UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Implementación de funcionalidades en la aplicación web de CRM especializada en el servicio de atención al cliente de una empresa suministradora de agua

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Jiménez Álvarez, José Daniel

Tutor/a: Escobar Román, Santiago

Cotutor/a externo: Ortí Balaguer, Esteban

CURSO ACADÉMICO: 2023/2024

Dedicatoria

A todas aquellas personas que me han ayudado a ser feliz y que forman parte de mi historia.

Agradecimientos

Deseo expresar mi mas sincero agradecimiento a Global Omnium por su confianza depositada en mí para colaborar en departamento de software en mi primera experiencia en el mundo laboral. En especial, querría darle las gracias a mis compañeros de equipo durante estos meses: Borja Salamanca Dominguez, Sergio Rubio Salvador, Carlos Perez Nuñez y Andrés Montserrat Sierra, por su profesionalidad y la forma en la que me han acogido; además de mi tutores, tanto a Santiago Escobar Román por su ayuda y orientación para este trabajo, como a Esteban Ortí Balaguer por confiar en mí para realizar tareas significativas a lo largo de estos meses con el fin de elaborar este trabajo.

Por otra parte, querría agradecer todo el cariño que me han brindado mis padres a lo largo de mi carrera y mi vida, ayudándome en mis momentos más difíciles. También agradezco infinitamente a mis amigos por sus ánimos , compañía y buenos momentos compartidos durante este periodo universitario.

Resumen

En el ámbito de la ingeniería de software, la forma de crear, desarrollar y evolucionar un producto software son muy cambiantes, dependiendo de multitud de factores, como la popularidad de los *frameworks* y lenguajes de programación al iniciar el proyecto. Durante la vida laboral de un desarrollador es habitual trabajar en proyectos ya iniciados, debiendo adaptarse a las tecnologías usadas en dicho proyecto y empresa, que pueden ser totalmente diferentes a las que está acostumbrado. Este Trabajo de Fin de Grado recopila los primeros pasos que un desarrollador atravesará durante su periodo de adaptación al unirse a un proyecto software de cualquier empresa. En primer lugar, comprendiendo la utilidad de la aplicación, al igual que las diferentes funcionalidades que la componen. En cuanto al desarrollo, será vital adaptarse a la metodología de trabajo de la empresa, utilizando sus herramientas de desarrollo, despliegue o análisis de calidad, además de aprender cómo funciona las integraciones del proyecto, como las bases de datos o servicios web. Todos estos aspectos son aplicados a la empresa Global Omnium, en particular a su aplicación web interna especializada en la gestión de relaciones con el cliente (CRM), que es un proyecto Maven escrito en Java para el *back-end* y en JSP para el *front-end*. La empresa utiliza un metodología ágil basada en DevSecOps, utilizando GitLab como repositorio git y herramientas como SonarQube para el análisis estático de código o Jenkins para la integración continua (CI) de código.

Palabras clave: proyecto software, CRM, metodología ágil, Git, SonarQube, Jenkins

Resum

En l'àmbit de l'enginyeria de programari, la manera de crear, desenvolupar i evolucionar un producte programari són molt canviants, depenent de multitud de factors, com la popularitat dels *frameworks* i llenguatges de programació en iniciar el projecte. Durant la vida laboral d'un desenvolupador és habitual treballar en projectes ja iniciats, havent d'adaptar-se a les tecnologies usades en este projecte i empresa, que poden ser totalment diferents a les que està acostumat. Este Treball de Fi de Grau recopila els primers passos que un desenvolupador travessarà durant el seu període d'adaptació en unir-se a un projecte programari de qualsevol empresa. En primer lloc, comprenent la utilitat de l'aplicació, igual que les diferents funcionalitats que la componen. Quant al desenvolupament, serà vital adaptar-se a la metodologia de treball de l'empresa, utilitzant les seues ferramentes de desenvolupament, desplegament o anàlisi de qualitat, a més d'aprendre com funciona les integracions del projecte, com les bases de dades o servicis web. Tots estos aspectes són aplicats a l'empresa Global Omnium, en particular a la seua aplicació web interna especialitzada en la gestió de relacions amb el client (CRM), que és un projecte Maven escrit a Java per al *back-end* i en JSP per al *front-end*. L'empresa utilitza un metodologia àgil basada en DevSecOps, utilitzant GitLab com a repositori git i ferramentes com SonarQube per a l'anàlisi estàtica de codi o Jenkins per a la integració contínua (CI) de codi.

Paraules clau: projecte programari, CRM, metodologia àgil, Git, SonarQube, Jenkins

Abstract

In the field of software engineering, the way of creating, developing and evolving a software product is highly changeable, depending on a multitude of factors, such as the popularity of the frameworks and programming languages at the start of the project. During a developer's working life, it is common to work on projects that have already begun, having to adapt to the technologies used in that project and company, which may be totally different to those they are used to. This Bachelor's Thesis compiles the first steps that a developer will go through during his adaptation period when joining a software project in any company. First of all, understanding the utility of the application, as well as the different functionalities that compose it. In terms of development, it will be vital to adapt to the company's work methodology, using its development, deployment or quality analysis tools, as well as learning how the project's integrations work, such as databases or web services. All these aspects are applied to the company Global Omnium, in particular to its internal web application specialised in customer relationship management (CRM), which is a Maven project written in Java for the *back-end* and in JSP for the *front-end*. The company uses an agile methodology based on DevSecOps, using GitLab as a git repository and tools such as SonarQube for static code analysis or Jenkins for continuous code integration (CI).

Key words: software project, CRM, agile methodologies, Git, SonarQube, Jenkins

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

En toda empresa que proporcione servicios presenciales u online, surge la necesidad de implantar una **infraestructura de datos** que permitan el consumo, almacenamiento y uso compartido de datos como clientes, empleados, recursos u otra multitud de registros dependiendo del contexto de la empresa. La manipulación de esta información se debe realizar a través de un **entorno simple y sencillo de usar**, con el que los empleados sean capaces de interactuar con las bases de datos de forma rápida y segura, sin tener ningún tipo de conocimiento ni de bases de datos ni de programación.

Con este fin, las empresas hacen uso de **aplicaciones de CRM** (Gestión de Relaciones con el Cliente) cuya principal finalidad es la recopilación, organización y análisis de información de los clientes, **integradas en una única plataforma**, ya sea una página web o una aplicación de escritorio. Con la ayuda de este tipo de aplicaciones, los empleados que interactúan con clientes, son capaces de realizar una serie de acciones solicitadas sobre el cliente en cuestión de forma eficiente y segura, al mismo tiempo que se registran automáticamente en el sistema las acciones llevadas a cabo, quedando constancia de toda la información relativa a la gestión.

En Global Omnium, se tomó lo decisión de **desarrollar un software web de CRM propio** para dotar a este sistema con todas las complejas funcionalidades que una empresa **suministradora de agua** necesita. Este proyecto se inició hace más de una década, por lo que el proyecto se encuentra en la etapa de mantenimiento y evolución. Algunas de las gestiones más sencillas que esta aplicación permite realizar son: dar una persona de alta o de baja, realizar el traspaso de un suministro de agua a otra persona, generar el documento físico de una factura, ver el historial de gestiones del cliente, entre otras muchas más.

A lo largo de este trabajo, se explicará en detalle en qué consisten este tipo de aplicaciones de gestión de relaciones con el cliente, siendo el objeto de estudio el creado por Global Omnium, analizando cómo está diseñado este proyecto, haciendo especial hincapié en los *frameworks* utilizados y las funciones más comunes del mismo. Como principal finalidad, se analizarán tres casos de uso con diferentes desarrollos del proyecto, siguiendo las metodologías ágiles instauradas en la empresa.

1.1 Motivación

Siempre he sentido curiosidad por cómo funcionan los sistemas informáticos a nivel de **diseño e infraestructura**. En particular, cómo están diseñadas las aplicaciones más populares de la actualidad, qué tecnologías se utilizaron para crear el *front-end* y *back-end*, cómo están implementados los diferentes módulos, qué servicios web implementan, cómo se *testean* y distribuyen las diferentes versiones de un aplicación, etc. Gracias a la empresa Global Omnium he tenido la oportunidad de aprender sobre todos estos aspectos de la ingeniería del software, gracias a mi colaboración en uno de sus proyectos internos. A pesar de haber estado colaborando únicamente en la aplicación de CRM, se me ofreció incorporarme a otros proyectos, como el desarrollo de la nueva aplicación móvil u otros desarrollos con diversas aplicaciones más especializadas en el ámbito de suministración del agua. Sin embargo, dada mi corta estancia de prácticas y la multitud de conceptos complejos necesarios para comprender el resto de aplicaciones, la opción más realista era la de enfocarse en el software de CRM.

1.2 Objetivos

El objetivo principal de este trabajo reside en **implementar nuevas funcionalidades en un producto software** que se encuentra en la etapa de mantenimiento y evolución.

Para llegar a este objetivo, será necesario:

- Comprender cómo está diseñado el proyecto.
- Entender cómo funciona la aplicación.
- Aprender qué tecnologías se hacen uso.
- Interiorizar la metodología de desarrollo.

1.3 Estructura de la memoria

En este trabajo se presentarán de forma líneal la forma los conceptos necesarios para adaptarse al ciclo de vida de un proyecto software. En primer lugar, en el capítulo *Sobre Aguas de Valencia: la actual Global Omnium* se hará un breve recorrido por la historia de la empresa para comprender el contexto actual en el que se sitúa, a través de su años de historia y su progresiva digitalización. Además, se incluirá, un Glosario en el contexto de la empresa que será imprescindible para entender ciertos conceptos vitales que serán repetidos a lo largo de este documento.

A continuación, en *Descripción de la herramienta* se expondrá tanto el concepto general de software de CRM, como la navegación y utilidad de las funcionalidades principales de la aplicación CRM que utiliza esta empresa. En el siguiente capítulo, en *Metodología de trabajo*, se examinarán todas las tecnologías y metodologías involucradas en el desarrollo de software del proyecto, para finalmente poder comprender adecuadamente los tres Casos de uso, exponiendo para cada uno qué errores se detectan, cómo se llega a elaborar el análisis de errores con su posibles soluciones, y la implementación final, probando que funcionan correctamente.

1.4 Convenciones

A lo largo de la memoria, se utilizarán las siguientes **normativas de marcado** para dotar a determinadas palabras o expresiones con significado adicional:

- Se utilizará la **negrita** en palabras o expresiones de vital importancia en ese preciso momento y contexto, o para destacar información de forma llamativa.
- Se utilizará la *cursiva* o " " (comillas) en expresiones en inglés; o en nombres o valores exactos extraídos de partes del código fuente o de la interfaz de la aplicación.
- Comenzarán por mayúscula aquellas palabras que en ese contexto aporten un significado relacionado con un componente u objeto de programación. Por ejemplo, Botón o Vista.
- Se utilizarán comentarios entre paréntesis para hacer aclaraciones relacionadas con elementos de la memoria ya explicados

Por otra parte, este trabajo contiene **información sensible** de clientes de la empresa, por lo que se aplicarán la siguiente estrategia para censurarlos:

Para cada imagen, se aplicará un **difuminado** sobre los siguientes datos de:

- Suministros: números de referencia y direcciones.
- Personas: usuarios, nombres y DNIs, teléfonos.
- Empresas: nombres, CIFs y recursos de contacto.
- Códigos de documentos.

El difuminado aplicado será **total o parcial** dependiendo de si esos datos deben ser distinguidos, bien sea para explicar su significado y posición en un interfaz, o bien para demostrar la veracidad de las imágenes en cada caso de uso. Para mostrar datos parcialmente censurados que estén escritos a la largo del trabajo, se utilizarán dos puntos suspensivos "‥" para reflejar que esa parte de información esta censurada.

Finalmente, en cuanto los extractos de **código fuente**, se incluirán las líneas de código necesarias para ayudar la comprensión del contexto en ese momento:

- Se incluirán fragmentos de código pequeños, de pocas líneas, cuando contenga información completa en el contexto que se esté empleando.
- Se incluirán fragmentos de código grandes, de aproximadamente una página, cuando contenga información relativa a la tecnología mencionada o cuando dicho fragmento de código sea equivalente a una función completa.
- Se utilizarán dos puntos suspensivos "‥" para omitir líneas de código irrelevantes en ese contexto, como pueden ser métodos que no interfieren en la función a modificar o fragmentos de código que permanecen intactos a la hora de realizar una modificación.

CAPÍTULO 2

Sobre Aguas de Valencia: la actual Global Omnium

Aguas de Valencia centra su actividad en la gestión del Ciclo Integral del Agua, desarrollando diversas líneas de negocio complementarias que generan las sinergias adecuadas para optimizar los recursos hídricos.

La empresa se constituyó en 1890. Nació bajo la denominación de Sociedad de Aguas Potables y Mejoras de Valencia, S.A., con el objetivo de modernizar el abastecimiento de la ciudad de Valencia mediante la construcción de balsas, filtros y depósitos a orillas del río Turia. En la actualidad, con más de 130 años de vida, disfruta del reconocimiento y la estabilidad necesaria para abordar un crecimiento sostenible en los nuevos territorios y contratos donde aporte valor.

Actualmente, gestiona los aspectos relacionados con la **captación, tratamiento y distribución de agua potable** en Valencia y en la mayoría de las poblaciones de su área metropolitana, así como desarrolla líneas de negocio complementarias para optimizar los recursos hídricos (abastecimiento y saneamiento, gestión de alcantarillado, depuración de aguas residuales, desnitrificación o gestión de regadío). Asimismo, también administra el Sistema Metropolitano de Abastecimiento que facilita el agua potable, en alta, a los municipios de diversas comarcas valencianas. Para ello, explota dos estaciones potabilizadoras que se abastecen del agua superficial proveniente de los ríos Júcar y Turia (La Presa de Manises y El Realón de Picasent).

Fruto de su larga experiencia y trabajo, Aguas de Valencia evidenció un crecimiento y una expansión que culminó con el **nacimiento del grupo Global Omnium**, una organización empresarial convertida en referente nacional e internacional en el sector del agua. A día de hoy, está presente en **15 Comunidades Autónomas de España** y con **presencia en USA, Latinoamérica, África y Oriente Medio**. A lo largo de su historia, para lograr el desarrollo de las actividades propias y su expansión tanto a nivel nacional como internacional, ha ido creando o participando en diversas sociedades de diferentes áreas geográficas y **diversificando su área de negocio**. Actualmente, agrupa 25 empresas especializadas, 13 firmas participadas y 56 UTEs en diferentes áreas relacionadas con la **gestión del agua, la ingeniería y el medio ambiente**.

Aguas de Valencia, la actual Global Omnium, desde su fundación, nació con la vocación de **abastecer a la ciudad de Valencia (gestionado por EMIVASA) y su área metropolitana** y, de manera progresiva, fue incorporándose a los municipios de las provincias de Valencia, Castellón y Alicante (EGEVASA, Global Omnium Medioambiente SL, Empresa de Aguas y Servicios Públicos de Morella, Aguas de Calpe, Aigües D'Altea). En poco tiempo amplió su proyección a otras comunidades autónomas, en donde también

gestiona servicios relacionados con el Ciclo Integral del Agua (Aguas de Teruel, SASTESA, Companyia General D'Aigües De Catalunya, Aigües d'Tortosa, Aigües d'Altafulla).

Asimismo, posee una presencia cada vez más consolidada en distintas áreas, como en la gestión de la calidad del agua a través del laboratorio (Gamaser), la implantación de sistemas de gestión de atención a clientes (ISG), la instalación, mantenimiento e implantación de contadores inteligentes (MACSA), otros servicios de mantenimiento (SAMAS) y obras hidráulicas y mantenimiento integral de infraestructuras hidráulicas (CCSA).

El Grupo Global Omnium gestiona, a lo largo de la geografía española, 28 plantas de tratamiento de agua potable, abasteciendo a más de 400 ciudades y más de 5 millones de personas. También gestiona 308 instalaciones de tratamiento de aguas residuales, 25 fosas sépticas, 155 bombeos exteriores y 2 instalaciones de fangos.

2.1 Digitalización de la empresa

En 2003, Global Omnium comenzó un proceso pionero de digitalización basado en la **extracción y análisis de los datos provenientes de la sensorización de las infraestructuras**, lo que desembocó en el desarrollo de la plataforma **GoAigua** dentro de la gestora. En la actualidad, esta solución está transformando la gestión no solo en Global Omnium, sino en otras operadoras de todo el mundo.

El desarrollo de la solución GoAigua ha beneficiado a la gestión de varias áreas de la empresa, como por ejemplo: Operaciones, Mantenimiento, Ingeniería, Clientes, Facturación y Contabilidad, aportándoles la experiencia y el saber hacer de una empresa con más de un siglo de historia. La solución GoAigua permite una gestión integrada, sostenible y eficiente del agua, en línea con los objetivos de la Agenda 2030 sobre el Desarrollo Sostenible.

Figura 2.1: Resumen de los beneficios de GoAigua

Ofrece soluciones tecnológicas innovadoras en cada uno de estos ámbitos para impulsar la eficiencia:

- **Abastecimiento** (*GoAigua Water*): Soluciones tecnológicas para optimizar los procesos de captación, potabilización, distribución del agua y ciclo comercial. Digitalización del abastecimiento urbano.

- **Saneamiento** (*GoAigua Wastewater*): Optimización de las plantas de tratamiento de aguas residuales, redes de alcantarillado y aguas pluviales para prevenir eventos y automatizar los procesos de tratamiento.

- **Agricultura** (*Agricultura*): Innovación y automatización de la infraestructura y explotación de riego a través de soluciones tecnológicas. Gestión centralizada del regadío y los usos agrarios.

- **Riego** (*GoAigua Irrigation*): Soluciones tecnológicas para el control integral del riego de zonas ajardinadas y optimización en el consumo de recursos en zonas urbanas.

- **Recursos hídricos** (*GoAigua Water Resources*): Servicio de alerta temprana que incorpora y procesa datos hidrometereológicos medidos o simulados en tiempo real para gestionar eventos en cuencas hidrográficas.

2.2 AS/400 como primer sistema informático

El AS/400, también conocido como IBM iSeries o IBM System i, es una plataforma informática de gama media desarrollada por IBM, lanzada inicialmente en 1988. Se trata de un sistema multiusuario, con una interfaz controlada mediante menús y comandos CL (Control Language) intuitivos que utiliza terminales y un sistema operativo basado en objetos y bibliotecas, denominado OS/400. Un punto fuerte del OS/400 es su integración con la base de datos DB2/400, siendo los objetos del sistema miembros de la citada base de datos. Dichas características suponían una gran revolución al integrar su propia base de datos relacional y soporte multiusuario, y pronto diversas ramas de la industria comenzaron a implantarlo de forma extensiva.

A día de hoy, esta plataforma es conocida por su **robustez, fiabilidad y capacidad para manejar grandes cargas de trabajo** empresariales. Es una máquina que, a pasar de su larga edad, es altamente eficiente e integrable a otros muchos lenguajes de programación actuales como Java, Python, C, C++ o Perl, motivo por el que muchas empresas **no se plantean migrar su información a unas bases de datos con tecnologías actuales como MySQL**.

Este dispositivo fue, como en muchas compañías en todo el mundo, la primera base de datos en Global Omnium, siendo a día de hoy el almacén de los datos más críticos de la empresa, como la información de los abonados, facturas, domiciliaciones bancarias, etc. En esta empresa sí se trabaja para poder migrar estos datos a otra tecnología, como la que ya se emplea para registrar datos relativos a las diversas aplicaciones de la empresa.

2.3 Glosario en el contexto de la empresa

A continuación, se explicarán diferentes términos importantes que se emplean en esta empresa, los cuales serán frecuentemente usados a lo largo de este trabajo:

- **SAC**: Software de CRM especializado en el área del Servicio de Atención al Cliente.

- **Cliente**: Cualquier persona que solicita ponerse en contacto con el servicio de atención al cliente, independientemente de si tiene los servicios de la empresa contratados o no. Si esta persona sí tiene contratada algún servicio de Global Omnium, recibe el nombre de **Abonado**.

- **Suministro**: Punto o lugar específico donde se proporciona el servicio de suministración de agua. Los puntos de Suministro se corresponden con una dirección postal completa. La persona que da de alta un suministro se convertirá en el Titular.
- **Número de Referencia**: Código que identifica un Suministro que está dado de Alta. Está compuesto por un Nia (número que identifica a un Abonado) y un SubNia (número que identifica el contrato del Abonado relacionado que este Suministro). Se representa este código separando el Nia y SubNia mediante el carácter "/", por ejemplo "123456/2" o "65656/1"
- **Funcionalidad**: Acciones que puede realizar un Cliente gracias al Servicio de Atención al Cliente. Por ejemplo, dar de alta un Suministro, pagar factura pendientes, modificar sus datos de abonado, etc.
- **Proceso o Gestión**: Se inicia un Proceso o Gestión cuando el Cliente solicita iniciar una de las funcionalidades. Por ejemplo, cuando el cliente solicita consultar el Histórico de sus Facturas, el personal del Servicio de Atención selecciona la Funcionalidad *Consultar Histórico de sus Facturas*, para que el sistema inicie el proceso para consultar todas las facturas del cliente.
- **Operación o Tarea**: Secuencia de acciones que se realizan durante un proceso. Por ejemplo, iniciar el proceso, seleccionar un suministro, etc.

CAPÍTULO 3
Descripción de la herramienta

Durante este periodo de prácticas, mi abanico de elecciones era relativamente amplio. El departamento de Desarrollo de Software de Aguas de Valencia tiene albergado en un repositorio de GitLab un macro-proyecto Maven escrito en Java llamado *gav* (Grupo Aguas de Valencia) el cual contiene todo el código fuente de 3 proyectos:

- Oficina Virtual (en página web)
- Software de Gestión de Relaciones con el Cliente (CRM)
- Futura nueva APP móvil

Es lógico que estos 3 proyectos estén unificados. La Oficina Virtual, el software de CRM y la APP móvil son plataformas que tienen prácticamente la misma necesidad: la gestión y relación con los clientes. Esto significa que estas plataformas, en cuanto a su desarrollo *back-end*, son muy similares, por lo que es muy conveniente tener todo el código fuente unificado en un mismo proyecto. Sin embargo, obviamente existen diferencias entre éstas, las cuales consisten en cómo el cliente interactúa con la empresa.

La Oficina Virtual y la APP móvil son plataformas públicas en las que el cliente puede realizar gestiones básicas de contratación como dar de alta un suministro, darlo de baja, traspasarlo a otra persona, pagar y consultar facturas, etc. Ambas plataformas están disponibles en cualquier hora del día porque son complemente online y no requieren de la atención del equipo de atención al cliente.

En cuanto al sofware de CRM, es una plataforma a la que solo tiene acceso determinados empleados de la empresa, que es utilizada cuando existe un contacto directo entre el cliente y el personal administrativo de la empresa. En determinados casos, hay gestiones que solo se pueden solucionar mediante esta interacción directa, como resolver una duda por un llamada telefónica, el pago de facturas con dinero en efectivo, o realizar una reclamación.

Dadas estas opciones, mi tutor de empresa y yo decidimos que dada mi experiencia universitaria y la dificultad de aprender la cantidad de servicios que ofrece la empresa, el mejor foco de trabajo seria **el software de CRM**, en el que sería capaz de en pocas semanas adaptarme a las funciones más frecuentes que éste ofrece.

3.1 ¿Qué es un Software de CRM?

En primer lugar, las siglas CRM provienen de "Customer Relationship Management", lo que se podría traducir como Gestión de Relaciones con el Cliente. En este área, se engloban conceptos de estrategias comerciales, prácticas y de soluciones tecnológicas que están enfocadas a la relación con los clientes de una empresa.

En cuanto a los sistemas CRM, son aquellos programas con los que pequeñas y grandes empresas tienen la posibilidad de **mantener organizadas sus relaciones con los clientes**, vinculando todas las gestiones que el cliente solicite realizar. Suelen ser programas que combinan múltiples tareas a la hora de tratar con los clientes, como pueden ser la gestión de ventas, el marketing o **la atención al cliente**.

Gracias a este tipo de software, una empresa es capaz de albergar toda la información sobre los clientes en una base de datos. Posteriormente, el CRM mantendrá un registro de todas tus interacciones con cada uno de ellos, y así **tener centralizada una visión de todos los clientes y sus estadísticas en una única herramienta.**

En cuanto a sus áreas principales, estos programas suelen estar enfocados a los departamentos de marketing y ventas, así como los de atención al cliente. Por lo tanto, los CRM permiten **realizar llamadas a los clientes** con fines comerciales o de solución de problemas, y luego guardan registros del historial de llamadas o de las interacciones de soporte técnico que se ha realizado con cada cliente.

Algunas soluciones de CRM también tienen aplicaciones para teléfonos móviles y tabletas, lo que permite que cualquier comercial o empleado de la empresa pueda consultar los datos de un cliente desde fuera de la oficina. Esto ayuda mucho en casos en los que las empresas tengan empleados que trabajan en remoto o que se desplacen por motivos laborales.

Además de esto, los CRM también son capaces de generar campañas de marketing, automatiza el envío de correos electrónicos y otras comunicaciones, y personaliza puntos de contacto con los clientes. También permite administrar el equipo y el rendimiento del equipo dentro de una empresa.

3.1.1. Motivación de ser implantado en una empresa

Este tipo de sistemas no solo son útiles para **centralizar la información de clientes**, sino que en grandes empresas llegan a ser vitales, porque todas las características que reúnen se centran en **incrementar exponencialmente la productividad** de los empleados y de la empresa:

- **Automatizan procesos de muchas tareas de gestión y administración.** Un CRM puede aumentar la productividad de los empleados, lo que a su vez puede traducirse en un aumento de la eficiencia operativa y una reducción de los costos.

- Permiten segmentar y **clasificar los clientes en diferentes grupos** según diversos criterios de forma sencilla. Gracias a esto, las compañías son capaces de dirigirse de manera más efectiva a grupos específicos de clientes con mensajes y ofertas personalizadas.

- Cuando una empresa ofrece una gran cantidad de servicios diferentes, el tiempo de aprendizaje que supone para una persona nueva es elevado. Sin embargo, cuando las funciones están automatizas y unificadas en un único software, **este tiempo de adaptación se ve disminuido drásticamente**.

Por tanto, en un único software las empresas pueden realizar todo tipo de interacciones con sus clientes.

3.1.2. Tipos

En cuanto a los diferentes tipos de software de CRM, existen varios. Estos son:

- *Operativo:* Se centra en la gestión del marketing, ventas y servicios al cliente. Todos estos procesos son denominados «Front Office» porque la empresa tiene contacto con el cliente.

- *Analítico:* Se corresponde con las diferentes aplicaciones y herramientas que proporcionan información de los clientes, por lo que el CRM analítico está ligado a un depósito de datos o información denominado Data Warehouse. Se utiliza con el fin de tomar decisiones relativas a productos y servicios, y evaluar resultados.

- *Colaborativo:* Su función es centralizar y organizar toda la información y los datos que el cliente proporciona a través de diferentes canales.

El software de Global Omnium se correspondería a un **CRM colaborativo**, pese a estar extendido con otros canales específicos en el ámbito de la empresa.

3.2 SAC de Global Omnium

En el caso de este grupo de aguas, hacen uso de un software interno con el cual tienen implementado todas las funcionalidades relacionadas con el Servicio de Atención al Cliente del área de CRM. Dicho software es al que **nos referimos en la empresa como SAC**.

Este software consiste en una plataforma web, en la que solo se puede acceder desde la red interna de Aguas de Valencia (o mediante su VPN). Además, para entrar a la vista principal es necesario iniciar sesión mediante el nombre de usuario y la contraseña de la empresa.

Al ser un software interno que solo usan los empleados:

- La máxima prioridad es mantener la aplicación estable. Para ello, cada modificación testeada exhaustivamente para certificar que la aplicación se comporte siempre como se espera.

- El diseño de la interfaz no es una prioridad. Si se realiza alguna modificación sobre la interfaz, solo se le presta atención a que tenga un uso intuitivo para los usuarios.

3.2.1. Interfaz inicial y canales

Una vez iniciado sesión con nuestras credenciales de la empresa en el entorno de desarrollo, veremos como primera pantalla todos los canales de comunicación que engloba la Servicio de Atención al Cliente, además de distintas secciones para configurar nuestro usuario:

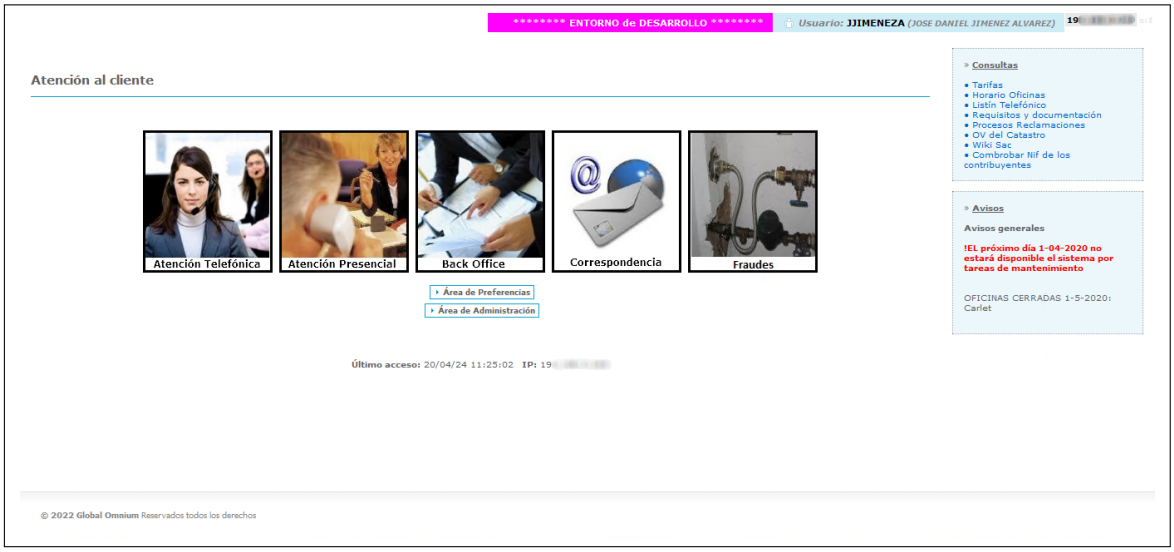


Figura 3.1: Vista principal del SAC

En la figura anterior, podemos detectar que los elementos que más destacan son las 5 imágenes que están colocadas una al lado de la otra, siguiendo una fila horizontal. Estas imágenes describen cada tipo de **canal de comunicación** posible con los clientes, los cuales serán descritos a partir de la próxima sección.

Aparte de los canales de comunicación, podemos distinguir los siguientes elementos:

- Cabecera con información relativa a:
  - Entorno actual (En este caso, de *Desarrollo*).
  - Usuario (En este caso, *JJIMENEZA*).
  - Dirección IP del equipo con el que se ha accedido (En este caso, *19...*).
- Paneles laterales:
  - *Consultas:* Listado de consultas generalmente frecuentes por todos los usuarios.
  - *Avisos:* Listado de mensajes que advierten de futuros cambios en el funcionamiento del sistema.
- *Área de Preferencias:* Apartado donde seleccionar ciertos valores predeterminados dentro del SAC, los cuales se vinculan a nuestro usuario:

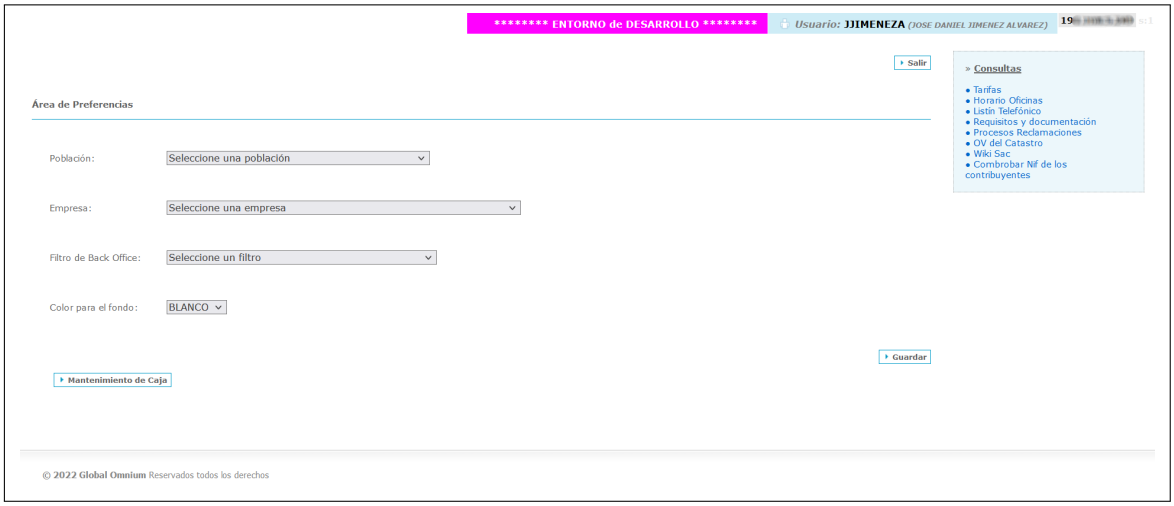


Figura 3.2: Área de preferencias

En la figura anterior se observa como se pueden aplicar un valor por defecto a los valores *Población*, *Empresa* y *Filtro de Back Office*, que es de gran utilidad para aquellos trabajadores que de forma muy recurrente realizan gestiones sobre la misma Población, Empresa o sobre el mismo Filtro dentro del canal Back Office.

Además, es posible cambiar el *color del fondo* de la aplicación y visualizar con el botón *Mantenimiento de Caja*, que permite visualizar en qué empresas el usuario está habilitado para gestionar cobros de facturas.

- *Área de Administración:* Apartado donde solo pueden acceder los administradores del sistema. Es utilizado para monitorizar ciertos aspectos de la aplicación.

- Cadena de texto que indica:

  - Fecha y Hora del último acceso (En este caso, de *20/04/24 11:25:02*).
  - Dirección IP del equipo con el que hemos accedido (En este caso, *19..*).

- Cadena de texto que indica que los *todos los derechos de la plataforma pertenecen a Global Omnium desde 2022*

3.2.2. Canales Telefónico y Presencial

El canal Telefónico y el canal Presencial son los **canales principales del SAC**, puesto que ambos son los utilizados para **registrar el contacto directo** entre el personal del Servicio de Atención al Cliente y un cliente.

La principal diferencia de entre estos dos canales es simplemente la de distinguir si el contacto es presencial o telefónico, dado que dependiendo del tipo del contacto:

- El sistema puede requerir algún dato en concreto. Por ejemplo, en la toma de datos de la persona interesada, si el contacto es telefónico, el sistema solicitará que se aporte un número de teléfono de forma obligatoria.

- Hay funcionalidades que no se pueden llevar a cabo en el telefónico. Normalmente, estas funcionalidades están relacionadas con gestiones que requieran la firma del solicitante, como por ejemplo, al pagar una factura con dinero en efectivo.

El comportamiento de ambos canales es el mismo. Al entrar a cualquiera de los dos, el sistema requerirá el número de identificación NIF/NIE si el cliente es una persona física, o CIF si se trata de una persona jurídica:

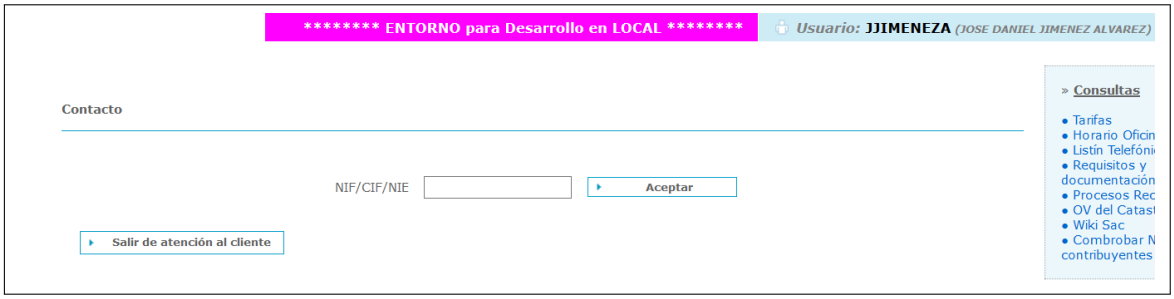


Figura 3.3: Primera vista del canal Telefónico

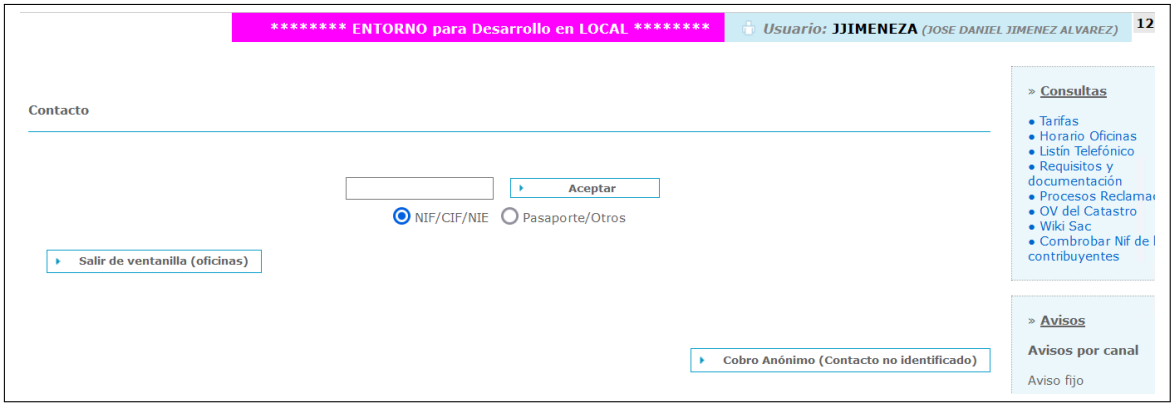


Figura 3.4: Primera vista del canal Presencial

Nótensen las siguientes diferencias:

- Solo en el canal Presencial se permite identificarse mediante el número de pasaporte, u otro número identificativo.

- Solo en el canal Presencial, se permite la opción de realizar un Cobro en el caso de que la persona interesada no se pueda identificar.

Una vez introducido el número identificativo y el sistema compruebe que es un número válido, el sistema detectará si dicha persona ya ha realizado previamente alguna gestión. Si esta persona ya está registrada en el SAC, se abrirá la misma vista de Toma de Datos[1], rellenando los campos que ya se rellenaron anteriormente. Si no lo está, se abrirá la misma ventana de Toma de Datos pero con todos los campos vacíos.

[1]En el canal Telefónico, el campo *Número de Teléfono* aparecería como *campo obligatorio*

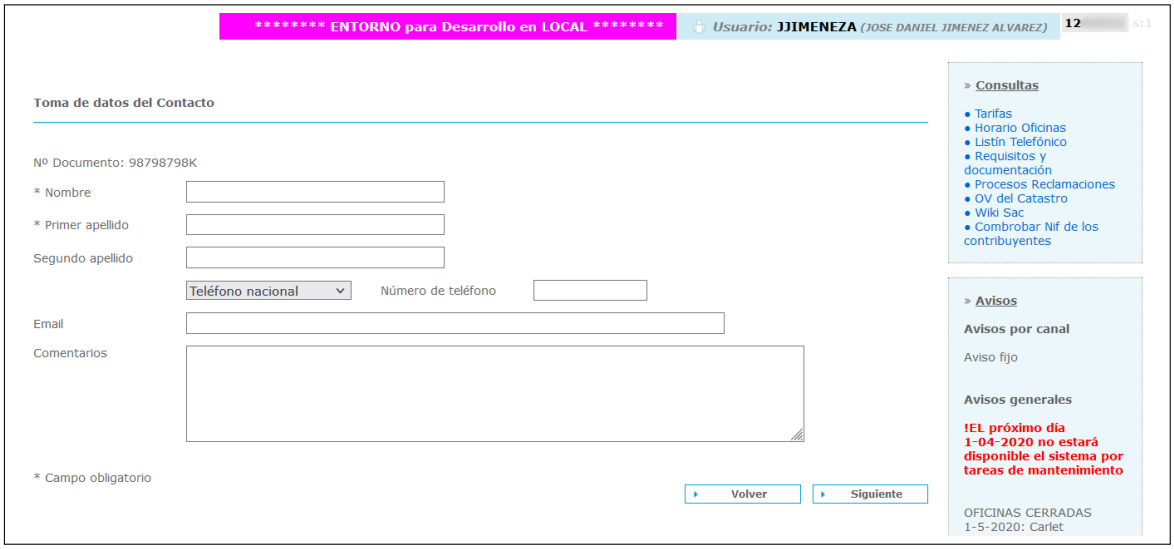


Figura 3.5: Vista de Toma de Datos en el canal Presencial

Al cumplimentar los datos necesarios y presionar el botón *Siguiente*, el SAC redirigirá a la vista principal del contacto:

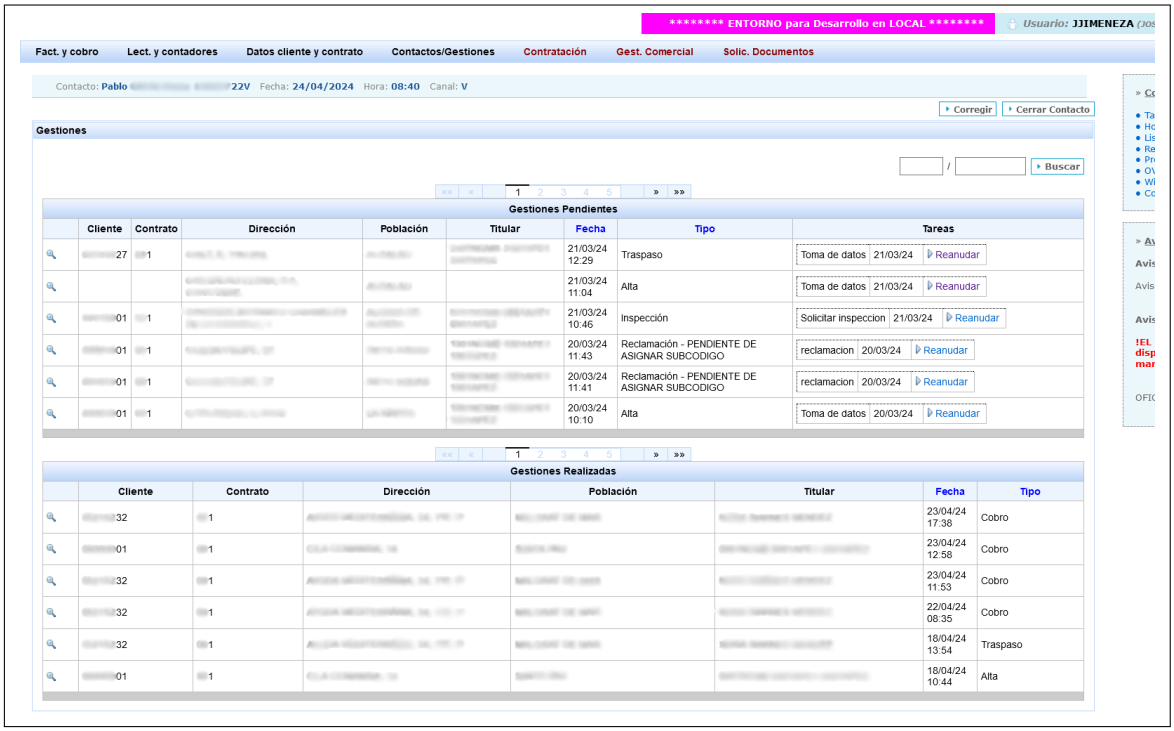


Figura 3.6: Vista principal del contacto

En la figura anterior, se observan los siguientes elementos:

- Menú de Gestiones: Cabecera que recopila todas las gestiones que la persona solicitante puede realizar.
- Cabecera de datos que resumen los datos del contacto. Estos son:
  - Nombre de la persona solicitante (En este caso, *Pablo*).
  - DNI de la persona solicitante (En este caso, acabado en *...22V*).

  - Fecha del contacto (En este caso, *24/04/2024*).
  - Fecha del inicio del contacto (En este caso, *08:40*).
  - Canal por el que se ha establecido el contacto (En este caso, *Ventanilla*). [2]

- Botón *Corregir* para modificar los datos del que se han introducido en la vista de *Toma de Datos*.
- Botón *Cerrar Contacto* para dar por finalizada la atención al cliente y volver a la vista principal del SAC.
- Buscador de Gestiones ya realizadas de la persona solicitante, finalizadas o no, mediante el Número de Referencia del suministro.
- Tabla con las *Gestiones Pendientes* de la persona interesada. Este tipo de gestiones son las que se han comenzado pero ha habido algún motivo, como la falta de documentación, que ha provocado que dicha gestión tenga que ser aplazada.
- Tabla con las *Gestiones Realizadas* de la persona interesada.

Al hacer click en el icono de la *Lupa* de cada gestión, se mostrará una nueva vista, mostrando los aspectos más relevantes de dicha gestión:

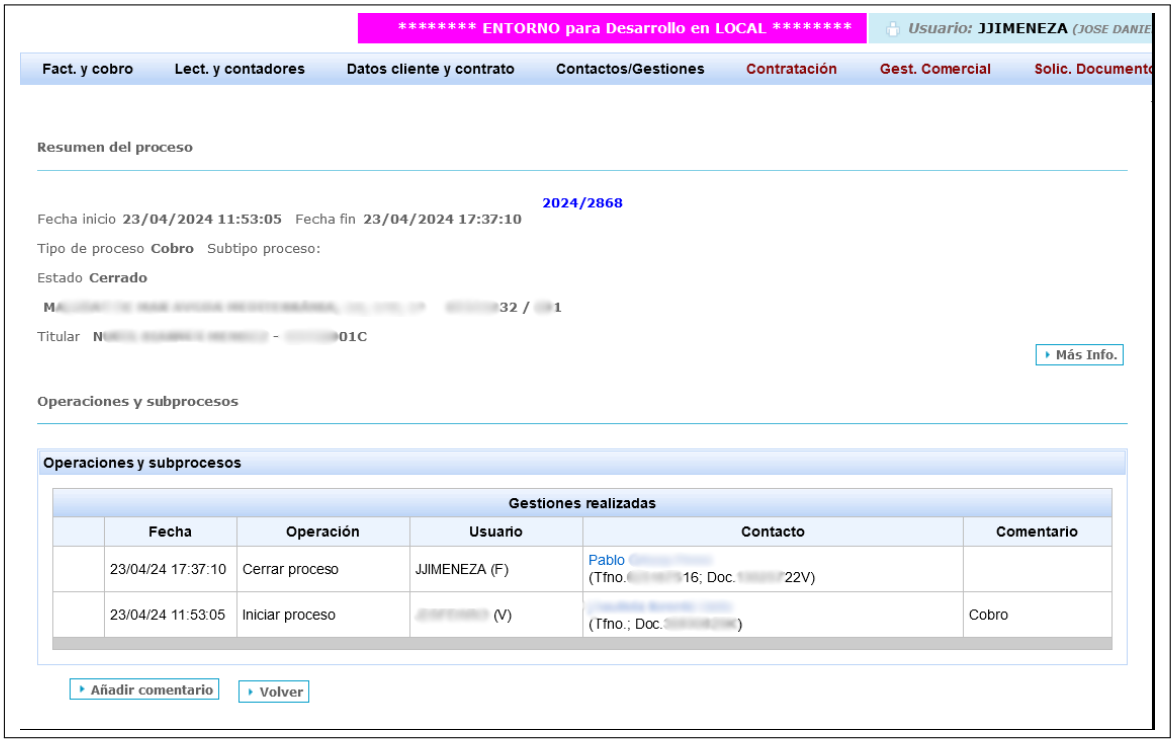


Figura 3.7: Vista del resumen de una gestión

Como se puede ver la captura de pantalla anterior, se observan los siguientes elementos:

- Resumen del proceso: Sección que contiene datos generales de la gestión realizada. Estos datos son:
  - *Código de Gestión*: Código que identifica la gestión (En este caso, *2024/2868*).

[2] El canal se representa mediante la letra *V* para el canal *Ventanilla* y con la *F* para el *Telefónico*.

- *Fecha de inicio* de la gestión, incluyendo la hora (En este caso, *23/04/2024 11:53:05*).
- *Fecha de fin* de la gestión, incluyendo la hora (En este caso, *23/04/2024 17:37:10*).
- *Tipo de proceso*: Nombre de la gestión realizada (En este caso, *Cobro*).
- *Subtipo de proceso*: Nombre de la subgestión si el tipo de proceso forma parte de una funcionalidad mas amplia.
- *Estado* de la gestión: Indica si dicha gestión está Abierta o Cerrada (En este caso, *Cerrada*).
- *Suministro* sobre el que se ha realizado la gestión (En este caso, *M...*).
- *Número de Referencia* del suministro: (En este caso, *...32 / ..1*).
- *Titular* del suministro (En este caso, *N...*).
- *DNI* del titular del suministro (En este caso, *...1C*).

- Botón *Más Info*, que abre una nueva vista que contiene datos más avanzados relativos al suministro.

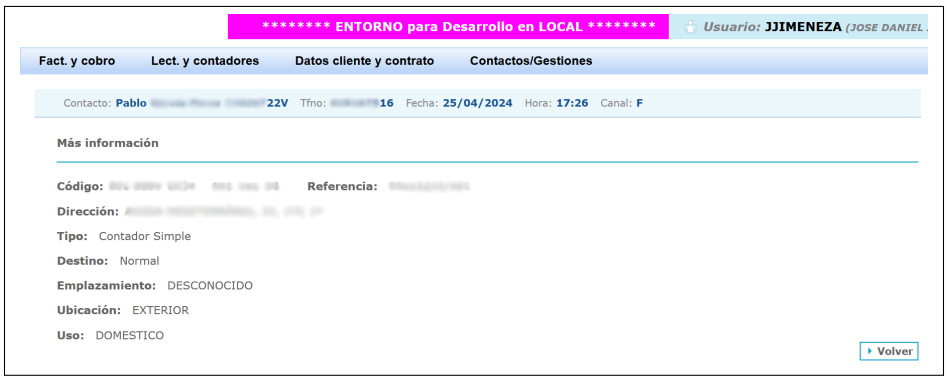


Figura 3.8: Vista del botón *Más Info.*

- *Operaciones y subprocesos*: tabla que contiene todas las operaciones que se han realizado durante las gestión (En este caso, solo se han realizado las operaciones de inicio y cierre de la gestión).

  Cada operación de la tabla es representada mediante los siguientes atributos:

  - *Fecha* y hora de inicio de la operación.
  - *Operación*: Nombre de la operación realizada.
  - *Usuario* que ha realizado la operación. Además, se incluye el *Canal*1 de contacto por el que se ha realizado el contacto.
  - *Contacto*: Nombre del cliente del contacto. También se incluyen su Teléfono y si Numero de DNI.
  - *Comentario* opcional sobre la operación.

  En cada operación, si tiene vinculado un contacto, se puede hacer click sobre el nombre del cliente para visualizar los **procesos que se han realizado durante todo el contacto**:

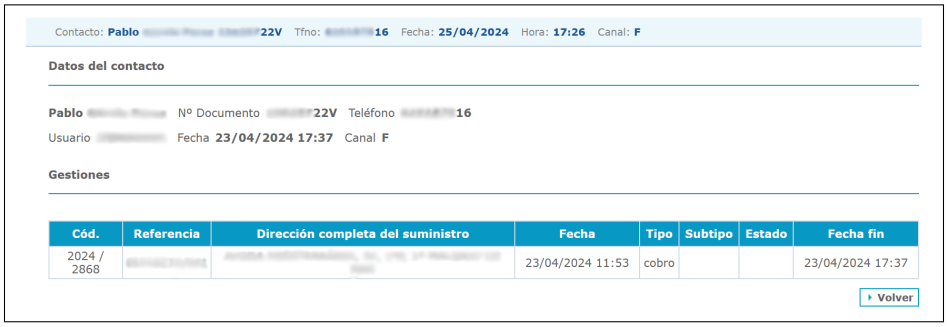


Figura 3.9: Vista de los procesos durante el contacto

- Botón *Añadir comentario*, que sirve para añadir a la tabla de Operaciones y Subprocesos un nuevo registro de nombre de Operación *Operación información*, con el Comentario que determinemos
- Botón *Volver* para regresar a la Vista principal del contacto

3.2.3. Canal Back Office

Mediante este canal, los usuarios del SAC son capaces de **visualizar los diferentes procesos que se han iniciado en el sistema**, pudiendo aplicar por **filtros** ya predefinidos o mediante una **búsqueda más avanzada**:

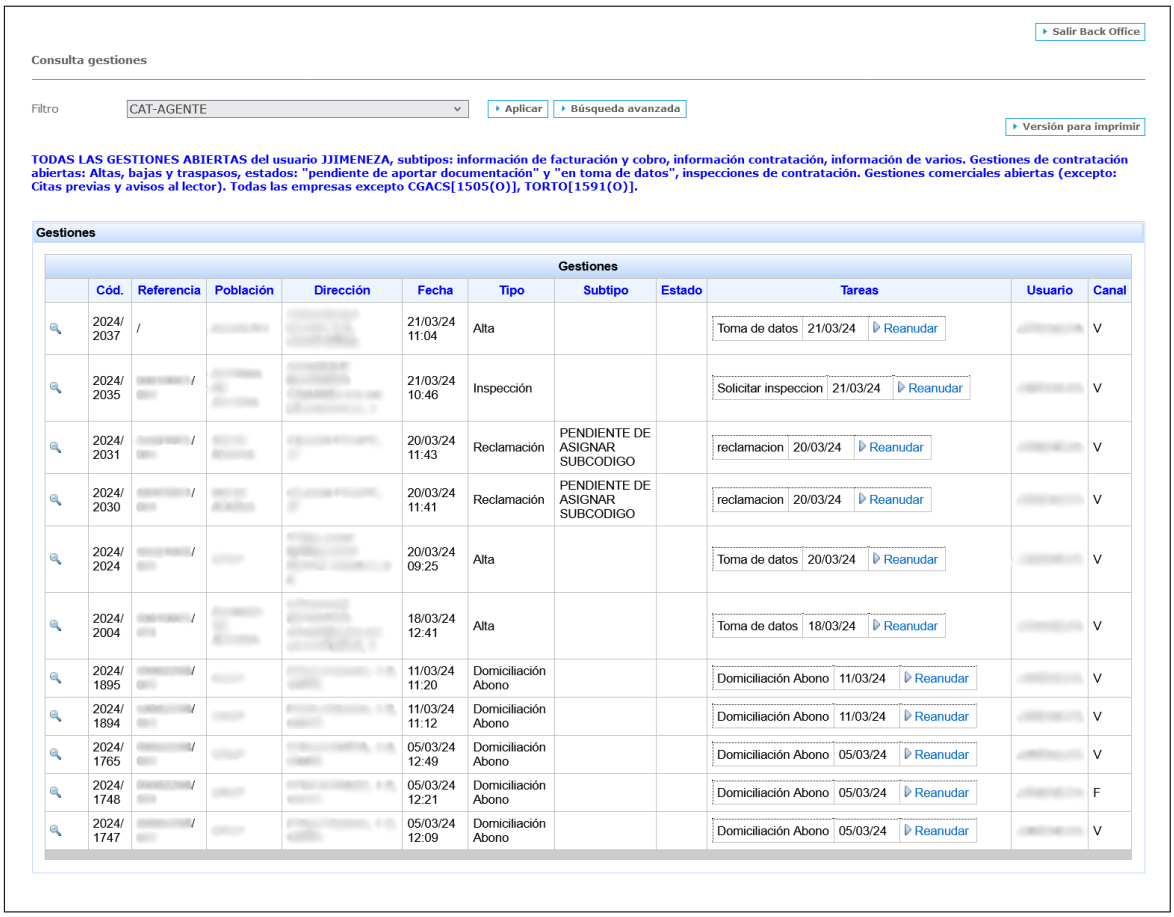


Figura 3.10: Vista del canal Back Office con el filtro CAT-AGENTE

En la figura anterior, se muestra un ejemplo de la Vista del canal Back Office, aplicando el filtro CAT-AGENTE, el cual muestra en la tabla las Gestiones Abiertas del usuario actual. Este tipo de filtros predefinidos son de gran utilidad para los usuarios, ya que engloban formas frecuentes de filtrar las gestiones.

En este canal, podemos distinguir los siguientes elementos:

- Botón *Salir Back Office* para volver a la Vista principal del SAC.
- Desplegable de selección con los filtros predefinidos.
- Botón *Aplicar* para actualizar la Vista con el filtro seleccionado.
- Botón *Búsqueda avanzada* para filtrar los procesos mediante valores determinados.

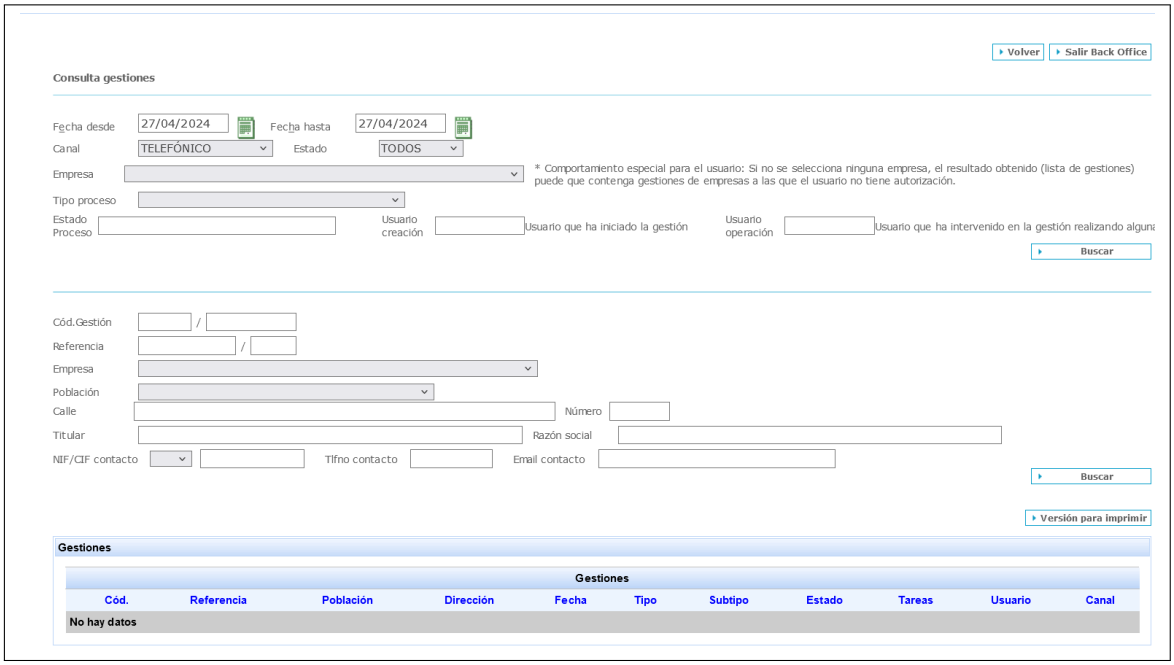


Figura 3.11: Vista de Búsqueda Avanzada del canal Back Office

En la imagen anterior se muestran multitud de valores que podemos establecer para filtrar las gestiones, relativas tanto a la Gestión como al punto de Suministro.

- Botón *Versión para imprimir* convierte toda la información que contiene esta Vista, en un único documento en formato PDF. Si la tabla tiene varias páginas (tiene tantos registros que no caben en la tabla, porque tienen un tamaño máximo), igualmente se muestran todos los registros en el documento resultante.
- Párrafo escrito en Azul que describe el filtro predefinido aplicado.
- Tabla resultante al aplicar un filtro. Cada gestión que esté abierta, será posible reanudarla *clickando* en el botón *Reanudar* de la columna *Tareas*.

3.2.4. Canal Correspondencia

En este canal, se realizan todas la acciones relacionadas con el envío de objetos de correspondencia. En esta empresa, los diferentes tipos de objetos de correspondencia son: correo electrónico, fax, burofax, correo ordinario, correo certificado, en mano o chat a través de un *bot* de mensajería.

3.2.5. Canal Fraudes

Este canal es específico del ámbito de la empresa. Su función es la de registrar y gestionar los fraudes que se comenten, que son modificaciones ilegales sobre las infraestructuras de los suministros, para redirigir de forma ilegal la ruta del agua evitando así que se registre correctamente la cantidad de agua consumida.

CAPÍTULO 4
Metodología de trabajo

En el departamento de software de Global Omnium se tiene establecida una metodología de trabajo muy estricta que se debe cumplir cada vez que se modifica el proyecto. En todo gran proyecto dentro de una empresa, es de vital importancia dejar constancia de todos los cambios que se han aplicado y de todos los datos relativos al por qué se ha llevado a cabo, como quién ha solicitado la modificación, qué desarrollador la ha implementado, qué modificaciones ha realizado, etc. Además, es importante seguir una metodología que, de la forma más eficiente posible, al final del proceso garantice que la modificación realizada no aporte ni errores ni código de mala calidad.

Con estos fines, en Global Omnium hace uso de una metodología de desarrollo ágil basada en la arquitectura **DevSecOps**, en las que se desean alcanzar los siguientes objetivos:

- Automatizar las tareas relacionadas con el desarrollo de software, como la compilación y el despliegue.
- Diseñar y calcular métricas que puedan evaluar el desempeño y la calidad del código.
- Disponer de trazabilidad en las acciones relacionadas con el ciclo de vida del software, y evidencias de cada acción clave.
- Ayudar a implantar el uso de Buenas Prácticas en el desarrollo.
- Proveer del contexto necesario a los equipos de desarrollo para mejorar la sostenibilidad y seguridad del código de las aplicaciones.
- Promover la cultura de la calidad dando soporte a la implantación de pruebas automatizadas.

Para cumplir con estos objetivos, surge la necesidad de implantar un modelo de Integración Continua Y Despliegue Continuo (CI/CD) para todos los proyectos de software de la empresa como parte de su proceso de mejora continua en la creación del software.

4.1 Integración continua (CI) y Entrega Continua (CD)

Esta fase del ciclo de desarrollo DevSecOps es de vital importancia en un proyecto tan grande. Prácticamente a diario se realizan cambios en el repositorio Git y en el entorno de Desarrollo, por lo que surge la necesidad automatizar todos estos procesos de compilación y de entrega, aportando las siguientes ventajas:

- Detección temprana y mejorada de errores, y métricas que le permiten abordar los errores con brevedad.
- Progreso continuo y demostrado para mejorar el feedback.
- Mejor colaboración en equipo; todos los miembros del equipo pueden cambiar el código, integrar el sistema y determinar rápidamente los conflictos con otras partes del software.
- Menor número de errores durante las pruebas del sistema.
- Se reducen los errores humanos gracias a los procesos que estén automatizados.
- Sistemas actualizados constantemente en los que realizar las pruebas.

Al igual que en muchas empresas del mundo, la piedra angular para integrar un modelo de CI/CD es la herramienta **Jenkins**. En Global Omnium se ha adaptado el siguiente modelo de CI:

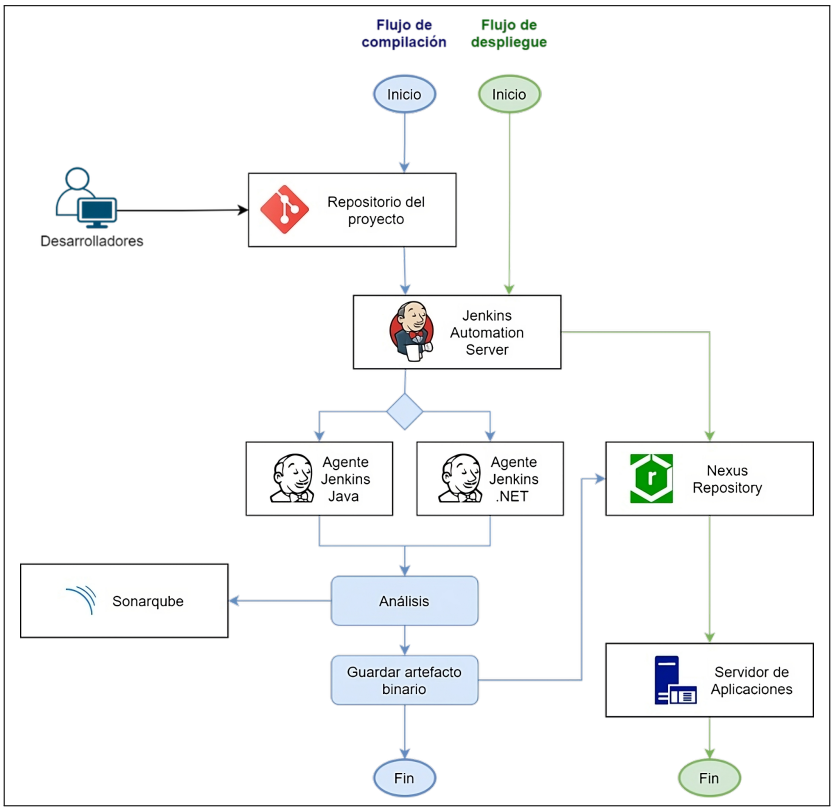


Figura 4.1: Modelo de Integración Continua en Global Omnium[8]

En este modelo, se distinguen dos tipos de flujo:

- Análisis: este proceso automático será la ejecución más habitual en el proceso de desarrollo. Como parte del procedimiento genera evidencias relacionadas con la calidad, pero no artefactos desplegables. Los *pipelines* de análisis se pueden ejecutar de forma automática con cada modificación realizada en el repositorio de código del proyecto.
- Liberación (Release): este *pipeline* no comprueba los requisitos de calidad del software, sino que genera binarios inmutables a partir de una etiqueta, que podrán ser desplegados en los diferentes entornos a posteriori mediante el flujo de despliegue.

Los artefactos que dan lugar a una nueva versión del sistema son almacenados de forma automática en el repositorio de artefactos **Nexus Repository**, el cual alberga copias de seguridad de todas las versiones pasadas de la aplicación.

4.2 Herramientas y tecnologías de desarrollo

Las siguientes herramientas son claves en el desarrollo DevSecOps de la empresa; sustituir alguna supondría un gran esfuerzo de migración.

4.2.1. Entornos de trabajo

En primer lugar, llamamos *entornos* a las diferentes copias que existen de la aplicación. El entorno principal es el que nos referimos como *entorno de Producción*, el cual es el que usan el personal del Servicio de Atención al Cliente para realizar los contactos reales con los clientes, quedando registrados en la base de datos de la aplicación.

Cabe destacar que los programadores no tenemos acceso al entorno de Producción, por diferentes motivos:

- Se puede registrar o eliminar información de forma no deseada.
- Al hacer una modificación, pueden haber errores o cambios que hagan que otras funcionalidades de la aplicación funcionen de forma incorrecta.
- Al *testear* las nuevas funcionalidades implementadas, las bases de datos se verían alteradas con nuevos registros, mezclando datos de producción (los datos reales) y de prueba.

Para evitar estos problemas, se utiliza una copia del entorno de Producción y de las bases de datos pero con los datos y referencias mezcladas, al que se le llamó *Entorno de Desarrollo*. Gracias a este nuevo entorno:

- Todas las modificaciones finales que se quieran implementar en el SAC, se aplicarán primero en este entorno, por lo que si existen errores, el entorno de producción no se vería afectado.
- Los usuarios del entorno de producción también tienen acceso al de desarrollo, por lo que estos podrán aprobar que las modificaciones se hayan implementado correctamente.
- Los programadores tienen acceso a las bases de datos de desarrollo, por lo que ahora se nos habilita la posibilidad de manipular los datos que se necesiten con el fin de testear las funcionalidades, sin que los datos de producción se vean afectados.

Además de este entorno, el equipo de desarrollo dispone de una copia de la aplicación, vinculada a las bases de datos del entorno de desarrollo, compilada para que pueda ser desplegada en el equipo local. En este *entorno local* es en el que los desarrolladores modifican y *testean* el código fuente de la aplicación, para posteriormente aplicar las modificaciones en el entorno de desarrollo.

El despliegue de este SAC Local se realiza mediante la herramienta JBoss, con la que una vez configurados los parámetros de despliegue en el IDE (en este caso Eclipse), la aplicación se inicia de forma sencilla y rápida.

Finalmente, a modo de emular al entorno de producción con el fin de testear las actualizaciones y asegurar que las mismas no corromperá la aplicación en los servidores en producción cuando estas sean desplegadas, se dispone de un *entorno de preproducción*, también conocido como de integración o *stagin*.

A modo de resumen, el siguiente esquema muestra la **relación entre los entornos y los desarrolladores**, mostrando los diferentes actores y bases de datos a los que están vinculados cada uno. Además, se representan el flujo de las modificaciones desde el entorno de desarrollo local hasta el de producción, quedando siempre actualizando el de desarrollo.

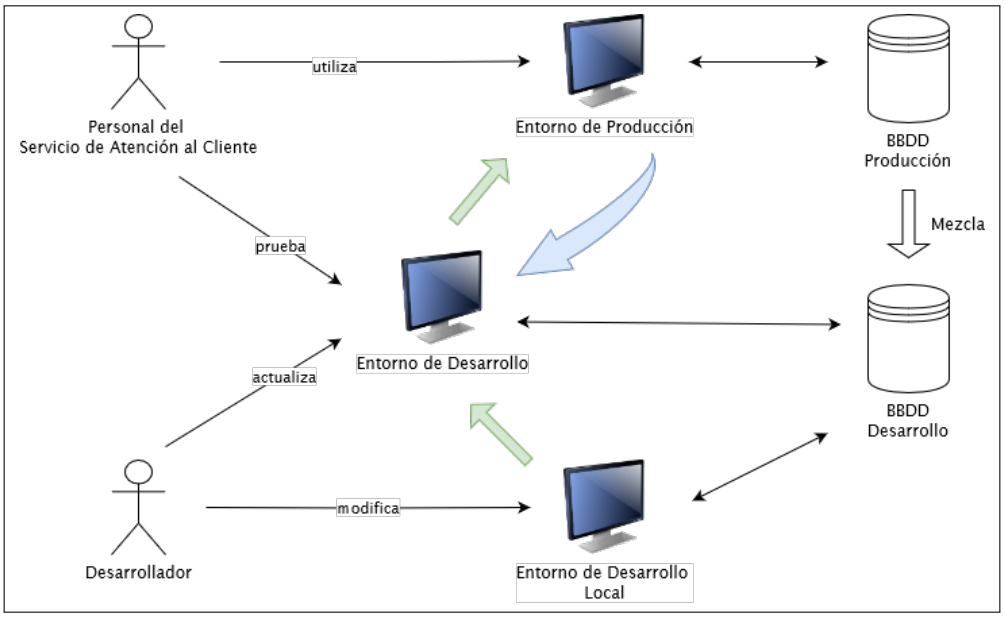


Figura 4.2: Esquema de relación de los entornos del SAC

En dicho esquema se observan:

- Los 3 entornos: *Producción*, *Desarrollo* y *Desarrollo Local*.
- Los 2 tipos de actores del SAC: *El personal del Servicio de Atención al Cliente* y *los Desarrolladores*.
- Las 2 bases de datos implementadas: Las relacionadas con el *entorno de producción* y las utilizas para *los dos entornos de desarrollo*.

En cuanto al entorno de Producción, se observa que está vinculado a la base de datos de Producción, que solo son accedidas por el Personal del Servicio de Atención al Cliente. Para el entorno de Desarrollo, se ha hecho una copia de las bases de datos de Producción para ser usado en el *testing* de las modificaciones. En este entrono, los Desarrolladores actualizan la versión del SAC con las nuevas modificaciones, para que los usuarios del entorno de Producción comprueben que funcionan correctamente. Las modificaciones son realizas en primera instancia en el entorno de desarrollo local de cada desarrollador. Cuando las modificaciones son aprobadas por los usuarios del entorno de Producción, los administradores actualizan la versión del entorno de producción con la de desarrollo.

Dicho esto, se deduce que:

- La versión más reciente del sistema siempre será la del Entorno de Desarrollo

- Cuando los usuarios de Producción aprueban las modificaciones del entorno de Desarrollo, estos cambios no se aplican automáticamente en el entorno de Producción, por lo que el tiempo que pueda transcurrir es indeterminado.

4.2.2. Repositorio Git: GitLab

Todo el proyecto *gav* se almacena en un repositorio Git hospedado en la plataforma DevSecOps GitLab.

Todos los proyectos de la empresa albergados en GitLab son accesibles desde el portal GoGitLab, al cual solo se puede acceder mediante un dispositivo que tenga instalado los Certificados de la Entidad de Global Omnium.

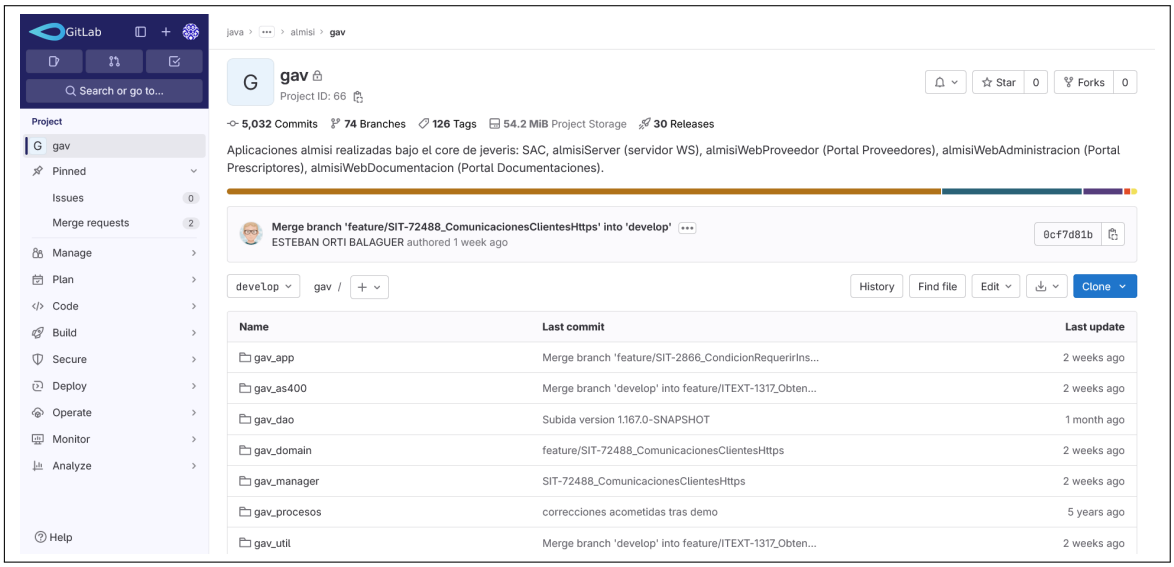


Figura 4.3: Vista del proyecto *gav* en *GoGitLab*

Mediante esta plataforma, los desarrolladores somos capaces de realizar todas las acciones de monitorización y modificación del proyecto:

- Obtener el enlace SSH y HTTPS para clonar en proyecto en nuestro IDE de trabajo.
- Posibilidad de visualizar todos los Commits que se han realizado y todos los datos relativos, como quien lo ha realizado, que cambios se han realizado, etc.

GitFlow

Gracias al uso de las ramas de Git, es posible tener implementados los diferentes Entornos de uso del SAC. Para tener independizados estos entornos en todo momento, se tienen establecidas 5 tipos de ramas[1]:

1. Rama `master`: rama que es la utilizada en todo momento en el entorno de **producción**. Teóricamente, como todas las modificaciones han sido probadas antes de actualizar esta rama, todo el código nuevo no debe contener errores.

   Cabe destacar que la nomenclatura concreta de la rama maestra puede ser variable. Históricamente se ha utilizado el término `master` para referirse a ella, pero las opciones `main`, `trunk` o `stable` son también aceptadas.

[1] Estas ramas son las ya diseñadas en el estándar del flujo de trabajo GitFlow

2. Rama `develop`: rama donde se han aplicado todas las modificaciones que ya se han probado que funcionan correctamente, considerando que la rama se encuentra en un estado estable y completo. Esta es la rama que es usada para crear la respectiva rama de versión `release/`.

3. Ramas de **versión**: Son el tipo de rama que es utilizada para liberar una nueva versión del proyecto. Cada vez que se decide actualizar el sistema con una nueva versión, se crea una copia de la rama `develop`, y se le asigna un nombre acorde a la siguiente nomenclatura:

   ```
   release/[NºVersión]
   ```

   A dicha rama, se le asignará un *tag* en el repositorio Git que coincidirá con el número de la nueva versión[2] del sistema que ya contiene su nombre.

   Gracias a estas ramas, se consigue tener un registro sencillo de todas las versiones de la aplicación, por lo que para actualizar la rama `master` solamente es necesario *mergear* la versión deseada.

4. Rama de **testing**: rama donde se aplican todas las modificaciones que van a ser probadas en el entorno de **desarrollo** por un usuario cualquiera del entorno de producción. El nombre de este tipo de ramas siguen la siguiente estructura:

   ```
   test/[NºVersión]
   ```

5. Ramas **de desarrollo**: son todas aquellas ramas que los desarrolladores crean a partir de la rama `develop` para poder realizar las modificaciones en el entorno de **desarrollo local**. El nombre que el desarrollador asigna a cada una de estas ramas de desarrollo sigue la siguiente nomenclatura:

   ```
   feature/[NºTicketJira]_[NombreDescriptivo]
   ```

6. Rama de **corrección**: Cuando se detecta que en una nueva versión del entorno de producción (rama `master`) se producen comportamientos o errores que no se han detectado en las fases de testing, es necesario crear una rama de corrección `hotfix/` a partir de la actual rama maestra para solucionar dicho comportamiento o error con la mayor brevedad posible. El nombre que el desarrollador asigna a cada una de estas ramas de correción sigue la siguiente nomenclatura:

   ```
   hotfix/[NºTicketJira]_[NombreDescriptivoDeCorreción]
   ```

A modo de resumen, omitiendo detalles de compilación y despliegue, si representamos las relaciones entre las ramas y los entornos durante el proceso de desarrollo, resultaría un esquema como el siguiente:

[2] Para establecer el número de cada versión, se utiliza el Versionado Semántico de Tom Preston-Werner[8]

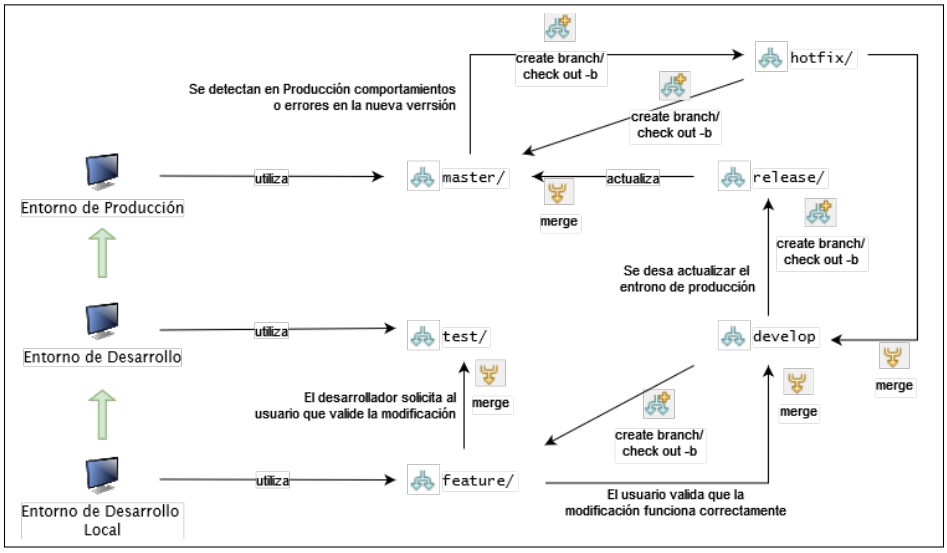


Figura 4.4: Esquema simplificado de relación entre Ramas y Entornos

NOTA: Antes de realizar un *merge*, es necesario realizar antes un *pull*. Es posible que durante el desarrollo, se hayan actualizado las ramas de `test/` o `develop`.

Para que los desarrolladores puedan realizar estos cambios en el repositorio, se hace uso de alguna herramienta que permita ejecutar todas estas acciones. Todas estas herramientas se pueden clasificar en 3 tipos:

- Extensión de Git para el IDE (en Eclipse, *EGit*)
- Aplicación de consola (por ejemplo, *Git Bash*)
- Aplicación con interfaz gráfica (por ejemplo, *GitHub Desktop*)

Con cualquiera de estas opciones, es posible trabajar sobre el repositorio Git, pero cada una está enfocada en tipos de desarrolladores diferente.

Por lo general, la opción más llamativa es usar la correspondiente extensión para el IDE, puesto que integra todas las funciones de Git en el proyecto de forma visual. Por ejemplo, es posible ver las ramas que están clonadas en el equipo de trabajo, realizar operaciones sobre ellas de forma visual, consultar los *commits* que se han realizado, comparar ramas, entre otras ventajas. A parte de que su uso es muy intuitivo, no es necesario realizar operaciones por consola, puesto que todas las operaciones se pueden realizar desde el IDE.

Por otra parte, trabajar con una aplicación con interfaz gráfica suele ser muy similar a hacerlo directamente desde el IDE, pero con estas aplicaciones separadas del entorno de trabajo se consigue ampliar la gama de herramientas gráficas, pudiendo escoger la que se considere más cómoda. Además, estar acostumbrado a una aplicación externa, sea gráfica o de consola, resulta muy conveniente para aquellas personas que trabajan con más de un entorno de programación.

Sin embargo, otros desarrolladores prefieren realizar todas las acciones por consola, ya que las personas más experimentadas prefieren ser totalmente conscientes de las accione que se realizan sobre el repositorio. Trabajar de forma prolongada con comandos resulta generalmente más eficiente que usando una interfaz gráfica.

4.2.3. Proyecto Maven

Apache Maven es una herramienta de código abierto de gestión de proyectos que se utiliza para la gestión de dependencias, como herramienta de compilación e incluso como herramienta de documentación.

Maven ofrece distintas funcionalidades, pero la más común es hacer uso de sus archivos **POM**.xml (Project Object Model) que se sitúan en la raíz de cada paquete del proyecto. En él, se definen:

- Dependencias de módulos.
- Dependencias con compones externos, como Spring Boot o JUnit.
- Procesos y orden de compilación para generar la *build*.
- Plugins del propio Maven.

En proyectos grandes, los POM pueden heredar la configuración de un POM padre, lo que simplifica la definición de módulos de cada paquete.

Gracias a estos POM se consigue tener toda la configuración del proyecto centralizada en un solo fichero XML, permitiendo así simplificar el desarrollo y los procesos de compilación y despliegue.

4.2.4. IDE Java: Eclipse

Como se ha avanzado anteriormente en este trabajo, en esta empresa se hace uso de Eclipse, uno de los Entornos de Desarrollo Integrado (IDE) para la programación en Java más usados en todo el mundo.

Eclipse, al ser de código abierto, es altamente personalizable, permitiendo configurar esta plataforma a medida para el correcto funcionamiento del proyecto.

4.2.5. Flujos de navegación con JEveris

En un proyecto donde la aplicación consta de tantas funcionalidades diferentes, es vital recopilar todos los posibles flujos de navegación de cada vista del sistema. En este proyecto, se hace uso de la libreria JEveris, el cual permite crear diagramas de flujo en formato XML y ser visualizado gráficamente mediante el visor predeterminado de flujos de Eclipse.

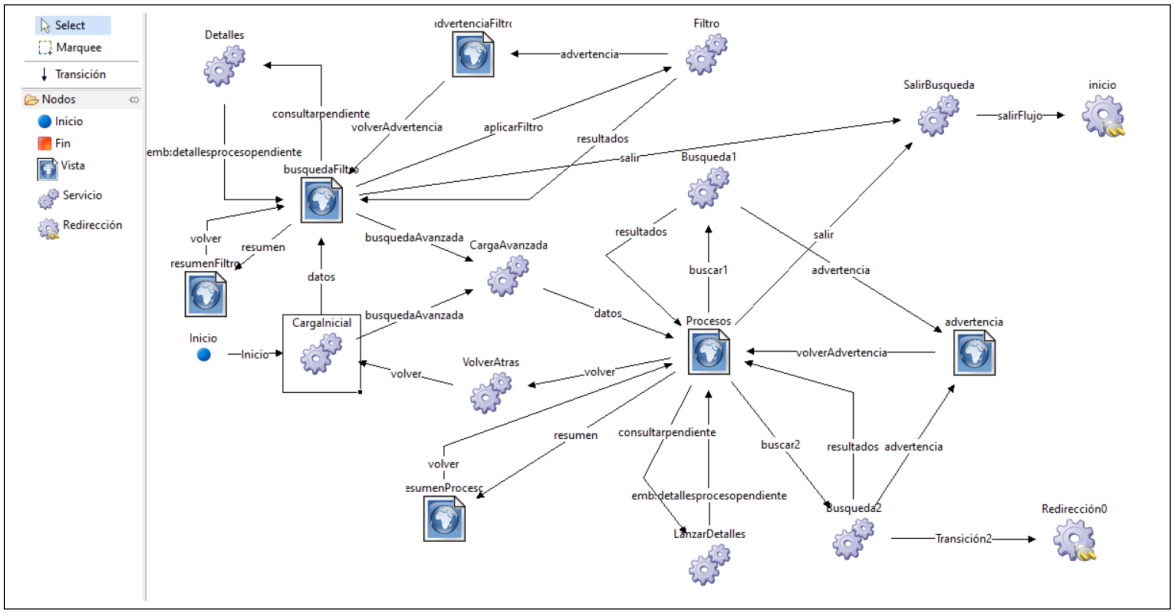


Figura 4.5: Flujo del canal Back Office

A modo de ejemplo, el grafo anterior representa el flujo de navegación al entrar en el canal Back Office.

Estos tipo de diagramas suelen estar formados por los siguientes elementos:

- **Punto de entrada al grafo**: Representa cuál es el punto de entrada al grafo. Este elemento es representado mediante el icono .
- **Punto de finalización del grafo**: Representan los puntos en los que se finaliza el grafo. Este elemento es representado mediante el icono .
- **Punto de redirección**: Representan los puntos en los que se redirecciona a otro elemento del grafo, o a un flujo diferente. Este elemento es representado mediante el icono .
- **Vista**: Representan las pantallas o páginas individuales que los usuarios ven y con las que interactúan dentro de la aplicación. Cada vista representa una interfaz de usuario única, vinculada a un fichero .jsp, mostrando información previamente calculada en los Servicios. Este elemento es representado mediante el icono .
- **Servicio**: Representa una clase .java que es utilizada para calcular todos los datos necesarios que serán mostrados en las Vistas. Este elemento es representado mediante el icono .

  Para utilizar una clase .java como un Servicio, se han de realizar las siguientes modificaciones:

  1. Cada clase ha de extender de la clase **ActionService**. Para ello se ha de importar el paquete `com.jeveris.pl.navigation.model.ActionService`.
  2. Cada clase ha de implementar el método `public String execute()` el cual contendrá el código a ejecutar cuando el flujo de navegación transicione a esta clase.

  Un ejemplo de un Servicio en JEveris podría ser el siguiente:

```java
import com.jeveris.pl.navigation.model.ActionService;

public class MiClase extends ActionService {

    @Override
    public String execute() {

        /**
        CODIGO A EJECUTAR
        */

        return "jeveris:siguienteRecursoJEveris";
    }

}

```

Listing 4.1: Ejemplo de Servicio JEveris

- **Transición**: Hacen referencia a cómo los usuarios se mueven de una Vista a otra dentro del sistema. Este elemento es representado mediante flechas unidireccionales.

Estos grafos sirven, a parte de visualizar el flujo de navegación de forma visual, para verificar que las transiciones que se escriben en cada parte del código sean idénticas a las que representa el grafo.

Podemos vincular los elementos del grafo con las clases del proyecto de forma sencilla desde el panel de propiedades de cada elementos:

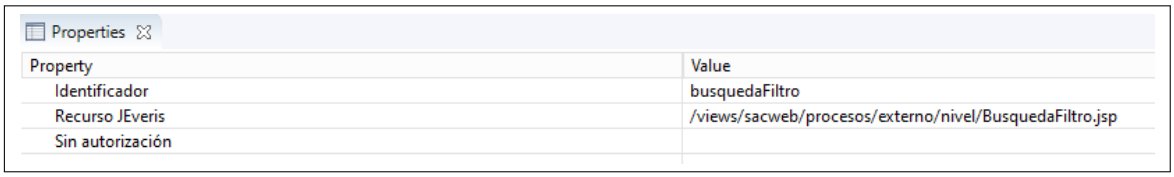


Figura 4.6: Propiedades de la Vista *BusquedaFiltro*

En cuanto al código, para representar una transición entre elementos simplemente se realiza mediante la estructura *return*, escribiendo como valor a devolver la cadena de texto *jeveris:* seguido del nombre de la transición a tomar.

```java
return "jeveris:resultados";
```

Figura 4.7: Ejemplo de una transición a un elemento del flujo

Si en lugar de transicionar a un elemento del flujo, se desea realizar una transición a otro flujo de navegación diferente, se ha de utilizar la cadena de texto *jeveris:emb:* seguido del nombre de flujo de navegación correspondiente:

```java
return "jeveris:emb:detallesprocesopendiente";
```

Figura 4.8: Ejemplo de una transición a otro flujo

En cada servicio claramente podemos definir variables y objetos, pero para poder ser usados en todos los servicios y vistas a las que se transicionen, se ha de utilizar en cada servicio, el paquete `com.jeveris.pl.context.PLContext`, el cual permite almacenar

cada variable dentro del flujo de navegación. Los métodos del paquete **PLContext** que tienen este uso son:

- `PLContext.putInFlowScope("nombreVariable", valorVariable)`: Define o actualiza la variable *nombreVariable* con el valor *valorVariable*.

- `(TipoVariable) PLContext.getFromFlowScope("nombreVariable")`: Obtiene del flujo el valor de la variable *nombreVariable*. Como al definir la variable no se define el tipo, al obtener el valor se ha de realizar una coerción explicita.

NOTA: En las vistas en formato JSP no es posible utilizar métodos consultores para acceder a los atributos de un objeto. Si se requiere hacer uso del resultado de un método consultor de un objeto, se deberá utilizar una variable en el flujo, con el resultado de dicho método.

4.2.6. Jira Service Management

En toda gran empresa surge la necesidad de implementar una herramienta integral capaz de gestionar los servicios de TI y otros servicios empresariales, como la Atención al Cliente, automatizando los máximos procesos posibles.

En Global Omnium se decidió adoptar la plataforma web Jira Service Management, la cual gestiona servicios ITSM mediante un **sistema de *tickets***. Principalmente está diseñada para ayudar a las empresas a gestionar y mejorar la entrega de servicios a sus clientes y usuarios internos, aportando multitud de enfoques diferentes en la forma de implementar la plataforma. El enfoque más usual en la empresa en general es la gestión de servicios ITSM, que incluyen:

- **Gestión de incidencias y problemas**: Permite a los distintos departamentos registrar, categorizar y resolver incidencias y problemas de manera eficiente. Proporciona flujos de trabajo personalizables para la gestión de incidencias, lo que facilita la identificación de problemas y la toma de medidas correctivas.

- **Gestión de cambios**: Facilita la planificación, evaluación y ejecución de cambios en los servicios de TI y otros sistemas empresariales. Ofrece una forma estructurada de solicitar, aprobar e implementar cambios, minimizando así el riesgo de interrupciones en los servicios.

Cualquier empleado de la empresa puede inicializar una de estas gestiones desde el portal **Jira Help Desk** de Global Omnium.

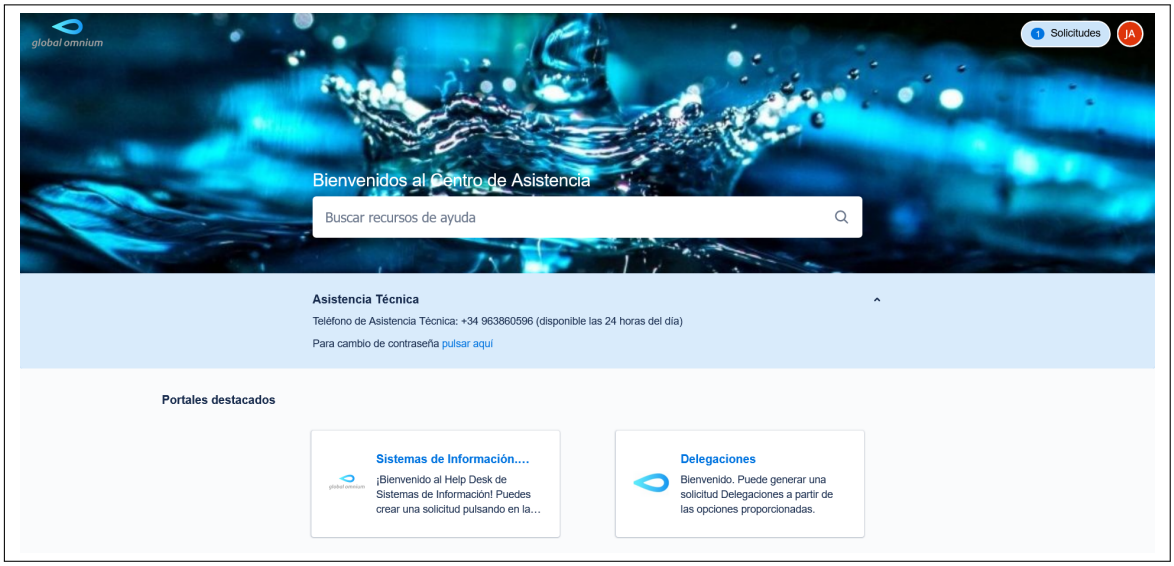


Figura 4.9: Vista inicial del Jira Help Desk de Global Omnium

En la Vista anterior, se muestran las dos áreas diferentes en las solicitudes pueden ser dirigidas: el área de **Sistemas de Información** (Soporte IT) y Delegaciones. Dependiendo del área, el sistema solicitará unos datos determinados para realizar la solicitud.

Para el área de Sistemas de Información, es necesario cumplimentar el siguiente formulario:

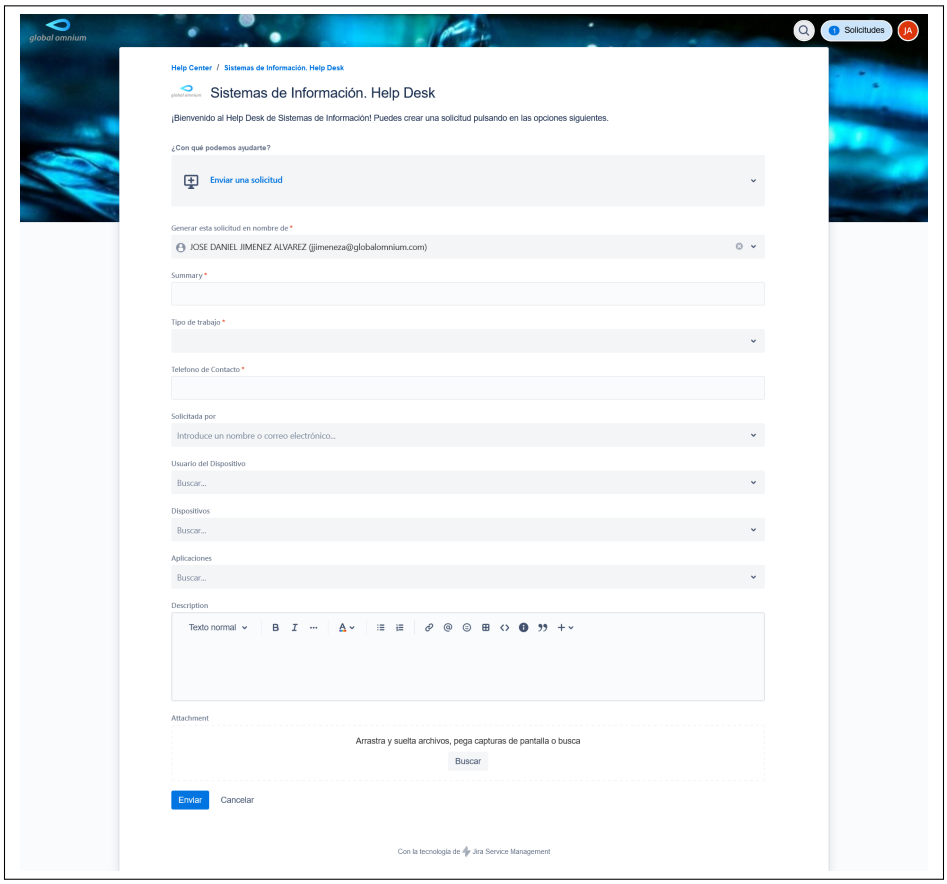


Figura 4.10: Formulario de solicitud al Soporte IT

Los campos de este formulario son los siguientes:

- *Generar esta solicitud en nombre de*: Persona que desea comunicar la incidencia, que puede ser distinta de la persona que la realiza .

- *Summary*: Título de la solicitud.

- *Tipo de trabajo*: *Presencial* o *Teletrabajo*.

- *Teléfono de Contacto*.

- *Solicitada por*: Persona que está cumplimentando el formulario, en el caso de que lo hago en nombre de otra persona.

- *Usuario del Dispositivo*: En el caso que la incidencia esté relacionada con algún dispositivo, indica el usuario de éste.

- *Dispositivos*: En el caso que la incidencia esté relacionada con un algún dispositivo, indica los dispositivos afectados.

- *Aplicaciones*: En el caso que la incidencia esté relacionada con un alguna aplicación, indica el nombre de las aplicaciones afectadas.

- *Descripción*: Sección donde se detalla en qué consiste la solicitud.

- *Attachment*: El solicitante podrá adjuntar documentos en esta sección si le resulta oportuno.

Una vez se rellene y se envíe le formulario, el sistema genera un identificador para la incidencia llamado **ticket** y se traslada la incidencia a la plataforma de **Jira Service Management** a la que solo tienen acceso los empleados encargados de responder a las solicitudes.

Dependiendo de los valores de los campos: *Tipo de Trabajo*, *Dispositivos* y *Aplicaciones*, el sistema filtrará la solicitud y la redirigirá a la **cola de solicitudes** correspondiente, previamente configuradas por la empresa. Además, desde Jira Service Management se podrá concretar el tipo de incidencia gracias a los atributos de *Área*, *Categoría* y *Servicios*.

En cuanto a las solicitudes relacionadas con la aplicación SAC, las solicitudes que en el campo *Aplicaciones* contenga la aplicación *SAC (att. cliente)* se redirigen a una cola llamada *CRM*, en las que se asignan los siguientes campos:

- *Área*: CRM

- *Categoría*: Mantenimento Sistema de Información

- *Servicios*: Mant. del SAC

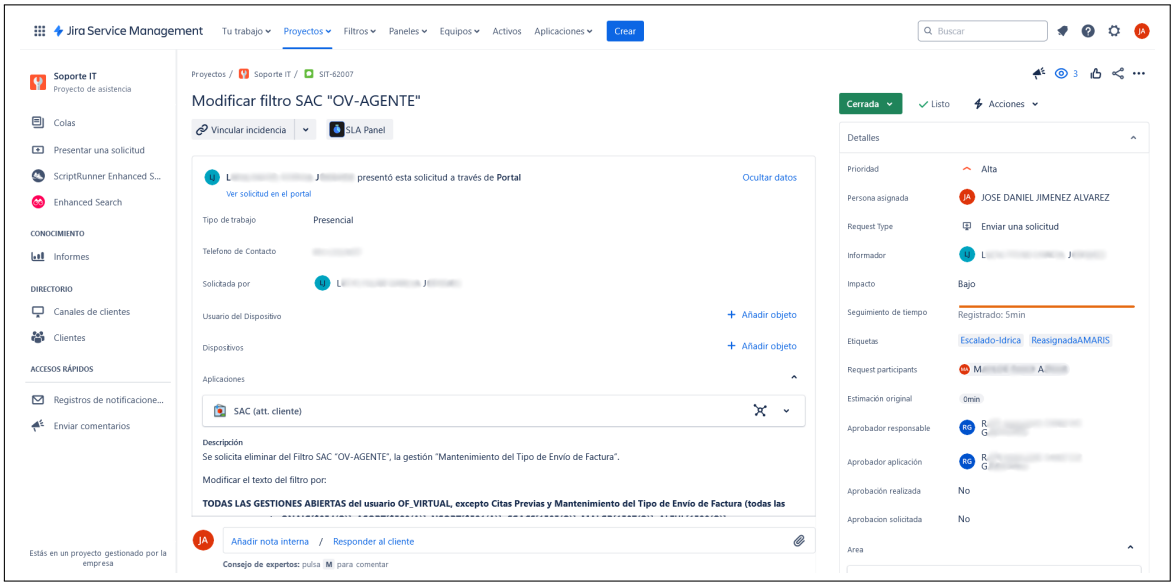


Figura 4.11: Ejemplo de una incidencia en Jira Service Management

En cuanto a la resolución de las solicitudes, Jira Service Management es muy fácil y intuitiva de usar. En cada solicitud, es posible:

- Modificar los campos de la solicitud original.
- Asignar prioridad, tiempo invertido en la resolución, participantes adicionales, aprobador responsable, etc.
- Vincular otras incidencias.
- Comunicarse con la persona solicitante o con un participante adicional mediante notas internas.

4.2.7. SonarQube

En cualquier proceso de desarrollo de software, es vital cuidar en cada modificación la **calidad del código**. En este ámbito, una herramienta como SonarQube es imprescindible en cualquier proyecto para garantizar un desarrollo de código de calidad y aplicando buenas prácticas.

SonarQube es una plataforma de código abierto para la inspección continua de la calidad del código a través de diferentes herramientas de **análisis estático de código fuente**. Proporciona **métricas** que permiten a los equipos de desarrollo mejorar el código de su programa, realizando seguimientos y detectando errores y vulnerabilidades de seguridad para mantener el código libre de errores.

Como es una herramienta diseñada para lograr la integración e implementación continuas del código, se basa en **analizar el código nuevo** que se ha introducido o modificado, basándose en la **métricas que se han configurado para el proyecto**. Las métricas engloban multitud de reglas que han de cumplir todo el código del proyecto, diferenciando el lenguaje de cada fichero, y su relevancia en la seguridad del proyecto

Algunas de las métricas ya pre-establecidas en SonarQube que determinan si el código nuevo es de calidad, pueden ser los siguientes:

- Los métodos "toString()"no deberían devolver null.

- Un método no puede solicitar más de 7 parámetros.
- No puede haber en una clase una secuencia de instrucciones repetida, puesto que se considera código repetido.
- El código comentado debe ser eliminado o descomentado.

Sin embargo, es posible que en ciertos puntos del código no se puedan respetar todas las normas definidas, por lo que SonarQube también permite la flexibilidad de ignorar estas normas si consideramos que no somos capaces de cumplirlas.

En Global Omnium, se tiene establecido un **umbral de calidad** especifico para todos sus proyectos[3], que consiste en cumplir los siguientes criterios de SonarQube:

- No más de un 3 % de duplicación de código.
- Calificación de Sostenibilidad "A".
- Calificación de Fiabilidad "A".
- Ningún riesgo de seguridad (Security Hotspots).
- Calificación de Seguridad "A".

Como esta herramienta está integrada en Jenkins Automation Server, los desarrolladores pueden iniciar el análisis de cualquier rama del repositorio de forma rápida y automática, visualizando los resultados desde esa misma plataforma.

4.2.8. Jenkins Automation Server

En un mundo empresarial cada vez más competitivo y dinámico, la eficiencia es clave. En este contexto, la **automatización de procesos** emerge como una herramienta esencial para optimizar el flujo de trabajo y maximizar la productividad. En este sentido, **Jenkins** es considerada la reina de las herramientas de automatización de procesos, ofreciendo una plataforma robusta y flexible para la implementación continua, la integración y la entrega de software.

Esta herramienta está escrita en Java, es multiplataforma, de código abierto, y es accesible mediante interfaz web. Además, cuenta con multitud de plugins para implementar procesos de otras herramientas, como el ánalisis de código con SonarQube

Jenkins es la plataforma más usada en el ámbito de **integración continua (CI) y entrega continua (CD)** de software, que son prácticas habituales en el desarrollo de software. Consiste en integrar frecuentemente mejoras en el código de un proyecto una vez han sido validadas con el objetivo de detectar errores lo antes posible. Esta automatización facilita a los desarrolladores en integrar cambios en un proyecto y entregar nuevas versiones a los usuarios de forma automática y sin errores humanos, siempre que Jenkins haya sido bien configurado.

Algunas de las funciones más importantes que ofrece Jenkins son:

- Automatizar la compilación, generación del ejecutable (*build*), el testeo y el despliegue de software.
- Notificar a los equipos correspondientes la detección de errores.

[3]Como estos controles se aplicarán únicamente sobre el código nuevo, se permite incluir un proyecto cuya deuda técnica exceda este control, y evitar que la calidad técnica siga empeorando con el tiempo.

- Llevar a cabo un seguimiento de la calidad del código y de la cobertura de las pruebas.
- Consultar en tiempo real los resultados de cada proceso que se está realizando.
- Generar la documentación de un proyecto.

Todas estas funciones se pueden iniciar cada vez que se detecte que se haya realizado un Commit en el repositorio, iniciando los procesos configurados de forma automática.

Sin embargo, en Global Omnium todo el proceso de CI/CD está semi-automatizado, dado que está configurado para que sea el desarrollador el que inicie los procesos.

Para el proyecto *gav* hay configuradas diversos procesos. En el caso de la aplicación SAC, solo se hacen uso de los dos siguientes procesos:

- GAV Análisis
- GAV Compilar y Desplegar Rama en DES

En *GAV Análisis*, simplemente permite seleccionar una rama del repositorio para analizarla mediante SonarQube, proceso que conlleva en torno a una hora y media. Un resultado podría ser el siguiente:

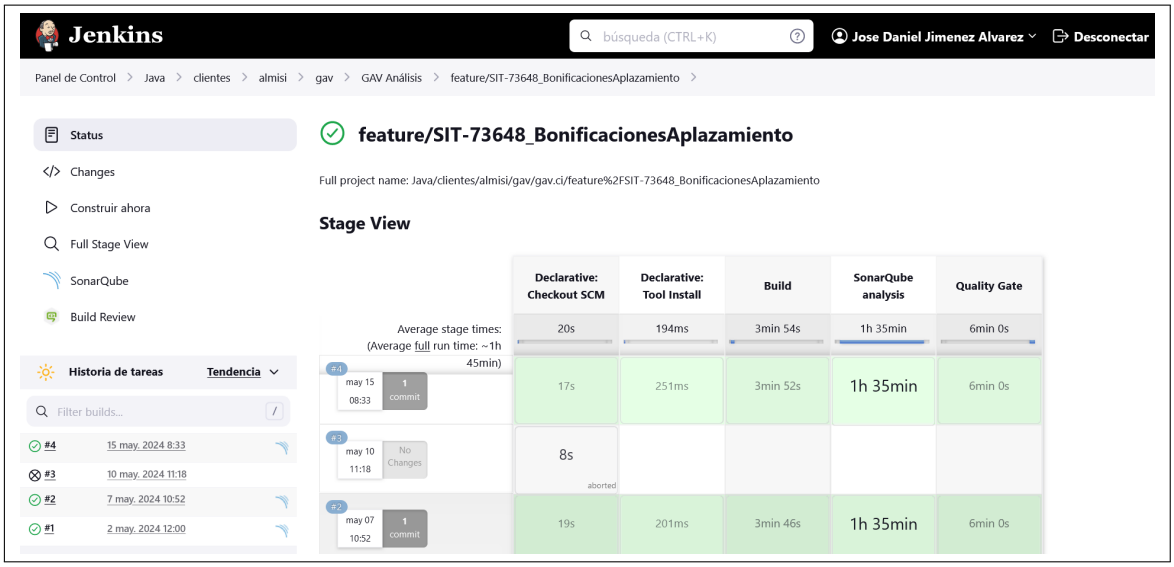


Figura 4.12: Ejemplo del proceso *GAV Análisis* en Jenkins

En cuanto a *GAV Compilar y Desplegar Rama en DES*, despliega en el entorno de Desarrollo la rama `test/` que seleccionemos, aproximadamente en unos diez minutos.

4.3 Checklist de desarrollo

En cada solicitud Jira en la que es necesario llevar a cabo un desarrollo, por muy insignificante que sea, se debe dejar documentada cada tarea de desarrollo que se ha realizado. Para ello, en cada solicitud es necesario añadir y cumplimentar las fases de la siguiente tabla:

FASE	ESTADO	DOCUMENTACIÓN
Análisis		
Desarrollo		
Revisión del código (análisis SonarQube)		
Plan de Pruebas		
Comunicación del despliegue en Pruebas		
Seguimiento del Plan de Pruebas		
Permiso para el despliegue en Producción		
Tareas previas a la implantación		
Merge Request		
Comunicación previa al despliegue		
Generación de la versión candidata		
Despliegue PROD		
Comunicación posterior a despliegue en PROD		

Tabla 4.1: *Checklist* de Desarrollo de una solicitud Jira

Como podemos ver, la tabla se compone de las tres siguientes partes:

- ***Fase*:** Nombre de todas las fases que ha de sufrir el desarrollo.
- ***Estado*:** Indica la situación en la que se encuentra cada fase. Los posibles valores de esta columna son:
  - *No realizado*: No se ha realizado la fase aún o porque se ha producido algún error en dicha fase.
  - *No computa*: Se ha realizado la fase, pero no ha sido necesario documentarla.
  - *En progreso*: En desarrollo se sitúa en dicha fase.
  - *Realizado*: Fase realizada correctamente.
- ***Documentación*:** Se añaden los documentos relacionados con cada fase que dejan constancia de las fases realizadas. Por ejemplo:
  - Correos electrónicos, tanto capturas de pantalla como el correo descargado.
  - Documentos de texto, imágenes, etc.
  - Enlaces a GitLab, SonarQube u otros recursos.

4.3.1. Análisis

La fase de Análisis consiste en dividir la modificación solicitada en subtareas. De esta forma:

- Se evalúa si la solución que se planea aplicar es viable.
- El desarrollador se asegura que todas las modificaciones que se proponen están contempladas.
- Si el desarrollo una vez empezado, pasa a estar en *stand by* durante un tiempo indefinido (ya sea porque se necesite solicitar información o por priorizar otra solicitud), esta fase ayuda al desarrollador a recordar todas tareas que ha de realizar.

Si el desarrollador considera que la modificación solicitada es muy fácil y rápida de realizar, es posible no pasar por esta fase, asignando al Estado de la *checklist* de desarrollo como *No computa*.

4.3.2. Rama de Desarrollo

Esta fase simplemente consiste en adjuntar el enlace a la rama de desarrollo en GitLab en la que se han implementado las modificaciones solicitadas.

4.3.3. Revisión del código (análisis SonarQube)

Esta fase simplemente consiste en adjuntar el enlace a Jenkins donde se ha analizado el nuevo código introducido en la rama de desarrollo correspondiente. Para ello, es necesario que la rama esté subida al repositorio de GitLab.

4.3.4. Plan de Pruebas

Una vez que el desarrollador considera que su implementación funciona correctamente y el resultado del análisis de la rama de desarrollo en SonarQube ha sido positiva, éste ha de elaborar un documento de texto (normalmente en Microsoft Word) dirigido a un usuario cualquiera del entorno de producción del SAC, que será el encargado de comprobar en el entorno de desarrollo que la implementación funciona como es esperado. Normalmente, este usuario aprobador será la misma persona que solicitó la modificación.

En este documento, el desarrollador explicará a nivel usuario las modificaciones que ha realizado y realizará una guía al usuario para que éste llegue a la conclusión de que las modificaciones realizadas son correctas.

Para ello, para cada aspecto a probar, se han de realizar estas 3 secciones:

1. **Reproducción:** Se especifican los pasos a seguir para llegar al punto de la aplicación donde se vaya a realizar la prueba. Normalmente se usan **casos de prueba** concretos que son los que el desarrollador utilizó para testear su desarrollo.

2. **Resultado esperado:** Se indican qué aspectos son los que el usuario ha de comprobar que son correctos.

3. **Resultado de la prueba:** El usuario declara si el resultado que ha obtenido coincide con el esperado.

4.3.5. Comunicación del despliegue en Pruebas

Una vez se haya finalizado de elaborar el Plan de Pruebas, se deberá actualizar respectiva rama `test/` para aplicar la modificación en el entorno de desarrollo, tal y como se muestra en la figura: Esquema simplificado de relación entre Ramas y Entornos.

Llegados a este punto, se le deberá comunicar al usuario a través del Jira correspondiente, que la **modificación solicitada ha sido desplegada en el entorno de desarrollo**, además de solicitarle que verifique mediante el Plan de Pruebas que la implementación realizada está funcionando correctamente.

En la documentación de esta fase, se deberá adjuntar:

1. El documento del Plan de Pruebas vacío.

2. Enlace al mensaje donde se le informa al usuario que la modificación está desplegada en el entorno de desarrollo.

4.3.6. Seguimiento del Plan de Pruebas

Si el usuario nos envía el Plan de Pruebas completado y verificando que todas las pruebas han sido validadas correctamente, se deberá documentar esta fase adjuntando el documento, que pasará a llamarse ***Resolución de Pruebas.***

Sin embargo, si en alguna de las pruebas el usuario detecta que el sistema no se comporta como es esperado, se deberá igualmente documentar esta Resolución de Pruebas **Fallida**, y posteriormente determinar la causa del error.

Si ha sido un error a la hora de elaborar el plan de Pruebas, simplemente se deberá rediseñar la sección correspondiente; pero si el error está relacionado con una mal implementación en el código, se deberán realizar todas las fases anteriores de nuevo para garantizar que toda modificación cumple con todas las fases realizadas.

4.3.7. Permiso para el despliegue en Producción

Cuando obtenemos la Resolución de Pruebas validada con éxito, se necesitará **permiso explícito del usuario** para comenzar el proceso de despliegue en el entorno de producción.

Los usuarios usualmente siempre contestarán que es posible iniciar el proceso, pero esta confirmación es estrictamente necesaria para documentar el usuario que solicita modificar el entorno de producción.

Como en el resto de comunicaciones, se le formulará directamente esta pregunta a través del Jira, y en la documentación de la fase se adjuntará un enlace a dicho mensaje.

4.3.8. Tareas previas a la implantación

En esta fase normalmente no se realizará ninguna opción, a no ser que se requiera alguna acción especial para su despliegue.

Cuando no se realice ninguna acción, se le asignará al campo *estado* el valor *No Computa*.

4.3.9. Merge Request

Para aplicar la modificación en la rama `develop` se deberá realizar un **Merge Request** desde GitLab, para que sea realizada por cualquier responsable del proyecto.

En este Merge Request, se asignarán estos valores:

- Rama origen: la rama `feature/` donde se ha llevado a cabo el desarrollo.
- Rama destino: `develop`.
- Persona asignada: Cualquier responsable del proyecto.[4]

Una vez realizado, el desarrollador habrá acabado todo su trabajo en el desarrollo. Las siguientes fases serán realizas por los responsables del proyecto, pero tendrán que ser documentadas en el Checklist de Desarrollo también por el desarrollador.

Para documentar esta fase, se deberá adjuntar el enlace al Merge Request realizado desde GitLab.

[4]En mi caso, al ser estudiante en practicas, siempre asigno a mi tutor de empresa (y cotutor de este TFG).

4.3.10. Comunicación previa al despliegue

Cuando se va a proceder a crear una nueva rama `release/` y con ello una nueva versión de la aplicación, un responsable del proyecto envía un **correo electrónico** a los administradores del sistema para informar de qué versión se solicita aplicar y qué modificaciones se han agregado.

Dicho correo se envía *Con Copia* a los desarrolladores que han participado en la `release/` para que sean conscientes de que sus modificaciones van a ser desplegadas en el entorno de producción y documentar esta comunicación en dichas solicitudes Jira.

4.3.11. Generación de la versión candidata

Esta fase hace referencia a la versión que se va a aplicar, ya comunicada en la Comunicación previa al despliegue.

Para documentar esta fase, simplemente se deberá añadir el enlace de GitLab de la versión generada.

4.3.12. Despliegue PROD

En este momento es cuando se despliega la nueva versión en el entorno de Producción, por parte de los administradores del sistema.

Como los desarrolladores no participan en esta fase, no se documentará nada pero se le asignará el estado: *Realizado*

4.3.13. Comunicación posterior a despliegue en PROD

Una vez desplegada la nueva versión del sistema en el entorno de Producción, un administrador del sistema responderá al correo electrónico de la Comunicación previa al despliegue comentando que el despliegue se ha realizado con éxito.

De igual forma que en la comunicación previa, se deberá documentar dicho correo en esta fase.

CAPÍTULO 5
Casos de uso

En este capítulo, se van a exponer algunas de las incidencias Jira sobre el SAC que mi tutor de empresa me asignó considerando que la inmensa mayoría de incendias me resultarían imposibles de comprender dada la compleja lógica de negocio de la empresa.

Estos casos de uso son algunos de los más complejos que he realizado, dada su relevancia, dificultad en el testeo o su dificultad de implementación.

5.1 Modificación del flujo en la funcionalidad de consultar las Tarifas Particulares de un suministro

En relación a la funcionalidad de consultar las Tarifas Particulares de un suministro, un usuario del SAC reportó una serie de errores que había observado en el comportamiento del flujo y en elementos de las vistas.

El usuario explicó en la solicitud con mucho detalle en qué consistían dichas anomalías. En primer lugar, indicó las tres vistas que estaban involucradas en el flujo: la vista de las Tarifas Particulares, la vista del Historial de una Tarifa Particular, y la vista de las Lecturas de una Tarifa ya pasada.

Al seleccionar un suministro, si tiene Tarifas Particulares se mostrará una vista como la siguiente. Si el suministro no tiene ninguna Tarifa Particular asociada, simplemente se muestra otra vista informando que no se han encontrado Tarifas. Para este caso de uso se utilizó como caso de prueba el suministro "..80/..1".

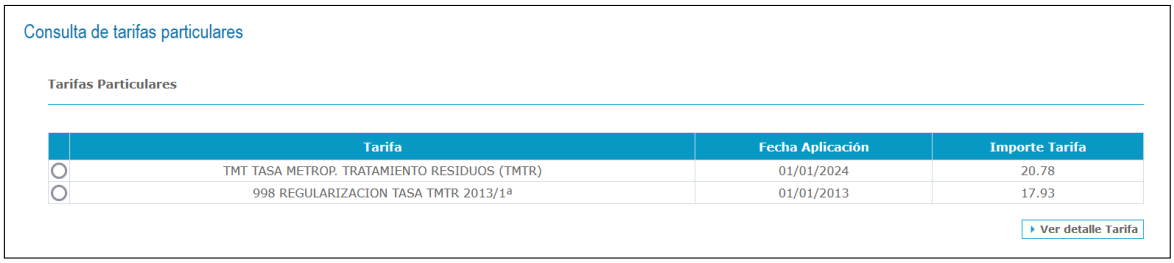


Figura 5.1: Vista de Tarifas Particulares

Al seleccionar una Tarifa Particular y pulsar en el botón *Ver Detalle Tarifa*, se abrirá la vista que muestra el Histórico de Tarifas (similar a un recibo):

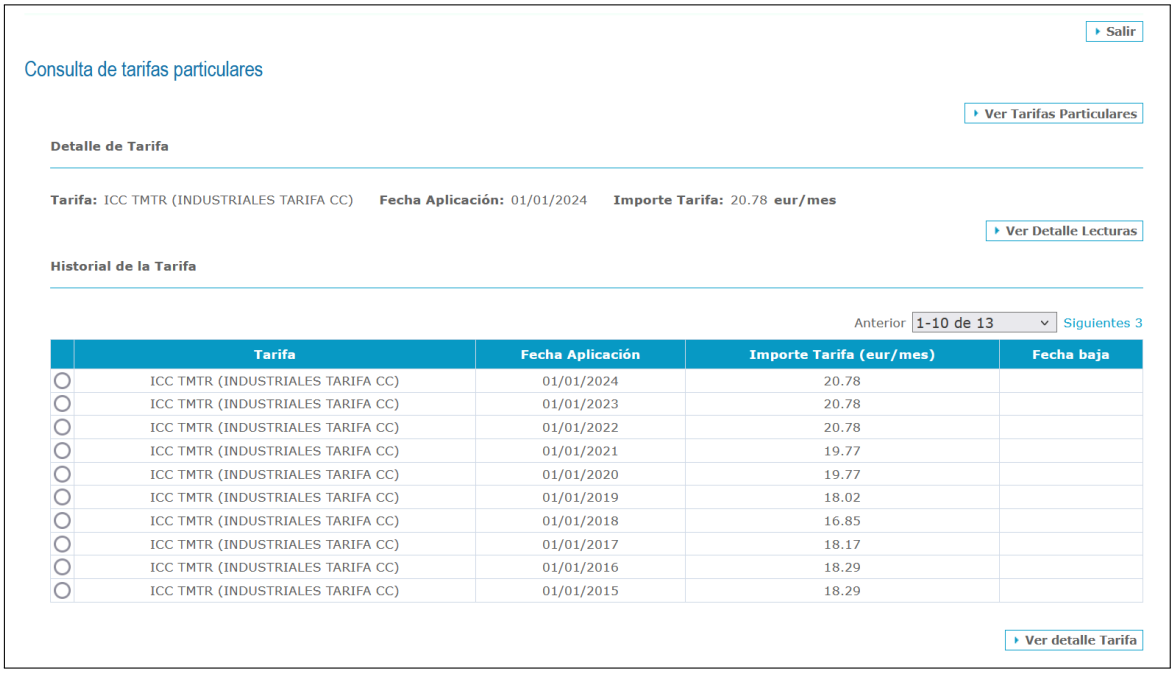


Figura 5.2: Vista del Histórico de una Tarifa Particular

En la vista anterior se observan los siguientes elementos:

- Botón *Ver Tarifas Particulares* para volver a la Vista de Tarifas Particulares
- Cabecera de datos ”Detalle de Tarifa” que muestra de forma resumida los datos asociados a la tarifa más reciente.
- Botón *Ver Detalle Lecturas* que abre la vista de las Lecturas de la **tarifa más reciente**
- Tabla "Historial de la Tarifa"que recopila todas las tarifas (recibos) asociadas a la Tarifa Particular
- Botón *Ver Detalle Tarifa* que **actualiza la cabecera** ”Detalle Tarifa” con los datos de la tarifa seleccionada de la tabla

Al seleccionar un registro de la tabla y pulsar en el botón *Ver Detalle Lecturas*, se abrirá la vista que muestra las lecturas asociadas a dicha factura. Si no tiene ninguna Lectura, se mostrará una tabla vacía:

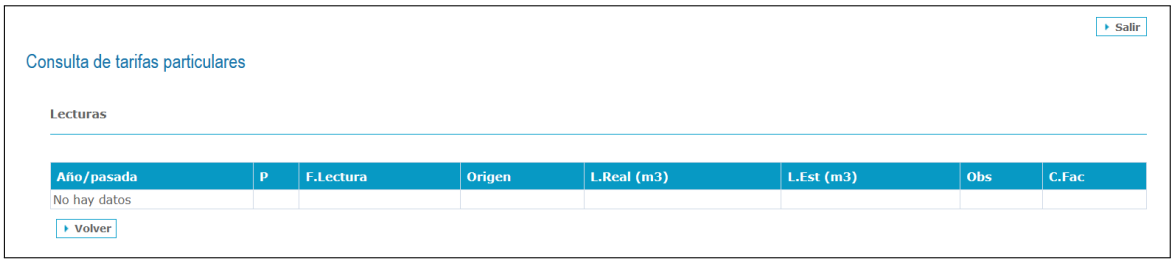


Figura 5.3: Vista de las Lecturas de una Tarifa

Una vez asimilado el funcionamiento del flujo, el usuario en base a la lógica del resto de la aplicación, reportó los dos siguientes problemas:

1. El botón *Ver Detalle Lecturas* debe actuar **sobre la tarifa seleccionada de la tabla**, no sobre la más reciente.

2. La cabecera de datos "Detalle de Tarifa" siempre debe de estar relacionada con la tarifa más reciente. Por lo que la implementación del Botón *Ver Detalle Tarifa* no es correcta.

5.1.1. Análisis

Una vez localizado el fichero XML que contiene el flujo de navegación, se deberá identificar qué elementos de JEveris corresponden con las vistas del SAC:

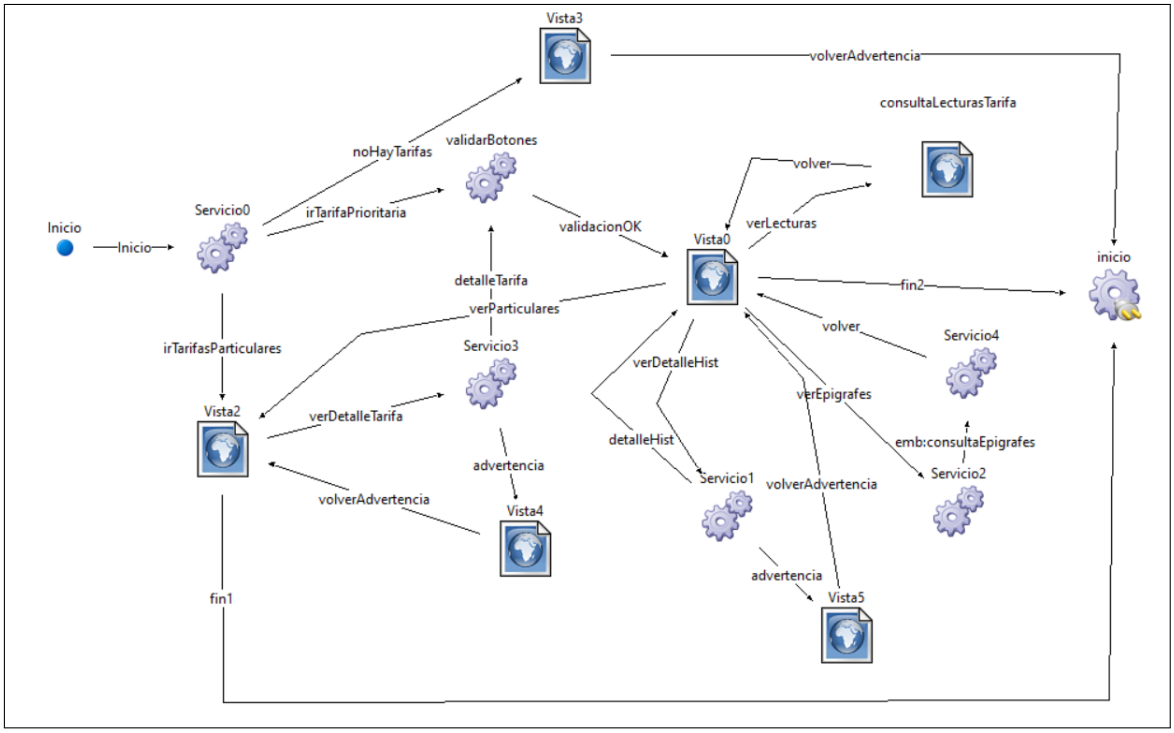


Figura 5.4: Flujo de la consulta de Tarifas Particulares

Como este grafo contiene poco elementos, resulta muy simple identificar las tres vistas anteriores:

Vista del SAC	Recurso de JEveris
Vista de Tarifas Particulares	Vista2
Vista del Histórico de una Tarifa Particular	Vista0
Vista de las Lecturas de una Tarifa	consultaLecturasTarifa

Tabla 5.1: Recursos JEveris de las vistas del flujo de consulta de Tarifas Particulares

Al continuar analizando el código y haciendo pruebas, al seleccionar la Tarifa Particular *998 REGULARIZACION TASA TMTR* de la Vista de Tarifas Particulares se detectaron errores en la tabla cuando ésta **solo tenia un registro**:

1. No es posible seleccionar el único registro de la tabla.

2. El botón *Ver Detalle Tarifa* aparece desactivado y sin formato.

Además, analizando la estructura de la Vista2, se observó que a parte de los datos visibles en la Vista del Histórico de una Tarifa Particular, se mostrarían tres datos más

en el caso de que la tarifa los tuviese: la Fecha de Baja, un Texto en Factura (una frase de pocas palabras) y el Consumo en m3.

Llegados a este punto, para completar el análisis faltaría decidir cómo se van a resolver los problemas solicitados.

Tras analizar detenidamente la mejor forma de aplicar esta modificación, en cuanto a facilidad y resultado final, se decidió cambiar varios elementos para hacer la navegación más intuitiva. Por tanto, le propusimos al usuario los siguientes cambios a realizar:

- En la Vista del Histórico de una Tarifa Particular: Como en la cabecera solo habían tres datos extra a los de la tabla, se propone **añadir todos los datos de la cabecera de datos "Detalle de Tarifa" en cada registro de la tabla**, dejando las columnas vacías si la tarifa no contenía dichos datos. De esta forma, se consigue que la cabecera de datos siempre contenga la información de la tarifa más reciente (Problema 2). Además gracias a esta modificación, es posible **eliminar el botón *Ver Detalle Tarifa***
- En la Vista de las Lecturas de una Tarifa: **Añadir una cabecera "Detalle de Tarifa"** pero con los datos de la tarifa seleccionada, ya que actualmente la vista no muestra la tarifa que se está consultando.

Dichos cambios son los que afectan a la funcionalidad del sistema, por lo que el usuario no requiere saber todos los cambios que se van a realizar. Sin embargo, es muy importante que el desarrollador tenga la buena practica de elaborar una *checklist* con todas estas modificaciones que debe aplicar. En este caso, estas son:

- En la Vista del Histórico de una Tarifa Particular:
  1. **Eliminar el botón *Ver Detalle Tarifa***.
  2. **Mover el botón *Ver Detalle Lecturas* en la parte inferior derecha de la tabla**, justo en la posición donde se situaba el botón *Ver Detalle Tarifa*.
  3. Permitir seleccionar el registro de la tabla cuando ésta solo tiene un registro.
  4. botón *Ver Detalle Lecturas* siempre visible y habilitado. Si no se selecciona un registro de la tabla, el sistema mostrará una advertencia.

5.1.2. Desarrollo

Es muy importante realizar los cambios de forma sistemática para asegurarse de realizar todas las modificaciones. Por este motivo, se decidió aplicar las modificaciones por orden de vistas, siendo la primera la Vista del Histórico de una Tarifa Particular (Vista0).

Esquematizando las modificaciones a realizar, únicamente son tareas de modificación de elementos ya existentes (textos, tablas, posición de elementos, etc.) y la modificación de la funcionalidad del botón *Ver Detalle Lecturas*. Por simplicidad, se decidió arreglar el botón como última implementación.

En resumen, en la Vista0 hay que realizar los siguientes cambios:

1. Tabla "Historial de la Tarifa":
   a) Añadir columnas *Texto en Tarifa* y *Consumo*
   b) Hacer que sea visible siempre
2. Cabecera "Detalle de Tarifa": *Texto en Tarifa* y *Consumo*

3. Botón *Ver Detalle Lecturas*:

   a) Reemplazar su posición a la de *Ver Detalle Tarifa*

   b) Cambiar su funcionalidad: las Lecturas a mostrar son las de la tarifa seleccionada de la tabla (Problema 1)

4. Botón *Ver Detalle Tarifa*: eliminarlo

Las modificaciones visuales son muy fáciles de aplicar, porque consisten en mostrar atributos ya existente de una forma que ya está definida en la vista, por lo que es buena idea implementar estos cambios primero.

En primer lugar, el fragmento de código que muestra la tabla es el siguiente:

```
<af:panelForm rendered="#{!empty FlowScope.mitablahist}">
<af:panelHeader text="#{msgs.Historial_de_Tarifa}"></af:panelHeader>
<af:objectSpacer height="8" width="0" />
<af:table rows="10" banding="row" id="nuevoId_nv" bandingInterval="3"
    emptyText="#{msgs.Listado_vacio}"
    binding="#{FlowScope.mitablahist.table}"
    value="#{FlowScope.mitablahist.tableMode}" var="row" width="100%">

    <af:column sortProperty="tarifa" sortable="false"
        headerText=" #{msgs.Tarifa}" formatType="icon">
        <af:outputText value="#{row.conceptoFTarifa} #{row.descConceptoF}"
    styleClass="" />
    </af:column>

    <af:column sortProperty="fecha_Aplicacion" sortable="false"
        headerText=" #{msgs.Fecha_Aplicacion}" formatType="icon">
        <af:outputText value="#{row.fechaAplTarifa}" styleClass="">
        <af:convertDateTime pattern="dd/MM/yyy"></af:convertDateTime>
        </af:outputText>
    </af:column>

    <af:column sortProperty="importe_Tarifa" sortable="false"
        headerText=" #{msgs.Importe_Tarifa} (eur/mes)" formatType="icon">
        <af:outputText value="#{row.importeTarifa}" styleClass="">
            <af:convertNumber minFractionDigits="2" maxFractionDigits="2"
    locale="en"></af:convertNumber>
        </af:outputText>
    </af:column>

    <af:column sortProperty="fecha_Baja" sortable="false"
        headerText=" #{msgs.Fecha_baja}" formatType="icon">
        <af:outputText value="#{row.fechaBajaTarifa}" styleClass="">
        <af:convertDateTime pattern="dd/MM/yyy"></af:convertDateTime>
        </af:outputText>
    </af:column>

    <f:facet name="selection">
        <af:tableSelectOne id="tableSelectOne2" disabled="#{!FlowScope.
    haySeleccion}" />
    </f:facet>
</af:table>
```

Listing 5.1: Tabla Historial de la Tarifa

El fragmento de código anterior se observa que se crea un panel que será visible únicamente si la tabla no esta vacía. En este caso, se vincula esta tabla a una variable *row* en que cada atributo se mostrará en la columna que se defina. En este caso, la tabla posee cuatro columnas, cuyos nombres provienen de una variable *msgs* que contiene todos

los textos que se muestran en la aplicación; y cuyos valores provienen de *row*. Como los nuevos atributos a mostrar ya estaban definidos en la aplicación, es posible saber con facilidad el nombre de estos atributos en el fragmento de código, ya que se muestran estos datos en la cabecera de este mismo fichero:

```
<af:panelHorizontal rendered="#{!empty FlowScope.miTarifa.textoTarifa}">
    <af:outputText value="#{msgs.Texto_Tarifa}: " inlineStyle="font-weight:
     bold;" />
    <af:objectSpacer height="0" width="5" />
    <af:outputText value="#{FlowScope.miTarifa.textoTarifa}" />
    <af:objectSpacer height="0" width="20" />
    <af:outputText value="#{msgs.Consumo}: " inlineStyle="font-weight: bold
    ;" />
    <af:objectSpacer height="0" width="5" />
    <af:outputText value="#{FlowScope.miTarifa.consumo}">
        <af:convertNumber maxFractionDigits="0" locale="en"></af:
    convertNumber>
    </af:outputText>
    <af:objectSpacer height="0" width="5" />
    <af:outputText value="m3" inlineStyle="font-weight: bold;" />
</af:panelHorizontal>
```

Listing 5.2: Fragmento de la cabecera Detalle Tarifa

Gracias a la figura anterior se deduce que los atributos **textoTarifa** y **consumo** solo son mostrados si *textoTarifa* no esta vacío.

Para aplicar estos cambios, únicamente se deberán añadir las columnas de forma análoga a las ya existentes, con los nuevos atributos *textoTarifa* y *consumo*. Una vez implantados, ya será posible eliminar el código que los mostraba en la cabecera.

Además, como se requería que si la tabla solo tuviese un elemento, éste debería ser seleccionable, se ha de **eliminar la propiedad *disabled*** de la faceta *selection* de la línea 49.

El resultado seria el siguiente:

```
<af:panelForm rendered="#{!empty FlowScope.mitablahist}">
<af:panelHeader text="#{msgs.Historial_de_Tarifa}"></af:panelHeader>
<af:objectSpacer height="8" width="0" />
<af:table rows="10" banding="row" id="nuevoId_nv" bandingInterval="3"
    emptyText="#{msgs.Listado_vacio}"
    binding="#{FlowScope.mitablahist.table}"
    value="#{FlowScope.mitablahist.tableMode}" var="row" width="100%">

    <af:column sortProperty="tarifa" sortable="false"
        headerText=" #{msgs.Tarifa}" formatType="icon">
        <af:outputText value="#{row.conceptoFTarifa} #{row.descConceptoF}"
    styleClass="" />
    </af:column>

    <af:column sortProperty="fecha_Aplicacion" sortable="false"
        headerText=" #{msgs.Fecha_Aplicacion}" formatType="icon">
        <af:outputText value="#{row.fechaAplTarifa}" styleClass="">
        <af:convertDateTime pattern="dd/MM/yyy"></af:convertDateTime>
        </af:outputText>
    </af:column>

    <af:column sortProperty="importe_Tarifa" sortable="false"
        headerText=" #{msgs.Importe_Tarifa} (eur/mes)" formatType="icon">
        <af:outputText value="#{row.importeTarifa}" styleClass="">
            <af:convertNumber minFractionDigits="2" maxFractionDigits="2"
    locale="en"></af:convertNumber>
        </af:outputText>
    </af:column>
```

```
27
28      <af:column sortProperty="fecha_Baja" sortable="false"
29          headerText=" #{msgs.Fecha_baja}" formatType="icon">
30          <af:outputText value="#{row.fechaBajaTarifa}" styleClass="">
31          <af:convertDateTime pattern="dd/MM/yyy"></af:convertDateTime>
32          </af:outputText>
33      </af:column>
34
35      <af:column sortProperty="texto_Tarifa" sortable="false"
36          headerText=" #{msgs.Texto_Tarifa}" formatType="icon">
37          <af:outputText value="#{row.textoTarifa}" styleClass="">
38          </af:outputText>
39      </af:column>
40
41      <af:column sortProperty="consumo" sortable="false"
42          headerText=" #{msgs.Consumo} (m3)" formatType="icon">
43          <af:outputText value="#{row.consumo}" styleClass="">
44              <af:convertNumber maxFractionDigits="0" locale="en"></af:
    convertNumber>
45          </af:outputText>
46      </af:column>
47
48      <f:facet name="selection">
49          <af:tableSelectOne id="tableSelectOne2" />
50      </f:facet>
51  </af:table>
```

Listing 5.3: Tabla Historial de la Tarifa modificada

En cuanto a los botones, como hay que reemplazar la posición de *Ver Detalle Tarifa*, que será eliminado, por el de *Ver Detalle Lecturas*, simplemente será necesario cortar el código del botón y sobrescribirlo en el botón a eliminar, cambiando los parámetros necesarios. El código original de *Ver Detalle Lecturas* es el siguiente

```
1  <af:commandButton action="jeveris:verLecturas" text="#{msgs.Ver_Lecturas}"
       rendered="#{FlowScope.verLecturas}"></af:commandButton>
```

Listing 5.4: Botón Ver Detalle Lecturas original

Una vez sobrescrito el código, se deberán cambiar los siguientes aspectos:

- Hacer que siempre sea visible
- Hacer que al ser pulsado, redirija a la Vista de las Lecturas de la tarifa seleccionada.

Para que siempre sea visible simplemente es suficiente con eliminar la propiedad **rendered**, pero para cambiar la funcionalidad es más complejo.

Observando el flujo, actualmente cuando se pulsa el botón *Ver Detalle Lecturas*, éste utiliza la transición *verLecturas* para mostrar la vista *consultaLecturasTarifa*, que muestra siempre las Lecturas de la tarifa más reciente. Desde esta vista, solo será posible volver a la Vista del Histórico de la Tarifa Particular. Analizando este bucle es lógico que actúe de esta forma, dado que no hay ningún servicio intermedio que detecte la tarifa de la tabla seleccionada.

Analizando el flujo más en profundidad, el comportamiento de detectar la tarifa de la tabla seleccionada y procesarla ya la realiza el **Servicio1** cuando se deseaba ver los detalles de una tarifa.

Reutilizar este servicio es la forma más simple y elegante de implementar la nueva funcionalidad deseada, simplemente será suficiente con:

1. Cambiarle el nombre a la transición *verDetalleHist* por otro más acorde a este caso o más general. Por ejemplo, *irServicio1*.

2. Mover la transición *verLecturas* para que su origen sea *Servicio1*, no Vista0.

Estos cambios hay que realizarlos en el flujo y posteriormente en el código, para que JEveris detecte que el grafo se corresponde con el código.

Por tanto, cambiando el flujo desde el editor de flujos, teniendo en cuenta que la transición *detalleHist* ya no existe y por tanto hay que eliminarla, se obtiene el siguiente resultado:

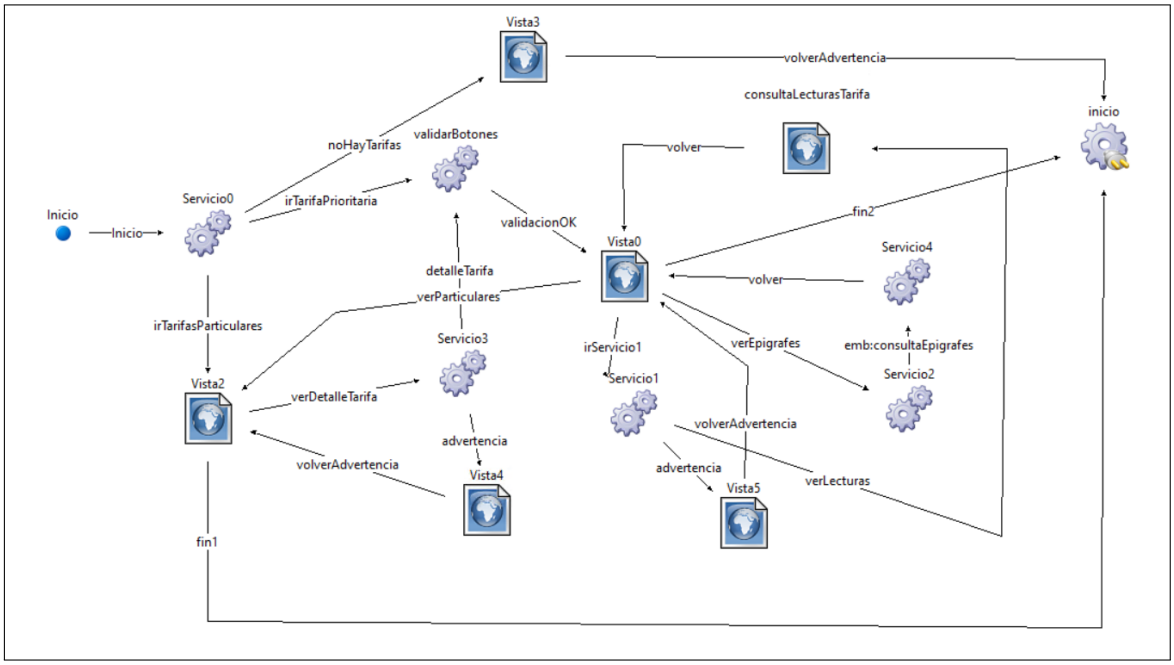


Figura 5.5: Nuevo flujo de la consulta de Tarifas Particulares

En primer lugar, en cuanto al código del Servicio1, originalmente realiza las siguientes acciones:

```
1  TablaSeleccionBean mitablahist = (TablaSeleccionBean) PLContext.
       getFromFlowScope("mitablahist");
2
3  List<TarifaParticular> seleccionado = (List<TarifaParticular>) mitablahist.
       getRowSelected();
4
5  boolean verLecturas = (Boolean) PLContext.getFromFlowScope("verLecturas");
6
7  if(seleccionado.size() > 0){
8
9      TarifaParticular miTarifa = seleccionado.get(0);
10
11     List<LecturaSGA> lecturas = null;
12
13     SessionScope misesion = (SessionScope) PLContext.get("SessionScope");
14     ObjetoSesion objetoSesion = (ObjetoSesion)misesion.getSessionObject(
       ObjetoSesion.identificadorObjetoSesion);
15     String canal = objetoSesion.getCanal();
16     ObjetoGestion objetoGestion = (ObjetoGestion)PLContext.getFromFlowScope
       ("objetoGestion");
17     String empresa = objetoGestion.getEmpresa();
18     String nia = objetoGestion.getNia();
```

```
19      String subnia = objetoGestion.getSubNia();
20
21      if (verLecturas) {
22
23          IConsultaLecturasAbonado miservicio = (IConsultaLecturasAbonado)
    Locator.getService("ConsultaLecturasAbonadoService");
24          String tipoLect = "";
25          String fechaApl = miTarifa.getFechaAplTarifa().substring(6, 10) +
    miTarifa.getFechaAplTarifa().substring(3, 5) + miTarifa.
    getFechaAplTarifa().substring(0, 2);
26          if (miTarifa.getConceptoTarifa()!= null && miTarifa.
    getConceptoTarifa().equals("TM1")) tipoLect = "T";
27          else if (miTarifa.getConceptoTarifa()!= null && miTarifa.
    getConceptoTarifa().equals("BAS")) tipoLect = "B";
28          else if (miTarifa.getConceptoTarifa()!= null && miTarifa.
    getConceptoTarifa().equals("TMT")) tipoLect = "C";
29          lecturas = miservicio.obtenerLecturasAbonado(canal, empresa, nia,
    subnia, tipoLect, fechaApl, "N");
30      }
31
32      TablaSeleccionBean tablaLecturas = new TablaSeleccionBean(lecturas);
33      PLContext.putInFlowScope("tablaLecturas", tablaLecturas);
34
35      PLContext.putInFlowScope("miTarifa", miTarifa);
36
37      return "jeveris:detalleHist";
38  }else{
39
40      PLContext.putInFlowScope("mensaje", "Debe seleccionar un registro de la
     tabla.");
41
42      return "jeveris:advertencia";
43  }
```

Listing 5.5: Código Servicio1 original

En dicho código, la variable *verLecturas* ya no tiene sentido, ya que es un valor que al principio del flujo se le asigna el valor *true* si la tarifa más reciente tiene Lecturas. Para implementar el nuevo diseño, se deberá:

1. **Añadir la variable de la tarifa seleccionada de la línea 9 en flujo** para ser usada en la Vista de las Lecturas de la tarifa seleccionada. Adicionalmente, se **adaptará su nombre** para facilitar la lectura del código; por ejemplo a *tarifaSeleccionada*.
2. El código de la sentencia *if* de la línea 21 deberá realizarse siempre. Es decir, se debe conservar el código pero **eliminando el la sentencia *if***.
3. El nombre del recurso JEveris a devolver en la línea 37 **debe ser *verLecturas***.

Aplicando los cambios, este sera el nuevo Servicio1:

```
1  TablaSeleccionBean mitablahist = (TablaSeleccionBean) PLContext.
       getFromFlowScope("mitablahist");
2
3  List<TarifaParticular> seleccionado = (List<TarifaParticular>) mitablahist.
       getRowSelected();
4
5  boolean verLecturas = (Boolean) PLContext.getFromFlowScope("verLecturas");
6
7  if(seleccionado.size() > 0){
8
9      TarifaParticular tarifaSeleccionada = seleccionado.get(0);
10
```

```
11      List<LecturaSGA> lecturas = null;
12
13      SessionScope misesion = (SessionScope) PLContext.get("SessionScope");
14      ObjetoSesion objetoSesion = (ObjetoSesion)misesion.getSessionObject(
        ObjetoSesion.identificadorObjetoSesion);
15      String canal = objetoSesion.getCanal();
16      ObjetoGestion objetoGestion = (ObjetoGestion)PLContext.getFromFlowScope
        ("objetoGestion");
17      String empresa = objetoGestion.getEmpresa();
18      String nia = objetoGestion.getNia();
19      String subnia = objetoGestion.getSubNia();
20
21      IConsultaLecturasAbonado miservicio = (IConsultaLecturasAbonado)Locator
        .getService("ConsultaLecturasAbonadoService");
22      String tipoLect = "";
23      String fechaApl = miTarifa.getFechaAplTarifa().substring(6, 10) +
        miTarifa.getFechaAplTarifa().substring(3, 5) + miTarifa.
        getFechaAplTarifa().substring(0, 2);
24      if (miTarifa.getConceptoTarifa()!= null && miTarifa.getConceptoTarifa()
        .equals("TM1")) tipoLect = "T";
25      else if (miTarifa.getConceptoTarifa()!= null && miTarifa.
        getConceptoTarifa().equals("BAS")) tipoLect = "B";
26      else if (miTarifa.getConceptoTarifa()!= null && miTarifa.
        getConceptoTarifa().equals("TMT")) tipoLect = "C";
27      lecturas = miservicio.obtenerLecturasAbonado(canal, empresa, nia,
        subnia, tipoLect, fechaApl, "N");
28
29      TablaSeleccionBean tablaLecturas = new TablaSeleccionBean(lecturas);
30      PLContext.putInFlowScope("tablaLecturas", tablaLecturas);
31
32      PLContext.putInFlowScope("tarifaSeleccionada", tarifaSeleccionada);
33
34    return "jeveris:verLecturas";
35  }else{
36
37      PLContext.putInFlowScope("mensaje", "Debe seleccionar un registro de la
         tabla.");
38
39      return "jeveris:advertencia";
40  }
```

Listing 5.6: Código Servicio1 original

Para transicionar a dicho servicio, será necesario implementar la transición *irServicio1* en la Vista0, **sustituyendo en el recurso JeEveris *detalleHist* por *irServicio1*** del Botón Ver Detalle Lecturas original:

```
1  <af:commandButton action="jeveris:irServicio1" text="#{msgs.Ver_Lecturas}">
       </af:commandButton>
```

Listing 5.7: Botón Ver Detalle Lecturas modificado

Finalmente, únicamente faltaría copiar la cabecera de la Vista del Histórico de una Tarifa Particular a la Vista de Lecturas. Para ello, se copiará toda la cabecera, cambiando el nombre de la variable utilizada por la que se hace uso en el Servicio1, es decir, *tarifaSeleccionada*: Botón Ver Detalle Lecturas original:

```
1  <af:panelHeader text="#{msgs.Detalle_Tarifa}"></af:panelHeader>
2  <af:objectSpacer height="8" width="0" />
3  <af:panelHorizontal>
4      <af:outputText value="#{msgs.Tarifa}: " inlineStyle="font-weight: bold;
       " />
5      <af:objectSpacer height="0" width="5" />
```

```
6       <af:outputText value="#{FlowScope.tarifaSeleccionada.conceptoFTarifa}
        #{FlowScope.tarifaSeleccionada.descConceptoF}" />
7       <af:objectSpacer height="0" width="20" />
8       <af:outputText value="#{msgs.Fecha_Aplicacion}: " inlineStyle="font-
        weight: bold;" />
9       <af:objectSpacer height="0" width="5" />
10      <af:outputText value="#{FlowScope.tarifaSeleccionada.fechaAplTarifa}" /
        >
11      <af:objectSpacer height="0" width="20" />
12      <af:outputText value="#{msgs.Importe_Tarifa}: " inlineStyle="font-
        weight: bold;" />
13      <af:objectSpacer height="0" width="5" />
14      <af:outputText value="#{FlowScope.tarifaSeleccionada.importeTarifa}">
15          <af:convertNumber minFractionDigits="2" maxFractionDigits="2"
        locale="en"></af:convertNumber>
16      </af:outputText>
17      <af:objectSpacer height="0" width="5" />
18      <af:outputText value="eur/mes" inlineStyle="font-weight: bold;" />
19  </af:panelHorizontal>
```

Listing 5.8: Nueva cabecera de la Vista de Lecturas

Llegados a este punto, ya están todas las modificaciones aplicadas. Las vistas modificadas tendrían la siguiente estructura:

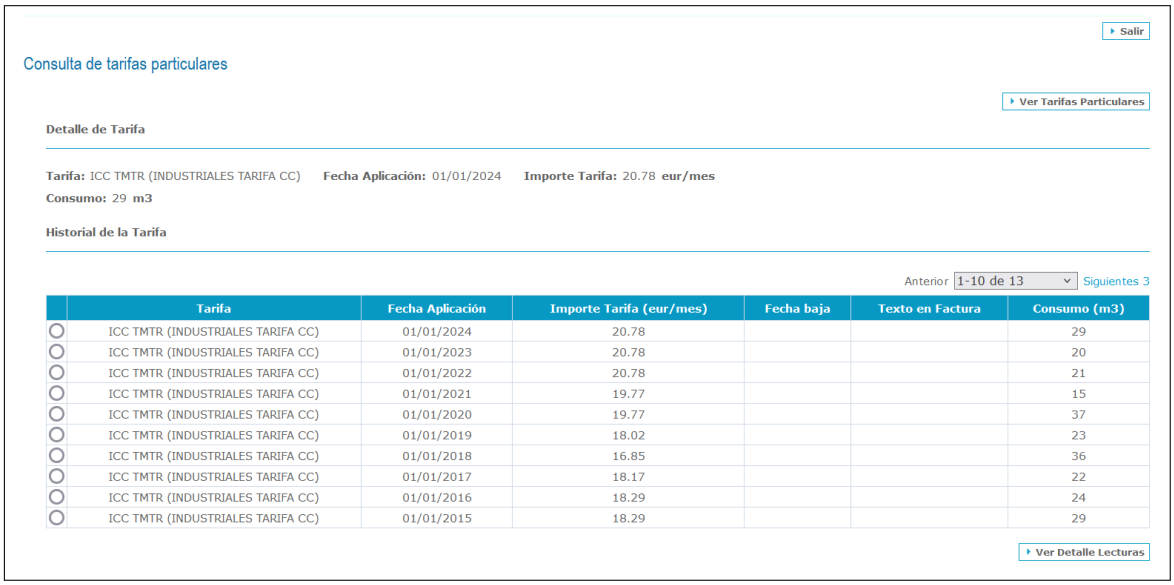


Figura 5.6: Vista del Histórico de una Tarifa Particular modificada

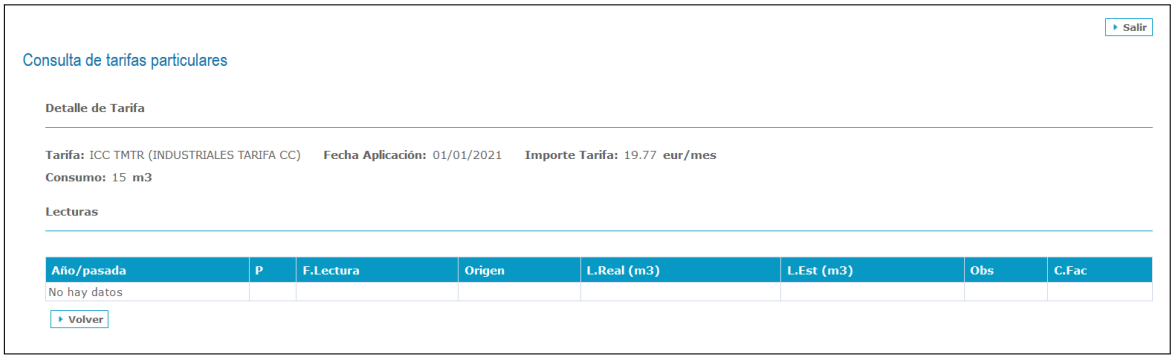


Figura 5.7: Vista de las Lecturas de una Tarifa modificada

5.1.3. Conclusiones

En una implementación como la que se ha llevado a cabo puede resultar peligroso evitar hacer el análisis de los cambios a realizar.

Como todos los cambios son sencillos de implementar y el resultado final es muy parecido al inicial, puede parecer que todo el desarrollo se puede terminar un muy poco tiempo, pero en un caso como este que ha sido necesario incluso modificar el flujo de navegación, **es vital tener la buena práctica de documentar los cambios que se van a realizar** para:

- Certificar que la solución propuesta es viable desde un primer momento.
- Realizar un seguimiento de las modificaciones pendientes.
- Explicar con precisión en qué consiste la modificación al usuario final.
- Optimizar el tiempo del desarrollo.
- Facilitar el testeo.

5.2 Implementación de direccionalidad de las llamadas telefónicas

En esta solicitud, el solicitante explica que debido a cambios que se van a producir en la forma en la que el servicio de atención al cliente se comunica con ellos, **los agentes podrán realizar llamadas telefónicas a los clientes**.

Esta nueva forma de contacto se necesita implementar en el SAC, además de realizar otros cambios en el funcionamiento del sistema como consecuencia.

Los cambios que se solicitan en la solicitud Jira son:

1. En el momento de iniciar un contacto por el Canal Telefónico, se deberá dar la opción de seleccionar si se trata de una llamada Entrante o Saliente, considerando las llamadas Salientes como el nuevo tipo de llamada. Por defecto debe ser Entrante.

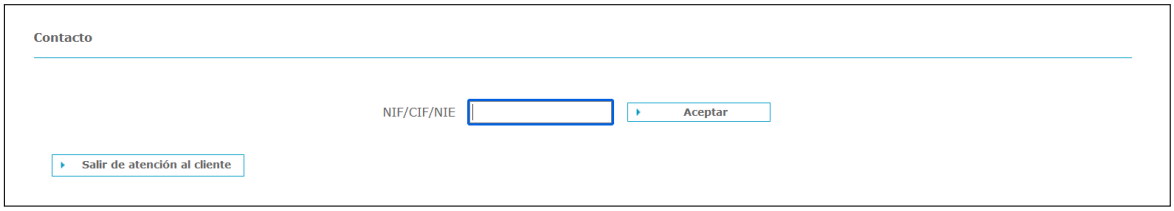


Figura 5.8: Vista original del inicio del Canal Telefónico

2. Al consultar los datos de un contacto telefónico, se debe indicar la direccionalidad de la llamada. Estos contactos están almacenados en la base de datos, por lo que cada dato que se muestra en la siguiente vista proviene de objetos ya registrados.

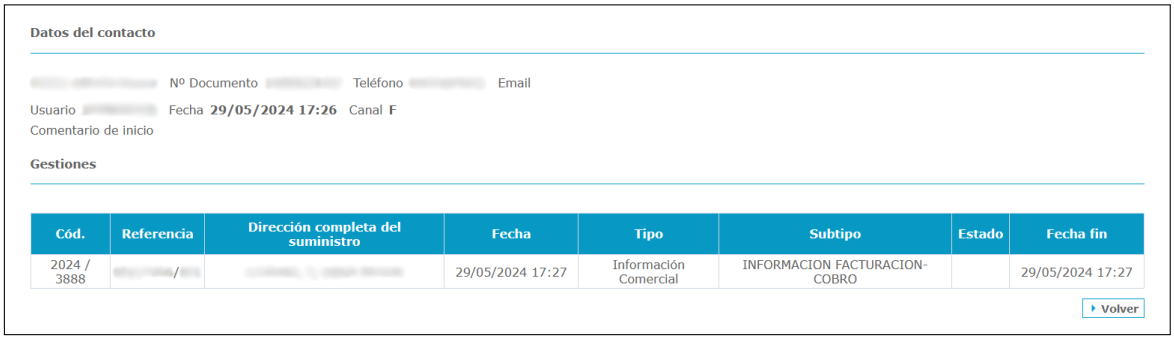


Figura 5.9: Ejemplo de la consulta de Datos de un Contacto

Esta diferenciación en la direccionalidad afectará a diversos servicios de la aplicación, principalmente en los criterios de filtrar Contactos. Para esta primera versión de la implementación, solo se solicita modificar el siguiente aspecto:

- Los datos que se utilizan para elaborar los informes de Función Comercial, no deben tener en cuenta los contactos salientes ni sus gestiones derivadas. Cada informe hace referencia a un tipo de Función Comercial.

5.2.1. Análisis

Este caso de uso puede parecer muy similar al primero, pero la principal diferencia entre ambos consiste en que en este se requiere **modificar el objeto del Contacto, tanto en el proyecto como en las bases de datos**.

En primer lugar, los cambios que se realizarán para dotar los contactos de direccionalidad son:

1. En el SAC:
   a) Añadir un nuevo atributo *booleano* al objeto Contacto del proyecto. Por ejemplo, *contactoSaliente*, que tendrá el valor *true* si el contacto es telefónico y saliente.
   b) Dar la opción en la Vista original del inicio del Canal Telefónico de seleccionar la direccionalidad de la llamada, siendo el valor por defecto Entrante.
   c) En el momento que se crea el Contacto, asignarle al nuevo atributo el valor indicado en el inicio del contacto.
   d) Al consultar los datos de un contacto, se deberá indicar la direccionalidad, junto al canal, solo si es Saliente.[1]
   e) De forma adicional, se añade la característica anterior en la cabecera de datos de la Vista principal del contacto.
   f) *Mapear* la variable en el proyecto con el de la relación en la base de datos.
2. En la Base de Datos:
   a) Añadir un nuevo atributo en la relación que almacena los contactos. El nombre de dicho campo será **GCCSAL**.

[1]Como usualmente prácticamente todas las llamadas son Entrantes, el sistema solo indicará la direccionalidad cuando sean Salientes

Para modificar el informe de función comercial, se requerirá modificar otro proyecto, *gav_exportarbbddparacm*, el cual contiene todas las **consultas SQL** que obtienen los datos registrados que serán empleados para elaborar los informes de Función Comercial.

Por tanto, cuando se hayan realizado las modificaciones anteriores, en el proyecto *gav_exportarbbddparacm* se deberá:

- Añadir en las consultas SQL pertinentes, la condición de que los Contactos sean Entrantes.

5.2.2. Desarrollo en SAC

En primera instancia, el objeto Contacto ha de ser modificado con el nuevo atributo ***contactoSaliente***. Simplemente se necesitará añadir este *booleano* junto a sus métodos *get* y *set*.

```
1  public class Contacto extends SessionObject implements Serializable {
2
3      ...
4      private boolean contactoSaliente;
5
6      ...
7      public boolean isContactoSaliente() {
8          return contactoSaliente;
9      }
10     public void setContactoSaliente(boolean contactoSaliente) {
11         this.contactoSaliente = contactoSaliente;
12     }
13 }
```

Listing 5.9: Modificación del objeto Contacto

Para realizar las modificaciones de la Vista original del inicio del Canal Telefónico, en primer lugar se necesita localizar el flujo que lo contiene. En este caso, esta vista solo es usada en el flujo **contacto.xml**.

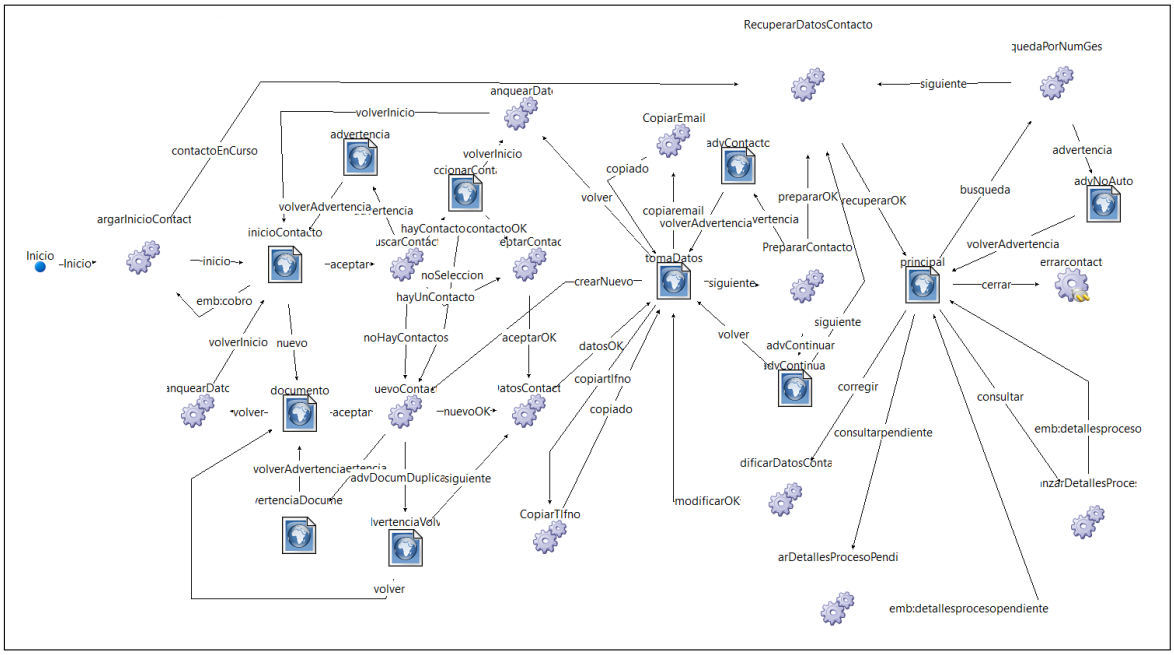


Figura 5.10: Flujo del inicio de un Contacto

Partiendo del inicio, se observa que el primer elemento es el servicio **CargarInicioContacto**, del cual parten dos caminos diferentes hasta llegar a la Vista principal del contacto. Para evitar errores en algún punto del flujo, se inicializara la variable en este servicio, siendo su valor predeterminado *false* si se va a iniciar un nuevo Contacto, y conservando su valor si este ya está en curso.

```
...

PLContext.putInFlowScope("contactoSaliente", false);
if (contacto != null){
    ...
    PLContext.putInFlowScope("contactoSaliente", contacto.
    isContactoSaliente());
    return "jeveris:contactoEnCurso";
}
...

return "jeveris:inicio";
```

Listing 5.10: Modificación del servicio CargarInicioContacto

En la Vista original del inicio del Canal Telefónico, se deberá cambiar el valor de dicha variable si el agente selecciona que el contacto es Saliente. Para ello, por simpleza se procede a implementar un **checkbox** debajo del *input* del NIF/CIF/NIE, el cual se mostrará solo en el Canal Telefónico y por defecto permanecerá desmarcado. Para su implementación, se añadirá el siguiente elemento en el código de la Vista.

```
<af:panelHorizontal halign="center" rendered="#{FlowScope.canal == 'F'}">
    <af:selectBooleanCheckbox id="checkBoxContactoSaliente" text="#{msgs.
    Es_Contacto_Saliente}" value="#{FlowScope.contactoSaliente}"></af:
    selectBooleanCheckbox>
</af:panelHorizontal>
```

Listing 5.11: Modificación de la Vista inicial del Canal Telefónico

Esta modificación se refleja en el SAC de la siguiente manera:

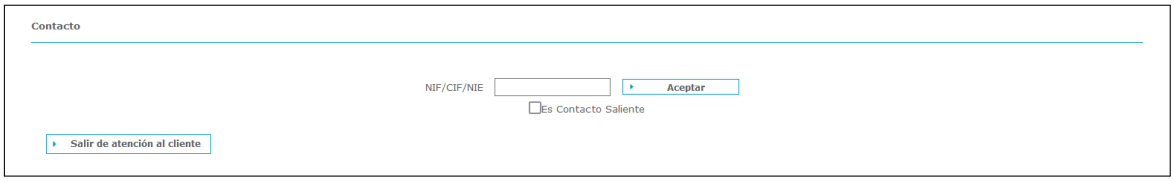


Figura 5.11: Vista modificada del inicio del Canal Telefónico

Analizando el resto del flujo, se observa que en el servicio **PrepararContacto** crea el objeto Contacto y lo actualiza en el flujo, utilizando los datos que se han recopilado en las vistas y servicios previos. Es lógico que sea en este servicio donde añadirle al Contacto el valor de la nueva variable, introduciendo el siguiente fragmento de código:

```
if (contacto.getCanal().equals("F")){
    boolean esLlamadaSaliente = (Boolean) PLContext.getFromFlowScope("
    esLlamadaSaliente");
    contacto.setContactoSaliente(esLlamadaSaliente);
}
```

Listing 5.12: Modificación en el servicio PrepararContacto

En este punto, el nuevo atributo ya está implementado en el proyecto. Para mostrarla en la cabecera de datos del contacto en curso, simplemente se deberá obtener el valor de

la variable **contacto**; pero para mostrarla en la consulta de datos del contacto, al ser datos que provienen de la base de datos, se deberá realizar la implementación una vez se haya añadido este atributo en la relación correspondiente.

Respecto a la cabecera de datos, simplemente en la vista que contiene la cabecera[2], se deberá añadir junto al canal, un texto indicando la direccionalidad de la llamada solo si es saliente:

```
<af:outputText value="&lt;li>Canal: &lt;span>" escape="false"
    rendered="#{FlowScope.contacto.canal != null and FlowScope.contacto.
    canal != ''}" />
<af:outputText styleClass=""
    value="#{FlowScope.contacto.canal} "
    rendered="#{FlowScope.contacto.canal != null and FlowScope.contacto.
    canal != ''}" />
<af:outputText rendered="#{FlowScope.contacto.canal == 'F' and FlowScope.
    contacto.contactoSaliente == true}"
    value="(#{msgs.Contacto_Saliente}) "
/>
```

Listing 5.13: Modificación de la cabecera de datos del contacto en curso

Con este cambio, la cabecera pasará a mostrarse, cuando la llamada sea saliente, de la siguiente forma:

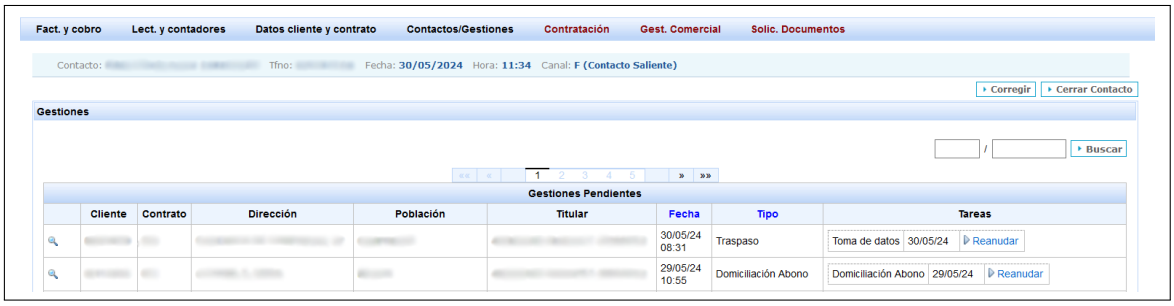


Figura 5.12: Cabecera modificada de los datos del contacto en curso

5.2.3. Desarrollo en la Base de Datos

Como se ha mencionado anteriormente, es necesario modificar la relación que almacena todos los registros que se crean cuando se inicializa un Contacto. Posteriormente, se deberá actualizar el proyecto *gav_exportarbbddparacm* para que las consultas SQL que obtienen los informes de Función Comercial, discriminen los contactos salientes.

En primer lugar, gracias a la herramienta MySQL Workbench es posible acceder a los esquemas utilizados en el proyecto[3]. En el esquema **sgae_des** se encuentran multitud de relaciones que registran todo tipo de acciones en el sistema, siendo una de ellas la relación **cfgccon**, la cual registra todos los contactos entre cliente y gestor del equipo del servicio de atención al cliente.

Desde esta misma herramienta, es posible modificar la relación de forma visual, para posteriormente obtener el ***script*** que la actualizará. Para añadir el nuevo atributo, se le asignará el nombre **CGCSAL** de tipo **BIT**, siendo **no nulo** y teniendo el valor por defecto el bit **0**, y se añadirá como **último atributo**.

[2]Las cabeceras son un tipo de vistas que están incrustadas en las vistas principales

[3]A los desarrolladores solo se les permite el acceso a los esquemas relacionados con el entorno de desarrollo, por lo que el resto de esquemas deberán de ser actualizados por los administradores de las bases de datos

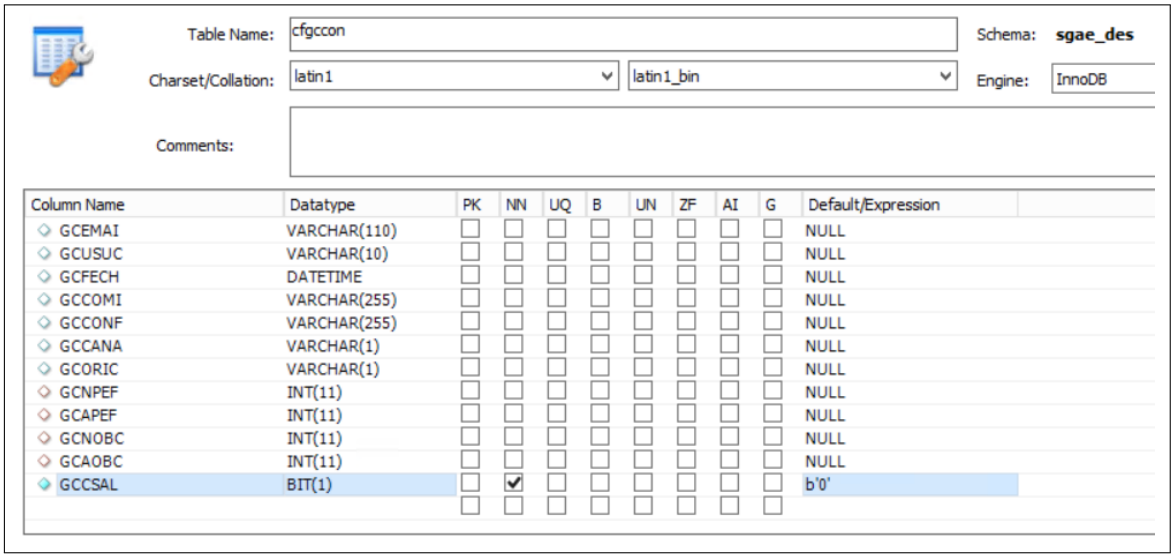


Figura 5.13: Creación del nuevo atributo GCCSAL en la relación cfgccon

Al guardar los cambios introducidos, la aplicación mostrará el *script* obtenido, el cual actualizará todos los registros de la relación CFGCON del esquema SGAE_DES[4].

```
1 ALTER TABLE `sgae_des`.`cfgccon`
2 ADD COLUMN `GCCSAL` BIT(1) NOT NULL DEFAULT 0 AFTER `GCAOBC`;
```

Listing 5.14: script obtenido

De esta forma, los registros de la relación **cfgccon** tendrán la siguiente estructura:

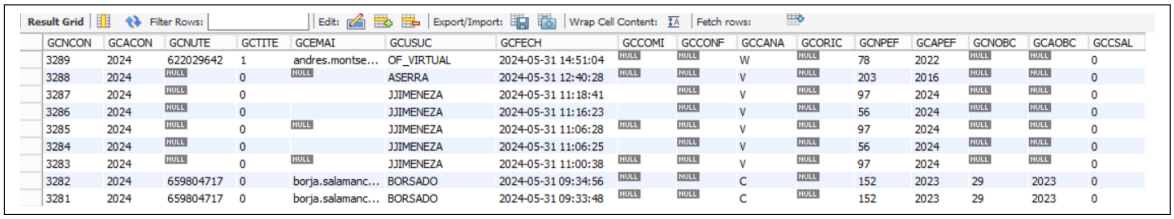


Figura 5.14: Relación cfgccon modificada con el nuevo atributo GCCSAL

A continuación, gracias a que el proyecto *gav* hace uso del *framework* de mapeo objeto-relacional **Hibernate**, el trabajo de mapear ambos campos será muy sencillo. Será suficiente con dirigirse al correspondiente archivo hbm.xml que hace uso cada objeto que se registra en la base de datos, en este caso **contacto.hbm.xml**, y relacionar ambos datos de forma análoga al resto de atributos del fichero. En este fichero XML están contenidas todas las configuraciones relacionadas con:

- El objeto a registrar.
- El nombre de la relación.
- Correspondencia entre los atributos del objeto a registrar y el nombre de los campos de la relación.
- Propiedades de los campos de la relación, como clave primaria, clave ajena, valor no nulo, etc.

[4]Este *script* solo ha modificado el entorno de desarrollo, por lo que se debe que cambiar el nombre del esquema para ser ejecutado en el entorno de producción y preproducción

- Tipo de relaciones con otros objetos del proyecto.
- Consultas SQL sobre esta relación.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
       3.0//EN"
3  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
4
5  <hibernate-mapping>
6      <class
7      name="es.aguasdevalencia.sac.modelo.gestorcontacto.Contacto"
8      table="CFGCCON">
9
10     <composite-id name="identificador" class="es.aguasdevalencia.sac.modelo
       .gestorcontacto.Identificador">
11       <key-property name="anyo" column="GCACON" length="4"></key-property>
12       <key-property name="id" column="GCNCON" length="7"></key-property>
13     </composite-id>
14
15     <property name="telefono" column="GCNUTE" length="15"/>
16     <property name="tipoTelefono" column="GCTITE" length="1"/>
17     <property name="email" column="GCEMAI" length="110"/>
18     <property name="usuario" column="GCUSUC" length="10"/>
19     <property name="fecha" column="GCFECH" length="26"/>
20     <property name="contactoInicio" column="GCCOMI" />
21     <property name="contactoFin" column="GCCONF"/>
22     <property name="canal" column="GCCANA" length="1"/>
23     <property name="origen" column="GCORIC" length="1"/>
24     <property name="contactoSaliente" column="GCCSAL"/>
25
26     <many-to-one name="personaFisicaJuridica" class="es.aguasdevalencia.sac
       .modelo.gestorcontacto.PersonaFisicaJuridica" lazy="false" fetch="join"
       >
27       <column name="GCAPEF" length="4"></column>
28       <column name="GCNPEF" length="7"></column>
29     </many-to-one>
30
31       ...
32
33     </class>
34
35     <query name="Contacto.buscarPorContacto">
36         <![CDATA[
37          select c
38          from es.aguasdevalencia.sac.modelo.gestorcontacto.Contacto as c
39          where c.identificador.id = :id
40          and c.identificador.anyo = :anyo
41         ]]>
42     </query>
43
44     ...
45
46 </hibernate-mapping>
```

Listing 5.15: Fragmento del fichero contacto.hbm.xml modificado

Gracias a tener todas las propiedades de la relación centralizadas con Hibernate, con solo introducir la línea 24 de la figura anterior en el fichero .hbm.xml correspondiente, ha sido suficiente para implementar el nuevo atributo.

En este momento ya es posible obtener dicho valor en la vista de Consulta de datos de un Contacto. Para ello, será necesario realizar una modificación análoga a la anterior-

mente aplicada en cabecera de datos del Contacto en curso. Simplemente observando en el navegador la dirección url de la vista a modificar, se deduce que ésta se llama **ListadoProcesosContactoPtes.jsp**. Sabiendo el nombre, es posible abrir el fichero en Eclipse y proceder a su modificación, añadiendo la palabra "Saliente" a la derecha del Canal, solo si el contacto ha sido telefónico y saliente:

```
<af:outputText value="#{msgs.Canal}" styleClass=""/>
<af:outputText value="#{FlowScope.contactoOperacion.canal}" inlineStyle="
    font-weight:bold; padding-left:5px"/>
<af:outputText rendered="#{FlowScope.contactoOperacion.canal == 'F' and
    FlowScope.contactoOperacion.contactoSaliente == true}"
     value="(#{msgs.Contacto_Saliente}) " inlineStyle="font-weight:bold;
    padding-left:5px"
/>
<af:objectSpacer height="0" width="10" />
```

Listing 5.16: Modificación de la vista de consulta de datos de un contacto

Ahora al iniciar un contacto telefónico saliente y consultar sus datos, la cabecera se mostrará de la siguiente manera:

Figura 5.15: Modificación en la consulta de Datos de un Contacto

5.2.4. Desarrollo en *gav_exportarbbddparacm*

Para acceder a *gav_exportarbbddparacm* desde Eclipse y posteriormente modificar las consultas SQL relacionadas con el informe de Función Comercial, se deberá en primera instancia clonar el proyecto mediante en enlace correspondiente de GitLab, y posteriormente acceder a la clase **Main.java**, que es la única clase que contiene este proyecto.

Analizando dicha clase, se observa que todas las consultas SQL relacionadas con la Función Comercial tiene definidas unas constantes que las representan:

```
//Constantes definidas para la Funcion Comercial. FICHEROS donde se
    generara la informacion en el AS/400:
//     --> BBWSGACFYD/SACUSUFUCO: BW Cuadro de Mando. Datos relacionados con
     la Funcion Comercial [Ratios por Usuario del SAC]
//     --> BBWSGACFYD/SACUSUARI:  BW Cuadro de Mando. Maestro Usuarios [para
     identificar el dato del Usuario que aparece en SACUSUFUCO]
private final static String funComTipoDatoNumeroGestionesDomiciliacion = "
    FCDOM";
private final static String funComTipoDatoNumeroGestionesCobro = "FCCOB";
private final static String
    funComTipoDatoNumeroGestionesAltaConDomiciliacion = "FCACD";
private final static String funComTipoDatoNumeroGestionesAlta = "FCALT";
private final static String
    funComTipoDatoNumeroGestionesTraspasoConDomiciliacion = "FCTCD";
private final static String funComTipoDatoNumeroGestionesTraspaso = "FCTRA"
    ;
private final static String funComTipoDatoNumeroGestionesCobroAnonimo = "
    FCCOA";
```

```
11 private final static String
       funComTipoDatoNumeroContactosRegistradosConDatoNuevoOrActualizadoTelefono
        = "FCCNT";
12 private final static String
       funComTipoDatoNumeroContactosRegistradosConDatoNuevoOrActualizadoEmail
       = "FCCNE";
13 private final static String
       funComTipoDatoNumeroContactosRegistradosConDatoMantenidoTelefono = "
       FCCMT";
14 private final static String
       funComTipoDatoNumeroContactosRegistradosConDatoMantenidoEmail = "FCCME"
       ;
15 private final static String
       funComTipoDatoNumeroContactosRegistradosSinDatoTelefono = "FCCST";
16 private final static String
       funComTipoDatoNumeroContactosRegistradosSinDatoEmail = "FCCSE";
17 private final static String funComTipoDatoNumeroContactosRegistrados = "
       FCCON";
18 private final static String funComTipoDatoNumeroGestionesAltaConFacturaE =
       "FCACF";
19 private final static String
       funComTipoDatoNumeroGestionesTraspasoConFacturaE = "FCTCF";
```

Listing 5.17: Tipos de Funciones Comerciales

En todas las consultas relacionadas con estas constantes, se mantiene la condición `PRO.GCCANA IN ("+canalVentanilla+","+canalTelefono+")`, siendo:

- **PRO: Alias dado a la relación CFGCPRO que registra todos los Procesos realizados en el SAC**.
- GCCANA: El campo de la relación que contiene el código del canal.
- canalVentanilla y canalTelefono: Constantes que representan los códigos del canal Ventanilla ('V') y el canal Telefónico ('F'), respectivamente.

```
1  sql = "SELECT DISTINCT PRO.GCUPRO, PRO.GCCANA, COUNT(*) FROM SGAE.CFGCPRO
       PRO "+
2       " WHERE PRO.GCTPRO = "+tipoProDomiciliacionAbono+
3       " AND (PRO.GCCANA IN ("+canalVentanilla+") OR (PRO.GCCANA IN ("+
       canalTelefono+") AND !PRO.GCCSAL)) "+
4       " AND PRO.END_ IS NOT NULL AND PRO.END_ BETWEEN '"+new java.sql.
       Timestamp(dateFechaDesde.getTime())+"' AND '"+new java.sql.Timestamp(
       dateFechaHasta.getTime())+"' "+
5       " AND NOT EXISTS( "+
6       "   SELECT * FROM SGAE.CFGCOPE COPE WHERE COPE.GCNPRH = PRO.GCNPRO AND
       COPE.GCAPRH = PRO.GCAPRO) "+
7       "GROUP BY PRO.GCUPRO, PRO.GCCANA";
8
9  rs = stmtmysql.executeQuery(sql);
10 while(rs.next()){
11      pasarDatosAlAS/400DatosMensuales(rs,
       funComTipoDatoNumeroGestionesDomiciliacion);
12 }
```

Listing 5.18: Consulta SQL para funComTipoDatoNumeroGestionesDomiciliacion

Para llevar a cabo la modificación se deberá refinar esta condición, filtrando por los Procesos cuyo Canal sea Ventanilla **o Telefónico si el Contacto relacionado no es saliente**.

Utilizando MySQL Workbench también es posible obtener gráficamente cómo se relacionan las relaciones CFGCCON y CFGCPRO, llegando a la conclusión que es necesario involucrar diversas relaciones, complicando la implementación, configuración y sus

mantenibilidad. Por este motivo, se decidió **modificar también CFGCPRO añadiéndole el campo GCCSAL, de igual forma que la relación CFGCCON**. Para ello, será necesario realizar las siguientes acciones, muy similares a las anteriores:

1. En el SAC:
   a) Añadir una variable *booleana* al objeto Proceso del proyecto. De igual forma que en Contacto.
   b) En el momento que se crea el Proceso, asignarle al nuevo atributo el mismo valor que la variable contactoSaliente del Contacto relacionado.
   c) *Mapear* la variable en el proyecto con el de la relación en la base de datos.
2. En la Base de Datos, usando MySQL Workbench:
   a) Modificar la relación CFGCPRO que almacena los procesos con la nueva variable. De igual forma que en **GCCSAL**.

```
ALTER TABLE `sgae_pre`.`cfgcpro`
ADD COLUMN `GCCSAL` BIT(1) NOT NULL DEFAULT 0 AFTER `END_`;
```

Listing 5.19: Script obtenido al modificar CFGCPRO

```
public class Proceso implements Serializable {

    ...
    private boolean contactoSaliente;

    ...
    public boolean isContactoSaliente() {
        return contactoSaliente;
    }
    public void setContactoSaliente(boolean contactoSaliente) {
        this.contactoSaliente = contactoSaliente;
    }
}
```

Listing 5.20: Modificación del objeto Contacto

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
    3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class
        name="es.aguasdevalencia.sac.modelo.gestorcontacto.Proceso"
        table="CFGCPRO">

    ...
      <property name="contactoSaliente" column="GCCSAL"/>

      ...

  </class>
</hibernate-mapping>
```

Listing 5.21: Fragmento del fichero proceso.hbm.xml modificado

Para *settear* el valor de la variable al Proceso, en este objeto no se crea de la misma manera que el Contacto, ya que los datos del contacto se recopilaban en distintas vistas

para que finalmente, en un único servicio, se realicen la validaciones necesarias para asignarlos al objeto. Sin embargo, los procesos se inician con un método **iniciarProceso** de la clase **GestorContacto**, que tiene la misma funcionalidad que el servicio que procesa los datos del Contacto. Este método inicia un Proceso dado los parámetros necesarios, y devuelve un objeto Tarea que representa la primera operación/tarea del proceso. En las líneas 23 y 30 se le asigna el valor correspondiente a la variable *contactoSaliente*, de forma análoga al resto del método.

```
1  public class GestorContacto implements Serializable, IGestorContacto {
2
3      ...
4
5      public Tarea iniciarProceso(Contacto contacto, ObjetoGestion
       objetoGestion,
6        Integer tipoProcesoId, Integer subTipoProcesoId, String usuario,
7        String comentario, String canal, Integer idUsuarioOV) {
8
9      TipoProceso tipoProceso = tipoProcesoDAO.findTipoProceso(tipoProcesoId)
       ;
10
11     ProcessInstance pI = jbpmDAO.startProcess(tipoProceso
12         .getProcessDefinitionName());
13
14     Proceso proceso = new Proceso();
15     // Por defecto el campo estado estara en blanco. Los procesos cerrados
16     // tienen fecha de cierre
17     proceso.setEstado("");
18     // Se anyade el usuario y el canal al proceso
19     proceso.setUsuario(usuario);
20     if (canal == null) {
21       if (contacto != null) {
22         proceso.setCanal(contacto.getCanal());
23         proceso.setContactoSaliente(contacto.isContactoSaliente());
24       } else {
25         proceso.setCanal("K");
26       }
27     } else {
28       proceso.setCanal(canal);
29       if (contacto != null) {
30         proceso.setContactoSaliente(contacto.isContactoSaliente());
31       }
32     }
33     proceso.setIdUsuarioOV(idUsuarioOV);
34
35     if (objetoGestion != null && objetoGestion.getIdentificador() == null)
36       objetoGestion = objetoGestionDAO.insert(objetoGestion);
37
38     SubtipoProceso subTipoProceso = null;
39     if (subTipoProcesoId != null) {
40       subTipoProceso = new SubtipoProceso();
41       subTipoProceso.setId(subTipoProcesoId);
42     }
43     proceso.setObjetoGestion(objetoGestion);
44     proceso.setSubTipoProceso(subTipoProceso);
45     proceso.setTipoProceso(tipoProceso);
46
47     proceso.setProcesoBpm(pI);
48     // Guardamos explicitamente las fechas de inicio/fin
49     proceso.setStart(pI.getStart());
50     proceso.setEnd(pI.getEnd());
51     // Insertamos en proceso
52     proceso = procesoDAO.insert(proceso);
53
```

```
54      // Recuperamos el tipo de operacion
55      TipoOperacion tipoOperacion = new TipoOperacion();
56      tipoOperacion.setId(TipoOperacion.TipoOperacionInicioProceso);
57      // Insertamos en operacion
58      operacionDAO.insert(proceso, null, contacto, tipoOperacion, usuario,
        comentario, canal);
59
60      //Grabamos el contacto para que WmtM pueda sincronizarselo si procede
61      grabarContactoWmtM(contacto, objetoGestion);
62
63      Collection<TaskInstance> listaResult = pI.getTaskMgmtInstance()
64          .getTaskInstances();
65      Tarea result = null;
66      if (listaResult != null && !listaResult.isEmpty())
67        result = convertTaskInstance(listaResult.iterator().next(), proceso);
68      return result;
69    }
70
71      ...
72  }
```

Listing 5.22: Modificacion en la creación de un Proceso

Finalmente, ya es posible refinar las consultas SQL de cada Función Comercial, modificando cada aparición de la condición

```
1  PRO.GCCANA IN ("+canalVentanilla+","+canalTelefono+")
```

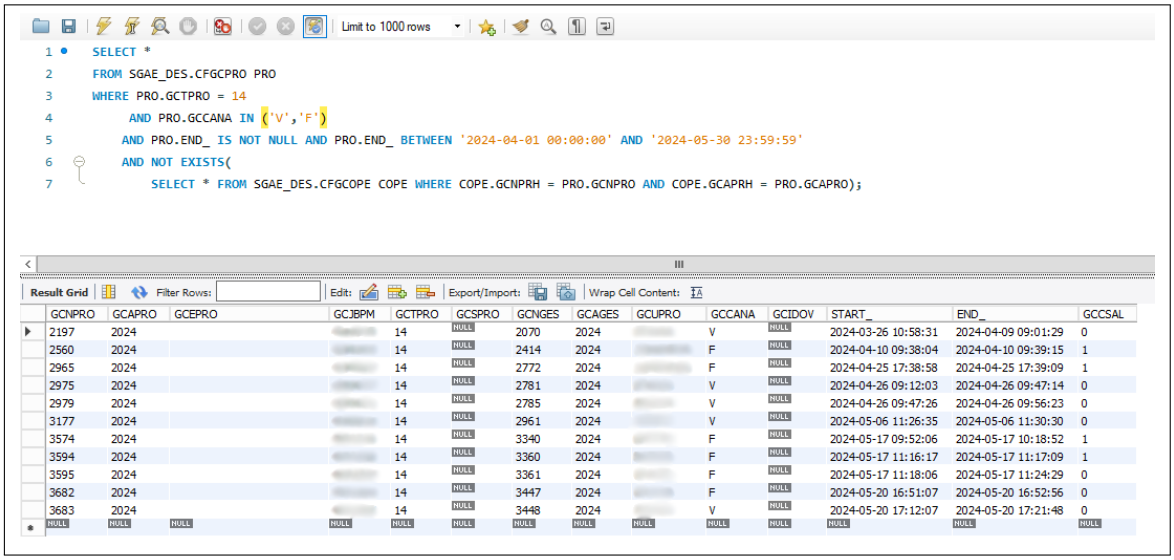
por la siguiente:

```
1  (PRO.GCCANA IN ("+canalVentanilla+") OR (PRO.GCCANA IN "+canalTelefono+")
       AND !PRO.GCCSAL))
```

5.2.5. Pruebas de las nuevas consultas SQL

Directamente desde MySQL Workbench, es posible ejecutar *querys* sobre cualquier relación.

Para *testear* las consultas modificadas, se procederá a comparar los resultados de las originales con los actuales. Por ejemplo, usando la *query* para la Función Comercial funComTipoDatoNumeroGestionesDomiciliacion, sustituyendo las variables y objetos java por ejemplos para ejecutar la *query* y modificándola para obtener todos los registros completos, se obtendrá un resultado como el siguiente:

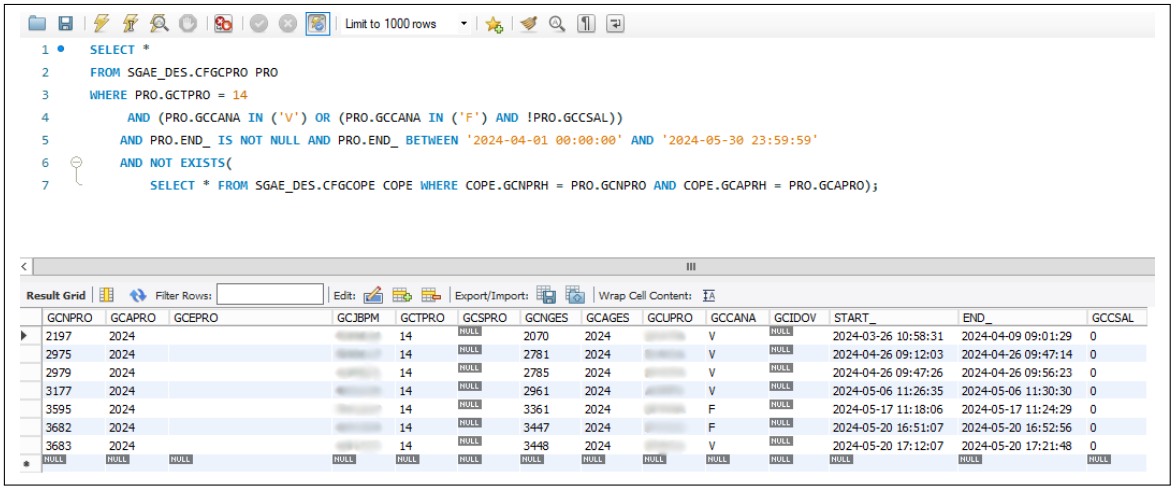


```
SELECT *
FROM SGAE_DES.CFGCPRO PRO
WHERE PRO.GCTPRO = 14
     AND PRO.GCCANA IN ('V','F')
    AND PRO.END_ IS NOT NULL AND PRO.END_ BETWEEN '2024-04-01 00:00:00' AND '2024-05-30 23:59:59'
    AND NOT EXISTS(
        SELECT * FROM SGAE_DES.CFGCOPE COPE WHERE COPE.GCNPRH = PRO.GCNPRO AND COPE.GCAPRH = PRO.GCAPRO);
```

GCNPRO	GCAPRO	GCEPRO	GCJBPM	GCTPRO	GCSPRO	GCNGES	GCAGES	GCUPRO	GCCANA	GCIDOV	START_	END_	GCCSAL
2197	2024			14	NULL	2070	2024		V	NULL	2024-03-26 10:58:31	2024-04-09 09:01:29	0
2560	2024			14	NULL	2414	2024		F	NULL	2024-04-10 09:38:04	2024-04-10 09:39:15	1
2965	2024			14	NULL	2772	2024		F	NULL	2024-04-25 17:38:58	2024-04-25 17:39:09	1
2975	2024			14	NULL	2781	2024		V	NULL	2024-04-26 09:12:03	2024-04-26 09:47:14	0
2979	2024			14	NULL	2785	2024		V	NULL	2024-04-26 09:47:26	2024-04-26 09:56:23	0
3177	2024			14	NULL	2961	2024		V	NULL	2024-05-06 11:26:35	2024-05-06 11:30:30	0
3574	2024			14	NULL	3340	2024		F	NULL	2024-05-17 09:52:06	2024-05-17 10:18:52	1
3594	2024			14	NULL	3360	2024		F	NULL	2024-05-17 11:16:17	2024-05-17 11:17:09	1
3595	2024			14	NULL	3361	2024		F	NULL	2024-05-17 11:18:06	2024-05-17 11:24:29	0
3682	2024			14	NULL	3447	2024		F	NULL	2024-05-20 16:51:07	2024-05-20 16:52:56	0
3683	2024			14	NULL	3448	2024		V	NULL	2024-05-20 17:12:07	2024-05-20 17:21:48	0

Figura 5.16: Consulta SQL de prueba con la condición original

Si en esta misma consulta, se modifica la condición de los canales para ignorar los contactos salientes, se obtiene el siguiente resultado:



```
SELECT *
FROM SGAE_DES.CFGCPRO PRO
WHERE PRO.GCTPRO = 14
     AND (PRO.GCCANA IN ('V') OR (PRO.GCCANA IN ('F') AND !PRO.GCCSAL))
    AND PRO.END_ IS NOT NULL AND PRO.END_ BETWEEN '2024-04-01 00:00:00' AND '2024-05-30 23:59:59'
    AND NOT EXISTS(
        SELECT * FROM SGAE_DES.CFGCOPE COPE WHERE COPE.GCNPRH = PRO.GCNPRO AND COPE.GCAPRH = PRO.GCAPRO);
```

GCNPRO	GCAPRO	GCEPRO	GCJBPM	GCTPRO	GCSPRO	GCNGES	GCAGES	GCUPRO	GCCANA	GCIDOV	START_	END_	GCCSAL
2197	2024			14	NULL	2070	2024		V	NULL	2024-03-26 10:58:31	2024-04-09 09:01:29	0
2975	2024			14	NULL	2781	2024		V	NULL	2024-04-26 09:12:03	2024-04-26 09:47:14	0
2979	2024			14	NULL	2785	2024		V	NULL	2024-04-26 09:47:26	2024-04-26 09:56:23	0
3177	2024			14	NULL	2961	2024		V	NULL	2024-05-06 11:26:35	2024-05-06 11:30:30	0
3595	2024			14	NULL	3361	2024		F	NULL	2024-05-17 11:18:06	2024-05-17 11:24:29	0
3682	2024			14	NULL	3447	2024		F	NULL	2024-05-20 16:51:07	2024-05-20 16:52:56	0
3683	2024			14	NULL	3448	2024		V	NULL	2024-05-20 17:12:07	2024-05-20 17:21:48	0

Figura 5.17: Consulta SQL de prueba con la condición modificada

Con esta sencilla prueba, se ha demostrado que la condición implementada funciona como es esperado, ignorando los contactos telefónicos y saliente. En este momento, ya es posible aplicar este cambio al resto de Funciones Comerciales y actualizar el proyecto *gav_exportarbbddparacm*

5.2.6. Conclusiones

Los cambios propuestos en esta solicitud han sido muy interesantes porque se han llevado a cabo el proceso completo de almacenar un objeto en la base de datos. Para ello, se han utilizado diferentes herramientas, como la aplicación MySQL Workbench o el *framework* Hibernate. Además, han surgido inconvenientes durante el desarrollo, como la complejidad de acceder a la relación CFGCCON desde CFGCPRO, por lo que ha llevado a tomas de decisiones.

5.3 Gestión del Recargo y la Devolución Bancaria de las Facturas. Corrección de comentarios

El error detectado en esta incidencia era relativamente alarmante, puesto que está relacionado con el **importe a cobrar en determinadas facturas.**

Este problema aparece en la funcionalidad de Cobro, en el momento que, una vez seleccionadas las facturas a cobrar, se desea gestionar los dos siguientes tipos de comisiones:

- **Recargo**: Gastos adicionales que reflejan que la factura no ha sido pagada antes de la fecha límite.

- **Devolución Bancaria**: Gastos adicionales que aparecen cuando la domiciliación bancaria vinculada al suministro no tenia fondos suficientes para pagar la factura.

La funcionalidad de Cobro comienza seleccionando el suministro al cual se le van a cobrar las facturas pendientes. Como casos de pruebas, se elegirán los suministros "..32/..1" y "..20/..2", ambos de la misma empresa. Al seleccionar más de un suministro con el botón *Adicionar*, ocurre que la gestión de Cobro se vincula al primero seleccionado, y por cada suministro adicional se crea una gestión hija de la gestión inicial.

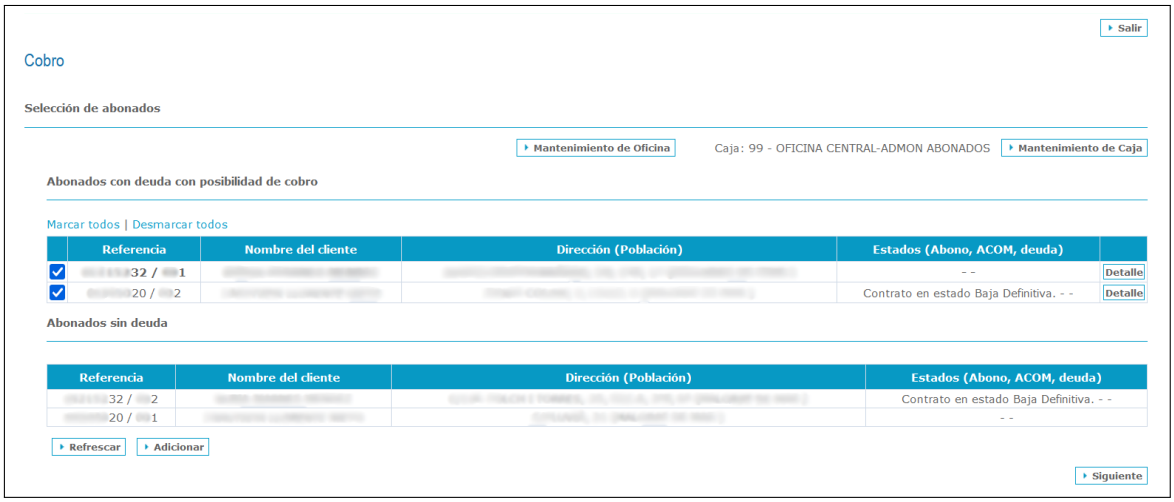


Figura 5.18: Suministros de prueba

Una vez seleccionados, se mostraran las facturas pendientes por cobrar de cada suministro, siendo el principal el representado más abajo.

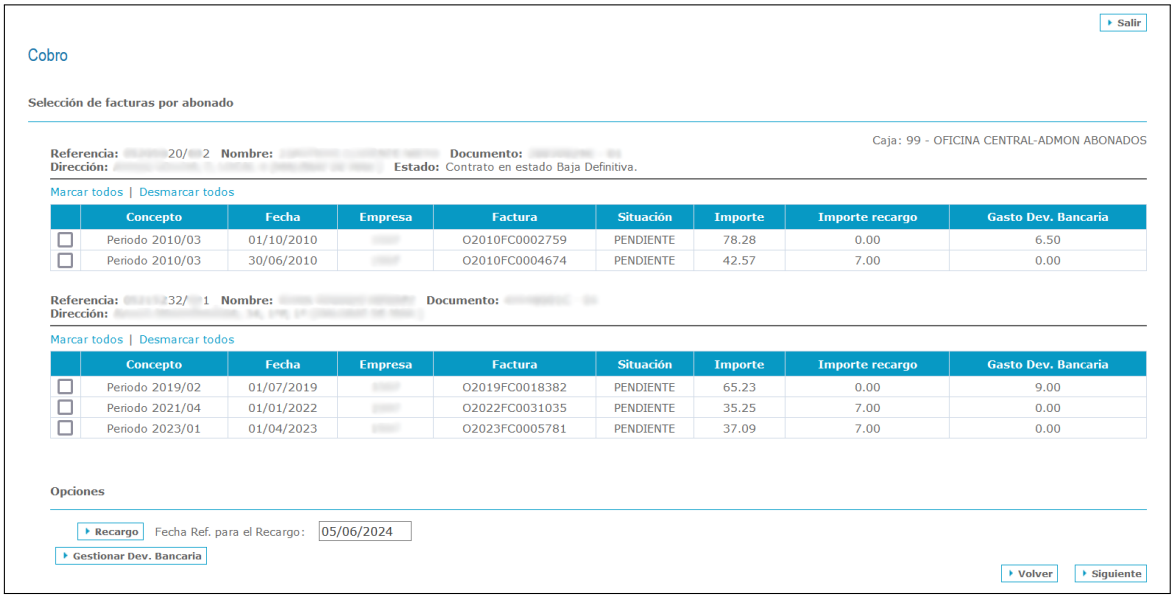


Figura 5.19: Conjunto de facturas a cobrar

En esta vista, se muestran los siguientes elementos:

- Las tablas que contienen las facturas pendientes de cobrar, todas en situación "PENDIENTE", con un Importe base y las comisiones relativas al Recargo y a la Dev. Bancaria.

- El botón ***Recargo***, cuya función es la de, **dada una fecha, mostrar el valor del recargo de la factura para dicha fecha**. El valor de la fecha de Recargo por defecto será el día en la que se esté realizando dicha gestión. Además, al utilizar esta función, el valor de los gastos de devolución bancaria **será el original**.

- El botón ***Gestionar Dev. Bancaria***, cuya función es la de eliminar los gastos de gestión bancaria de las facturas seleccionadas y asignarles el importe de recargo, si debiesen tener.

El problema reportado proviene precisamente del **algoritmo que gestiona las devoluciones bancarias**, puesto que sí elimina estos gastos, **pero no obtiene el valor correspondiente a los gastos de Recargo, devolviendo "0.00"**.

Al iniciar dicha gestión pulsando el botón *Gestión Dev. Bancaria*, se mostrará una vista como la anterior, pero mostrando únicamente las facturas a las que son posibles aplicarles la devolución de los gastos de dev. bancaria:

Figura 5.20: Selección de facturas para descartar los gastos de dev. bancaria

Al presionar el botón *Descartar*, se mostrará otra vista en la que el sistema solicita de forma obligatoria que se introduzca un motivo de facturación para proceder al descarte. Si no se escribe ningún comentario, el sistema mostrará un mensaje solicitando un valor. De forma adicional, también se muestran las facturas seleccionadas en la vista anterior.

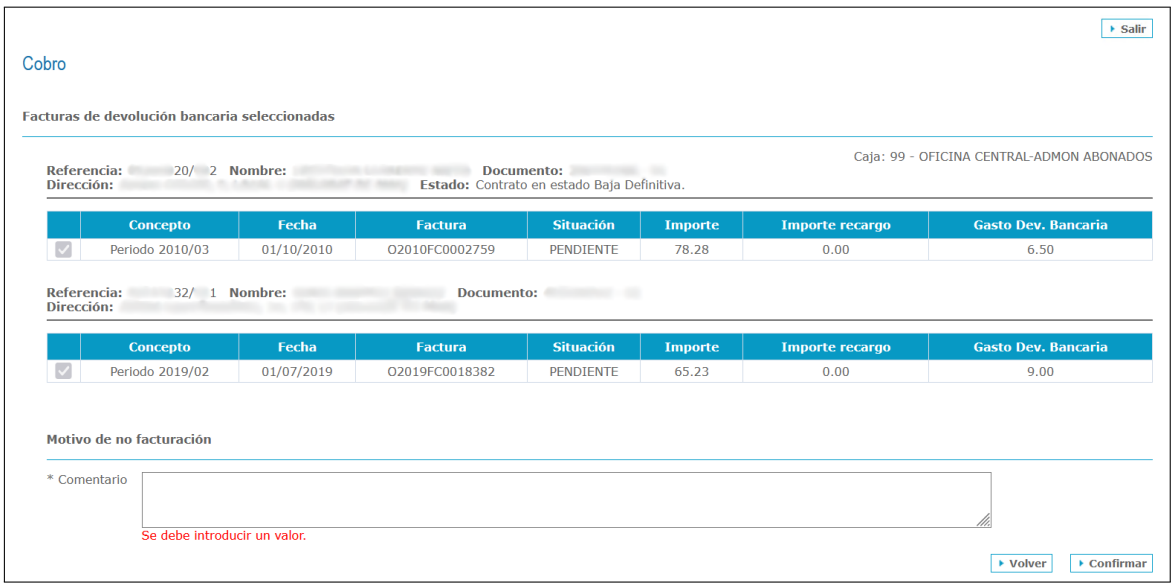


Figura 5.21: Motivo de descarte de los gastos de dev. bancaria

Al introducir un motivo y presionar el botón *Confirmar*, el sistema actualizará los datos en la tabla del conjunto de facturas a cobrar de forma incorrecta.

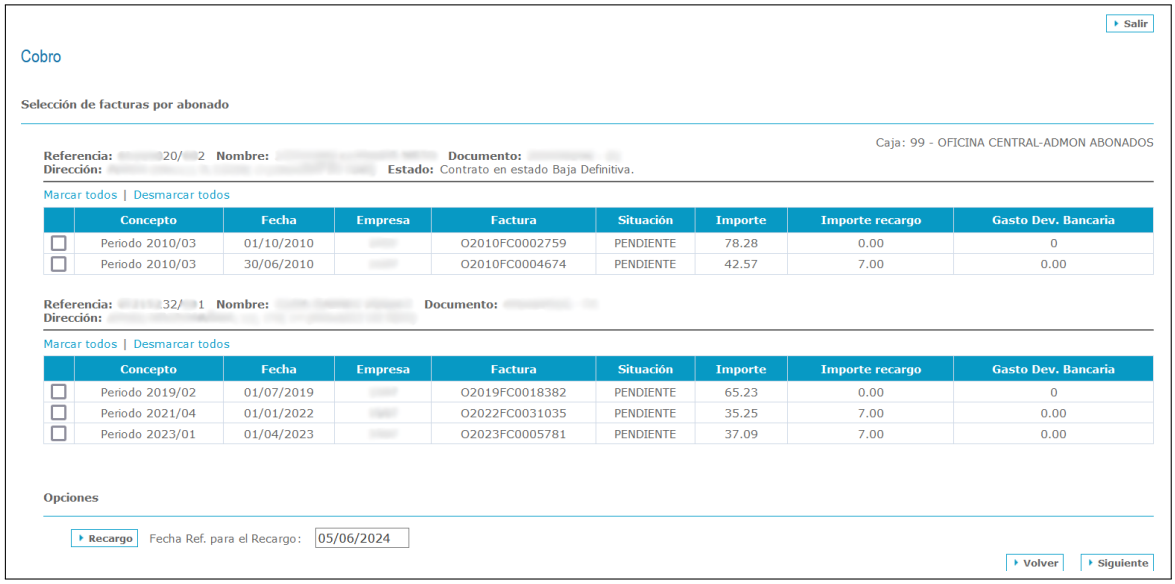


Figura 5.22: Gestión de la devolución bancaria incorrecta

Este problema se magnifica en el momento de generar las facturas en PDF, ya que al continuar con el proceso se cobro y seleccionar el botón *Documento para Cobro Ventanilla Banco* (CVB), el cual simplemente genera el PDF correspondiente, los valores que se muestran **son los que deberían mostrarse en el SAC**. Es decir, el documentos CVB sí muestra los datos reales tras la gestión de devolución bancaria.

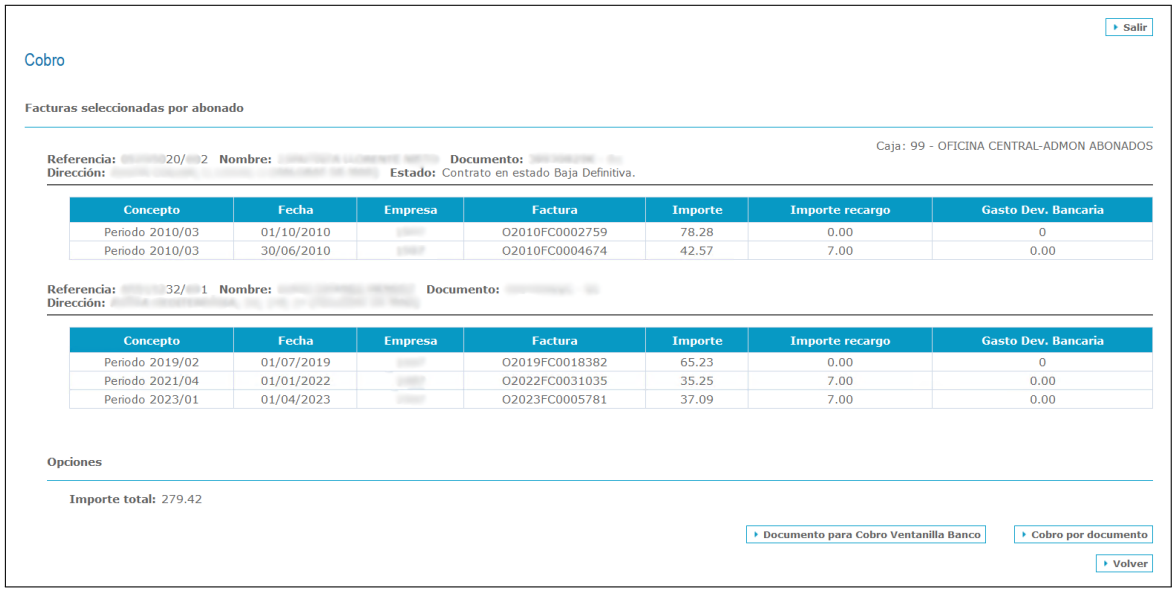


Figura 5.23: Facturas para generar el CVB

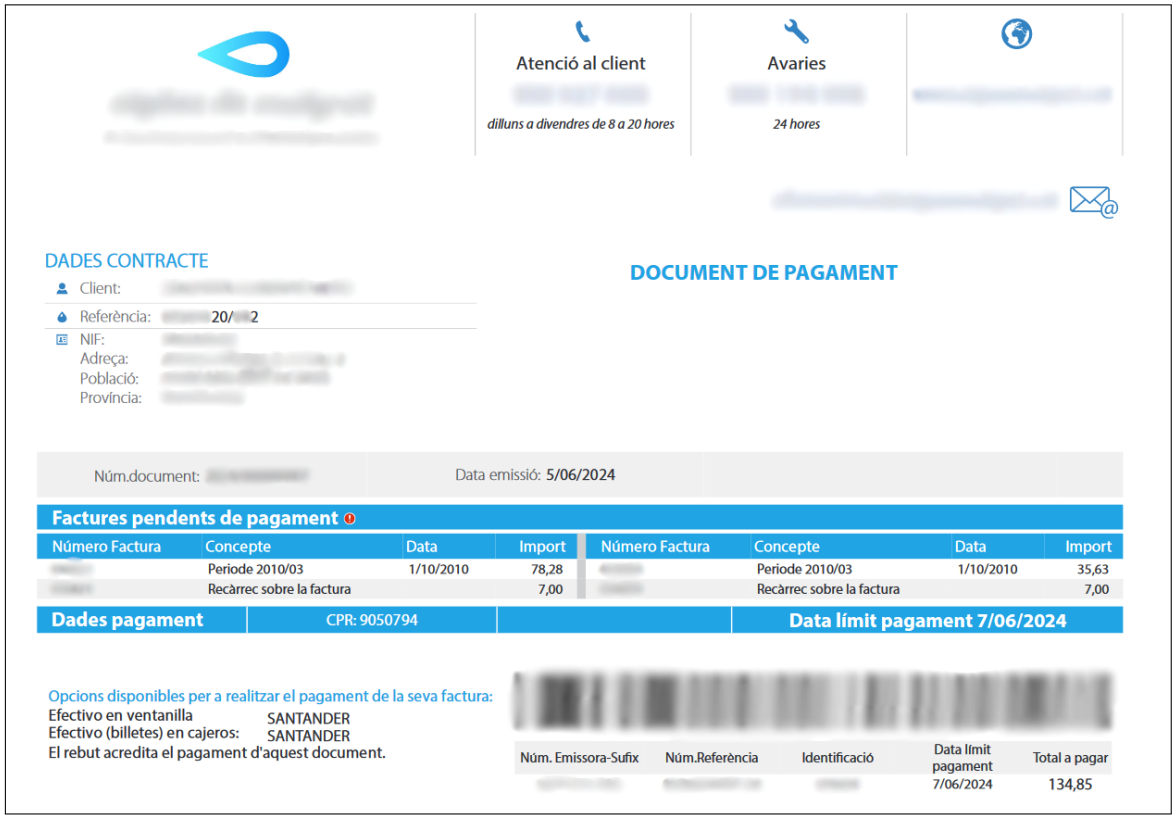


Figura 5.24: Facturas sin dev. bancarias del suministro ..20/..2

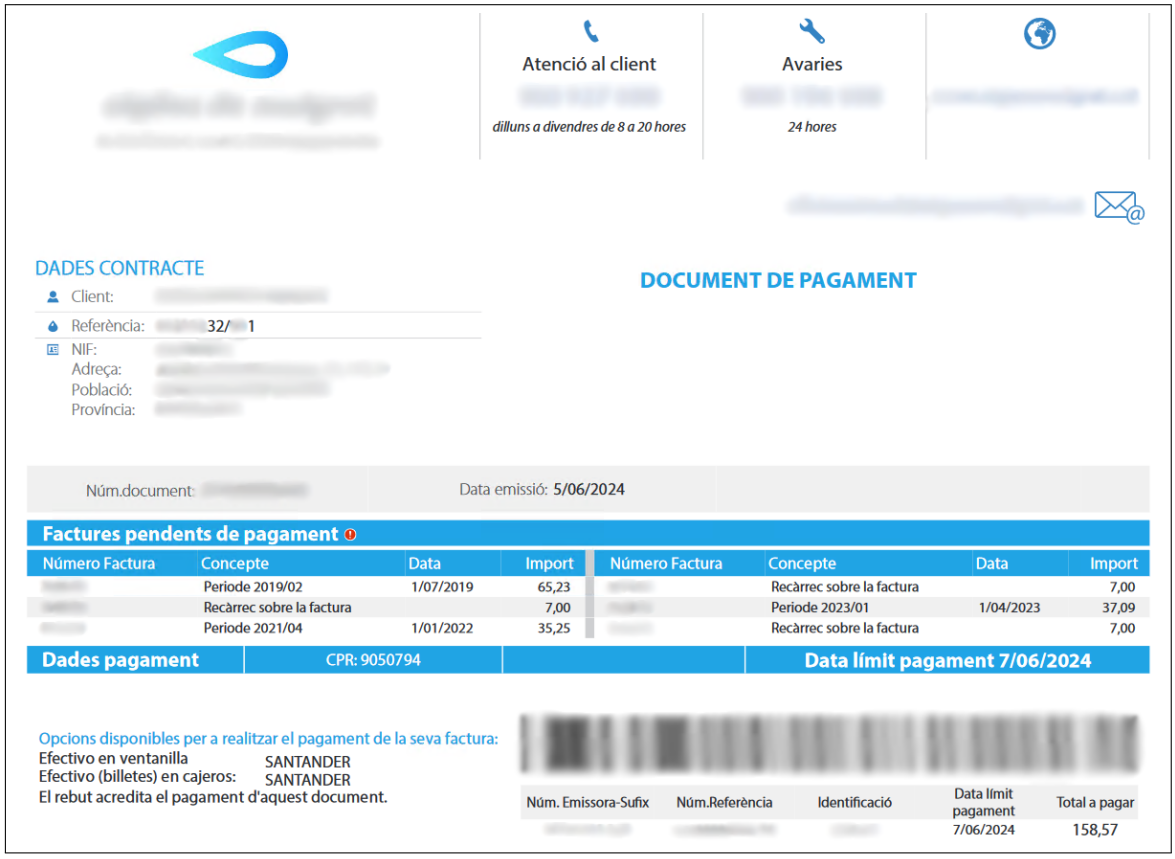


Figura 5.25: Facturas sin dev. bancarias del suministro ..32/..1

Continuando las pruebas del comportamiento del sistema en este flujo, se observan más errores cuando se **gestionan los gastos de devolución bancaria junto a la actualización del recargo**.

Si una vez descartados los gastos de devolución bancaria se decide actualizar la fecha de recargo, ya sea a la misma fecha o una anterior, los valores se van a actualizar correctamente. Sin embargo, el sistema **no detecta que debe volver a mostrar el botón *Gestionar Dev. Bancaria* previamente usado, ya que han vuelto a aparecer estos gastos**.

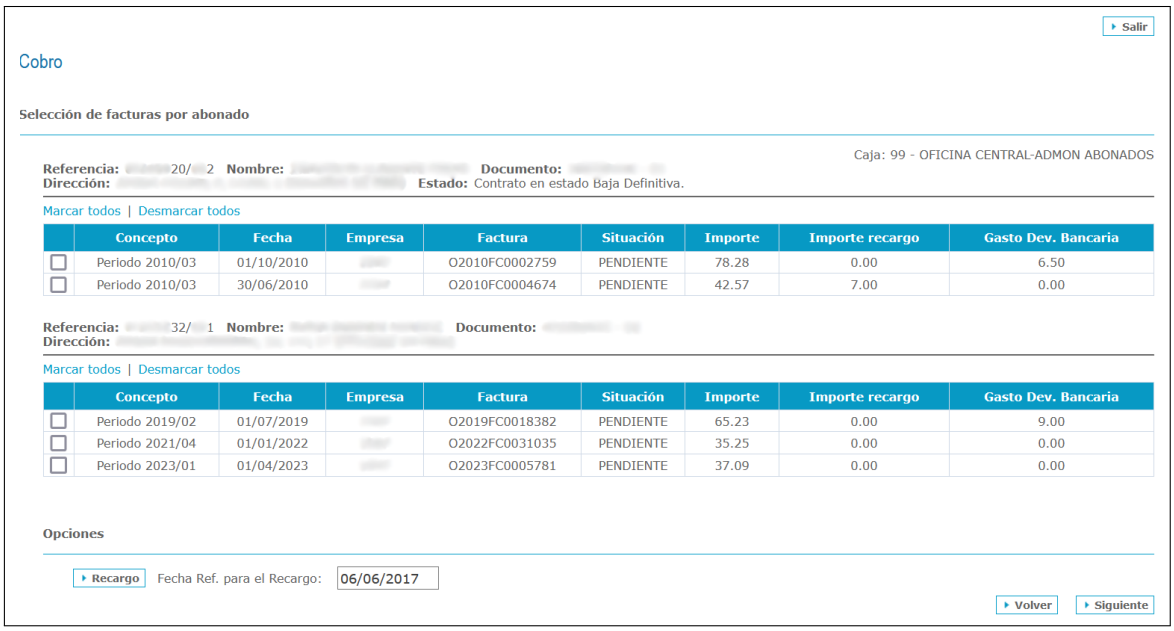
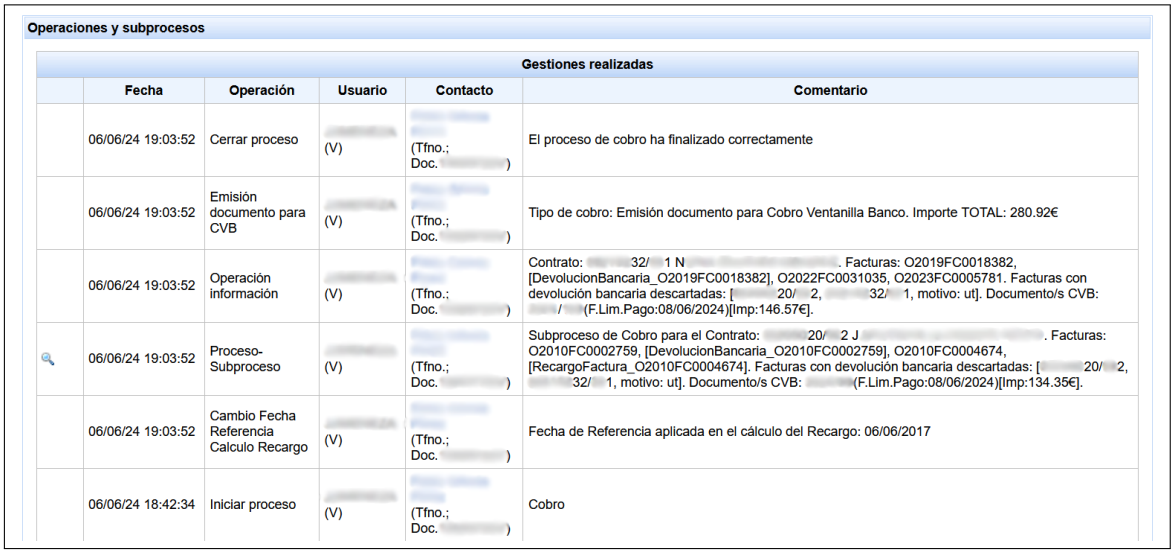


Figura 5.26: El Botón Gestionar Dev. Bancaria no reaparece

Al generar el Documento CVB vuelven a aparecer los valores esperados, que en este caso son los valores de la tabla mostrada en el SAC. Sin embargo, al acceder al registro de la gestión realizada y comprobar los procesos realizados durante el cobro, los comentarios que genera el sistema de forma automática para las gestiones que generan los documentos CVB[5] no son correctos:

[5] Aunque se genere un documento para ambos suministros, el sistema crea un documento CVB para cada suministro, siendo el PDF resultante la unión de todos los generados.



Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	06/06/24 19:03:52	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	06/06/24 19:03:52	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 280.92€
	06/06/24 19:03:52	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/1 N. Facturas: O2019FC0018382, [DevolucionBancaria_O2019FC0018382], O2022FC0031035, O2023FC0005781. Facturas con devolución bancaria descartadas: [20/2, 32/1, motivo: ut]. Documento/s CVB: /(F.Lim.Pago:08/06/2024)[Imp:146.57€].
	06/06/24 19:03:52	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/2 J. Facturas: O2010FC0002759, [DevolucionBancaria_O2010FC0002759], O2010FC0004674, [RecargoFactura_O2010FC0004674]. Facturas con devolución bancaria descartadas: [20/2, 32/1, motivo: ut]. Documento/s CVB: (F.Lim.Pago:08/06/2024)[Imp:134.35€].
	06/06/24 19:03:52	Cambio Fecha Referencia Calculo Recargo	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 06/06/2017
	06/06/24 18:42:34	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.27: No deberían aparecer comentarios de gestión de dev. bancaria

Cada párrafo de comentario en los dos procesos de generación del documento CVB se compone de las siguientes partes:

1. Si es un subproceso, se indica al principio del párrafo.
2. Datos del Contrato. Se muestra el número de referencia del suministro y el nombre completo de su titular.
3. Facturas seleccionadas para generar dicho documento. Si alguna tiene comisiones de recargo o de devolución bancaria, se indica a la derecha de cada factura.
4. Si se ha realizado alguna gestión de devolución bancaria, se indican **las facturas afectadas y el motivo de la gestión**, escrito por el gestor que está realizando el cobro.
5. Datos del propio documentos CVB generado: número de documento, fecha límite de pago e importe a pagar.

Para estos dos comentarios, se observan los siguientes errores:

- No deberían aparecer información relativa a los descartes de gastos de devolución bancaria porque **al actualizar la fecha de recargo, se anulan dichos descartes previos**.
- Dentro de los corchetes que reflejan los descartes de devolución bancaria, se muestran los suministros, cuando se **deberían mostrar los números de factura**.
- Dentro de los corchetes que reflejan los descartes de devolución bancaria, **solo debe aparecer información relacionada con el suministro al que pertenece el comentario**.

Analizando más en profundidad el comportamiento del sistema en esta funcionalidad, en concreto cuando:

- Los descartes de gatos de dev. bancaria se realizan de una en una.
- Se actualiza la fecha de recargo y posteriormente se gestionan gastos de dev. bancaria.

- Se generar el documento CVB sin seleccionar todas las facturas disponibles.

Para dichos escenarios, solo se han detectado anomalías para el último caso. En los pendientes de cobrar, cuando **se deberían mostrar solo las seleccionadas para generar el documento**.

Por ejemplo, descartando los gatos de devolución bancaria en ambas facturas de una en una, procediendo a generar el documento incluyendo solo una de las dos facturas, se muestra que se han descartado en ambas, **pero únicamente deben reflejarse para aquella utilizada en el documento CVB**:

Figura 5.28: Los comentarios solo deben contener información de las facturas utilizadas

5.3.1. Análisis

En resumen, los errores detectados son:

1. La función de gestionar la devolución bancaria actualiza los datos correctamente a la hora de cobrar las facturas, pero no durante el proceso de cobro del SAC.

2. En la función de actualizar la fecha de recargo, no comprueba si el botón *Gestionar Dev. bancaria* debe ser mostrado de nuevo.

3. En los comentarios de la gestión:

   a) Al actualizar la fecha de recargo, se deben anular los descartes de dev. bancaria previos.

   b) Dentro de los corchetes que reflejan los descartes de devolución bancaria, se muestran los suministros, cuando se deben mostrar los números de factura.

   c) Dentro de los corchetes que reflejan los descartes de devolución bancaria, solo debe aparecer información relacionada con el suministro al que pertenece el comentario.

   d) Solo se debe mostrar los comentarios de descarte de dev. bancaria para las facturas utilizadas en generar el documento CVB.

En primer lugar, el flujo de navegación que engloba todo el proceso de cobro es realmente complejo, compuesto por una gran cantidad de vistas y servicios. A pesar de ello, los posibles servicios afectados se pueden visualizar en el siguiente fragmento del grafo:

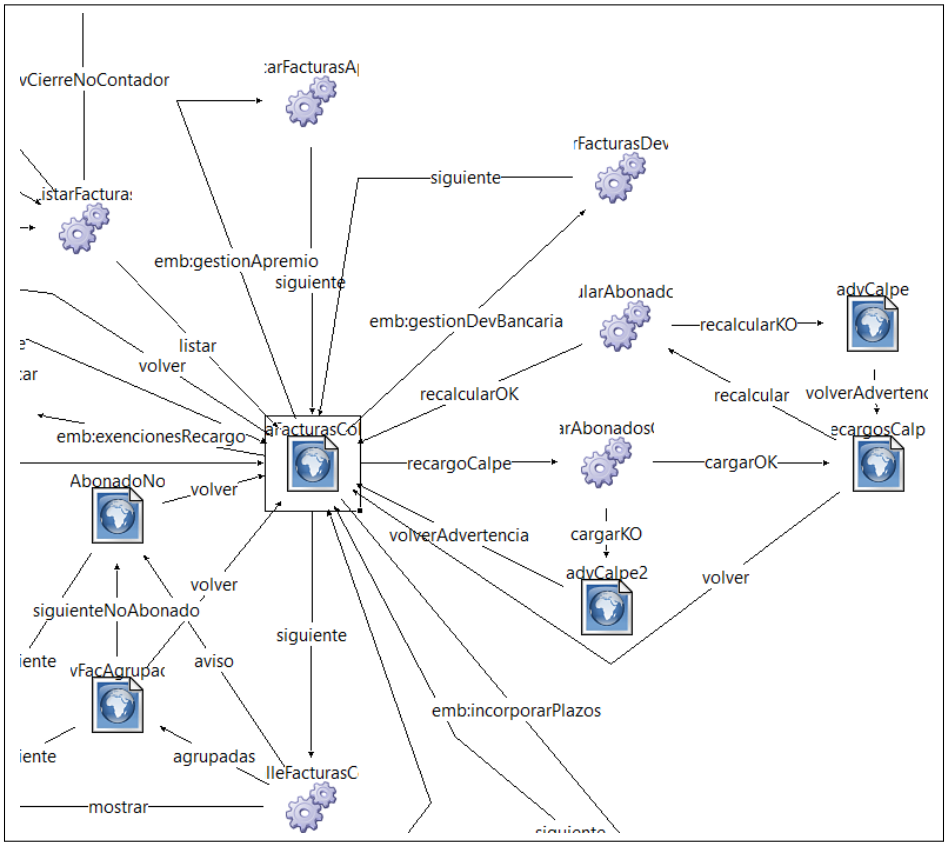


Figura 5.29: Flujo de navegación desde la vista del conjunto de facturas a cobrar

Desde la vista principal en este caso de uso, cuyo nombre es **listaFacturasCobro.jsp**, se intuye que para actualizar la fecha de Recargo, se utiliza la transición **recargoCalpe**[6], y los datos a mostrar nuevamente en la vista principal provendrá de la transición *recalcularOK*, es decir, del servicio **RecalcularRecargoCalpe**. Para la gestión de devolución bancaria, se redirige al flujo embebido **gestionDevBancaria**, y finaliza en el servicio *RefrescarFacturasDevBancarias*, el cual solo comprueba si se debe volver a mostrar el botón *Gestionar Dev. Bancaria*.

El grafo **gestionDevBancaria.xml** se corresponde con el siguiente grafo:

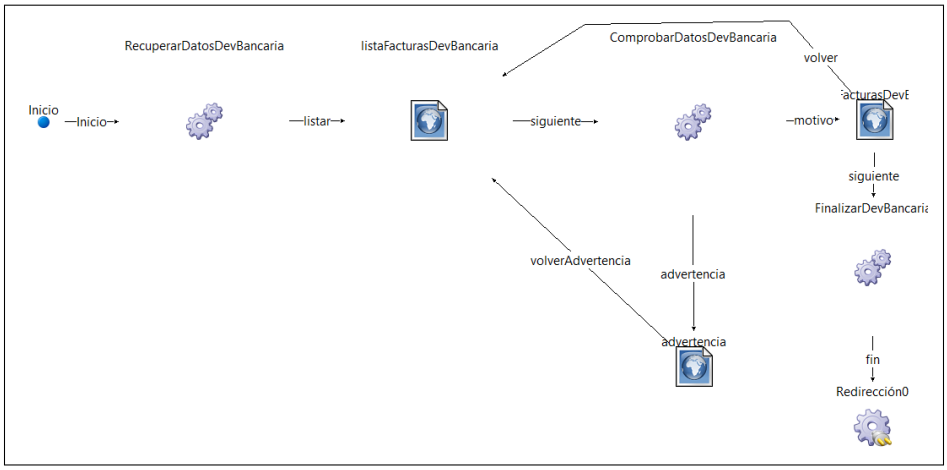


Figura 5.30: Flujo **gestionDevBancaria.xml**

[6]Pese a tener este nombre, es el servicio que se emplea para todos los municipios que engloban la aplicación

En el servicio **FinalizarDevBancaria** es donde se actualizan los valores de Devolución Bancaria, por lo que es en este algoritmo donde se debe aplicar la solución.

```
Boolean tieneFacturasDevBancariaDescartadas = (Boolean)PLContext.
    getFromFlowScope("tieneFacturasDevBancariaDescartadas");
List<AbonadoCobroListaFacturasSGA> listaAbonadoCobroListaFacturasSGA = (
    List<AbonadoCobroListaFacturasSGA>)PLContext.getFromFlowScope("
    listaAbonadoCobroListaFacturasSGABean");
List<FacturasCobroBean> listaFacturasDevBancariasNoCobradas = (List<
    FacturasCobroBean>)PLContext.getFromFlowScope("
    listaFacturasDevBancariasNoCobradas");

if (tieneFacturasDevBancariaDescartadas) {

    // Almacenamos estas facturas con su comentario para tener constancia
    de esta accion en el log
    List<FacturasDevBancariaDescartadasCobroBean>
    listaFacturasDevBancariaDescartadasCobroBean =
            (List<FacturasDevBancariaDescartadasCobroBean>)PLContext.
    getFromFlowScope("listaFacturasDevBancariaDescartadasCobroBean");
    if (listaFacturasDevBancariaDescartadasCobroBean == null)
    listaFacturasDevBancariaDescartadasCobroBean = new ArrayList<
    FacturasDevBancariaDescartadasCobroBean>();

    String motivoNoFacturacion = (String)PLContext.getFromFlowScope("
    motivoNoFacturacion");
    List<FacturasCobroBean> facturasAnuladasEnMomento = new ArrayList<
    FacturasCobroBean>();

    // Actualizar las facturas descartadas
    for (AbonadoCobroListaFacturasSGA abonadoCobroListaFacturasSGA :
    listaAbonadoCobroListaFacturasSGA) {
        if (abonadoCobroListaFacturasSGA.getTieneImporteDevBancaria()) {
            for (FacturasCobroBean facturasCobroBean :
    abonadoCobroListaFacturasSGA.getListaFacturasBean()) {
                for (FacturasCobroBean facturaDevBancariaNoCobrada :
    listaFacturasDevBancariasNoCobradas) {
                    // filtramos la busqueda para facturas de dev. bancaria
    , ya que son las unicas que se han podido ver alteradas
                    if (facturasCobroBean.getImporteDevBancaria().compareTo
    (BigDecimal.ZERO) > 0 &&
                facturasCobroBean.getEmpresa() != null &&
    facturaDevBancariaNoCobrada.getEmpresa() != null && facturasCobroBean.
    getEmpresa().equals(facturaDevBancariaNoCobrada.getEmpresa())) {
                    if (facturasCobroBean.getNia().trim().equals(
    facturaDevBancariaNoCobrada.getNia().trim()) &&
                        facturasCobroBean.getNumFactura() != null && !
    facturasCobroBean.getNumFactura().trim().isEmpty() &&
    facturaDevBancariaNoCobrada.getNumFactura() != null && !
    facturaDevBancariaNoCobrada.getNumFactura().trim().isEmpty() &&
                        facturasCobroBean.getNumFactura().trim().equals(
    facturaDevBancariaNoCobrada.getNumFactura().trim())) {
                    if (facturasCobroBean.getSubnia() != null &&
    facturaDevBancariaNoCobrada.getSubnia() != null &&
                        facturasCobroBean.getSubnia().trim().equals(
    facturaDevBancariaNoCobrada.getSubnia().trim())) {
                                    facturasCobroBean.getSubnia().trim().
    equals(facturaDevBancariaNoCobrada.getSubnia().trim())) {

                                facturasAnuladasEnMomento.add(
    facturasCobroBean);
                                facturasCobroBean.setImporteDevBancaria(
    BigDecimal.ZERO);

```

```
33                              } else if (facturasCobroBean.getSubnia() ==
    null && facturaDevBancariaNoCobrada.getSubnia() == null) {
34
35                                  facturasAnuladasEnMomento.add(
    facturasCobroBean);
36                                  facturasCobroBean.setImporteDevBancaria(
    BigDecimal.ZERO);
37                              }
38                          }
39                      }
40                  }
41              }
42          }
43      }
44
45      // Actualizar lista final de descartadas con su comentario
46      if (facturasAnuladasEnMomento.size() > 0) {
47          FacturasDevBancariaDescartadasCobroBean
    facturasDevBancariaDescartadas =
48                  new FacturasDevBancariaDescartadasCobroBean(
    facturasAnuladasEnMomento, motivoNoFacturacion);
49          listaFacturasDevBancariaDescartadasCobroBean.add(
    facturasDevBancariaDescartadas);
50          PLContext.putInFlowScope("
    listaFacturasDevBancariaDescartadasCobroBean",
    listaFacturasDevBancariaDescartadasCobroBean);
51      }
52
53  }
54
55  return "jeveris:EndFlow";
```

Listing 5.23: Código del servicio FinalizarDevBancaria

El funcionamiento de este método es el siguiente:

1. De las líneas 1 a la 15, se obtienen del flujo y se declarar las siguientes variables:
   a) tieneFacturasDevBancariaDescartadas: es *true* si alguna factura contiene gastos de dev. bancaria.
   b) listaAbonadoCobroListaFacturasSGA: lista de objetos AbonadoCobroListaFacturasSGA, los cuales representan al abonado y sus facturas pendientes de cobro. En este caso de prueba únicamente hay dos: el correspondiente al suministro ..32/..1 y ..20/..2. Por cada AbonadoCobroListaFacturasSGA, en la vista del conjunto de facturas a cobrar se mostrará una cabecera de datos y una tabla de facturas.
   c) listaFacturasDevBancariasNoCobradas: lista de FacturasCobroBean, que corresponde a cada factura seleccionada en la vista de Selección de facturas para descartar los gastos de dev. bancaria.
   d) listaFacturasDevBancariaDescartadasCobroBean: lista vacía al que se le añadirán las facturas de la listaFacturasDevBancariasNoCobradas, que junto al motivoNoFacturacion, formarán el objeto a utilizar para crear los comentarios de la gestión.
   e) motivoNoFacturacion: cadena de texto que corresponde al motivo de descarte de dev. bancaria.
   f) facturasAnuladasEnMomento: lista vacía al que se añadirán las facturas de listaFacturasDevBancariasNoCobradas, que son las que se le deben descartar la dev. bancaria.

2. De las líneas 16 a la 23, se recorren todas las facturas pendientes de cobro para cada Abonado, y se intenta reconocer si dichas facturas se han seleccionado para descartar sus gastos de dev. bancaria.

3. De las líneas 24 a la 28, si se detecta que son iguales, a la factura a descartar los gastos de dev. bancaria se copia a la lista facturasAnuladasEnMomento y se procede a asignarle a la original, el valor 0. Claramente se observa que no se actualiza el recargo.

4. De las líneas 29 a la 40 se realiza otra comprobación adicional para determinar si se está comparado la misma factura, ya que en esta lógica de negocio las facturas no son fácilmente comparables y no hay un método *equals* ya definido para este objeto.

5. De las líneas 40 a la 49, se construye la lista listaFacturasDevBancariaDescartadas-CobroBean, vinculando las facturas descartadas con su motivo de descarte. Dado los errores ya obtenidos en los comentarios de las gestiones, se deberá cambiar dicha forma de agrupar las facturas descartadas.

Por otra parte, investigando de donde provienen los datos originales de las facturas, resultan de la **invocación a un servicio del AS/400**, el cual devuelve los datos de la factura directamente para cobrar, es decir, que **si tiene gastos de dev. bancaria, se ignora el recargo**.

Esto implica que utilizando únicamente dicho servicio resulta imposible actualizar los datos en el SAC, por lo que es necesario que **en el AS/400 se genere un nuevo servicio que devuelva las facturas pero conservando tanto el recargo como la dev. bancaria**. Para su implementación, se le debería comunicar esta necesidad al departamento relacionado con el mantenimiento del AS/400.

5.3.2. Desarrollo

El departamento de mantenimiento del AS/400 nos comunicó que actualizaron el servicio ya existente para obtener las facturas de la forma deseada en este caso de uso.

En dicho servicio se utilizaba un parámetro **PRDESL** de tipo *char* que ya devolvia los datos de las facturas en función de la funcionalidad en la que se iban utilizar. En este desarrollo implementaron el posible valor G, el cual devolvía los campos de recargo y devolución bancaria originales.

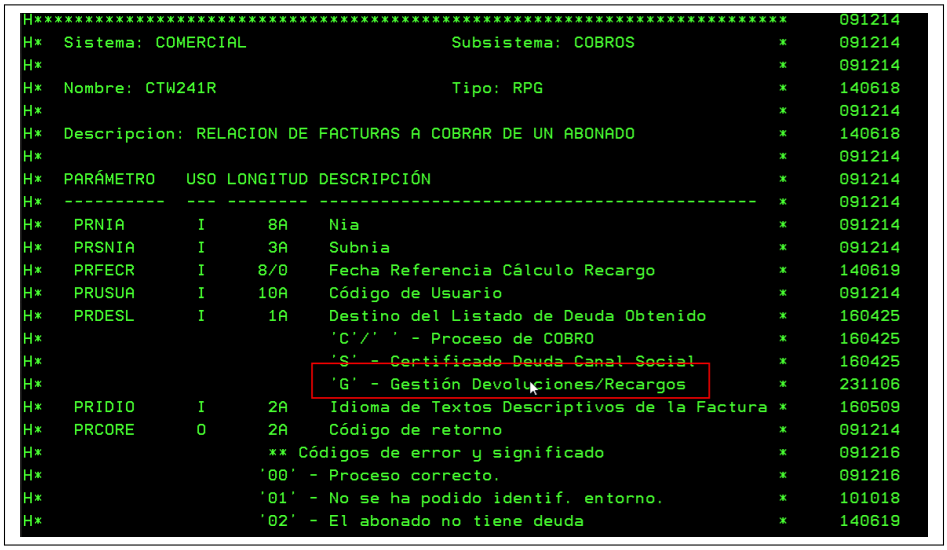


Figura 5.31: Actualización del servicio en el AS/400

Este servicio CTW241R ya estaba implementado en el SAC, el cual para ser invocado se utilizan unos servicios de más alto nivel, en el que se utilizan los parámetros deseados.

```
1  public List<FacturaCobroSGA> relacionFacturasAcobrar(String empresa, String
       nia, String subnia, Date fechaCalculoRecargo, String usuario, String
       idioma) throws AppException {
2      String destinoListadoFacturas = "C";  // Listado de Facturas para el
       Cobro
3      return relacionFacturasA(empresa, nia, subnia, fechaCalculoRecargo,
       usuario, destinoListadoFacturas, idioma);
4  }
5
6  public List<FacturaCobroSGA> relacionFacturasAcobrar(String empresa, String
       nia, String subnia, Date fechaCalculoRecargo, String usuario) throws
       AppException {
7      String destinoListadoFacturas = "C";  // Listado de Facturas para el
       Cobro
8      String idioma = null;
9      return relacionFacturasA(empresa, nia, subnia, fechaCalculoRecargo,
       usuario, destinoListadoFacturas, idioma);
10 }
11
12 public List<FacturaCobroSGA>
       relacionFacturasParaCertificadoDeudaCanalSocial(String empresa, String
       nia, String subnia, Date fechaCalculoRecargo, String usuario, String
       idioma) throws AppException {
13     String destinoListadoFacturas = "S";  // Listado de Facturas para el
       CANAL SOCIAL
14     return relacionFacturasA(empresa, nia, subnia, fechaCalculoRecargo,
       usuario, destinoListadoFacturas, idioma);
15 }
16
17 private List<FacturaCobroSGA> relacionFacturasA(String empresa, String nia,
       String subnia, Date fechaCalculoRecargo, String usuario, String
       destinoListadoFacturas, String idioma) throws AppException {
18
19     if (subnia == null) subnia = "";
20     String fechaCalculoRecargoString = AppUtils.convertirAString(
       fechaCalculoRecargo);
21
22     if (idioma == null || idioma.trim().isEmpty()) idioma = "ES";
23
```

```
24      return implCTW241R.relacionFacturasAcobrar(empresa, nia, subnia,
        fechaCalculoRecargoString, usuario, destinoListadoFacturas, idioma);
25  }
```

Listing 5.24: Métodos relacionados con CTW241R en GestionCobro.java

El método relacionFacturasA es la capa entre el SAC y el AS/400, por los que el resto de métodos del SAC, si quieren utilizar este servicio del AS/400, deben invocar a relacionFacturasA. Esto es lo que sucede en los otros tres servicios, cuyo nombres reflejan la utilidad del método, invocando relacionFacturasA con los valores deseados.

Para la implementación del nuevo método, simplemente se creará uno nuevo de forma análoga a los ya implementados. A este nuevo método se le llamará **relacionFacturasAcobrarRecargoODevBancaria**.

```
1  public List<FacturaCobroSGA> relacionFacturasAcobrarRecargoODevBancaria(
       String empresa, String nia, String subnia, Date fechaCalculoRecargo,
       String usuario) throws AppException {
2      String destinoListadoFacturas = "G";  // Listado de Facturas con datos
       de Recargo Y Dev bancaria
3      String idioma = null;
4      return relacionFacturasA(empresa, nia, subnia, fechaCalculoRecargo,
       usuario, destinoListadoFacturas, idioma);
5  }
```

Listing 5.25: Métodos relacionados con CTW241R en GestionCobro.java

Con este método, la estrategia escogida para modificar el Código del servicio FinalizarDevBancaria consiste en **obtener una copia de la variable facturasCobroBean pero con los datos originales de ambas comisiones**. Para ello, se realizarán las siguientes modificaciones:

1. Para rediseñar el primer bucle de la línea 16, se almacenará previamente el valor de *abonadoCobroListaFacturasSGA.getListaFacturasBean()* en una variable llamada **listaFacturasAbonado**. Cualquier factura de esta lista contendrá todos los datos necesarios para invocar al nuevo servicio **relacionFacturasAcobrarRecargoODevBancaria**, el cual para este caso **obtendrá una lista idéntica a listaFacturasAbonado**, con la única diferencia de que **la nueva lista contendrá los valores originales de recargo y devolución bancaria**. De esta forma, cambiando el tipo del bucle a uno con variable temporal, en cada iteración sera posible obtener, además de la factura proveniente de *abonadoCobroListaFacturasSGA.getListaFacturasBean()*, **la respectiva factura pero con los valores originales de recargo y dev. bancaria.**. A esta nueva factura se le llamará ***facturaConRecargo***.

2. El segundo bucle de la línea 19, conservará la funcionalidad, únicamente se le cambiará el tipo de bucle a uno con variable temporal para mantener la consistencia entre ambos bucles.

3. Cuando *facturasCobroBean* y *facturaConRecargo* sean la misma factura, aparte de asignarle al valor de la dev. bancaria un **0 con dos decimales**, **se le sustituirá el valor de recargo con el de facturaConRecargo**.

Con estos cambios, el fragmento de código alterado tendría el siguiente aspecto

```
1  Boolean tieneFacturasDevBancariaDescartadas = (Boolean)PLContext.
       getFromFlowScope("tieneFacturasDevBancariaDescartadas");
2  List<AbonadoCobroListaFacturasSGA> listaAbonadoCobroListaFacturasSGA = (
       List<AbonadoCobroListaFacturasSGA>)PLContext.getFromFlowScope("
       listaAbonadoCobroListaFacturasSGABean");
```

```
List<FacturasCobroBean> listaFacturasDevBancariasNoCobradas = (List<FacturasCobroBean>)PLContext.getFromFlowScope("listaFacturasDevBancariasNoCobradas");

IGestionCobro servicioGestionCobro = (IGestionCobro) Locator.getService("GestionCobroService");
String usuario = (String) PLContext.getFromFlowScope("usuario");

if (tieneFacturasDevBancariaDescartadas) {

    // Almacenamos estas facturas con su comentario para tener constancia de esta accion en el log
    List<FacturasDevBancariaDescartadasCobroBean> listaFacturasDevBancariaDescartadasCobroBean = (List<FacturasDevBancariaDescartadasCobroBean>)PLContext.getFromFlowScope("listaFacturasDevBancariaDescartadasCobroBean");
    if (listaFacturasDevBancariaDescartadasCobroBean == null) listaFacturasDevBancariaDescartadasCobroBean = new ArrayList<FacturasDevBancariaDescartadasCobroBean>();

    String motivoNoFacturacion = (String)PLContext.getFromFlowScope("motivoNoFacturacion");
    List<FacturasCobroBean> facturasAnuladasEnMomento = new ArrayList<FacturasCobroBean>();

    // Actualizar las facturas descartadas
    for (AbonadoCobroListaFacturasSGA abonadoCobroListaFacturasSGA : listaAbonadoCobroListaFacturasSGA) {
        if (abonadoCobroListaFacturasSGA.getTieneImporteDevBancaria()) {

            List<FacturasCobroBean> listaFacturasAbonado = abonadoCobroListaFacturasSGA.getListaFacturasBean();
            FacturasCobroBean facturaEjemplo = listaFacturasAbonado.get(0);
            List<FacturaCobroSGA> listaFacturasAbonadoConRecargoYDevBancaria = servicioGestionCobro.relacionFacturasAcobrarRecargoODevBancaria(facturaEjemplo.getEmpresa(), facturaEjemplo.getNia(), facturaEjemplo.getSubnia(), facturaEjemplo.getFechaCalculoRecargo(), usuario);

              for (int i = 0; i <= listaFacturasAbonado.size() - 1; i++) {
                  FacturasCobroBean facturasCobroBean = listaFacturasAbonado.get(i);
                  FacturaCobroSGA facturaConRecargo = listaFacturasAbonadoConRecargoYDevBancaria.get(i);

                  for (int j = 0; j <= listaFacturasDevBancariasNoCobradas.size() - 1; j++) {
                    FacturasCobroBean facturaDevBancariaNoCobrada = listaFacturasDevBancariasNoCobradas.get(j);
                    // filtramos la busqueda para facturas de dev. bancaria, ya que son las unicas que se han podido ver alteradas
                    if (facturasCobroBean.getImporteDevBancaria().compareTo(BigDecimal.ZERO) > 0 && ...){

                      facturasAnuladasEnMomento.add(facturasCobroBean);
                      facturasCobroBean.setImporteDevBancaria(BigDecimal.ZERO.setScale(2));
                      facturasCobroBean.setImporteRecargo(facturaConRecargo.getImporteRecargo());
                    }
                    else if (facturasCobroBean.getSubnia() == null && facturaDevBancariaNoCobrada.getSubnia() == null) {

                      facturasAnuladasEnMomento.add(facturasCobroBean);
```

```
                        facturasCobroBean.setImporteDevBancaria(BigDecimal.
    ZERO.setScale(2));
                        facturasCobroBean.setImporteRecargo(
    facturaConRecargo.getImporteRecargo());
                    }
                    ...

return "jeveris:EndFlow";
```

Listing 5.26: Código del servicio FinalizarDevBancaria modificado

Estos cambios ya solucionan el problema en la función de descarte de la dev. bancaria, por lo que si se realizan de nuevo las pruebas anteriores, descartando estos gastos en ambas facturas a la vez, el SAC muestra los datos correctos:

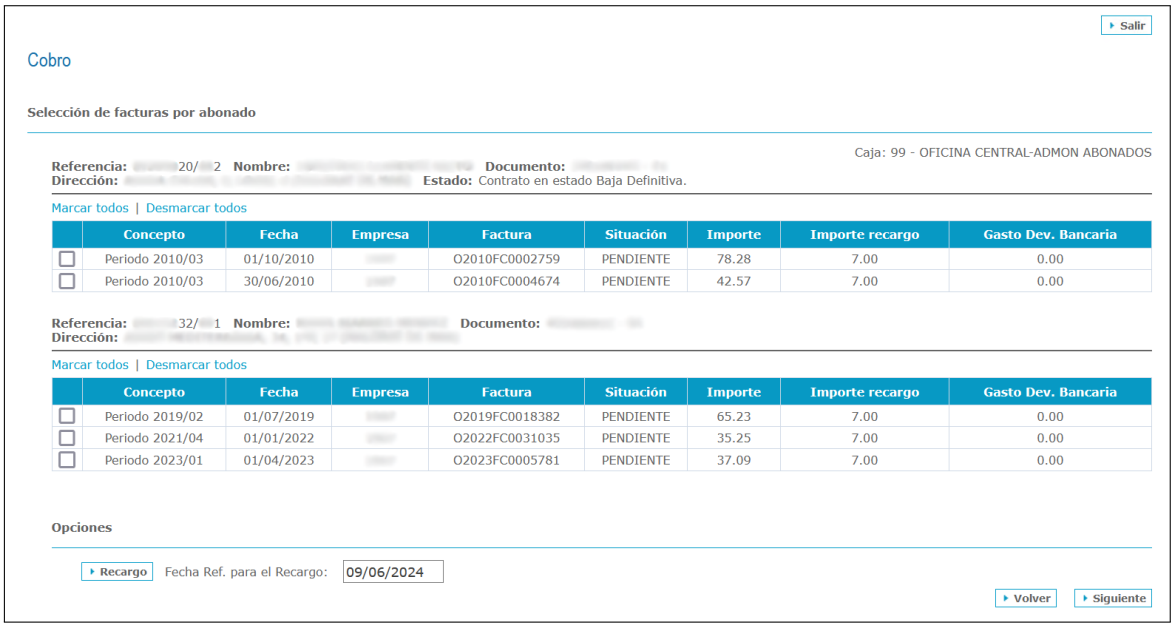


Figura 5.32: Gestión de la devolución bancaria correcta

Por otra parte, para conseguir que el botón *Gestionar Dev. Bancaria* se muestre si es necesario al actualizar la fecha de recargo, se necesitará modificar el servicio **RecalcularRecargoCalpe**, cuyo algoritmo es el siguiente:

```
SessionScope misesion = (SessionScope) PLContext.get("SessionScope");
ObjetoSesion objSesion = (ObjetoSesion)misesion.getSessionObject(
    ObjetoSesion.identificadorObjetoSesion);
String usuario = objSesion.getUsuario();

List<AbonadoRecargoSGA> listaAbonados = (List<AbonadoRecargoSGA>)PLContext.
    getFromFlowScope("listaAbonadosRecargoCalpe");
Date fechaRecargo = (Date)PLContext.getFromFlowScope("fechaRecargo");

//Solo son validas fechas no futuras
if (fechaRecargo.after(new Date())){
    PLContext.putInFlowScope("mensaje",  MessagesUtils.getMessage("common",
     "Fecha_incorrecta"));
    return "jeveris:recalcularKO";
}

IGestionCobro servicioGestionCobro = (IGestionCobro) Locator.getService("
    GestionCobroService");
//Recuperamos todos los abonados del cobro y sus facturas asociadas
```

```
List<AbonadoCobroListaFacturasSGA> listaAbonadoCobroListaFacturasSGA = (List<AbonadoCobroListaFacturasSGA>)PLContext.getFromFlowScope("listaAbonadoCobroListaFacturasSGABean");

//Refrescamos las facturas de estos abonados con la nueva fecha
for (AbonadoRecargoSGA abonadoRecargoCalpe : listaAbonados) {
    //Para cada abonado implicado se busca en el listado completo
    for (AbonadoCobroListaFacturasSGA abonadoCobroListaFacturasSGA : listaAbonadoCobroListaFacturasSGA) {
        AbonadoCobroSGA abonadoCobro = abonadoCobroListaFacturasSGA.getAbonadoCobroSGA();
        if (abonadoCobro.getNia()!= null && abonadoCobro.getNia().trim().equals(abonadoRecargoCalpe.getNia())
                && abonadoCobro.getSubnia() != null && abonadoCobro.getSubnia().trim().equals(abonadoRecargoCalpe.getSubnia())){
            //Recuperamos de nuevo las facturas del abonado con la nueva fecha para actualizar la lista en pantalla
            List<FacturaCobroSGA> listafacturasCobroSGA = new ArrayList<FacturaCobroSGA>();
            try{
                listafacturasCobroSGA = servicioGestionCobro.relacionFacturasAcobrar(abonadoRecargoCalpe.getEmpresa(), abonadoRecargoCalpe.getNia(), abonadoRecargoCalpe.getSubnia(), fechaRecargo, usuario);
            }catch (Exception ex){
                String mensajeError = ex.getMessage();
                String codigoError = "";
                if(ex.getMessage()!= null){
                    String[] mensajeServicio = ex.getMessage().split(AppUtils.SEPARADOR_EXCEPCION);
                    if(mensajeServicio != null && mensajeServicio.length > 2){
                        codigoError = mensajeServicio[1];
                        mensajeError = mensajeServicio[2];
                    }
                }
                PLContext.putInFlowScope("mensaje", MessagesUtils.getMessage("common", "Se_ha_producido_un_error") + ": " + codigoError + " " + mensajeError);
                return "jeveris:recalcularKO";
            }

            if (abonadoCobro.getSubnia() == null) abonadoCobro.setSubnia("");

            // llamamos a nuestro propio metodo para asignar el orden y seleccionar las facturas obligatorias
            List<FacturasCobroBean> miListaFacturas = servicioGestionCobro.asignarSeleccionadaOrdenFacturas(listafacturasCobroSGA, abonadoCobro);

            abonadoCobroListaFacturasSGA.setListaFacturasBean(miListaFacturas);

            //Salimos del bucle
            break;
        }
    }
}

//Refrescamos la lista del flujo
PLContext.putInFlowScope("listaAbonadoCobroListaFacturasSGABean", listaAbonadoCobroListaFacturasSGA);
```

```
60 PLContext.putInFlowScope("fechaRecargo", fechaRecargo);
61
62 return "jeveris:recalcularOK";
```

Listing 5.27: Código del servicio RecalcularRecargoCalpe

El funcionamiento del servicio es correcto, simplemente se deberá añadir un fragmento de código que compruebe **si una factura modificada vuelve a tener dev. bancaria para volver a mostrar el botón.** Para ello:

1. Se obtendrá del flujo el valor actual que determina si el botón debe ser mostrado o no, cuyo nombre es ***mostrarBotonDevBancaria***.

2. En el momento en el que se actualizan todas las facturas de un Abonado, es decir, al finalizar la condición *if* de la línea 23, se comprobará si alguna de sus factura tiene gastos de dev. bancaria. Si es así, se actualizará en el flujo y en el propio servicio para que no se vuelvan a realizar las comprobaciones para el resto de facturas.

Estos cambios implementados se pueden visualizar con la siguiente figura:

```
1  ...
2  Boolean mostrarBotonDevBancaria = (Boolean) PLContext.getFromFlowScope("
       mostrarBotonDevBancaria");
3  ...
4
5      /Para cada abonado implicado se busca en el listado completo
6    for (AbonadoCobroListaFacturasSGA abonadoCobroListaFacturasSGA :
       listaAbonadoCobroListaFacturasSGA) {
7          ...
8
9
10         //Actualizar boton Gestionar Dev Bancaria
11         if (abonadoCobroListaFacturasSGA.getTieneImporteDevBancaria() && !
       mostrarBotonDevBancaria) {
12             for (FacturasCobroBean facturasCobroDevBancaria :
       abonadoCobroListaFacturasSGA.getListaFacturasBean()) {
13                 if (facturasCobroDevBancaria.getImporteDevBancaria().
       compareTo(BigDecimal.ZERO) > 0) {
14                     mostrarBotonDevBancaria = true;
15                     PLContext.putInFlowScope("mostrarBotonDevBancaria",
       mostrarBotonDevBancaria);
16                     break;
17                 }
18             }
19         }
20     }
21
22
23 ...
24
25 return "jeveris:recalcularOK";
```

Listing 5.28: Código del servicio RecalcularRecargoCalpe modificado

Con este fragmento de código, al actualizar la fecha de recargo cuando previamente se han descartado todos los gastos de dev. bancaria, el botón *Gestionar Dev. Bancaria* volverá a mostrar si es necesario.

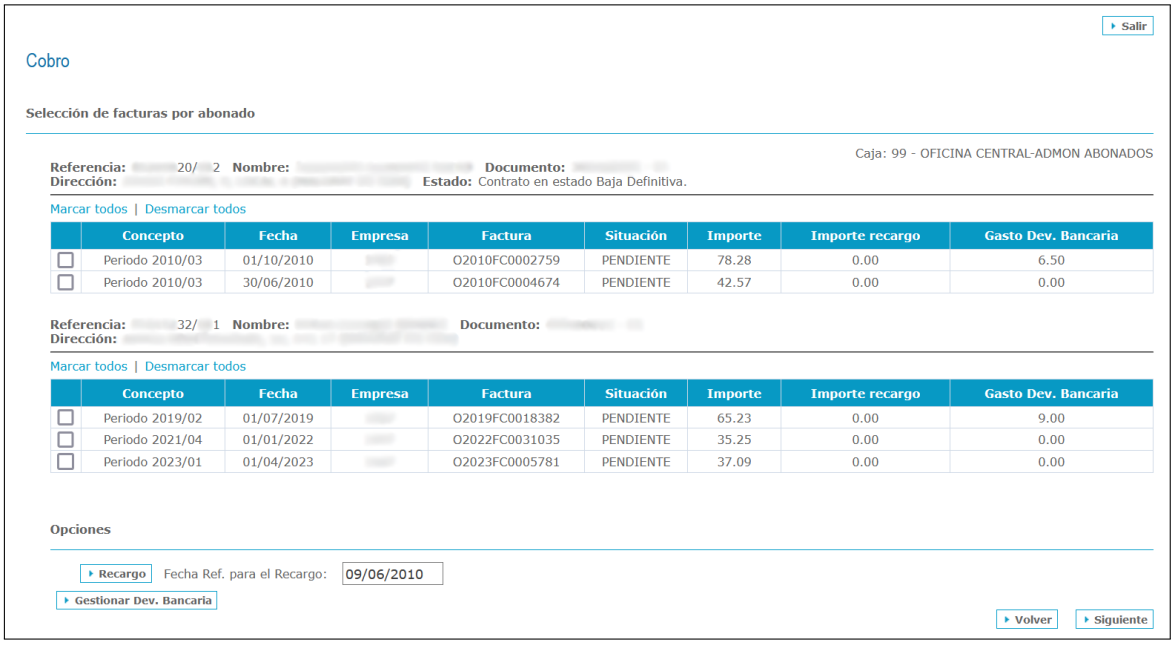


Figura 5.33: El botón *Gestionar Dev. Bancaria* vuelve a parecer

5.3.3. Corrección de los comentarios

En primer lugar, como cambio más significativo a realizar, se debe modificar la manera en que se actualiza el objeto **listaFacturasDevBancariaDescartadasCobroBean** que es utilizado al final del mismo flujo para generar los comentarios. En concreto, es usado en el servicio **Cobro**, el cual completa la gestión de cobro para cada factura seleccionada y genera los comentarios acorde a cada subproceso que realiza. El párrafo que forma el comentario consiste en una variable *comentario* de tipo **StringBuilder** que dependiendo de los procesos que se realicen, construirá progresivamente el comentario final.

En el caso de de los comentarios que reflejan las facturas a las que se le han descartado los gastos de dev. bancaria, se generan en el siguiente fragmento de código:

```
1  //Facturas con devolucion bancaria descartadas
2  List<FacturasDevBancariaDescartadasCobroBean>
       listaFacturasDevBancariaDescartadasCobroBean = (List<
       FacturasDevBancariaDescartadasCobroBean>)PLContext.getFromFlowScope("
       listaFacturasDevBancariaDescartadasCobroBean");
3  if (listaFacturasDevBancariaDescartadasCobroBean != null &&
       listaFacturasDevBancariaDescartadasCobroBean.size() > 0) {
4      comentario.append(" Facturas con devolucion bancaria descartadas: ");
5      for (FacturasDevBancariaDescartadasCobroBean
       facturasDevBancariaDescartadasCobroBean :
       listaFacturasDevBancariaDescartadasCobroBean) {
6          comentario.append("[");
7          for (int i = 0; i < facturasDevBancariaDescartadasCobroBean.
       getListaFacturasCobroBean().size(); i++) {
8              comentario.append(facturasDevBancariaDescartadasCobroBean.
       getListaFacturasCobroBean().get(i).getNia() + "/" +
       facturasDevBancariaDescartadasCobroBean.getListaFacturasCobroBean().get
       (i).getSubnia());
9              if (i < facturasDevBancariaDescartadasCobroBean.
       getListaFacturasCobroBean().size() - 1) comentario.append(", ");
10         }
11         comentario.append(", motivo: " +
       facturasDevBancariaDescartadasCobroBean.getMotivo());
12         comentario.append("].");
```

```
    }
}
```

Listing 5.29: Código que genera los comentarios de los descartes de dev. bancaria

División del comentario por suministros

Para modificar dicho objeto, se optará por la siguiente estrategia:

1. Como la variable *facturasDevBancariaDescartadas*, de tipo **FacturasDevBancariaDescartadasCobroBean**, es un objeto que se usa exclusivamente en esta función de la aplicación, **es posible modificarlo sin poner en riego otras secciones del sistema**. Por tanto, a parte de tener los parámetros correspondientes a *facturasAnuladasEnMomento* y *motivoNoFacturacion*, **se le añadirán dos más, para el Nia y el subNia del suministro**.

```
public class FacturasDevBancariaDescartadasCobroBean implements
    Serializable {
    ...

    private List<FacturasCobroBean> listaFacturasCobroBean;
    private String motivo;
    private String niaSuministro;
    private String subniaSuministro;

    public FacturasDevBancariaDescartadasCobroBean() { }

    public FacturasDevBancariaDescartadasCobroBean(List<
    FacturasCobroBean> listaFacturasCobroBean, String motivo,
            String nia, String subnia) {
        super();
        this.listaFacturasCobroBean = listaFacturasCobroBean;
        this.motivo = motivo;
        this.niaSuministro = nia;
        this.subniaSuministro = subnia;
    }
    ...
}

```

Listing 5.30: Objeto FacturasDevBancariaDescartadasCobroBean modificado

2. Con esta estrategia, *facturasAnuladasEnMomento* pasará a ser el conjunto de facturas con descarte de dev. bancaria de un mismo suministro. Esto significa antes de iniciar el proceso para las facturas de otro abonado, **esta lista se deberá vaciar**.

3. La **listaFacturasDevBancariaDescartadasCobroBean** se actualizará en el flujo antes de cambiar de vista.

Con estos cambios aplicados, el código modificado en *FinalizarDevBancaria* se puede representar de esta manera:

```
Boolean tieneFacturasDevBancariaDescartadas = (Boolean)PLContext.
    getFromFlowScope("tieneFacturasDevBancariaDescartadas");
List<AbonadoCobroListaFacturasSGA> listaAbonadoCobroListaFacturasSGA = (
    List<AbonadoCobroListaFacturasSGA>)PLContext.getFromFlowScope("
    listaAbonadoCobroListaFacturasSGABean");
List<FacturasCobroBean> listaFacturasDevBancariasNoCobradas = (List<
    FacturasCobroBean>)PLContext.getFromFlowScope("
    listaFacturasDevBancariasNoCobradas");
```

```
4
5  IGestionCobro servicioGestionCobro = (IGestionCobro) Locator.getService("
       GestionCobroService");
6  String usuario = (String) PLContext.getFromFlowScope("usuario");
7
8  if (tieneFacturasDevBancariaDescartadas) {
9
10     ...
11
12     // Actualizar las facturas descartadas
13     for (AbonadoCobroListaFacturasSGA abonadoCobroListaFacturasSGA :
       listaAbonadoCobroListaFacturasSGA) {
14         if (abonadoCobroListaFacturasSGA.getTieneImporteDevBancaria()) {
15
16             List<FacturasCobroBean> listaFacturasAbonado =
       abonadoCobroListaFacturasSGA.getListaFacturasBean();
17             ...
18
19             for (int i = 0; i <= listaFacturasAbonado.size() - 1; i++) {
20                 ...
21             }
22
23             //se acaban de actualizar todas las facturas de un abonado
24             listaFacturasDevBancariaDescartadasCobroBean.add(new
       FacturasDevBancariaDescartadasCobroBean(facturasAnuladasEnMomento,
       motivoNoFacturacion, abonadoCobroListaFacturasSGA.getAbonadoCobroSGA().
       getNia(), abonadoCobroListaFacturasSGA.getAbonadoCobroSGA().getSubnia()
       ));
25             facturasAnuladasEnMomento = new ArrayList<FacturasCobroBean>();
26
27         }
28     }
29     PLContext.putInFlowScope("listaFacturasDevBancariaDescartadasCobroBean"
       , listaFacturasDevBancariaDescartadasCobroBean);
30
31 }
32 return "jeveris:EndFlow";
```

Listing 5.31: Modificación de generar el objeto listaFacturasDevBancariaDescartadasCobroBean

La única modificación restante respecto a la variable listaFacturasDevBancariaDescartadasCobroBean, seria el tener en cuenta que **el contenido de dicha variable debe ser eliminado en el momento que se actualiza la fecha de recargo**. Para ello simplemente se debe realizar la siguiente modificación antes de tomar la transición *recalcularOK*:

```
1  ...
2
3  //Al se actualiza la tabla, se descartan los descartes de dev previamente
       ya realizados
4  PLContext.putInFlowScope("listaFacturasDevBancariaDescartadasCobroBean",
       null);
5
6  return "jeveris:recalcularOK";
```

Listing 5.32: Código del servicio RecalcularRecargoCalpe modificado

Con estos cambios implementados, ya es posible rediseñar la forma en que se construye el comentario en el servicio Cobro. Una posible solución para este desarrollo, teniendo en cuenta todas las características que deben cumplir, podría ser la siguiente:

```
1  // Facturas con devolucion bancaria descartadas
2  List<FacturasDevBancariaDescartadasCobroBean>
       listaFacturasDevBancariaDescartadasCobroBean = (List<
```

```
    FacturasDevBancariaDescartadasCobroBean>)PLContext.getFromFlowScope("
    listaFacturasDevBancariaDescartadasCobroBean");
if (listaFacturasDevBancariaDescartadasCobroBean != null &&
    listaFacturasDevBancariaDescartadasCobroBean.size() > 0) {

    for (FacturasDevBancariaDescartadasCobroBean
    facturasDevBancariaDescartadasCobroBean :
    listaFacturasDevBancariaDescartadasCobroBean) {
        if (abonadoCobro.getNia().equals(
    facturasDevBancariaDescartadasCobroBean.getNiaSuministro ()) && (
                (   (abonadoCobro.getSubnia() == null || (abonadoCobro.
    getSubnia() != null && abonadoCobro.getSubnia().trim().isEmpty())) &&
                        (facturasDevBancariaDescartadasCobroBean.
    getNiaSuministro () == null || (facturasDevBancariaDescartadasCobroBean
    .getNiaSuministro o() != null &&
    facturasDevBancariaDescartadasCobroBean.getNiaSuministro ().trim().
    isEmpty()))) ||
                (abonadoCobro.getSubnia() != null &&
    facturasDevBancariaDescartadasCobroBean.getNiaSuministro () != null &&
    abonadoCobro.getSubnia().trim().equals(
    facturasDevBancariaDescartadasCobroBean.getSubniaSuministro ().trim()))
    ) )
        {

            List<FacturasCobroBean>
    facturasDevBancariaDescartadasDelAbonado =
    facturasDevBancariaDescartadasCobroBean.getListaFacturasCobroBean();

            //Determinar si se esta cobrando alguna factura que se le ha
    descartado la devolucion bancaria
            int posicionPrimeraFacturaACobrar = -1;
            for (int i = 0; i <= facturasDevBancariaDescartadasDelAbonado.
    size()-1 && posicionPrimeraFacturaACobrar < 0; i++) {
                if (listaFacturas.contains(
    facturasDevBancariaDescartadasDelAbonado.get(i))) {
                    posicionPrimeraFacturaACobrar = i;
                }
            }

            if (posicionPrimeraFacturaACobrar >= 0) {
                comentario.append(" Facturas con devolucion bancaria
    descartadas: [");

                for (int i = posicionPrimeraFacturaACobrar; i <=
    facturasDevBancariaDescartadasDelAbonado.size() - 1; i++) {

                    if (listaFacturas.contains(
    facturasDevBancariaDescartadasDelAbonado.get(i))) {
                        comentario.append(
    facturasDevBancariaDescartadasDelAbonado.get(i).getNumFactura());
                        if (i < facturasDevBancariaDescartadasCobroBean.
    getListaFacturasCobroBean().size() - 1) {comentario.append(", ");}
                    }

                }
                comentario.append(", motivo: " +
    facturasDevBancariaDescartadasCobroBean.getMotivo() + "].");
            }


        }
    }
}
```

Listing 5.33: Código que genera los comentarios de los descartes de dev. bancaria

Con esta solución:

1. Con La condición *if* de la línea 6, se verifica que solo se vaya a incluir en el comentario las facturas vinculadas al Nia y subNia del suministro. Por tanto, **se creará un comentario por cada suministro que tenga facturas a cobrar**.

2. En el fragmento de código de las líneas 12 a la 20 se comprueba mediante el valor de la variable ***posicionPrimeraFacturaACobrar*** si alguna de las facturas a las que se han descartado los gastos de dev. bancaria, está en la lista **listaFacturas** que contiene todas las facturas para generar el documento CVB. Gracias a esta variable y a la condición *if* de la línea 22, solo se comenzará a crear el comentario **si hay como mínimo una factura con descarte de dev. bancaria que se va a cobrar**.

3. En el bucle *for* de la línea 25 sigue comprobando si existen más facturas con descarte de dev. bancaria a ser cobradas. Si sí existen, en el comentario se van enumerando **usando el numero de factura en lugar del número de referencia**.

Pruebas para cada posible escenario

Para verificar la correcta generación de los comentarios, se debe analizar de forma exhaustiva el comportamiento del sistema para verificar que los 4 errores detectados en el Análisis están corregidos en **todos los escenarios posibles**. Estos escenarios se pueden reducir a las siguientes siguientes pruebas:

1. Cobro por defecto, sin realizar descartes de dev. bancaria y sin actualizar la fecha de recargo, cobrando todas las facturas pendiente.

2. Cobro por defecto, sin realizar descartes de dev. bancaria y sin actualizar la fecha de recargo, no cobrando todas las facturas pendiente.

3. Cobro descartando gastos de dev. bancaria a la vez, cobrando todas las facturas.

4. Cobro descartando gastos de dev. bancaria de una en una, cobrando todas las facturas.

5. Cobro descartando gastos de dev. bancaria a la vez, no cobrando todas las facturas a las que se le han descartado dichos gastos.

6. Cobro descartando gastos de dev. bancaria y posteriormente actualizando la fecha de recargo, cobrando todas las facturas pendiente.

Para el primer escenario, solamente se procede a cobrar el documento CVB para todas las facturas del caso de prueba.

Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	10/06/24 19:39:24	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	10/06/24 19:39:23	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 294.92€
	10/06/24 19:39:23	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/ 1 . Facturas: O2019FC0018382, [DevolucionBancaria_O2019FC0018382], O2022FC0031035, [RecargoFactura_O2022FC0031035], O2023FC0005781, [RecargoFactura_O2023FC0005781]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:160.57€].
	10/06/24 19:39:23	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/ 2 . Facturas: O2010FC0002759, [DevolucionBancaria_O2010FC0002759], O2010FC0004674, [RecargoFactura_O2010FC0004674]. Documento/s CVB: (F.Lim.Pago:12/06/2024) [Imp:134.35€].
	10/06/24 19:39:23	Operación información	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 10/06/2024
	08/06/24 10:21:54	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.34: Prueba del escenario 1

Para el segundo escenario, solo se procederán a cobrar las facturas O2022FC0031035 y O2010FC0002759, que serán **las únicas facturas reflejadas en los comentarios**.

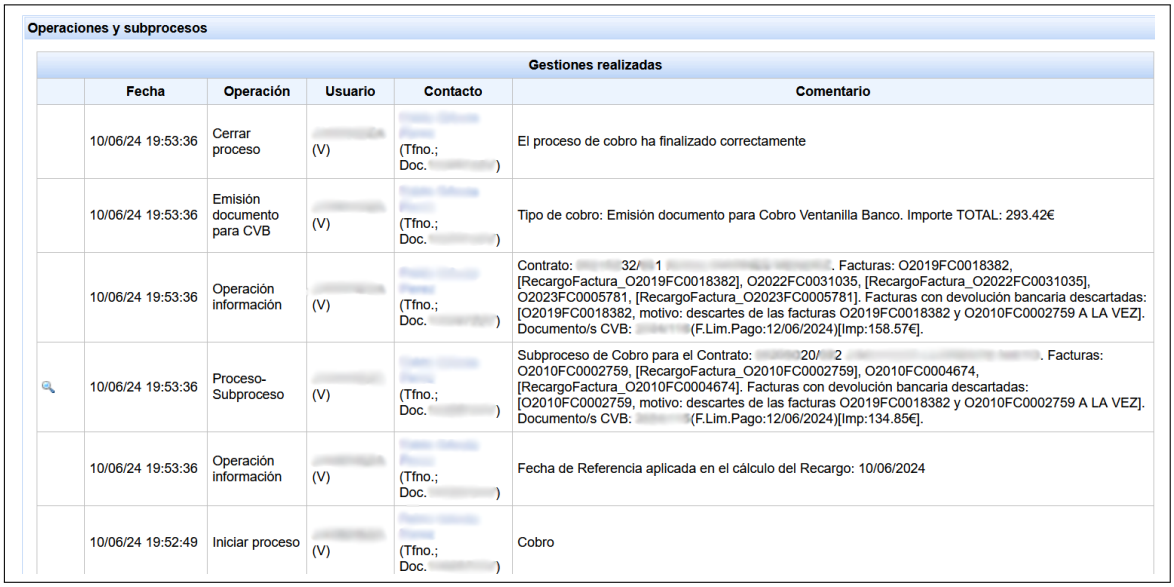
Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	10/06/24 19:44:41	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	10/06/24 19:44:41	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 127.03€
	10/06/24 19:44:41	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/ 1 . Facturas: O2022FC0031035, [RecargoFactura_O2022FC0031035]. Documento/s CVB: (F.Lim.Pago:12/06/2024) [Imp:42.25€].
	10/06/24 19:44:41	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/ 2 . Facturas: O2010FC0002759, [DevolucionBancaria_O2010FC0002759]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:84.78€].
	10/06/24 19:44:41	Operación información	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 10/06/2024
	10/06/24 19:42:08	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.35: Prueba del escenario 2

Para el tercer escenario, simplemente se descartarán todos los gatos de dev. bancaria a la vez, mostrando el comentario del descarte de dev. bancaria **únicamente las facturas del suministro correspondiente**.



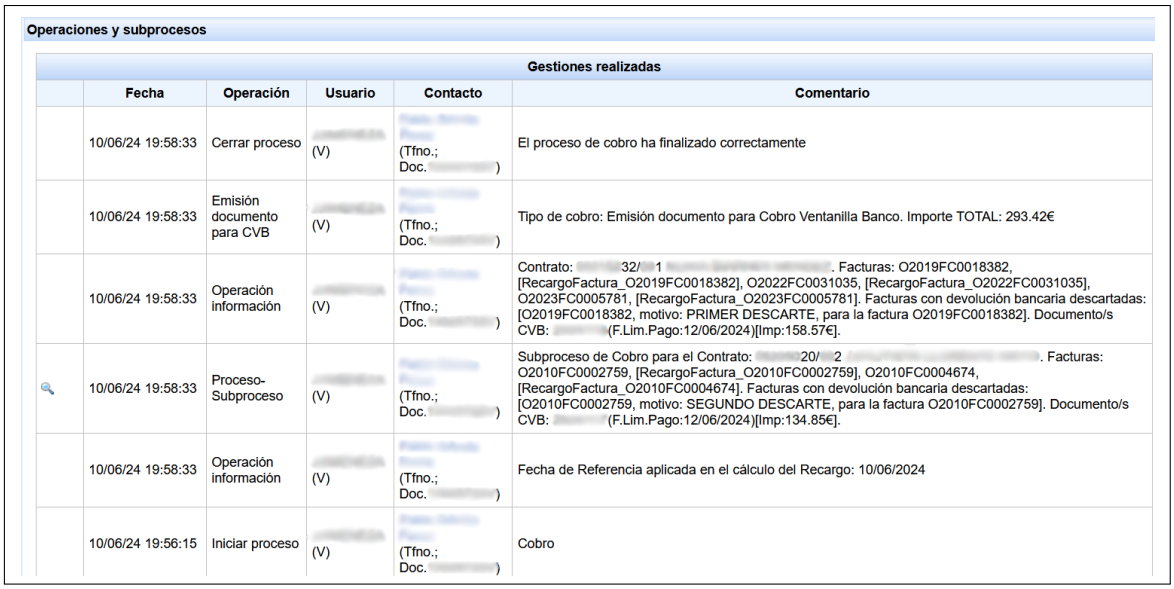
Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	10/06/24 19:53:36	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	10/06/24 19:53:36	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 293.42€
	10/06/24 19:53:36	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/1. Facturas: O2019FC0018382, [RecargoFactura_O2019FC0018382], O2022FC0031035, [RecargoFactura_O2022FC0031035], O2023FC0005781, [RecargoFactura_O2023FC0005781]. Facturas con devolución bancaria descartadas: [O2019FC0018382, motivo: descartes de las facturas O2019FC0018382 y O2010FC0002759 A LA VEZ]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:158.57€].
	10/06/24 19:53:36	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/2. Facturas: O2010FC0002759, [RecargoFactura_O2010FC0002759], O2010FC0004674, [RecargoFactura_O2010FC0004674]. Facturas con devolución bancaria descartadas: [O2010FC0002759, motivo: descartes de las facturas O2019FC0018382 y O2010FC0002759 A LA VEZ]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:134.85€].
	10/06/24 19:53:36	Operación información	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 10/06/2024
	10/06/24 19:52:49	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.36: Prueba del escenario 3

Para el cuarto escenario, se realizará como la anterior, pero cobrando descartando los gastos de una en una. Es decir, con **dos motivos diferentes**.



Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	10/06/24 19:58:33	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	10/06/24 19:58:33	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 293.42€
	10/06/24 19:58:33	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/1. Facturas: O2019FC0018382, [RecargoFactura_O2019FC0018382], O2022FC0031035, [RecargoFactura_O2022FC0031035], O2023FC0005781, [RecargoFactura_O2023FC0005781]. Facturas con devolución bancaria descartadas: [O2019FC0018382, motivo: PRIMER DESCARTE, para la factura O2019FC0018382]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:158.57€].
	10/06/24 19:58:33	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/2. Facturas: O2010FC0002759, [RecargoFactura_O2010FC0002759], O2010FC0004674, [RecargoFactura_O2010FC0004674]. Facturas con devolución bancaria descartadas: [O2010FC0002759, motivo: SEGUNDO DESCARTE, para la factura O2010FC0002759]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:134.85€].
	10/06/24 19:58:33	Operación información	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 10/06/2024
	10/06/24 19:56:15	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.37: Prueba del escenario 4

El quinto escenario, se compone de forma parcial de las pruebas de los cuatro escenarios anteriores.

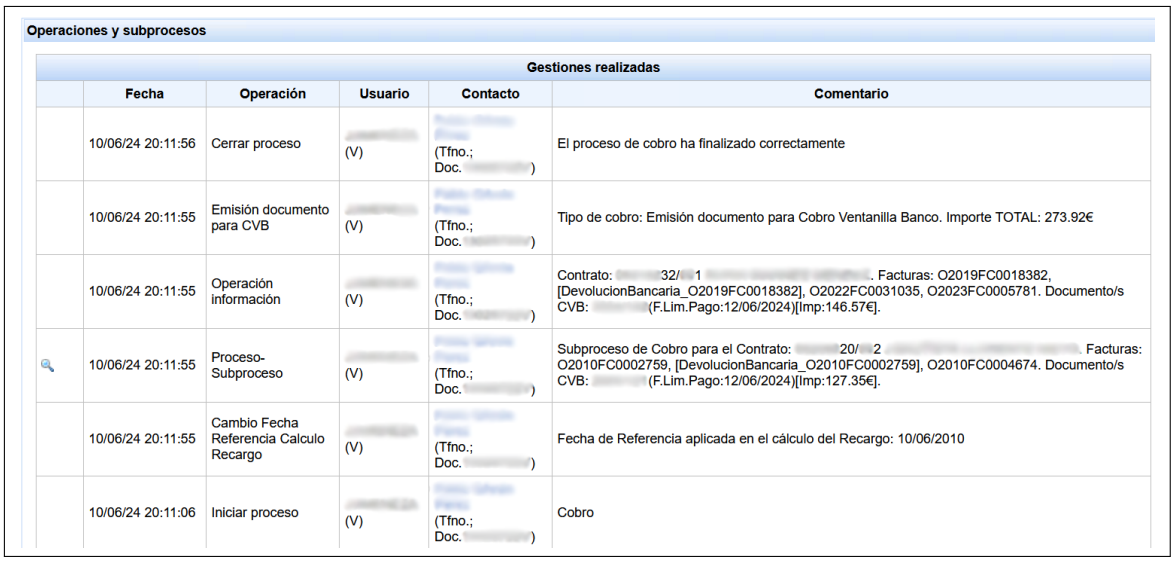
Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	10/06/24 20:08:10	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	10/06/24 20:08:09	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 171.62€
	10/06/24 20:08:09	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/1. Facturas: O2022FC0031035, [RecargoFactura_O2022FC0031035], O2023FC0005781, [RecargoFactura_O2023FC0005781]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:86.34€].
	10/06/24 20:08:09	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/2. Facturas: O2010FC0002759, [RecargoFactura_O2010FC0002759]. Facturas con devolución bancaria descartadas: [O2010FC0002759, motivo: descarte de todas]. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:85.28€].
	10/06/24 20:08:09	Operación información	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 10/06/2024
	10/06/24 20:06:56	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.38: Prueba del escenario 5

Para el ultimo escenario, se actualizará la fecha de recargo al 10/06/2010, por lo que **los comentarios de dev. bancaria deberían desaparecer**.



Operaciones y subprocesos

Gestiones realizadas

	Fecha	Operación	Usuario	Contacto	Comentario
	10/06/24 20:11:56	Cerrar proceso	(V)	(Tfno.; Doc.)	El proceso de cobro ha finalizado correctamente
	10/06/24 20:11:55	Emisión documento para CVB	(V)	(Tfno.; Doc.)	Tipo de cobro: Emisión documento para Cobro Ventanilla Banco. Importe TOTAL: 273.92€
	10/06/24 20:11:55	Operación información	(V)	(Tfno.; Doc.)	Contrato: 32/1. Facturas: O2019FC0018382, [DevolucionBancaria_O2019FC0018382], O2022FC0031035, O2023FC0005781. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:146.57€].
	10/06/24 20:11:55	Proceso-Subproceso	(V)	(Tfno.; Doc.)	Subproceso de Cobro para el Contrato: 20/2. Facturas: O2010FC0002759, [DevolucionBancaria_O2010FC0002759], O2010FC0004674. Documento/s CVB: (F.Lim.Pago:12/06/2024)[Imp:127.35€].
	10/06/24 20:11:55	Cambio Fecha Referencia Calculo Recargo	(V)	(Tfno.; Doc.)	Fecha de Referencia aplicada en el cálculo del Recargo: 10/06/2010
	10/06/24 20:11:06	Iniciar proceso	(V)	(Tfno.; Doc.)	Cobro

Figura 5.39: Prueba del escenario 6

Como se puede observar en cada imagen, en todos los escenarios de prueba se observan los resultados esperados.

5.3.4. Conclusiones

Este es el caso de uso más completo que he realizado, ya que:

1. Se han vinculado un servicio del AS/400 con el SAC

2. Se ha realizado un análisis profundo para comprender como funcionaban los algoritmos modificados, puesto que habían sufrido muchos cambios con el paso del tiempo y no era fácil su interpretación. De la misma manera ha ocurrido con las

variables utilizadas, puesto que no era intuitivo relacionarlos con los elementos visibles del SAC.

3. Se ha rediseñado completamente algoritmos y métodos ya existentes, intentando modificar la menor parte posible del resto de código para evitar posibles errores no esperados.

4. Se han encontrado fallos colaterales con los comentarios, los cuales ha requerido otro análisis con sus respectivo desarrollo y pruebas.

Además, ha sido una corrección importante en el que se ven afectadas facturas reales.

CAPÍTULO 6
Conclusiones

Todo este periodo de prácticas de empresa se ha centrado en el uso de **buenas prácticas** durante el proceso de **mantenimiento y evolución de un proyecto software**.

En primer lugar, ha sido necesario comprender toda la **infraestructura del proyecto** con el que trabajar, desde la división de **entornos de trabajo** o el ciclo DevOps relacionado con el **desarrollo y la integración continua**, hasta aprender las **herramientas y tecnologías empleadas**, como SonarQube, Jenkins, Jira o JEveris.

Por otra parte, el proceso de adaptación a la aplicación en la que he estado trabajando no ha sido fácil, puesto que la lógica de negocio en un ámbito como el de Global Omnium es **realmente compleja**, siendo este el motivo por el que los desarrollos que he realizado afectaban a secciones del código relativamente sencillas de comprender.

En cuanto a los desarrollos en el código fuente, he podido aprender cómo está estructurado un macro-proyecto que engloba tantas funcionalidades, aprendiendo a utilizar en detalle Eclipse y los diferentes *frameworks* utilizados. Para cada desarrollo ha sido imprescindible cumplimentar adecuadamente cada *checklist de desarrollo* para tener constancia de quién y cómo se modificaba el código fuente del proyecto.

Al margen del proceso de desarrollo, al **comunicarse con clientes** mediante cada Plan de Pruebas era inevitable adquirir *softskills* como la **asertividad**, **adaptabilidad del lenguaje** o la **claridad** y **precisión**, ya que en cada Plan de Pruebas nos ponemos en contacto con usuarios que no conciben la forma de interactuar con una apliación de igual forma que un desarrollador, por lo que es vital **empatizar** con la forma de entender de los usuarios.

Bibliografía

[1] Aguas de Valencia, Nuestra Organización. Información oficial en AguasdeValencia.es

[2] Global Omnium, Nuestra Organización. Información oficial en GlobalOmnium.com

[3] Emivasa, Historia. Información oficial en Emivasa.es

[4] Qué es un CRM y para qué sirve este tipo de programas. Articulo en Xakata.com

[5] Qué es un CRM. MDAnetArchivos.com

[6] Apache Maven. Artículo en GeeksforGeeks.org

[7] Flujo de trabajo de ramas de función en Git. Articulo en Atlassian.com

[8] Manual de uso para los proyectos de Global Omnium. Información sobre procedimientos, CI/CD, umbrales de calidad, etc. Material restringido.

[9] Introducción a Jenkins: ¿qué es, para qué sirve y cómo funciona? Artículo en Sentrio.io

[10] Qué es SonarQube: Verifica y analiza la calidad de tu código Artículo en Sentrio.io

[11] AS/400. Wikipedia.org

[12] AS/400 y los viejos rockeros que nunca mueren: sigue en activo décadas después de nacer, y hay demanda de profesionales. Artículo en Xakata.com

APÉNDICE A

Reflexión sobre los Objetivos de Desarrollo Sostenibles (ODS)

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS):

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**			X	
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**		X		
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**		X		
ODS 6. **Agua limpia y saneamiento.**	X			
ODS 7. **Energía asequible y no contaminante.**	X			
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**		X		
ODS 11. **Ciudades y comunidades sostenibles.**	X			
ODS 12. **Producción y consumo responsables.**	X			
ODS 13. **Acción por el clima.**	X			
ODS 14. **Vida submarina.**		X		
ODS 15. **Vida de ecosistemas terrestres.**	X			
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados:

En el desarrollo de este TFG únicamente se han desarrollado funcionalidades relacionadas con la gestión de relaciones con el cliente (CRM), cuya relación nace de la suministración de agua a un abonado. Todos los suministros, tanto nacionales como internacionales, gestionados por Global Omnium están diseñados desde hace años con tecnología vanguardista, gracias a la gran apuesta de la empresa por la digitalización de sus infraestructuras.

Esta empresa se involucra de forma directa en alcanzar todos los ODS. Sin embargo, en relación con este TFG, los Objetivos de Desarrollo Sostenible más involucrados podrían ser los siguientes:

- ODS 6. Agua limpia y saneamiento: Personalmente considero que este trabajo refleja el valor de no desaprovechar el agua y hacer un consumo eficiente del mismo, por lo que Global Omnium se apoya en esta aplicación para mejorar la eficiencia en la gestión del agua, suministrando el agua con la máxima calidad posible a todos los abonados.
- ODS 9. Industria, innovación e infraestructuras: Esta es en mi opinión la ODS más vinculada con este trabajo. Este software de CRM existe gracias a que todas las infraestructuras están digitalizadas y son monitorizarles mediante este tipo de aplicaciones.
- ODS 11. Ciudades y comunidades sostenibles: Claramente todas las ciudades y comunidades que engloba el grupo Global Omnium gozan de toda la tecnología de la empresa, permitiendo así a todas las infraestructuras un funcionamiento eficiente y mantenible a lo largo de los años.