(getClass(), "Destruyendo tarea: " + te.getTarea().getCodigo() + ", Campana: " + te.get
                                    te.actualizarEstado(Constantes.ESTADO_TAREA.PARADO, Constantes.KEY_MESSAGE.OBS_STOPPED_BY_LIMPIADOR_INACTIVOS
                                    SwingUtil.executeInSwingThread(new FormAction() {
                                        @Override
                                        public void execute() {
                                            VariablesGlobales.FRM_PRINCIPAL_CONTROLLER.getForm().repaint();
                                        }
                                    });
                                    PrintLogUtil.printInfo(getClass(), "Finalizado Destruyendo tarea: " + te.getTarea().getCodigo() + ", Campana:
                                } catch (Exception e) {
                                    PrintLogUtil.printError(getClass(), "Error al invocar INTERRUPT de Thread: " +
                                            te.getTarea().getCodigo() + ", Campana: " + te.getTarea().getIdCampana());
                                    PrintLogUtil.printError(getClass(), e);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
```

Figura 60 Thread "ThreadTaskExecutorInactivos": Se crea una única instancia de este thread para destrucción de thread inactivos.

Fuente: (Subversion SVN Atento Perú, 2019)



```java
package timertask;

import java.util.ArrayList;

public class ThreadExecutor extends Thread{
    private GenericSenderController senderController = null;
    private SchedulerTaskService serviceSchedulerTaskService = (SchedulerTaskService)SingletonFactory
            .obtieneInstancia(SchedulerTaskServiceImpl.class);

    private TareaImpl tarea = null;
    private EstadoTarea estado = Constantes.ESTADO_TAREA.PAUSA;
    private String estadoCaller = null;
    private Integer currentIdBaseCaller = null;
    private boolean alive = true;
    private List<Hilo> hilos = new ArrayList<Hilo>();

    //***PERIODOS DE PAUSA***
    private Long periodoHilos = null;
    private Long periodTareas = null;
    //***FIN PERIODOS DE PAUSA***

    private Date ultimoTiempoInactivo = null;
    private int numeroVecesInactivos = 0;
    private String observacion = "";


    public ThreadExecutor(GenericSenderController senderController) {
        super();
        this.senderController = senderController;
        setConfiguracionThread(senderController);
        if(tarea != null) {
            this.setName(tarea.getCodigo() + " - " + tarea.getCampana());
        }
    }

    private void setConfiguracionThread(GenericSenderController senderController){
        this.tarea = senderController.getTarea();
```

Figura 61 Thread "ThreadExecutor": Se crea múltiples instancias de este thread dependiendo del número de canales de comunicación.

Fuente: (Subversion SVN Atento Perú, 2019)

```java
@Override
public void run() {
    try {
        while(alive){
            if(this.estado.equals(Constantes.ESTADO_TAREA.CORRIENDO)){
                if(senderController != null){
                    PrintLogUtil.printInfo(getClass(), "Corriendo TAREA " + tarea + "-->" + new Date());
                    procesaHilos();
                    if(!Constantes.DEFAULT_PERIODO_TAREAS.equals(this.periodTareas)) {
                        Thread.sleep(this.periodTareas);//SI TIENE PERIODO CONFIGURADO, ESPERA EL PERIODO ESTABLECIDO DESPUES DE CAD
                    }
                }
            }
        }
    } catch (Exception e) {
        PrintLogUtil.printError(getClass(), e);
    }
}

private void procesaHilos(){
    try {
        int size = 0;
        this.senderController.setFromNuevoProceso(true);
        for(int i = 0; i < (size = hilos.size()); i++){
            if(Constantes.ESTADO_TAREA.PAUSA.equals(this.estado)) {
                break;
            }
            Hilo hilo = hilos.get(i);
            PrintLogUtil.printInfo(getClass(), "Corriendo TAREA " +  tarea + "--> Hilo " + (i + 1) + new Date());
            senderController.setCurrentHilo(hilo);
            senderController.enviaCaso();
            this.senderController.setFromNuevoProceso(false);
            if((i + 1) < size){
                Thread.sleep(periodoHilos);
            }
        }
```

Figura 62 Thread "ThreadExecutor": Se crea múltiples instancias de este thread dependiendo del número de canales de comunicación.

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.5 CAPA DE SERVICIO.-

3.2.3.4.3.5.1 INTERFACES.-

```java
package pe.com.atento.scheduler.service;

import pe.com.atento.scheduler.domain.ConfiguracionValidaCampana;

public interface ConfiguracionCampanaService {
    public ConfiguracionValidaCampana obtieneConfiguracionPorCampanaId(Integer campanaId, Integer configId,
            String tipoConexion) throws Exception;
}
```

Figura 63 INTERFACE "ConfiguracionCampanaService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service;

import java.util.List;

public interface EnviaEmailService {
    public void enviaEmails(List<CasoEnvio> listCasosEnviar, TareaImpl tarea, Hilo currentHilo,
            String plantilla, String mailOrigen, String asunto,
            String inbox, boolean fromNuevoProceso) throws ServiceException;
}
```

Figura 64 INTERFACE "EnviaEmailService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service;

import java.util.List;

public interface EnviaIvrService{
    public void disparaLlamadasDesdeCaller(TareaImpl tarea) throws ServiceException;
    public void enviaIvr(List<CasoEnvio> listaAenviar, Hilo currentHilo, TareaImpl tarea,
            boolean fromNuevoProceso) throws ServiceException;
    public void actualizaEstadoBaseCaller(Integer idBaseCaller, String estado, String us) throws ServiceException;
}
```

Figura 65 INTERFACE "EnviaIvrService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service;

import java.util.List;

public interface EnviaSmsService{
    public void enviaSmsViaStore(List<CasoEnvio> listaAenviar, Hilo currentHilo, TareaImpl tarea, Object smsConfigure, boolean fromNuevoPro
    public void enviaSmsViaSoap(List<CasoEnvio> listaAenviar, Hilo currentHilo, TareaImpl tarea, Object smsConfigure, boolean fromNuevoProc
    public void enviaSmsViaInfoBip(List<CasoEnvio> listaAenviar, Hilo currentHilo, TareaImpl tarea, Object smsConfigure, boolean fromNuevoF
}
```

Figura 66 INTERFACE "EnviaSmsService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service;

import java.util.List;

public interface EnviaWhatsappService {
    public void enviaWhatsApp(List<CasoEnvio> listaAenviar, TareaImpl tarea, Hilo currentHilo, Object whatsappConfigure, boolean fromNuevo
}
```

Figura 67 INTERFACE "EnviaWhatsappService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service;

import java.util.LinkedList;

public interface SchedulerTaskService {
    public LinkedList<TareaImpl> obtieneListadoTareasActivasPorCodigo(String estado, String codigo) throws ServiceException;
    public void creaTareas(List<TareaImpl> tareas) throws ServiceException;
    public ThreadExecutor getThreadTask(TareaImpl tarea);
    public TareaImpl obtieneTareaPorCampanaYCodigo(Integer campanaId, String codigoTarea, String estado);
    public void instalaNuevasTareas() throws ServiceException;
    public void cambiaEstadoBaseCaller(TareaImpl tarea, String estado) throws ServiceException;
}
```

Figura 68 INTERFACE "SchedulerTaskService"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.5.2 IMPLEMENTACIÓN DE INTERFACES

```java
package pe.com.atento.scheduler.service.impl;

import pe.com.atento.scheduler.dao.ConfiguracionCampanaDao;

public class ConfiguracionCampanaServiceImpl implements ConfiguracionCampanaService {
    private ConfiguracionCampanaDao configuracionCampanaDao = (ConfiguracionCampanaDao)SingletonFactory
            .obtieneInstancia(ConfiguracionCampanaDaoImpl.class);

    @Override
    public ConfiguracionValidaCampana obtieneConfiguracionPorCampanaId(
            Integer campanaId, Integer configId, String tipoConexion) throws Exception{
        try {
            valida(campanaId, configId);
            return configuracionCampanaDao.getConfiguracionPorCampanaId(campanaId, configId, tipoConexion);
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), e);
            throw new ServiceException(e);
        }
    }

    private void valida(Integer campanaId, Integer configId) throws Exception{
        if(campanaId == null)
            throw new Exception("No es posible otener configuracion valida campana: ID CAMPANA es null");

        if(configId == null)
            throw new Exception("No es posible otener configuracion valida campana: ID CONFIGURACION es null");
    }
}
```

Figura 69 CLASE "ConfiguracionCampanaServiceImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service.impl;

import java.math.BigDecimal;

/**
 *
 * @author mvasquezma
 *
 */
public class GenericService {
    private CasosEnvioDao casosEnvioDao = (CasosEnvioDao)SingletonFactory.obtieneInstancia(CasosEnvioDaoImpl.class);

    /**
     * Este método genera casos en base al codigo de la propia tarea. Devuelve un listado de los casos generados
     * @param tarea
     * @return
     * @throws ServiceException
     */
    public List<CasoEnvio> obtieneListaCasosPropiosAEnviar(TareaImpl tarea) throws ServiceException{
        try {
            validaGeneracionPropios(tarea);
            return casosEnvioDao.getListCasosEnvioPropios(tarea);
        } catch (Exception e) {
            throw new ServiceException(e);
        }
    }
    /**
     * Este método obtiene casos en base a la tarea a monitorear configurada
     * @param tarea
     * @return
     * @throws ServiceException
     */
    public List<CasoEnvio> obtieneListaCasosAtensesoresAEnviar(TareaImpl tarea) throws ServiceException{
        try {
            validaGeneracionAntesesores(tarea);
            return casosEnvioDao.getListCasosAntesesores(tarea);
        } catch (Exception e) {
            throw new ServiceException(e);
```

Figura 70 CLASE "GenericService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.scheduler.service.impl;

import java.util.LinkedList;

public class SchedulerTaskServiceImpl extends GenericService implements SchedulerTaskService{
    private SchedulerTaskDao daoSchedulerTask = (SchedulerTaskDao)SingletonFactory.obtieneInstancia(SchedulerTaskDaoImpl.class)
    private ConfiguracionCampanaDao configuracionCampanaDao = (ConfiguracionCampanaDao)SingletonFactory.obtieneInstancia(Configu

    @Override
    public LinkedList<TareaImpl> obtieneListadoTareasActivasPorCodigo(String estado, String codigo) throws ServiceException {
        try {
            if(codigo != null){
                LinkedList<TareaImpl> listadoTareas = daoSchedulerTask.obtieneListaTareasPorEstadoYCodigo(estado, codigo);
                return listadoTareas;
            }
            return null;
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), e);
            throw new ServiceException(e);
        }
    }

    @Override
    public void creaTareas(List<TareaImpl> tareas) throws ServiceException {
        try {
            if(ListUtil.isNotEmpty(tareas)){
                for(int i = 0; i < tareas.size(); i++){
                    TareaImpl t = tareas.get(i);
                    addCampanaListaGlobal(t);
                    try {
                        ThreadExecutor threadTarea = getThreadTask(t);
                        if(threadTarea != null){
                            t.setThreadExecutor(threadTarea);
                            threadTarea.start();
                        }
                    } catch (Exception e) {
                        PrintLogUtil.printError(getClass(), "Error al configurar TAREA. CAMPANA: " + t.getIdCampana() + "; TAREA
```

Figura 71 CLASE "SchedulerTaskServiceImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.6 CAPA DE VISTA Y CONTROLADOR.-

3.2.3.4.3.6.1 VISTAS.-

```
1  package pe.com.atento.scheduler.view.form;
2
3  import java.util.Vector;
24
25 @SuppressWarnings({ "rawtypes", "serial" })
26 public class FrmPrincipal extends JFrame{
27     private JComboBox cmbCampana;
28     private Vector<JSColumnTable> tblProcesoColumnas = new Vector<JSColumnTable>();
29     private JSTreeTableModel tblProcesosModel = null;
30     private JSTreeTable tblProcesos;
31     private JScrollPane scrollPane;
32     private JButton btnResume;
33     private JButton btnSuspender;
34     private JButton btnParar;
35     private JButton btnRefresh;
36     private JComboBox cmbEstados;
37     private JCheckBox chkSelectAll;
38     private JButton btnStopCaller;
39     private JButton btnPauseCaller;
40     private JPanel pnlEstadosCaller;
41     private JButton btnActivoCaller;
42
43     public FrmPrincipal() {
44         getContentPane().setBackground(Color.decode("#E3E9ED"));
45
46         setTitle("MOTOR DE TAREAS DE ENVÍOS");
47         setIconImage(Toolkit.getDefaultToolkit().getImage(FrmPrincipal.class.getResource("/pe/com/atento/scheduler/commons/icons/
48         initGui();
49         this.setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
50         this.setSize(1027, 720);
51         this.setResizable(false);
52         tblProcesoColumnas.add(new JSColumnTable("PROCESO", 180, JLabel.CENTER));
53         tblProcesoColumnas.add(new JSColumnTable("<html>MONITOREANDO</html>", 120, JLabel.CENTER));
54         tblProcesoColumnas.add(new JSColumnTable("<html>ENVÍOS<br>A PROCESAR</html>", 120, JLabel.CENTER));
55         tblProcesoColumnas.add(new JSColumnTable("<html>ENVÍOS<br>PROCESADOS</html>", 120, JLabel.CENTER));
56         tblProcesoColumnas.add(new JSColumnTable("AVANCE %", 90, JLabel.CENTER));
57         tblProcesoColumnas.add(new JSColumnTable("<html>TOTAL<br>PROCESADO<br>DESDE INICIO</html>", 90, JLabel.CENTER));
```

Figura 72 CLASE "FrmPrincipal"

Fuente: (Subversion SVN Atento Perú, 2019)

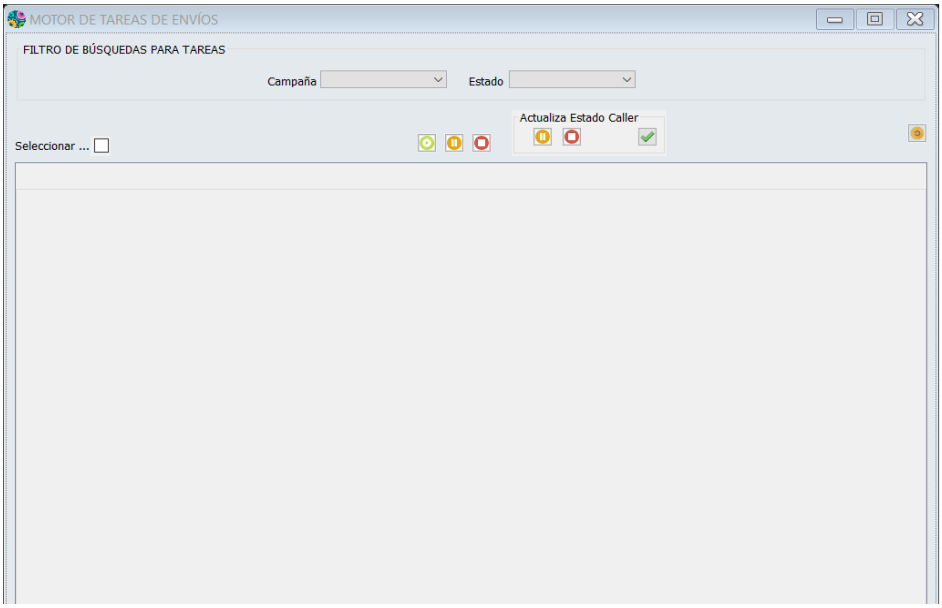


Figura 73 INTERFAZ MONITOR DE THREADS

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.3.6.2 CONTROLADORES.-

```
package pe.com.atento.scheduler.view;

import java.awt.Window;

public class FrmPrincipalController extends FormController{
    private FrmPrincipal form;
    private SchedulerTaskService schedulerTaskService = (SchedulerTaskService)SingletonFactory
            .obtieneInstancia(SchedulerTaskServiceImpl.class);

    /**
     * @wbp.parser.entryPoint
     */
    @Override
    public void show(FormController padre, FormAction formAction) {
        try {
            form = new FrmPrincipal();
            form.getBtnSuspender().setEnabled(false);
            form.getPnlEstadosCaller().setVisible(false);
            form.getBtnParar().setEnabled(false);
            form.getBtnResume().setEnabled(false);
            form.getBtnParar().setToolTipText("Click para DETENER o \"MATAR\" el proceso");
            form.getBtnResume().setToolTipText("Click para INICIAR el proceso");
            form.getBtnSuspender().setToolTipText("Click para PAUSAR el proceso");
            form.getBtnRefresh().setToolTipText("Click para ACTUALIZAR el monitor con NUEVAS TAREAS CONFIGURADAS");
            loadComboEstados();
            initListener();
            loadComboCampana();
            form.setLocationRelativeTo(null);
            form.setVisible(true);
            form.setAlwaysOnTop(true);
        } catch (Exception e) {
            MessageDialog.mostrarMensajeError(e, form);
        }

    }
```

Figura 74 Clase "FrmPrincipalController"

Fuente: (Subversion SVN Atento Perú, 2019)

```
@Override
protected void initListener() {

    form.getCmbCampana().addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            cmbCampanaActionListener();
        }
    });

    form.getCmbEstados().addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            cmbEstadoTareaActionListener();
        }
    });

    form.getBtnResume().addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            btnResumeActionListener();
        }
    });

    form.getBtnSuspender().addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            btnPauseActionListener();
        }
    });
```

Figura 75 Clase "FrmPrincipalController"

Fuente: (Subversion SVN Atento Perú, 2019)

```
private void btnPauseCallerActionListener(){
    SwingUtil.setWaitCursor(form);
    try {
        PrintLogUtil.printInfo(getClass(), "CLICK BTN RESUME CALLER");
        int selectedRow = form.getTblProcesos().getSelectedRows()[0];
        ThreadExecutor thread = (ThreadExecutor)(this.form.getTblProcesos().getDominioTreeTableEntryByIndex(s
        TareaImpl tarea = thread.getTarea();
        schedulerTaskService.cambiaEstadoBaseCaller(tarea, Constantes.CALLER_ESTADO_BASE.PAUSE);
        thread.setEstadoCaller(Constantes.CALLER_ESTADO_BASE.PAUSE);
        refreshTableProcesos();
    } catch (Exception e) {
        MessageDialog.mostrarMensajeError(e, form);
    } finally {
        SwingUtil.setDefaultCursor(form);
    }
}

private void btnStopCallerActionListener(){
    SwingUtil.setWaitCursor(form);
    try {
        PrintLogUtil.printInfo(getClass(), "CLICK BTN RESUME CALLER");
        int selectedRow = form.getTblProcesos().getSelectedRows()[0];
        ThreadExecutor thread = (ThreadExecutor)(this.form.getTblProcesos().getDominioTreeTableEntryByIndex(s
        TareaImpl tarea = thread.getTarea();
        schedulerTaskService.cambiaEstadoBaseCaller(tarea, Constantes.CALLER_ESTADO_BASE.STOP);
        thread.setEstadoCaller(Constantes.CALLER_ESTADO_BASE.STOP);
        refreshTableProcesos();
    } catch (Exception e) {
        MessageDialog.mostrarMensajeError(e, form);
    } finally {
        SwingUtil.setDefaultCursor(form);
    }
}
```

Figura 76 Clase "FrmPrincipalController"

Fuente: (Subversion SVN Atento Perú, 2019)

```
package pe.com.atento.scheduler.view;

import pe.com.atento.scheduler.domain.Hilo;

public abstract class GenericSenderController {
    protected Hilo currentHilo;
    protected boolean fromNuevoProceso = false;
    protected TareaImpl tarea = null;

    public abstract void enviaCaso();
    protected abstract void instalaConfiguracion() throws Exception;
    public abstract GenericSenderController getNewInstance(TareaImpl tarea);

    public Hilo getCurrentHilo() {
        return currentHilo;
    }
    public void setCurrentHilo(Hilo currentHilo) {
        this.currentHilo = currentHilo;
    }
    public boolean isFromNuevoProceso() {
        return fromNuevoProceso;
    }
    public void setFromNuevoProceso(boolean fromNuevoProceso) {
        this.fromNuevoProceso = fromNuevoProceso;
    }
    public TareaImpl getTarea() {
        return tarea;
    }
    public void setTarea(TareaImpl tarea) {
        this.tarea = tarea;
    }
}
```

Figura 77 Clase "GenericSenderController"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.4 BUNDLE “SMSTASK”

3.5.5.4.1 ARCHIVO MANIFIESTO.-

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: SmsTask
Bundle-SymbolicName: SmsTask
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: smstask.Activator
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Require-Bundle: Scheduler;bundle-version="1.0.0"
Bundle-ClassPath: .

```

Figura 78 MANIFIEST.MF SMSTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.2 CLASE ACTIVATOR.-

```
package smstask;

import org.osgi.framework.BundleActivator;

public class Activator implements BundleActivator {

    private static BundleContext context;

    static BundleContext getContext() {
        return context;
    }

    public void start(BundleContext bundleContext) throws Exception {
        Launcher.main(null);
    }

    public void stop(BundleContext bundleContext) throws Exception {
        Activator.context = null;
    }

}
```

Figura 79 ACTIVATOR SMSTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.3 ESTRUCTURA DE CÓDIGO.-

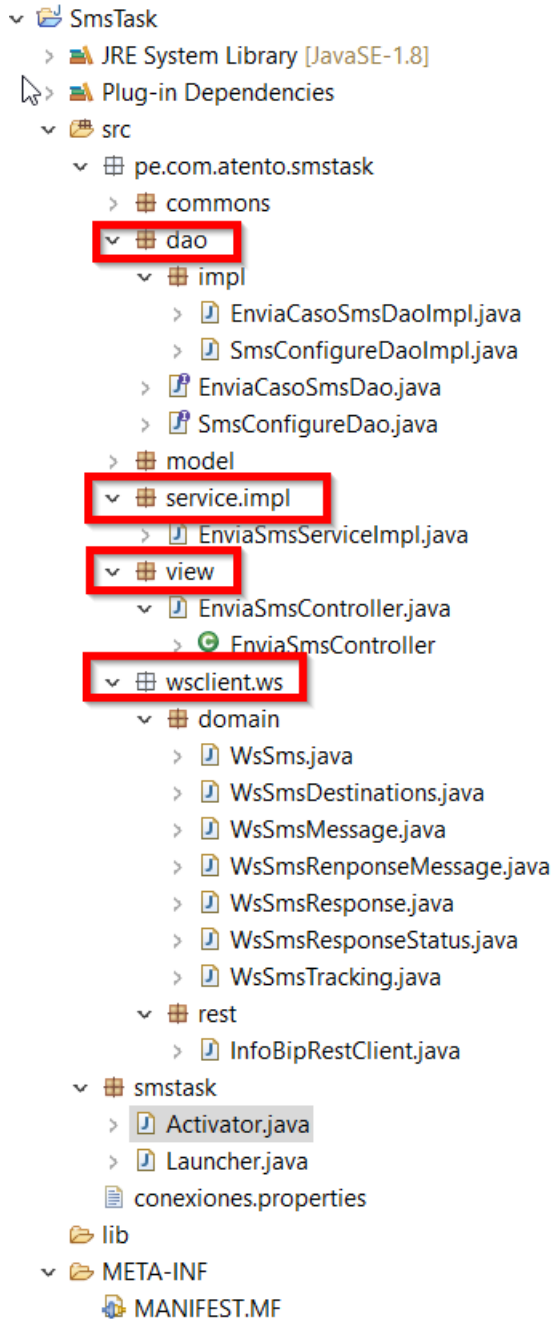


Figura 80 ESTRUCTURA CÓDIGO SMSTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.4 CAPA DAO.-

3.5.5.4.4.1 INTERFACES.-

```
package pe.com.atento.smstask.dao;

public interface EnviaCasoSmsDao {
    public Integer registraSms(String fromNewMessage, String toNewMessage, String ccNewMessage, String ccoNewMessage,
            String subjectNewMessage, String dateAppNewMessage, String meesageNew, Integer countAttachmentNew,
            Integer idMovile,String operador,Integer accoount,Integer temaCodigo,Integer arbolId) throws Exception;
}

```

Figura 81 INTERFACE "EnviaCasoSmsDao"

Fuente: (Subversion SVN Atento Perú, 2019)

```
package pe.com.atento.smstask.dao;

import pe.com.atento.smstask.model.SmsConfigure;

public interface SmsConfigureDao {
    public SmsConfigure getSmsConfigurePorId(Integer id, String tipoConexion) throws Exception;
}
```

Figura 82 INTERFACE "SmsConfigureDao"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.4.2 IMPLEMENTACIÓN DE INTERFACES.-

```java
package pe.com.atento.smstask.dao.impl;

import java.sql.Connection;

public class EnviaCasoSmsDaoImpl implements EnviaCasoSmsDao{

    @Override
    public Integer registraSms(String fromNewMessage, String toNewMessage, String ccNewMessage, String ccoNewMessage,
            String subjectNewMessage, String dateAppNewMessage, String meesageNew, Integer countAttachmentNew,
            Integer idMovile,String operador,Integer accoount,Integer temaCodigo,Integer arbolId) throws Exception{

        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection cn = SmsVariablesGlobales.POOL_CONEXION_SMS.getConnection();
        try {
            PrintLogUtil.printInfo(getClass(), "registraSms " + fromNewMessage + ", " + toNewMessage + ", "
                    + ccNewMessage + ", " + ccoNewMessage + ", " + subjectNewMessage + ", " + dateAppNewMessage + ", "
                    + meesageNew + ", " + countAttachmentNew + ", " + idMovile + ", " + operador + ", " + accoount + ", "
                    + temaCodigo + ", " + arbolId);

            String query = "{call setNew(?, ? , ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ? )}";
            ps = cn.prepareStatement(query);

            ps.setString(1, fromNewMessage);
            ps.setString(2, toNewMessage);
            ps.setString(3, ccNewMessage);
            ps.setString(4, ccoNewMessage);
            ps.setString(5, subjectNewMessage);
            ps.setString(6, dateAppNewMessage);
            ps.setString(7, meesageNew);
            ps.setObject(8, countAttachmentNew);
            ps.setObject(9, idMovile);
            ps.setString(10, operador);
            ps.setObject(11, accoount);
            ps.setObject(12, temaCodigo);
            ps.setObject(13, arbolId);
            rs = ps.executeQuery();
```

Figura 83 CLASE "EnviaCasoSmsDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.smstask.dao.impl;

import java.sql.Connection;

public class SmsConfigureDaoImpl implements SmsConfigureDao{

    @Override

    public SmsConfigure getSmsConfigurePorId(Integer id, String tipoConexion) throws Exception {
        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection cn = null;
        try {
            PrintLogUtil.printInfo(getClass(), "getSmsConfigurePorId " + id + ", " + tipoConexion );

            cn = VariablesGlobales.getPoolConnectionPorTipo("T").getConnection();
            ps = cn.prepareStatement("{call MG_TRAER_CONFIGURACION_SMS(?)}");
            ps.setInt(1, id);

            rs = ps.executeQuery();
            if(rs != null && rs.next()){
                Integer trataId = rs.getInt(1);
                String fieldTelefono = rs.getString(2);
                String fieldParametro = rs.getString(3);
                String smsConfSmsSpeech = rs.getString(4);
                String fromNewMessage = rs.getString(5);
                String ccNewMessage = rs.getString(6);
                String ccoNewMessage = rs.getString(7);
                Integer countAttachmentNewMessage = ObjectUtil.getInteger(rs.getString(8));
                Integer idAccount = rs.getInt(9);
                String fieldMovil = rs.getString(10);
                String subjectNewMessage = rs.getString(11);
                Integer codigoPais = rs.getInt(12);

                SmsConfigure smsConfigure = new SmsConfigure(trataId, fieldTelefono, fieldParametro, smsConfSmsSpeech,
                        fromNewMessage, ccNewMessage, ccoNewMessage, subjectNewMessage, countAttachmentNewMessage, fieldMovil, i
                return smsConfigure;
            }
```

Figura 84 CLASE "SmsConfigureDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.4.3 CLIENTE WEBSERVICE REST - INFOBIP.-

```
public class InfoBipRestClient {

    private String usuarioPass;
    public static URL URL = null;
    public static final int TIME_OUT = 15000;
    public static final String ERROR_CODE_START = "4";
    static {
        try {
            URL = new URL(SmsConstantes.INFO_BIP.WS_URL_INFOBIP);
        } catch (Exception e) {
            PrintLogUtil.printError(InfoBipRestClient.class, e);
        }
    }

    public InfoBipRestClient(String usuarioPass) {
        this.usuarioPass = usuarioPass;
    }

    public WsSmsResponse consume(WsSms wsSms, String codigoHash) {
        HttpURLConnection conn = null;
        try {
            String tramaJson = JsonUtil.objectToJsonString(wsSms);
            conn = configureHttpURLConnection();
            agregaHeader(conn);
            agregarJson(conn, tramaJson);

            PrintLogUtil.printInfo(getClass(), "TRAMA ENVIADA " + codigoHash + ": " + tramaJson);

            String response = getResponse(conn);

            PrintLogUtil.printInfo(getClass(), "RESPUESTA INFOBIP " + codigoHash + ": " + response);
            if(StringUtils.isNotBlank(response)) {
                WsSmsResponse wsSmsResponse = (WsSmsResponse)JsonUtil.jsonStringToObject(response, WsSmsResponse.class);
                return wsSmsResponse;
            }
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), e);
```

Figura 85 CLASE "InfoBipRestClient"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.5 CAPA DE SERVICIO.-

3.5.5.4.5.1 INTERFACES.-

Todas las interfaces de servicio para el bundle “SMSTASK” fueron declaradas en el BUNDLE “SCHEDULER”.

3.5.5.4.5.2 IMPLEMENTACIÓN DE INTERFACES

```
package pe.com.atento.smstask.service.impl;

import java.util.ArrayList;

public class EnviaSmsServiceImpl extends GenericService implements EnviaSmsService {
    private EnviaCasoSmsDao enviaCasoSmsDao = (EnviaCasoSmsDaoImpl) SingletonFactory.obtieneInstancia(EnviaCasoSmsDaoImpl.class);

    @Override
    public void enviaSmsViaStore(List<CasoEnvio> listaAenviar, Hilo currentHilo, TareaImpl tarea, Object smsConfigure,
            boolean fromNuevoProceso) throws ServiceException {
        try {
            ThreadExecutor te = tarea.getThreadExecutor();
            if (ListUtil.isNotEmpty(listaAenviar)) {
                reiniciaValoresInactivos(te);
                monitorTareaValoresIniciales(listaAenviar, currentHilo, tarea, fromNuevoProceso);

                SmsConfigure smsConfig = ((SmsConfigure)smsConfigure);
                String speech = smsConfig.getSmsConfSmsSpeech();

                if (StringUtils.isNotBlank(speech)) {
                    String idOperador = tarea.getIdOperador();

                    for (CasoEnvio envio : listaAenviar) {
                        String codigoSms = null;
                        try {
                            String codigoHash = envio.getCodigoHash();
                            if (StringUtils.isNotBlank(codigoHash)) {
                                String speechTemp = speech.replaceFirst(Pattern.quote(Constantes.KEY_MESSAGE.LBL_CODIGO_HASH),codigoHa
                                codigoSms = enviaViaStore(envio, speechTemp, smsConfig, idOperador);
                            }
                            String estado = Constantes.ESTADO_ENVIO.ENVIADO;
                            if (StringUtils.isBlank(estado)) {
                                estado = Constantes.ESTADO_ENVIO.FALLIDO;
                            }
                            this.actualizaEstadoCasoEnvio(envio, codigoSms, estado, tarea);
                        } catch (Exception e) {
                            PrintLogUtil.printError(getClass(), "Error al ENVIAR SMS. CAMPANA: " + tarea.getCampana()
                            + "; CODIGO TAREA: " + tarea.getCodigo() + "; CODIG HASH CASO: "  + envio.getCodigoHash());
```

Figura 86 CLASE “EnviaSmsServiceImpl”

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.4.6 CAPA DE VISTA Y CONTROLADOR.-

3.5.5.4.6.1 CONTROLADORES.-

```
package pe.com.atento.smstask.view;

import java.util.List;

public class EnviaSmsController extends GenericSenderController{
    private static final Integer ID_SMS_SOAP = 365;
    private static final Integer ID_SMS_INFOBIP = 376;
    private boolean viaSoap = false;
    private boolean viaInfoBip = false;
    private SmsConfigure smsConfigure = null;

    private SmsConfigureDao smsConfigureDao = (SmsConfigureDao) SingletonFactory.obtieneInstancia(SmsConfigureDaoImpl.class);
    private ConfiguracionCampanaService configuracionCampanaService = (ConfiguracionCampanaService) SingletonFactory.obtieneInsta
    private EnviaSmsService enviaSmsService = (EnviaSmsService)SingletonFactory.obtieneInstancia(EnviaSmsServiceImpl.class);

    public EnviaSmsController(TareaImpl tarea) throws Exception {
        if(tarea != null){
            this.tarea = tarea;
            instalaConfiguracion();
        }
    }

    @Override
    public void enviaCaso() {
        try {
            List<CasoEnvio> listaAenviar = null;
            String codigoTareaAntesesora = tarea.getCodigoTareaAtensesora();

            if (StringUtils.isBlank(codigoTareaAntesesora)) {// GENERA PROPIOS CASOS
                listaAenviar = ((GenericService)enviaSmsService).obtieneListaCasosPropiosAEnviar(tarea);
            } else {// GENERA CASOS DE TAREA ANTESESORA QUE NO HAYAN SIDO VISTOS
                listaAenviar = ((GenericService)enviaSmsService).obtieneListaCasosAtensesoresAEnviar(tarea);
            }
            if(viaInfoBip) {
                enviaSmsService.enviaSmsViaInfoBip(listaAenviar, currentHilo, tarea, smsConfigure, this.fromNuevoProceso);
                return;
            }
            if(viaSoap) {
```

Figura 87 CLASE "EnviaSmsController"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.5 BUNDLE "EMAILTASK".-

3.5.5.5.1 ARCHIVO MANIFIESTO.-

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: EmailTask
Bundle-SymbolicName: EmailTask
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: emailtask.Activator
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Require-Bundle: Scheduler;bundle-version="1.0.0"
Bundle-ClassPath: .
```

Figura 88 CLASE "EnviaSmsController"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.2 CLASE ACTIVATOR.-

```
package emailtask;

import org.osgi.framework.BundleActivator;

public class Activator implements BundleActivator {

    private static BundleContext context;

    static BundleContext getContext() {
        return context;
    }

    public void start(BundleContext bundleContext) throws Exception {
        Activator.context = bundleContext;
        Launcher.main(null);
    }

    public void stop(BundleContext bundleContext) throws Exception {
        Activator.context = null;
    }

}
```

Figura 89 ACTIVATOR EMAILTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.3 ESTRUCTURA DE CÓDIGO.-

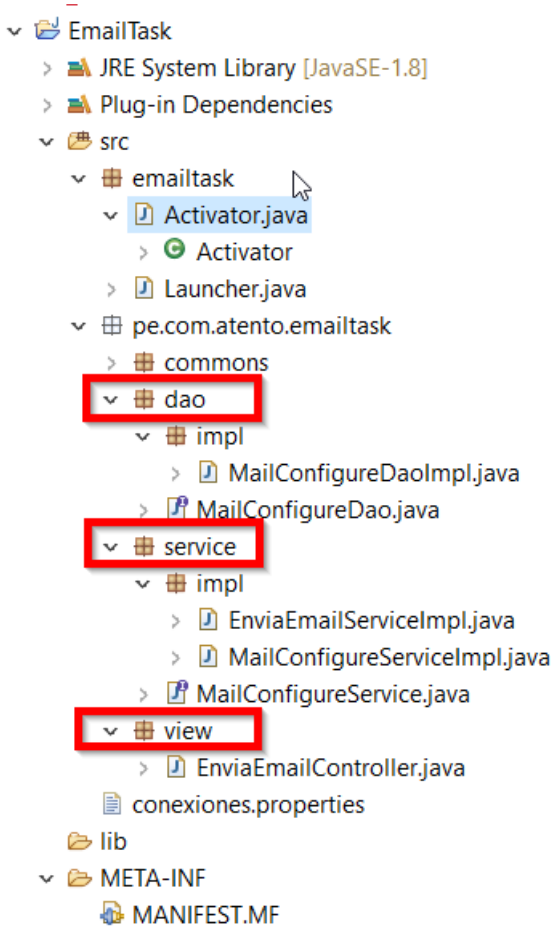


Figura 90 ESTRUCTURA CÓDIGO EMAILTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.4 CAPA DAO.-

3.5.5.5.4.1 INTERFACES.-

```
package pe.com.atento.emailtask.dao;

import pe.com.atento.scheduler.domain.ConfiguracionValidaCampana;

public interface MailConfigureDao {
    public ConfiguracionValidaCampana getConfigMensaje(String idGrupo, String idCodigo, String tipoConexion) throws Exception;
    public Integer getCodigoMail(String action) throws Exception;
    public void insertaMail(Integer codigoMensaje, String mailOrigen, String mailDestino, String copia, String copiaOculta,
            String desAsunto, String desMensaje, Integer contador, String idInbox, String fechaHora, String action, Integer esta
}
```

Figura 91 INTERFACE "MailConfigureDao"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.4.2 IMPLEMENTACIÓN DE INTERFACES.-

```java
package pe.com.atento.emailtask.dao.impl;

import java.sql.Connection;

public class MailConfigureDaoImpl implements MailConfigureDao{

    @Override
    public ConfiguracionValidaCampana getConfigMensaje(String idGrupo, String idCodigo, String tipoConexion) throws Exception {

        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection cn = null;
        try {
            PrintLogUtil.printInfo(getClass(), "getConfigMensaje " + idGrupo + ", " + idCodigo + ", " + tipoConexion);

            cn = VariablesGlobales.getPoolConnectionPorTipo(tipoConexion).getConnection();
            ps = cn.prepareStatement("{call MG_DETALLE_CONFIGURACION(?, ?)}");

            ps.setString(1, idGrupo);
            ps.setString(2, idCodigo);

            rs = ps.executeQuery();
            if(rs != null && rs.next()){
                String campanaResultId = rs.getString(3);
                String campanaDescrip = rs.getString(4);
                String campanaEstado = rs.getString(5);
                String campanaTemaId = rs.getString(6);
                String campanaVdn = rs.getString(1);
                String printXmlConfig = rs.getString(2);

                ConfiguracionValidaCampana config = new ConfiguracionValidaCampana();
                config.setCampanaResultId(campanaResultId);
                config.setCampanaDescrip(campanaDescrip);
                config.setCampanaEstado(campanaEstado);
                config.setCampanaTemaId(campanaTemaId);
                config.setCampanaVdn(campanaVdn);
                config.setPrintXmlConfig(printXmlConfig);
                return config;
            }
```

Figura 92 CLASE "MailConfigureDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.5 CAPA DE SERVICIO.-

3.5.5.5.5.1 INTERFACES.-

```
package pe.com.atento.emailtask.service;

import pe.com.atento.scheduler.domain.ConfiguracionValidaCampana;

public interface MailConfigureService {
    public ConfiguracionValidaCampana obtenerConfigMensaje(String idGrupo, String idCodigo, String tipoConexion) throws Servi
    public Integer obtenerCodigoMail() throws ServiceException;
    public void creaMail(Integer codigoMensaje, String mailOrigen, String mailDestino, String copia, String copiaOculta,
        String desAsunto, String desMensaje, Integer contador, String idInbox, String fechaHora) throws ServiceException;
}
```

Figura 93 INTERFACE "MailConfigureService"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.5.2 IMPLEMENTACIÓN DE INTERFACES

```
package pe.com.atento.emailtask.service.impl;

import java.util.Date;

public class EnviaEmailServiceImpl extends GenericService implements EnviaEmailService{
    private MailConfigureService mailConfigureService = (MailConfigureService)SingletonFactory.obtieneInstancia(MailConfigureServi

    public void enviaEmails(List<CasoEnvio> listCasosEnviar, TareaImpl tarea, Hilo currentHilo,
            String plantilla, String mailOrigen, String asunto, String inbox, boolean fromNuevoProceso) throws ServiceException{
        try {
            ThreadExecutor te = tarea.getThreadExecutor();
            if(ListUtil.isNotEmpty(listCasosEnviar)){
                reiniciaValoresInactivos(te);
                monitorTareaValoresIniciales(listCasosEnviar, currentHilo, tarea, fromNuevoProceso);

                for (CasoEnvio casoEnvio : listCasosEnviar) {
                    Integer codigoEmail = null;
                    try {
                        codigoEmail = mailConfigureService.obtenerCodigoMail();
                        if(codigoEmail == null)
                            throw new Exception("No se ha podido obtener CODIGO EMAIL");
                        String codigoHash = casoEnvio.getCodigoHash();
                        PrintLogUtil.printInfo(getClass(), "Enviado email '" + codigoHash + "', codigo email: " + codigoEmail);
                        String mailDestino = casoEnvio.getEmail();
                        String msg = plantilla.replaceFirst(Pattern.quote(Constantes.KEY_MESSAGE.LBL_CODIGO_HASH), casoEnvio.getCo
                        String fecha = String.format(MailConstantes.FORMAT_FECHA_MAIL_CENTER, new Date());

                        mailConfigureService.creaMail(codigoEmail, mailOrigen, mailDestino, null, null, asunto, msg, 0,
                            inbox, fecha);

                        PrintLogUtil.printInfo(getClass(), "Email enviado '" + codigoHash + "', codigo email: " + codigoEmail);

                        actualizaEstadoCasoEnvio(casoEnvio, codigoEmail, Constantes.ESTADO_ENVIO.ENVIADO, tarea);
                    } catch (Exception e) {
                        PrintLogUtil.printError(getClass(), "Error al enviar Email. " + tarea);
                        actualizaEstadoCasoEnvio(casoEnvio, codigoEmail, Constantes.ESTADO_ENVIO.FALLIDO, tarea);
                        PrintLogUtil.printError(getClass(), e);
                    }
```

Figura 94 CLASE "MailConfigureService"

Fuente: (Subversion SVN Atento Perú, 2019)

```java
package pe.com.atento.emailtask.service.impl;

import pe.com.atento.emailtask.commons.MailConstantes;

public class MailConfigureServiceImpl implements MailConfigureService{
    private MailConfigureDao mailConfigureDao = (MailConfigureDao)SingletonFactory.obtieneInstancia(MailConfigureDaoImpl.class);

    @Override
    public ConfiguracionValidaCampana obtenerConfigMensaje(String idGrupo, String idCodigo, String tipoConexion) throws ServiceE
        try {
            if(StringUtils.isBlank(idGrupo))
                throw new ValidacionException("Error obtener CONFIGURACIÓN MENSAJE. ID GRUPO:" + idGrupo);

            if(StringUtils.isBlank(idCodigo))
                throw new ValidacionException("Error obtener CONFIGURACIÓN MENSAJE. idCodigo:" + idCodigo);

            if(StringUtils.isBlank(tipoConexion))
                throw new ValidacionException("Error obtener CONFIGURACIÓN MENSAJE. tipoConexion:" + tipoConexion);

            return mailConfigureDao.getConfigMensaje(idGrupo, idCodigo, tipoConexion);
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), e);
            throw new ServiceException(e);
        }
    }

    @Override
    public Integer obtenerCodigoMail() throws ServiceException {
        try {
            return mailConfigureDao.getCodigoMail(MailConstantes.ACTION_MAIL_CENTER);
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), e);
            throw new ServiceException(e);
        }
    }
```

Figura 95 CLASE "MailConfigureServiceImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.5.6 CAPA DE VISTA Y CONTROLADOR.-

3.5.5.5.6.1 CONTROLADORES.-

```java
package pe.com.atento.emailtask.view;

import java.util.List;

public class EnviaEmailController extends GenericSenderController{
    private String mailOrigen;
    private String inbox;
    private String asunto;
    private String plantilla;

    private ConfiguracionCampanaService configuracionCampanaService = (ConfiguracionCampanaService)SingletonFactory.obtieneInsta
    private MailConfigureService mailConfigureService = (MailConfigureService)SingletonFactory.obtieneInstancia(MailConfigureSer
    private EnviaEmailService enviaEmailService = (EnviaEmailService) SingletonFactory.obtieneInstancia(EnviaEmailServiceImpl.cl

    public EnviaEmailController(TareaImpl tarea) throws Exception {
        if(tarea != null){
            this.tarea = tarea;
            instalaConfiguracion();
        }
    }

    @Override
    public void enviaCaso() {
        try {
            if(StringUtils.isNotBlank(mailOrigen) && StringUtils.isNotBlank(inbox)
                    && StringUtils.isNotBlank(asunto) && StringUtils.isNotBlank(plantilla)){

                String codigoTareaAntesesora = tarea.getCodigoTareaAtensesora();
                List<CasoEnvio> listCasosEnviar = null;
                if(StringUtils.isBlank(codigoTareaAntesesora)){
                    listCasosEnviar = ((GenericService)enviaEmailService).obtieneListaCasosPropiosAEnviar(tarea);
                }
                else{
                    listCasosEnviar = ((GenericService)enviaEmailService).obtieneListaCasosAtensesoresAEnviar(tarea);
                }
                enviaEmailService.enviaEmails(listCasosEnviar, this.tarea, this.currentHilo,
                        plantilla, mailOrigen, asunto, inbox, this.fromNuevoProceso);
            }
```

Figura 96 CLASE "EnviaEmailController"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.4.6 BUNDLE “WHATSAPPTASK”.-

3.5.5.6.1 ARCHIVO MANIFIESTO.-

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: WhatsappTask
Bundle-SymbolicName: WhatsappTask
Bundle-Version: 1.0.0.qualifier
Bundle-Activator: whatsapptask.Activator
Bundle-RequiredExecutionEnvironment: JavaSE-1.8
Import-Package: org.osgi.framework;version="1.3.0"
Bundle-ActivationPolicy: lazy
Require-Bundle: Scheduler;bundle-version="1.0.0"

```

Figura 97 MANIFIEST.MF WHATSAPPTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.6.2 CLASE ACTIVATOR.-

```
package whatsapptask;

import org.osgi.framework.BundleActivator;

public class Activator implements BundleActivator {

    private static BundleContext context;

    static BundleContext getContext() {
        return context;
    }

    public void start(BundleContext bundleContext) throws Exception {
        Activator.context = bundleContext;
        Launcher.main(null);
    }

    public void stop(BundleContext bundleContext) throws Exception {
        Activator.context = null;
    }

}
```

Figura 98 ACTIVATOR WHATSAPPTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.6.3 ESTRUCTURA DE CÓDIGO.-

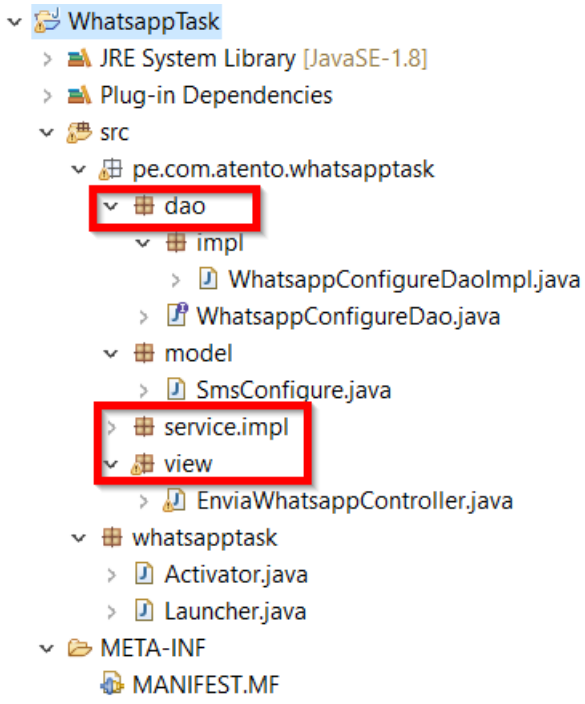


Figura 99 ESTRUCTURA DE CÓDIGO WHATSAPPTASK

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.6.4 CAPA DAO.-

3.5.5.6.4.1 INTERFACES.-

```
package pe.com.atento.whatsapptask.dao;

import pe.com.atento.whatsapptask.model.SmsConfigure;

public interface WhatsappConfigureDao {
    public SmsConfigure getWhatsappConfigurePorId(Integer id, String tipoConexion) throws Exception;
}
```

Figura 100 INTERFACE "WhatsappConfigureDao"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.6.4.2 IMPLEMENTACIÓN DE INTERFACES.-

```java
package pe.com.atento.whatsapptask.dao.impl;

import java.sql.Connection;

public class WhatsappConfigureDaoImpl implements WhatsappConfigureDao{

    @Override

    public SmsConfigure getWhatsappConfigurePorId(Integer id, String tipoConexion) throws Exception {
        PreparedStatement ps = null;
        ResultSet rs = null;
        Connection cn = null;
        try {
            PrintLogUtil.printInfo(getClass(), "getSmsConfigurePorId " + id + ", " + tipoConexion );

            cn = VariablesGlobales.getPoolConnectionPorTipo("T").getConnection();
            ps = cn.prepareStatement("{call MG_TRAER_CONFIGURACION_SMS(?)}");
            ps.setInt(1, id);

            rs = ps.executeQuery();
            if(rs != null && rs.next()){
                Integer trataId = rs.getInt(1);
                String fieldTelefono = rs.getString(2);
                String fieldParametro = rs.getString(3);
                String smsConfSmsSpeech = rs.getString(4);
                String fromNewMessage = rs.getString(5);
                String ccNewMessage = rs.getString(6);
                String ccoNewMessage = rs.getString(7);
                Integer countAttachmentNewMessage = ObjectUtil.getInteger(rs.getString(8));
                Integer idAccount = rs.getInt(9);
                String fieldMovil = rs.getString(10);
                String subjectNewMessage = rs.getString(11);
                Integer codigoPais = rs.getInt(12);

                SmsConfigure smsConfigure = new SmsConfigure(trataId, fieldTelefono, fieldParametro, smsConfSmsSpeech,
                        fromNewMessage, ccNewMessage, ccoNewMessage, subjectNewMessage, countAttachmentNewMessage, fieldMovil, id
                return smsConfigure;
```

Figura 101 CLASE "WhatsappConfigureDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.6.5 CAPA DE SERVICIO.-

3.5.5.6.5.1 INTERFACES.-

Todas las interfaces de servicio para el bundle "WHATSAPPTASK" fueron declaradas en el BUNDLE "SCHEDULER".

3.5.5.6.5.2 IMPLEMENTACIÓN DE INTERFACES

```
package pe.com.atento.whatsapptask.service.impl;

import java.util.List;

public class EnviaWhatsappServiceImpl extends GenericService implements EnviaWhatsappService{

    public void enviaWhatsApp(List<CasoEnvio> listaAenviar, TareaImpl tarea, Hilo currentHilo, Object whatsappConfigure, boolean
        try {
            ThreadExecutor te = tarea.getThreadExecutor();
            if(ListUtil.isNotEmpty(listaAenviar)){
                reiniciaValoresInactivos(te);
                monitorTareaValoresIniciales(listaAenviar, currentHilo, tarea, fromNuevoProceso);
                SmsConfigure whatsappConfig = ((SmsConfigure)whatsappConfigure);
                String speech = whatsappConfig.getSmsConfSmsSpeech();
                if (StringUtils.isNotBlank(speech)) {
                    for (CasoEnvio casoEnvio : listaAenviar) {
                        try {
                            String codigoHash = casoEnvio.getCodigoHash();
                            if (StringUtils.isNotBlank(codigoHash)) {
                                String tempSpeech = speech.replaceFirst(Pattern.quote(Constantes.KEY_MESSAGE.LBL_CODIGO_HASH),co
                                enviaMsgWhatsapp(casoEnvio, tempSpeech, whatsappConfig);
                            }
                        } catch (Exception e) {
                            PrintLogUtil.printError(getClass(), "Error al ENVIAR WHATSAPP. CAMPANA: " + tarea.getCampana()
                            + "; CODIGO TAREA: " + tarea.getCodigo() + "; CODIG HASH CASO: "  + casoEnvio.getCodigoHash());
                            PrintLogUtil.printError(getClass(), e);
                            actualizaEstadoCasoEnvio(casoEnvio, null, Constantes.ESTADO_ENVIO.FALLIDO, tarea);
                        }
                        monitorTareaValoresLuegoDeProcesamiento(currentHilo, tarea);
                        actualizaInterfazMonitor();
                    }

                }
            }
            else {
                incrementaValoresInactivos(te);
                //verificaDestruccionDeTarea(te);
            }
```

Figura 102 CLASE "WhatsappConfigureDaoImpl"

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.5.6.6 CAPA DE VISTA Y CONTROLADOR.-

3.5.5.6.6.1 CONTROLADORES.-

```
package pe.com.atento.whatsapptask.view;

import java.util.List;

public class EnviaWhatsappController extends GenericSenderController{
    private EnviaWhatsappService enviaWhatsappService = (EnviaWhatsappService)SingletonFactory.obtieneInstancia(EnviaWhatsappServ
    private WhatsappConfigureDao whatsappConfigureDao = (WhatsappConfigureDao)SingletonFactory.obtieneInstancia(WhatsappConfigurel
    private SmsConfigure whatsappConfigure = null;

    public EnviaWhatsappController(TareaImpl tarea) throws Exception {
        if(tarea != null) {
            this.tarea = tarea;
            instalaConfiguracion();
        }
    }

    @Override
    public void enviaCaso() {
        try {
            List<CasoEnvio> listaAenviar = null;
            String codigoTareaAntesesora = tarea.getCodigoTareaAtensesora();
            if(StringUtils.isBlank(codigoTareaAntesesora)){//GENERA PROPIOS CASOS
                listaAenviar = ((GenericService)enviaWhatsappService).obtieneListaCasosPropiosAEnviar(tarea);
            }
            else{//GENERA CASOS DE TAREA ANTESESORA QUE NO HAYAN SIDO VISTOS
                listaAenviar = ((GenericService)enviaWhatsappService).obtieneListaCasosAtensesoresAEnviar(tarea);
            }
            enviaWhatsappService.enviaWhatsApp(listaAenviar, this.tarea, this.currentHilo, whatsappConfigure,this.fromNuevoProces
        } catch (Exception e) {
            PrintLogUtil.printError(getClass(), "Error al enviar casos por WHATSAPP. " + tarea);
        }
    }

    @Override
    protected void instalaConfiguracion() throws Exception{

        try {
            Integer campana = tarea.getIdCampana();
```

Figura 103 CLASE "EnviaWhatsappController"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.3.5 CONSTRUCCIÓN DE LA APLICACIÓN WEB.-

3.5.6.1 INTEGRACIÓN HIBERNATE, SPRING Y JSF.-

3.5.6.1.1 Modificaciones en Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
    version="3.1">
    <display-name>TemplateJSF</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            file:${configPath}/spring/application-context*.xml
        </param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>
    <context-param>
        <param-name>log4jConfigLocation</param-name>
        <param-value>file:${configPath}/log4j.properties</param-value>
    </context-param>
    <context-param>
        <param-name>log4jExposeWebAppRoot</param-name>
        <param-value>false</param-value>
    </context-param>
    <context-param>
        <param-name>primefaces.THEME</param-name>
        <param-value>ui-lightness</param-value>
    </context-param>
    <context-param>
        <param-name>javax.faces.DATETIMECONVERTER_DEFAULT_TIMEZONE_IS_SYSTEM_TIMEZONE</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
        <param-name>com.sun.faces.writeStateAtFormEnd</param-name>
        <param-value>false</param-value>
    </context-param>
```

Define donde buscar los archivos de configuración de spring

Define integración con extensión JSF: Primefaces

Figura 104 Web.xml

Fuente: (Subversion SVN Atento Perú, 2019)

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener</listener-class>
</listener>
<listener>
    <listener-class>org.springframework.web.util.Log4jConfigListener</listener-class>
</listener>
```

Declara Lisrtener de Spring

Figura 105 Web.xml

Fuente: (Subversion SVN Atento Perú, 2019)

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

Declara los Servlets de JSF

DEFINE PATRONES DE URL PARA REQUEST DEL SERVLET

Figura 106 Web.xml

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.1.2 MODIFICACIONES EN "FACES-CONFIG.XML".-

```
<?xml version="1.0" encoding="UTF-8"?>

<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd"
    version="2.1">

    <application>
        <el-resolver>org.springframework.web.jsf.el.SpringBeanFacesELResolver</el-resolver>
        <locale-config>
            <default-locale>es</default-locale>
            <supported-locale>es</supported-locale>
        </locale-config>
    </application>
    <lifecycle>
        <phase-listener>pe.atentoweb.util.web.LifeCycleListener</phase-listener>
    </lifecycle>
</faces-config>
```

Delega a SPRING la administración de BEANS

Define listener propio para manejar ciclo de vida JSF

Figura 107 faces-config.xml

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.1.3 ARCHIVOS DE CONFIGURACIÓN SPRING.-

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context              http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/tx                   http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
        http://www.springframework.org/schema/aop                  http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

    <!-- Activa el scanning del @Autowired -->
    <context:annotation-config/>

    <!-- Activa el scanning de las clases marcadas con @Component, @Repository y @Service -->
    <context:component-scan base-package="pe.atentoweb.dao" />
    <ehcache:annotation-driven cache-manager="cacheManager" />

    <ehcache:config cache-manager="cacheManager">
        <ehcache:evict-expired-elements interval="1" />
    </ehcache:config>

    <bean id="cacheManager"
        class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
        <property name="configLocation" value="classpath:ehcache.xml" />
    </bean>

    <!-- Transaction manager para session factory -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <!-- SessionFactory -->
```

Figura 108 application-context-dao.xml

Fuente: (Subversion SVN Atento Perú, 2019)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context              http://www.springframework.org/schema/context/spring-context-4.0
        http://www.springframework.org/schema/tx                   http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
        http://www.springframework.org/schema/aop                  http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
        http://www.springframework.org/schema/mvc                  http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <!-- Activa el scanning del @Autowired -->
    <context:annotation-config/>
    <!-- Activa el scanning de las clases marcadas con @Component, @Controller, @Repository y @Service -->
    <context:component-scan base-package="pe.atentoweb.service" />
    <aop:config>
        <aop:pointcut id="txManagerMethods"
            expression="execution(public * pe.atentoweb.service.*.*(..))"/>
        <aop:advisor advice-ref="txAdvice" pointcut-ref="txManagerMethods"/>
    </aop:config>

    <tx:advice id="txAdvice" transaction-manager="transactionManager">
        <tx:attributes>
            <tx:method name="obtener*" read-only="true" />
            <tx:method name="listar*" read-only="true" />
            <tx:method name="validar*" read-only="true" />
            <tx:method name="buscar*" read-only="true" />
            <tx:method name="*" rollback-for="Throwable" propagation="REQUIRED" />
        </tx:attributes>
    </tx:advice>
    <!-- ACTIVA TRANSACIONES MEDIANTE ANOTACIONES -->
    <tx:annotation-driven transaction-manager="transactionManager" />
</beans>
```

Figura 109 application-context-service.xml

Fuente: (Subversion SVN Atento Perú, 2019)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:context="http://www.springframework.org/schema/context"
    xmlns:tx="http://www.springframework.org/schema/tx" xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd
        http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

    <!-- Activa el scanning del @Autowired -->
    <context:annotation-config/>
    <!-- Activa el scanning de las clases marcadas con @Component, @Controller, @Repository y @Service -->
    <context:component-scan base-package="pe.atentoweb.web" />
    <bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">
        <property name="scopes">
            <map>
                <entry key="view">
                    <bean class="pe.atentoweb.util.web.ViewScope"/>
                </entry>
            </map>
        </property>
    </bean>
</beans>
```

Figura 110 application-context-web.xml

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.1.4 ADMINISTRADOR DE CACHE HIBERNATE "ehcache.xml"

ehcache.xml

templatejsf/src/main/resources/ehcache.xml

```xml
<?xml
<ehcache>

    <defaultCache maxElementsInMemory="100" eternal="false"
        timeToIdleSeconds="120" timeToLiveSeconds="200" />

    <cache name="cache01"
        maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120"
        timeToLiveSeconds="120" />

    <cache name="cache02"
        maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120"
        timeToLiveSeconds="120" />

    <cache name="cache03"
        maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120"
        timeToLiveSeconds="120" />

    <cache name="cache04"
        maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120"
        timeToLiveSeconds="120" />
</ehcache>
```

Figura 111 ehcache.xml.xml

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.2 ESTRUCTURA DE CÓDIGO.-

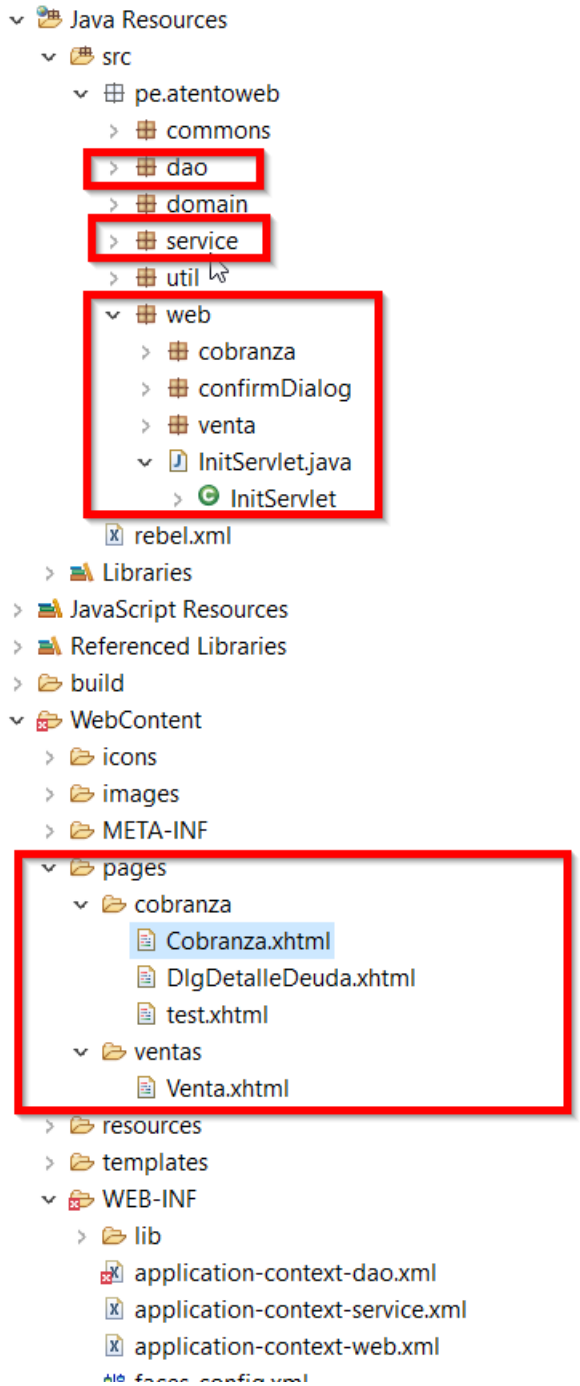


Figura 112 ESTRUCTURA DE CÓDIGO APP WEB

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.3 CAPA DAO INTERFACES.-

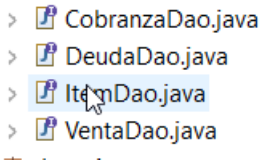


Figura 113 INTERFACES DAO

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.4 CAPA DAO IMPLEMENTACIONES

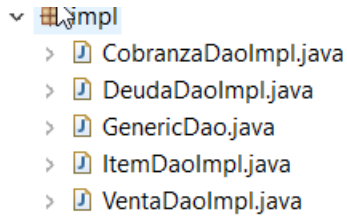


Figura 114 IMPLEMENTACIONES DAO

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.5 CAPA DE SERVICIO INTERFACES

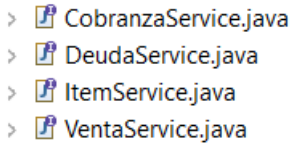


Figura 115 INTERFACES SERVICE

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.6 CAPA DE SERVICIO IMPLEMENTACIONES

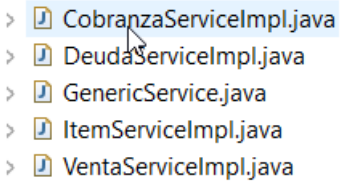


Figura 116 IMPLEMENTACIONES SERVICE

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.7 VISTAS.-

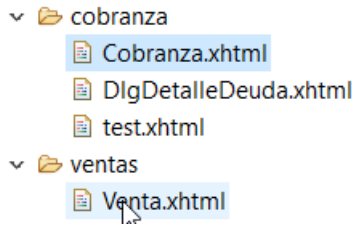


Figura 117 VISTAS

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.8 CONTROLADORES.-

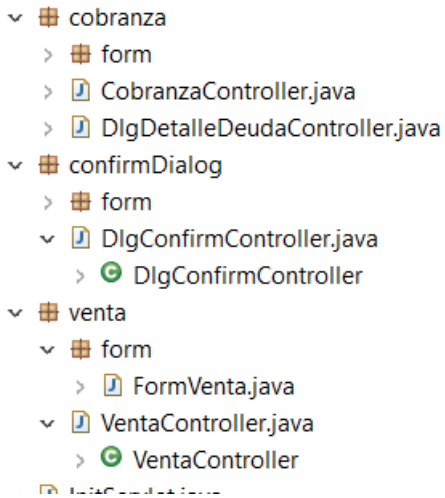


Figura 118 CONTROLADORES

Fuente: (Subversion SVN Atento Perú, 2019)

3.5.6.9 LISTENER CICLO DE VIDA JSF.-

LifeCycleListener.java

```
17
18 public class LifeCycleListener  implements PhaseListener{
19
20     private static final long serialVersionUID = -2107601277145449199L;
21
22     @Override
23     public void afterPhase(PhaseEvent event) {
24
25     }
26
27     @Override
28     public void beforePhase(PhaseEvent event) {
29         if(!event.getFacesContext().getPartialViewContext().isAjaxRequest()){
30             HttpServletRequest request = JsfUtil.getRequest();
31             String ru = request.getRequestURI();
32             boolean isPageCobranza = ru.equals(Constantes.PAGE_COBRANZA);
33             boolean isPageVenta = ru.equalsIgnoreCase(Constantes.PAGE_VENTA);
34             if(request.getMethod() != "HEAD" && (isPageCobranza || isPageVenta)){
35                 String codigo = request.getParameter(Constantes.PARAM_CODIGO);
36                 JsfUtil.getSessionMap().put(Constantes.PARAM_CODIGO, codigo);
37                 if(StringUtil.isBlank(codigo))
38                     return;
39                 try {
40                     ELContext elContext = FacesContext.getCurrentInstance().getELContext();
41                     ELResolver elResolver = FacesContext.getCurrentInstance().getApplication().getELResolver();
42                     GenericService genericService = (GenericService)(elResolver.getValue(elContext, null, "genericService"));
43                     genericService.actualizaContadorYEstado(codigo);
44                     if(isPageVenta) {
45                         VentaController vc = (VentaController)(elResolver.getValue(elContext, null, "ventaController"));
46                         vc.show();
47                     }
48                     else if(isPageCobranza) {
49                         CobranzaController cc = (CobranzaController)(elResolver.getValue(elContext, null, "cobranzaController"));
50                         cc.show();
51                     }
52                 } catch (Exception e) {
53                     PrintLogUtil.printError(getClass(), e);
54                 }
```

Figura 119 CLASE LIFE LISTENER "LifeCycleListener"

Fuente: (Subversion SVN Atento Perú, 2019)

3.2.4 PRUEBAS.-

Las pruebas de un desarrollo en ATENTO PERÚ consisten en una demo del aplicativo con llevada a cabo por el líder de equipo del GRUPO CRM y el desarrollador, con la participación de los supervisores y jefes de la unidad de negocio dedicada al cliente CENCOSUD CHILE.

Las pruebas consistieron en lo siguiente:

a. Generar registros en base de datos de cuatro (04) clientes potencialmente morosos. Se utilizó números telefónicos de chile y correos electrónicos pertenecientes a los jefes de negocio de CENCOSUD CHILE.

b. El motor de cobranzas inicio el proceso de cobranza mediante llamadas telefónicas (IVRTASK).

c. Algunos números no respondieron las llamadas.

d. Para aquellas llamadas sin respuesta, la tarea de cobranza SMS TASK realizo el reintento de cobranza mediante SMS.

e. Todos los números de teléfono que recibieron el SMS no registraron una fecha de pago mediante la aplicación web.

f. Para aquellos SMS que no hayan registrado fecha de pago, la tarea EMAILTASK realizo el reintento de cobranza.

g. La tarea WHATSAPPTASK realizo el reintento de cobranza para los emails que no hayan registrado fecha de pago mediante la aplicación web.

3.2.5 DESPLIEGUE

3.7.1 EVIDENCIA FORMATO DE PASE.-

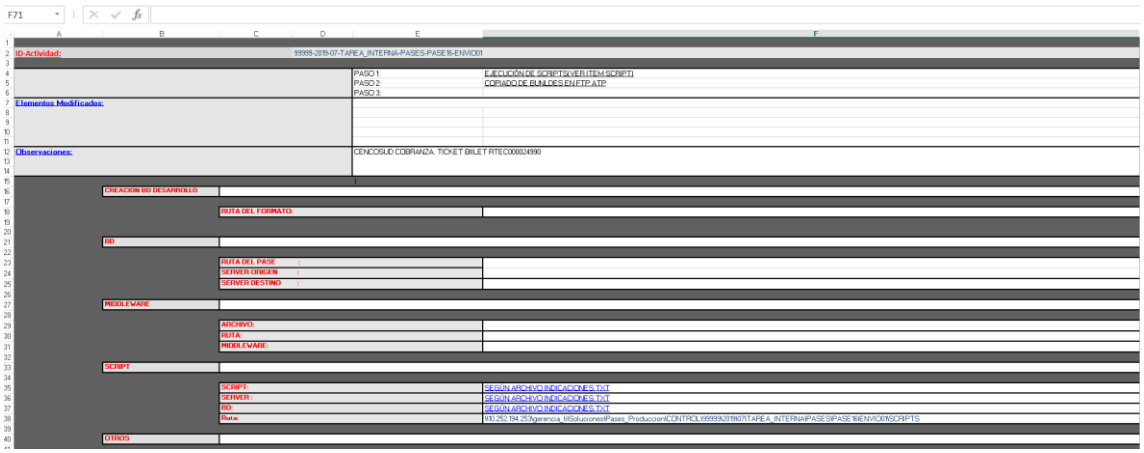


Figura 120 FORMATO PASE A PRODUCCIÓN ENVIADO

Fuente: (Atento Perú, 2019)

Figura 121 FORMATO PASE A PRODUCCIÓN ENVIADO

Fuente: (Atento Perú, 2019)

3.7.2 SCRIPTS PASE A PRODUCCIÓN.-

1.- CREATE TABLE SCHEDULER_TASK INSERTS.sql	10/07/2019 11:18	Archivo SQL	1 KB
2.- CREATE TABLE [dbo].[scheduler_task_x_campana].sql	10/07/2019 11:20	Archivo SQL	2 KB
3.- create procedure [dbo].[obtiene_tarea_por_campana_codigo].sql	10/07/2019 12:05	Archivo SQL	2 KB
4.- create procedure [dbo].[creaActualizaSchedulerTask].sql	10/07/2019 12:02	Archivo SQL	4 KB
5.- create PROCEDURE [dbo].[BORRA_SchedulerTask].sql	10/07/2019 12:03	Archivo SQL	1 KB
6.- CREATE PROCEDURE [dbo].[SP_VALIDA_CAMPANAS_POR_CAMPANA_ID].sql	10/07/2019 11:39	Archivo SQL	2 KB
7. create procedure [dbo].[scheduler_obtieneListaTareasPorEstadoCodigo].sql	10/07/2019 11:33	Archivo SQL	2 KB
8. create procedure [dbo].[obtiene_tarea_por_campana_codigo].sql	10/07/2019 11:33	Archivo SQL	2 KB
indicaciones	12/07/2019 15:41	Documento de tex...	1 KB

1.- CREATE TABLE [dbo].[maestro_envio_...	12/07/2019 14:45	Archivo SQL	1 KB
2.- add columnas estados.sql	12/07/2019 14:46	Archivo SQL	1 KB
3.- CREATE SEQUENCE seq_idproceso_en...	9/05/2019 11:52	Archivo SQL	1 KB
4.- create table mt_token.sql	22/05/2019 17:28	Archivo SQL	1 KB
5.- CREATE procedure [dbo].[getFirstIdCa...	10/07/2019 11:50	Archivo SQL	1 KB
6.- CREATE procedure [dbo].[genera_mt_...	10/07/2019 11:53	Archivo SQL	2 KB
7.- CREATE procedure [dbo].[generaMae...	10/07/2019 11:42	Archivo SQL	12 KB
8.- CREATE PROCEDURE [dbo].[ACTUALI...	10/07/2019 11:43	Archivo SQL	1 KB
9. create procedure [dbo].[update_mt_res...	10/07/2019 13:50	Archivo SQL	3 KB
indicaciones	12/07/2019 15:06	Documento de tex...	1 KB

3.7.3 BUNDLES.-

EMAILTASK	31/05/2019 10:44	Executable Jar File	1,886 KB
IVRTASK	31/05/2019 10:44	Executable Jar File	2,356 KB
SCHEDULERTASK	31/05/2019 10:44	Executable Jar File	5,108 KB
SMSTASK	31/05/2019 10:44	Executable Jar File	2,356 KB
WHATSAPPTASK	31/05/2019 10:44	Executable Jar File	1,110 KB

3.7.4 EVIDENCIA CORREO DE PASE A PRODUCCIÓN

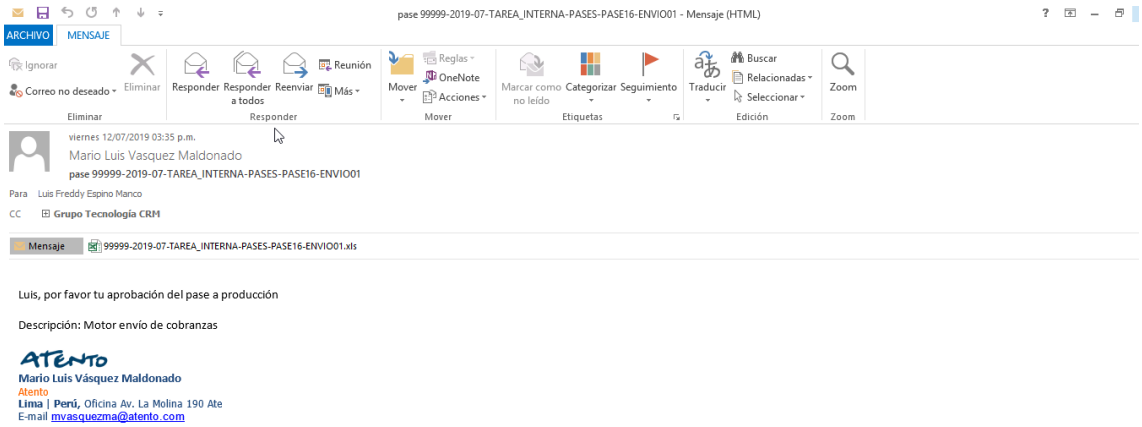


Figura 122 FORMATO CORREO PASE A PRODUCCIÓN ENVIADO

Fuente: (Atento Perú, 2019)

3.7.5 EVIDENCIA CORREO CONFIRMACIÓN PASE A PRODUCCIÓN.-

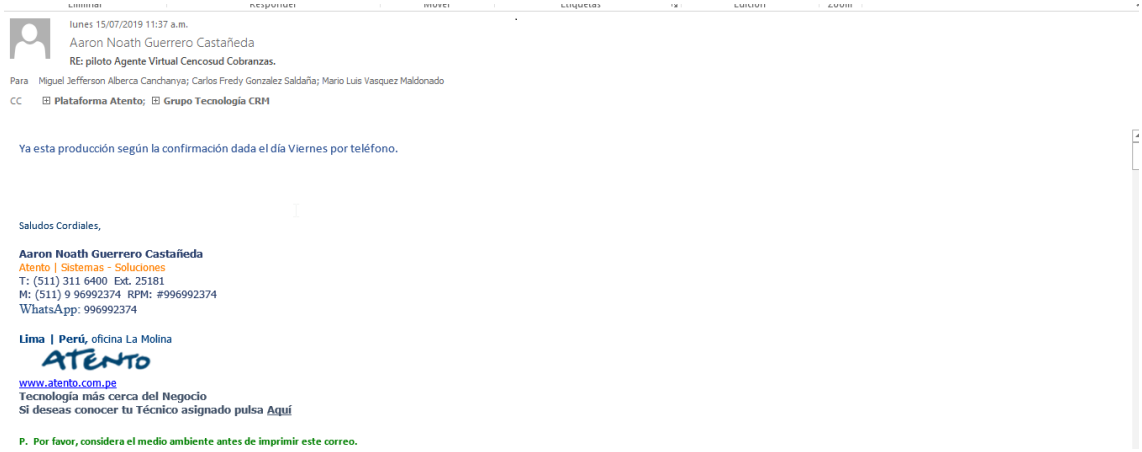


Figura 123 FORMATO CORREO CONFIRMACIÓN PASE A PRODUCCIÓN

Fuente: (Atento Perú, 2019)

4. LECCIONES APRENDIDAS Y PROYECCIÓN PROFESIONAL

4.1 LECCIONES APRENDIDAS.-

Debo decir que, al terminar la universidad tuve mucha suerte de iniciar mi carrera profesional en la consultora de software CONASTEC, puesto que aprendí a desarrollar software de calidad respetando patrones de diseño para la construcción de software, aplicando los más altos estándares de programación java; de hecho teníamos un analista funcional con más de 15 años programando en java quien constantemente revisaba el código fuente producido y hacía notar su voz de enojo en cuanto identificaba algún desperfecto en el código. En CONASTEC todo la LÓGICA DEL NEGOCIO estaba implementada a nivel de APLICACIÓN JAVA, la PROGRAMACIÓN en BASE DE DATOS era nula; se podría decir que se seguía la filosofía "LA BASE DE DATOS DEBE SER TONTA".

En ATENTO PERÚ, si bien es cierto no se construye SOTFWARE DE CALIDAD, gran volumen de la LOGICA DEL NEGOCIO se encuentra a nivel de BASE DE DATOS, se explota casi al máximo todo el potencial de una BASE DE DATOS y esto es algo que muchos programadores de LENGUAJES DE ALTO NIVEL -como JAVA- omiten. Gracias a la programación extensiva en BASE DE DATOS llevada a cabo en ATENTO PERÚ, pude comprender la importancia de usar índices en tablas, optimización de consultas, cómo evitar bloquear tablas con consultas pesadas, consumo de webservices desde un procedimiento almacenado, bloqueos implícitos de tablas que genera una operación de UPDATE, INSERT o DELETE, manipulación de estructuras XML en consultas, uso de LINK SERVER para ejecutar consultas a bases de datos remotas y más.

Entonces, ¿qué es mejor, implementar toda la lógica del negoció en un lenguaje de alto nivel o, por el contrario, se debe implementar en el lenguaje de base de datos? Personalmente creo que se debe buscar un equilibrio. La gran mayoría de la lógica del negocio debe recaer sobre un lenguaje de alto nivel – es más fácil resolver incidencias e identificar problemas en un lenguaje de alto nivel-; sin embargo, se debe aprovechar el poder de procesamiento de una base de datos para tareas o procesos pesados que podrían ejecutarse en horarios de baja transaccionalidad, siempre tratando de evitar la programación excesiva del lado del servidor de base de datos. Este equilibrio entre base de datos y código java se deja evidenciar en el desarrollo del motor de cobranza y el aplicativo web.

4.2 PROYECCIÓN PROFESIONAL.-

Al finalizar mis estudios en la escuela de Computación y Sistemas de UPAO, no se me pasaba por la mente que iba a dedicarme al mundo de la programación, puesto que fui un pésimo alumno en todos los cursos de programación. Estuve cerca de 6 meses postulando a puestos de trabajo que en nada guardaban relación con la carrera que había estudiado: Cajero bancario, asistente administrativo, soporte técnico o asesor telefónico; no logrando calificar para ninguno de esos puestos de trabajo. Dado que no lograba calificar para ningún trabajo, entonces me dispuse a aprender programación y consultas SQL por mi cuenta para así poder aplicar a trabajos afines a la carrera de Computación y Sistemas. Me encerré cerca de dos meses en mi habitación para aprender, por mi cuenta, la programación Java y el lenguaje SQL server en ORACLE.

Desde ese entonces, luego del “encierro forzoso”, soy un apasionado de la programación Java y me proyecto a estar siempre involucrado en el mundo JAVA, desarrollando software de calidad y promoviendo siempre las buenas prácticas de programación. En un futuro me veo dirigiendo mi propio equipo de programadores calificados.

5. FUENTES DE CONSULTA.-

IBM Knowledge Center. (Junio de 2020). *OSGi bundle manifest file*. Obtenido de https://www.ibm.com/support/knowledgecenter/en/SSAW57_9.0.5/com.ibm.websphere.osgi.nd.multiplatform.doc/ae/ra_bundle_mf.html

infobip. (2016). *SMS API Integration Manual*. Obtenido de https://cf-cdn.infobip.com/assets/downloads/SMS_API_Integration_Manual_2016_ES.pdf

infobip. (2020). *programmable communications SMS*. Obtenido de https://dev.infobip.com/#programmable-communications/sms

journaldev. (Mayo de 2015). *JSF Spring Hibernate Integration*. Obtenido de https://www.journaldev.com/7122/jsf-spring-hibernate-integration-example-tutorial

livebook. (Marzo de 2017). *Working with OSGi services*. Obtenido de https://livebook.manning.com/book/spring-dynamic-modules-in-action/chapter-5/

Oracle. (Marzo de 2014). *Especificación Java*. Obtenido de https://docs.oracle.com/javase/specs/

primefaces. (2019). *primefaces showcase*. Obtenido de https://www.primefaces.org/showcase/

spring.io. (Julio de 2015). *Spring transaction manager*. Obtenido de https://docs.spring.io/spring/docs/4.2.x/spring-framework-reference/html/transaction.html

Sun Microsystems. (2006). *Interface PhaseListener*. Obtenido de https://docs.oracle.com/cd/E17802_01/j2ee/j2ee/javaserverfaces/1.2/docs/api/javax/faces/event/PhaseListener.html

tirthalpatel. (Febrero de 2014). Obtenido de http://tirthalpatel.blogspot.com/2014/02/overview-of-osgi-and-how-to-get-started.html

tutorialspoint. (2014). *Java - Multithreading*. Obtenido de https://www.tutorialspoint.com/java/java_multithreading.htm

vogella. (Junio de 2016). *Getting Started with OSGi Declarative Services*. Obtenido de http://blog.vogella.com/2016/06/21/getting-started-with-osgi-declarative-services/

vogella. (17 de Abril de 2020). *vogella*. Obtenido de https://www.vogella.com/tutorials/OSGi/article.html