Universidad Central “Marta Abreu” de Las Villas

EnerguX
Control de Portadores Energéticos

Tesis Presentada en Opción al Título Académico de
Máster en Computación Aplicada

Autor: Lic. Ariel Pérez Rodríguez
Tutor: Dra. Gheisa Ferreira Lorenzo

- 2010 -

RESUMEN

El trabajo consiste en la programación de un sistema informático capaz de establecer el control de los portadores energéticos utilizados en las empresas cubanas. Completamente diseñada con herramientas de código abierto, se propone una solución alternativa a una aplicación existente en la actualidad y que se beneficia de cinco años de uso por parte de los clientes de la empresa Desoft (Empresa Nacional del Software).

El desarrollo de esta solución se centra en modernas tecnologías de programación en Java que permiten dividir el desarrollo de una aplicación orientada a la Web en las tradicionales tres capas de arquitectura de desarrollo, que proveen a la vez múltiples ventajas para el desarrollador, calidad en la interfaz de usuario y facilidad de mantenimiento, utilizando las potencialidades de los marcos de trabajo existentes.

ABSTRACT

This paper is about a computer application that intends to control the usage of energetic sources in Cuban enterprises. It was completely designed and programmed using open source technologies. At the same time, it is showed the whereabouts of a new application that stands by itself as a viable substitution for a current product developed by Desoft (*Empresa Nacional del Software*), currently leading the market in this field of interest for five years now.

The entire development of this solution is based on Java, an outstanding open source language programming. The techniques described in here allows to divide application development into three traditional-and-well-defined layers bound together to easy the design of a Web application, which provides several advantages for the developer, it produces an easy-to-use user interface and advantages for future versions of this software.

CONTENIDO

INTRODUCCIÓN

Los sistemas computacionales de gestión existen desde los principios de la actual era informática. Las empresas estatales cubanas en su gran mayoría hacen uso de las tecnologías de la información, teniendo que subsistir con simples programas genéricos, tecnológicamente muy limitados para cualquier tipo de negocio, con la consiguiente pérdida de productividad debido a que en lugar de adaptar un sistema a sus procesos acaban realizando lo contrario. Un sistema de este tipo debe permitir gestionar la información de la empresa desde el inicio de los procesos hasta su finalización.

No es secreto que en Cuba se le concede una importancia vital al control de los portadores energéticos con el objetivo de planificar de forma eficiente y prioritaria la distribución del combustible necesario para el trabajo empresarial diario. La informática en el mundo cada día gana en desarrollo y en particular en Cuba. Se hacen grandes esfuerzos por llevarla a todos los niveles de la sociedad para contribuir con la economía. Con este avance, se consigue que el desarrollo del trabajo y el procesamiento de la información sean superiores en cuanto a inmediatez y fiabilidad de los resultados.

El desempeño por lograr aplicaciones más cercanas al usuario final, así como la facilidad a la hora de manipularlas, es el mayor propósito de los programadores y el soporte de las nuevas tecnologías, específicamente tecnologías basadas en la Web, más aún cuando la Web 2.0 es una realidad y toda la tecnología que trae aparejada este concepto.

La idea de controlar la energía consumida no es novedosa. Muchos son los sistemas informáticos existentes en el país. En el entorno cubano existe uno que constituye el antecesor natural del que se propone. Se trata del *Celador S2C*, solución informática comercializada y desarrollada por Desoft, en explotación desde el año 2004 y que se explota, en la actualidad, por parte de unos 80 clientes, con más de 100 licencias en todo el territorio nacional. El sistema, sobre todo, pondera su uso en el control del consumo de combustibles por tarjeta magnética, funcionalidad principal que tiene la solución. Su versión más reciente es la 3.4 de marzo del 2010.

Las principales funcionalidades del sistema son:

- Control de consumo de Combustible por Tarjetas Magnéticas, que incluye:
    - Facturación
    - Control de consumo del combustible.
    - Cambio de portador
    - Ajustes
    - Traspaso de saldo entre tarjetas
- Control de hojas de ruta, que incluye la elaboración de Hojas de Ruta por vehículos, incluyendo la introducción de los detalles de viajes de cada una de ellas y los planes de consumo mensuales por vehículos.
- Emisión de comprobantes contables para uso del departamento económico de la entidad, a partir de las fuentes de información almacenadas en el sistema.

Sin embargo, como parte de un proceso que se comenzó en todo el país en el año 2008, se hace necesario migrar los sistemas informáticos actuales a tecnologías de código abierto. Además, a pedidos de los clientes se hace necesario incorporar o mejorar algunas funcionalidades no presentes como: asociar los consumos de los vehículos a actividades propias de la empresa, la automatización del proceso de carga comenzando por la facturación de combustible, las operaciones descritas por la Resolución 60 del 2009 y la configuración de los niveles y subniveles de cuentas para la generación del comprobante contable.

Problema de Investigación:

A partir de una aplicación de escritorio, programada con herramientas de software propietario, se desea diseñar una aplicación Web -realizada en su totalidad con herramientas de código abierto utilizando marcos de trabajo- para el control de los Portadores Energéticos que posea una interfaz de usuario amigable, personalizable y

que posibilite controlar los principales objetivos de asignación y consumo de combustible por Tarjeta Magnética, que mejore e incorpore algunas de las funcionalidades actuales.

Objetivo General:

Diseñar e implementar una aplicación Web que controle el proceso de asignación y consumo de combustible por Tarjetas Magnéticas, amplíe las funcionalidades del *Celador S2C* y cumpla con los estándares de código abierto apoyándose fundamentalmente en la utilización de marcos de trabajo.

Objetivos Específicos:

1. Realizar un estudio sobre la utilización de marcos de trabajo de Java para el desarrollo de aplicaciones Web.
2. Definir los procesos de negocio fundamentales para el control de consumo de combustible.
3. Realizar un análisis de las funcionalidades del sistema existente para mejorar o incorporar características no presentes en la aplicación *Celador S2C*.
4. Definir la arquitectura de la aplicación.
5. Implementar la aplicación Web utilizando los marcos de trabajo de desarrollo de Java seleccionados.

El presente trabajo no se enmarca en el tema de solucionar el problema de una entidad específica. El valor agregado de la herramienta que se propone consiste en su generalidad y extensibilidad a una gran cantidad de empresas cubanas. Se pretende que *Energux*, la herramienta que será objeto de análisis en las siguientes páginas, sea fácilmente extensible; esté basado en software *open source* o de código abierto, ampliamente utilizado y probado, con una gran comunidad de usuarios, lo que proporciona un soporte de gran calidad; disponga de mecanismos de seguridad y gestión de permisos; promueva la separación de roles entre los desarrolladores gracias al

patrón MVC (*Model-View-Controller*) y la separación en capas y módulos funcionales, que soporta la práctica totalidad de los sistemas de gestión de bases de datos relacionales.

Se pretende que este trabajo sea referencia para futuras consultas en el sentido:

- Práctico, por permitir la implementación de una herramienta automatizada que solucionará un flujo de negocio empresarial.
- Metodológico, por instruir desde el punto de vista teórico sobre las acciones a realizar para realizar con éxito la unión y configuración de los marcos de trabajo seleccionados, que permitan llevar a buen término el desarrollo de una aplicación Web.

El trabajo de tesis está estructurado en cuatro capítulos, distribuidos de la siguiente forma:

Capítulo 1: Aplicaciones Web con Java. Marcos de trabajo.

Capítulo 2: Análisis y diseño del proceso de control de Portadores Energéticos.

Capítulo 3: Implementación del sistema.

Capítulo 4: *Energux*: Control de Portadores Energéticos.

En el primer capítulo se hace la revisión bibliográfica fundamental, especificando el uso de los marcos de trabajo utilizados en la aplicación, las razones de la selección así como sus ventajas y características generales.

El segundo capítulo describe el problema real, se procede al análisis y diseño del sistema desde el punto de vista computacional, con el uso de los diagramas propios de cada etapa del diseño de la ingeniería de software, así como la estructura de la base de datos que sirve de soporte a la gestión de la información de la aplicación.

En el tercer capítulo se detallan técnicamente las herramientas usadas y las interioridades de la programación con los marcos de trabajo en Java que proporcionaron realizar la implementación de este sistema informático.

En el último capítulo se presenta la interfaz de usuario del sistema.

Finalmente se presentan las conclusiones, recomendaciones y la bibliografía utilizada, así como un conjunto de anexos que permiten apoyar los temas desarrollados en el contenido del documento.

CAPÍTULO 1. APLICACIONES WEB CON JAVA

MARCOS DE TRABAJO

En los últimos años las aplicaciones Web han tenido gran auge gracias, en gran parte, a Internet y la proliferación de sitios web por toda la red, principalmente con el fin de fomentar el comercio electrónico. Su facilidad de administración centralizada las hace ideales tanto para su despliegue en redes de amplio alcance como en redes corporativas. La facilidad de uso de las interfaces Web y el hecho de que cada día más personas están acostumbradas a la navegación por Internet hace que el tiempo de aprendizaje se reduzca considerablemente respecto a las tradicionales aplicaciones de escritorio.

Por otra parte, es de fecha más reciente el creciente auge (en aumento acelerado) de multitud de marcos de trabajo de código abierto o libre, lo que hace que su desarrollo sea sencillo y que un gran número de desarrolladores tengan al menos una experiencia con alguno de ellos. Otro hecho a tener en cuenta es que una vez realizada una aplicación Web para uso interno de una empresa, por ejemplo en una Intranet, el poner esa funcionalidad, o incluso funcionalidades nuevas, a disposición de empleados o el público general tiene un costo mínimo a la vez que una potencial proyección mundial. (Pérez, 2009)

1.1 Aplicaciones en capas

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas de aplicaciones complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad e integración. Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa que servirá para guardar los datos (modelo), una capa para centralizar la lógica de negocio (control) y por último una interfaz gráfica que facilite al usuario el uso del sistema (presentación). (Albin, 2003)

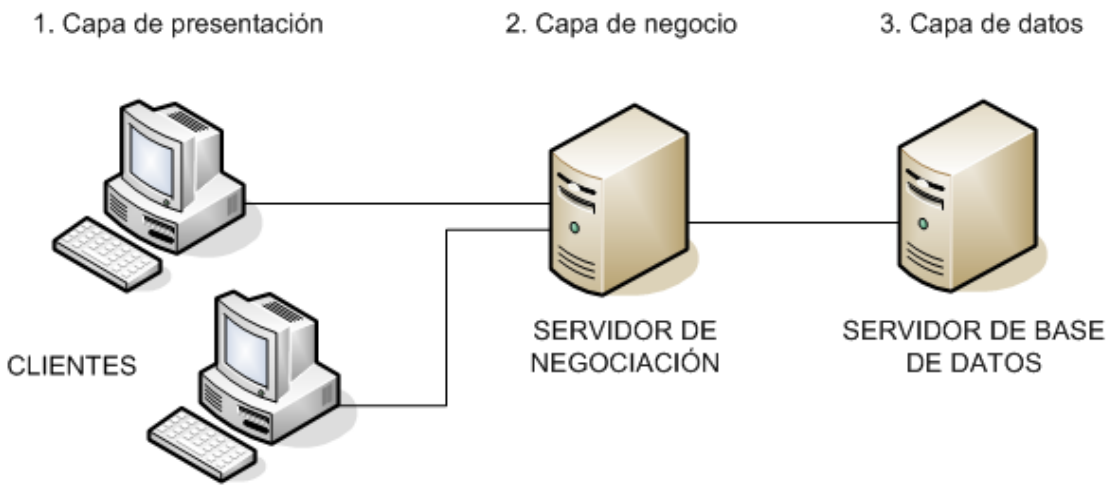

Figura 1.1. Arquitectura tradicional en tres capas.

Si se establece una separación entre la capa de interfaz gráfica (cliente), replicada en cada uno de los entornos de usuario, y la capa del modelo, que quedaría centralizada en un servidor de base de datos, según el diagrama que se muestra en la Figura 1.1, , se obtiene una potente arquitectura que otorga algunas ventajas:

- Centralización de los aspectos de seguridad y transaccionalidad, que serían responsabilidad del modelo.
- No replicación de la lógica de negocio en los clientes, lo que permite que las modificaciones y mejoras sean automáticamente aprovechadas por el conjunto de los usuarios, reduciendo los costos de mantenimiento.
- Mayor sencillez de los clientes.

La mayoría de las aplicaciones web comunes utilizan una arquitectura basada en la de tres capas extendida a las particularidades de la web.

1.2 La arquitectura MVC

La arquitectura *Model-View-Controller* surgió como patrón arquitectónico para el desarrollo de interfaces gráficas de usuario en entornos *Smalltalk*. Su concepto se basaba en separar el modelo de datos de la aplicación, de su representación de cara al usuario y

de la interacción de éste con la aplicación, mediante la división de la aplicación en tres partes fundamentales:

- El modelo, que contiene la lógica de negocio de la aplicación
- La vista, que muestra al usuario la información que éste necesita.
- El controlador, que recibe e interpreta la interacción del usuario, actuando sobre modelo y vista de manera adecuada para provocar cambios de estado en la representación interna de los datos, así como en su visualización.

Esta arquitectura ha demostrado ser muy apropiada para las aplicaciones web y especialmente adaptarse bien a las tecnologías proporcionadas por la plataforma J2EE (*Java Platform, Enterprise Edition*), de manera que:

- El modelo, que contiene la lógica de negocio, sería modelado por un conjunto de clases Java, existiendo dos claras alternativas de implementación, utilizando objetos Java tradicionales llamados POJOs (*Plain Old Java Objects*) o bien utilizando EJB (*Enterprise JavaBeans*) en sistemas con mayores necesidades de concurrencia o distribución.
- La vista proporcionará una serie de páginas web dinámicamente al cliente, siendo para él simples páginas HTML. Existen múltiples marcos de trabajo o *frameworks* que generan estas páginas web a partir de distintos formatos, siendo el más extendido el de páginas JSP (*JavaServer Pages*), que mediante un conjunto de etiquetas XML proporcionan un interfaz sencillo y adecuado a clases Java y objetos proporcionados por el servidor de aplicaciones. Esto permite que sean sencillas de desarrollar por personas con conocimientos de HTML. Entre estas etiquetas tienen mención especial la biblioteca estándar JSTL (*JavaServer Pages Standard Tag Library*) que proporciona una gran funcionalidad y versatilidad.
- El controlador en la plataforma J2EE se desarrolla mediante *servlets*, que hacen de intermediarios entre la vista y el modelo, más versátiles que los JSP para esta función al estar escritos como clases Java normales, evitando mezclar código

visual (HTML, XML...) con código Java. Para facilitar la implementación de estos *servlets* también existe una serie de marcos de trabajo que proporcionan soporte a los desarrolladores, entre los que cabe destacar *Struts*, que con una amplia comunidad de usuarios se ha convertido en el estándar *de facto* en este rol. (Camacho, 2004)

Con todo lo anterior, el funcionamiento de una aplicación web J2EE que utilice el patrón arquitectural MVC se puede descomponer en una serie de pasos:

El usuario realiza una acción en su navegador, que llega al servidor mediante una petición HTTP y es recibida por un *servlet* (controlador). Esa petición es interpretada y se transforma en la ejecución de código java que delegará al modelo la ejecución de una acción de éste.

1. El modelo recibe las peticiones del controlador, a través de un interfaz o fachada que encapsulará y ocultará la complejidad del modelo al controlador. El resultado de esa petición será devuelto al controlador.
2. El controlador recibe del modelo el resultado, y en función de éste, selecciona la vista que será mostrada al usuario, y le proporcionará los datos recibidos del modelo y otros datos necesarios para su transformación a HTML. Una vez hecho esto, el control pasa a la vista para la realización de esa transformación.
3. En la vista se realiza la transformación tras recibir los datos del controlador, elaborando la respuesta HTML adecuada para que el usuario la visualice.

Esta arquitectura de aplicaciones otorga varias ventajas clave al desarrollo de aplicaciones web, destacando que:

- Al separar de manera clara la lógica de negocio (modelo) de la vista permite la reusabilidad del modelo, de modo que la misma implementación de la lógica de negocio que maneja una aplicación pueda ser usado en otras aplicaciones, sean éstas web o no.

- Permite una sencilla división de roles, dejando que sean diseñadores gráficos sin conocimientos de programación o desarrollo de aplicaciones los que se encarguen de la realización de la capa vista, sin necesidad de mezclar código Java entre el código visual que desarrollen (tan sólo utilizando algunos *tags*, no muy diferentes de los usados en el código HTML). (Sanchez, 2004)

1.3 La plataforma Java

Java es el nombre de un entorno o plataforma de computación originaria de *Sun Microsystems*, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a código intermedio y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de bibliotecas estándares que ofrecen funcionalidad común.

La plataforma así llamada incluye:

- Edición Estándar (*Java Platform, Standard Edition*), o Java SE (antes J2SE)
- Plataforma Java, Edición Empresa (*Java Platform, Enterprise Edition*), o Java EE (antes J2EE)
- Plataforma Java, Edición Micro (*Java Platform, Micro Edition*), o Java ME (antes J2ME)

Desde 2006, la versión actual de la Plataforma *Java Standard Edition* se le conoce como Java SE 6 como versión externa, y 1.6 como versión interna. Sin embargo, se prefiere el término versión 6. (Pérez, 2009)

La Plataforma Java se compone de un amplio abanico de tecnologías, cada una de las cuales ofrece una parte del complejo de desarrollo o del entorno de ejecución en tiempo real. Por ejemplo, los usuarios finales suelen interactuar con la máquina virtual de Java y el conjunto estándar de bibliotecas. Además, las aplicaciones Java pueden usarse de forma variada, como por ejemplo ser incrustadas en una página Web. Para el desarrollo

de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (*Java Development Kit*, o herramientas de desarrollo para Java). (Froufe, 1997)

1.3.1 Java Runtime Environment

Un programa destinado a la Plataforma Java necesita dos componentes en el sistema donde se va a ejecutar: una máquina virtual de Java (*Java Virtual Machine, JVM*), y un conjunto de bibliotecas para proporcionar los servicios que pueda necesitar la aplicación. La JVM que proporciona Sun Microsystems, junto con su implementación de las bibliotecas estándares, se conocen como *Java Runtime Environment* (JRE) o Entorno en tiempo de ejecución para Java. El JRE es lo mínimo que debe contener un sistema para poder ejecutar una aplicación Java sobre el mismo.

En el concepto de máquina virtual se encierra el concepto común de un procesador "virtual" que ejecuta programas escritos en el lenguaje de programación Java. En concreto, ejecuta el código resultante de la compilación del código fuente, conocido como bytecode. Este "procesador" es la máquina virtual de Java o JVM, que se encarga de traducir (interpretar o compilar al vuelo) el bytecode en instrucciones nativas de la plataforma destino. Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas con arquitecturas distintas, siempre que con una implementación adecuada de la JVM. Este hecho es lo que ha dado lugar a la famosa frase: "*write once, run anywhere*" (escriba una vez, ejecute en cualquier parte). La condición es que no se utilicen llamadas nativas o funciones especificas de una plataforma y aún así no se asegura completamente que se cumpla una verdadera independencia de plataforma. (Pérez, 2009)

Desde la versión 1.2 de JRE, la implementación de la máquina virtual de Sun incluye un compilador JIT (Just In Time). De esta forma, en vez de la tradicional interpretación del código bytecode, que da lugar a una ejecución lenta de las aplicaciones, el JIT convierte el bytecode a código nativo de la plataforma destino. Esta segunda compilación del código penaliza en cuanto a tiempo, pero el código nativo resultante se ejecuta de forma más eficaz y rápida que si fuera interpretado. Otras técnicas de compilación dinámica

del código durante el tiempo de ejecución permiten optimizar más aún el código, dejando atrás el estigma que caía sobre Java en cuanto a su lentitud y en sus últimas versiones la JVM se ha optimizado a tal punto que ya no se considera una plataforma lenta en cuanto a ejecución de aplicaciones.

Java no fue la primera plataforma basada en el concepto de una máquina virtual, aunque es la que de más amplia difusión ha gozado. El empleo de máquinas virtuales se había centrado principalmente en el uso de emuladores para ayudar al desarrollo de hardware en construcción o sistemas operativos, pero la JVM fue diseñada para ser implementada completamente en software, y al mismo tiempo hacer que fuera portable a todo tipo de hardware. (Campo, 1999)

1.3.2 Bibliotecas de Java

En la mayoría de los sistemas operativos actuales, se ofrece una cantidad de código para simplificar la tarea de programación. Este código toma la forma, normalmente, de un conjunto de bibliotecas dinámicas que las aplicaciones pueden llamar cuando lo necesiten. Pero la plataforma Java está pensada para ser independiente del sistema operativo subyacente, por lo que las aplicaciones no pueden apoyarse en funciones dependientes de cada sistema en concreto. Lo que hace la plataforma Java, es ofrecer un conjunto de bibliotecas estándares, que contiene mucha de las funciones reutilizables disponibles en los sistemas operativos actuales.

Las bibliotecas de Java tienen tres propósitos dentro de la plataforma Java. Al igual que otras bibliotecas estándares, ofrecen al programador un conjunto bien definido de funciones para realizar tareas comunes, como manejar listas de elementos u operar de forma sofisticada sobre cadenas de caracteres. Además, las bibliotecas proporcionan una interfaz abstracta para tareas que son altamente dependientes del hardware de la plataforma destino y de su sistema operativo. Tareas tales como manejo de las funciones de red o acceso a ficheros, suelen depender fuertemente de la funcionalidad nativa de la plataforma destino. En el caso concreto anterior, las bibliotecas java.net y java.io implementan el código nativo internamente, y ofrecen una interfaz estándar para que

aplicaciones Java puedan ejecutar tales funciones. Finalmente, no todas las plataformas soportan todas las funciones que una aplicación Java espera. En estos casos, las bibliotecas bien pueden emular esas funciones usando lo que esté disponible, o bien ofrecer un mecanismo para comprobar si una funcionalidad concreta está presente.

1.3.3 Lenguaje de programación Java

La palabra Java, por sí misma, se refiere habitualmente al lenguaje de programación Java, que fue diseñado para usar con la Plataforma Java. Los lenguajes de programación se encuentran fuera del ámbito de lo que es una "plataforma", aunque el lenguaje de programación Java es uno de los componentes fundamentales de la propia plataforma. El propio lenguaje y el entorno en tiempo de ejecución suelen considerarse una única entidad.

1.4 Marcos de trabajo en Java

Lo novedoso de las tecnologías usadas es uno de los puntos fuertes de este proyecto. Todas ellas son tecnologías Java de código abierto. El aprendizaje y familiarización ha ocupado la mayor parte del tiempo del proyecto. A continuación se describen los marcos de trabajo disponibles para la implementación de las diferentes capas.

1.4.1 Capa de datos

Los marcos de trabajo disponibles para Java para la capa de datos basan su implementación en el mapeo objeto-relacional, en inglés ORM (*Object-Relational Mapping*) que es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos, básicamente herencia y polimorfismo. (Wikipedia, 2007)

Hibernate

Hibernate es un potente mapeador objeto/relacional y servicio de consultas para Java. Es la solución ORM más popular en el mundo Java. Permite desarrollar clases persistentes a partir de clases comunes, incluyendo asociación, herencia, polimorfismo, composición y colecciones de objetos. El lenguaje de consultas de HQL (*Hibernate Query Language*), diseñado como una mínima extensión orientada a objetos de SQL, proporciona un puente elegante entre los mundos objetual y relacional. *Hibernate* también permite expresar consultas utilizando SQL nativo o consultas basadas en criterios. Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones *standalone*. Posee un alto rendimiento, tiene una caché de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa (*lazy*) de objetos y colecciones y soporta la generación automática de llaves primarias. (Wikipedia, 2007)

TopLink

Es un marco de trabajo ORM que almacena objetos Java en una base de datos relacional o convierte objetos Java a documentos XML. Está liberado con la licencia *Oracle Licence* y dentro de sus características fundamentales están: consultas que soportan *Query by Example* (QBE), EJB QL, SQL, y procedimientos almacenados, transacciones a nivel de objeto, cache avanzado que asegura la identidad de los objetos, mapeo de objetos a XML, soporte para fuentes de datos no relacionales y un editor visual. La última versión es del 2007.

Apache Cayenne

Es un marco de trabajo con licencia Apache, que proporciona mapeo objeto-relacional y servicios de interacción remota. *Cayenne* une uno o más esquemas de base de datos directamente a los objetos Java. Soporta generación de SQL, relaciones y secuencias.

La persistencia de los objetos se puede conservar en los clientes a través de *Web Services* o con serialización nativa en XML. Soporta la ingeniería inversa de bases de datos y la generación de código. Todas estas funciones pueden ser controladas directamente a través del *Cayenne Modeler*, una herramienta visual completamente funcional. No se requiere de archivos XML o de anotaciones para lograr configurarlo.

Tiene otras características que incluye el almacenamiento en caché, una completa sintaxis de consulta de objetos, la herencia de objetos, detección automática de bases de datos y objetos persistentes genéricos. Lo más importante es que *Cayenne* puede ser escalable a prácticamente cualquier tipo de proyecto, ya sea grande o pequeño.

iBATIS

Está basado en capas y es desarrollado por *Apache Software Foundation*. Se ocupa de la capa de persistencia y se sitúa entre la lógica de negocio y la capa de la base de datos. Puede ser implementado en Java y .NET y también existe una versión para *Ruby on Rails*. *iBATIS* asocia objetos de modelo (*JavaBeans*) con sentencias SQL o procedimientos almacenados mediante ficheros descriptores XML, simplificando la utilización de bases de datos. La capa de persistencia se configura mediante un fichero XML de configuración, sql-map-config.xml. Además cada objeto de modelo, que representa al objeto en la aplicación, se relaciona con un fichero del tipo sqlMap.xml, que contiene sus sentencias SQL.

1.4.2 Capa de negocio

Spring

Es un marco de trabajo de aplicaciones Java/J2EE desarrollado por Rod Jonson, Juergen Hoeller, Justin Gehtland y Bruce A. Tate. Proporciona una potente gestión de configuración basada en *JavaBeans*, aplicando los principios de Inversión de Control (*IoC*). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener *singletons* ni ficheros de configuración, una aproximación consistente y elegante. Esta factoría de *beans* puede ser usada en cualquier entorno, desde *applets* hasta

contenedores J2EE. Estas definiciones de *beans* se realizan en lo que se llama el contexto de aplicación. Presenta una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción enchufables (*pluggables*), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel. Se incluyen estrategias genéricas para JTA y un único JDBC DataSource. En contraste con el JTA simple o EJB CMT, el soporte de transacciones de *Spring* no está atado a entornos J2EE.

Spring además posee una gran integración con *Hibernate*, JDO e *iBatis SQL Maps* en términos de soporte a implementaciones DAO y estrategias con transacciones. Especial soporte a *Hibernate* añadiendo convenientes características de *IoC*, y solucionando muchos de los comunes problemas de integración de *Hibernate*. Todo ello cumpliendo con las transacciones genéricas de *Spring* y la jerarquía de excepciones DAO. (Agüero, 2007)

La Programación Orientada a Aspectos (*Aspect Oriented Programming*, AOP, por sus siglas en inglés) está presente en el marco de trabajo y se encuentra totalmente integrada en la gestión de configuración de *Spring*. Se puede aplicar AOP a cualquier objeto gestionado, añadiendo aspectos como gestión de transacciones declarativa.

Está basado sobre un marco de trabajo MVC (*Model-View-Controller*), construido sobre el núcleo de *Spring*. Este marco de trabajo es altamente configurable a través de interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP, *Velocity*, *Tiles*, *iText* o POI. De cualquier manera una capa modelo realizada con *Spring* puede ser fácilmente utilizada con una capa web basada en cualquier otro framework MVC, como *Struts*, *WebWork* o *Tapestry*.

Toda esta funcionalidad puede usarse en cualquier servidor J2EE, y la mayoría de ella ni siquiera requiere su uso. El objetivo central de *Spring* es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos.

La arquitectura en capas de *Spring* (Figura 1.2) ofrece gran flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar

la gestión de configuración basada en JavaBeans sin utilizar el marco de trabajo MVC o el soporte AOP. (Bagüés, 2009)

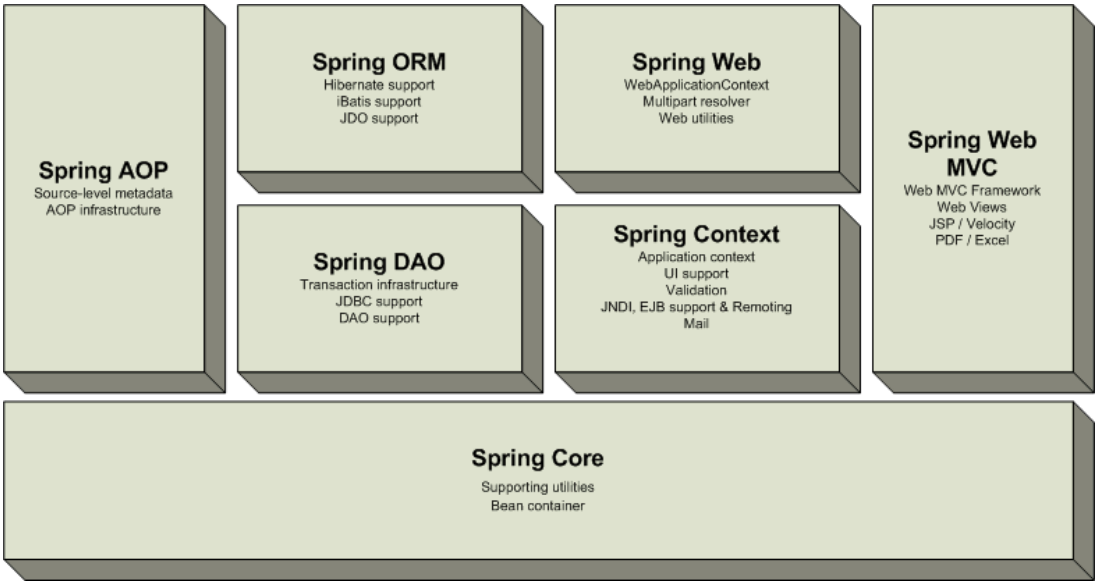

Figura 1.2. Arquitectura en capas de Spring

Google Guice

Liberado por Google con licencia Apache, tiene soporte para inyección de dependencia usando anotaciones para configurar los objetos Java. *Guice* permite que las clases implementadas se enlacen a una interfaz y luego se inyecten en los constructores, métodos o atributos usando la anotación *@Inject*

EJB

Enterprise JavaBeans (también conocidos por sus siglas EJB) es uno de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE, inicialmente desarrollado por *Sun Microsystems*). Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJB. Proporcionan, en resumen, un modelo de componentes distribuido estándar del lado del servidor, y su objetivo es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, etc.) para centrarse en el desarrollo de la lógica de negocio en sí.

El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables. (Alur et al., 2003)

1.4.3 Capa de presentación

Java Server Pages

Es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo. Es un desarrollo de la compañía Sun Microsystems. La especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la especificación JSP 2.1.

Permiten la utilización de código Java mediante scripts y permite la utilización de algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de bibliotecas de etiquetas (*TagLibs* o *Tag Libraries*) externas e incluso personalizadas.

JSP puede considerarse como una manera alternativa, y simplificada, de construir *servlets*. Es por ello que una página JSP puede hacer todo lo que un *servlet* puede hacer, y viceversa. El funcionamiento general de la tecnología JSP es que el servidor de aplicaciones interpreta el código contenido en la página JSP para construir el código Java del *servlet* a generar. Este será luego el que genere el documento (típicamente HTML) que se presentará en la pantalla del navegador del usuario.

El rendimiento de una página JSP es el mismo que tendría el servidor equivalente, ya que el código es compilado como cualquier otra clase Java. A su vez, la máquina virtual compilará dinámicamente a código de máquina las partes de la aplicación que lo requieran. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

La principal ventaja de JSP frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar

en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML en el archivo JSP. (Wikipedia, 2010)

Otra ventaja es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios. Es común incluso que los desarrolladores trabajen en una plataforma y que la aplicación termine siendo ejecutada en otra.

Java Server Faces

Es un marco de trabajo para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE. Usa *Java Server Pages* (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también se puede acomodar a otras tecnologías.

Su implementación incluye un conjunto de APIs para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de navegación de las páginas y dar soporte para internacionalización y accesibilidad, un conjunto por defecto de componentes para la interfaz de usuario, dos bibliotecas de etiquetas personalizadas para *JavaServer Pages* que permiten expresar una interfaz *JavaServer Faces* dentro de una página JSP, un modelo de eventos en el lado del servidor.

La especificación de JSF fue desarrollada por la *Java Community Process*. (Apache, 2010)

Richfaces

RichFaces es una biblioteca de componentes visuales basada en *Java Server Faces* (JSF) programada tomando como base en marco de trabajo de código abierto *Ajax4jsf*. Permite una integración rápida y sencilla de las posibilidades de Ajax en el desarrollo de aplicaciones empresariales.

Creado y diseñado en el 2005 por Alexander Smirnov, empleado de la compañía Exadel. Lo que un principio constituyó un marco de trabajo comercial se convirtió en marzo del 2007 en uno de los proyectos para Java más ambiciosos e interesantes, ahora bajo la

tutela de JBoss, una división de Red Hat. *RichFaces* sería desde ese momento software de código abierto y gratuito. En septiembre del propio año, se decidió unir *Ajax4jsf* –que había continuado su desarrollo de forma paralela- y *RichFaces* con el nombre de este último.

El marco de trabajo está implementado de forma que se adicionen las posibilidades de Ajax dentro de las páginas existente, de manera que un desarrollador no tuviese que escribir código en *JavaScript*. Por tanto, el desarrollador puede definir el evento en la página que invoca una petición de *Ajax* y las áreas de la página que deben sincronizarse con el árbol de dependencia de JSF.

Facelets

Facelets es un marco de trabajo de código abierto liberado con la licencia Apache y una alternativa tecnológica para visualizar páginas codificadas con *Java Server Faces* (JSF). Esta herramienta requiere un documento XML válido para poder funcionar. *Facelets* suporta todos los componentes visuales de JSF y construye su propio árbol de componentes. *Facelets* es bastante similar a *Apache Tapestry* y tomó algunos de las ideas de esta herramienta que había sido publicada unos años antes. Tiene facilidades de elaboración de plantillas comunes para muchos de los elementos de una aplicación.

Stripes

Está basado en el patrón Modelo-Vista-Controlador, tiene como propósito ser un marco de trabajo más ligero que *Struts* usando tecnologías de Java como anotaciones y genéricos que fueron introducidas a partir de la versión 1.5, para alcanzar el concepto de "convención mejor que configuración". Hecho con la idea de que una marca que se use reduce la cantidad de archivos de configuración a utilizar. En la práctica, esto significa que una aplicación *Stripes* apenas necesita archivos de configuración lo que reduce ostensiblemente el desarrollo y el trabajo de mantenimiento.

1.4.4 Otros marcos de trabajo

Spring AOP

La Programación Orientada a Aspectos, más conocida como AOP por su nombre en inglés *Aspect Oriented Programming*, es un modelo de programación que aborda un problema específico: capturar las partes de un sistema que los modelos de programación habituales obligan a que estén repartidos a lo largo de distintos módulos del sistema. Estos fragmentos que afectan a distintos módulos son llamados aspectos y los problemas que solucionan, problemas cruzados (*crosscutting concerns*). (O'Regan, 2004)

Usando un lenguaje que soporte AOP, podemos capturar estas dependencias en módulos individuales, obteniendo un sistema independiente de ellos y podemos utilizarlos o no sin tocar el código del sistema básico, preservando la integridad de las operaciones básicas.

Los principales campos de aplicación de la AOP son:

- rastreo de la ejecución (*tracing*)
- medida de tiempos y optimización (*profiling*)
- pruebas (*testing*)

Spring AOP es uno de los marcos de trabajo existentes para implantar la programación orientada a aspectos.

Spring Security

Proporciona servicios de seguridad dentro de *Spring*. Proporciona un sistema de autenticación a través del cual los usuarios pueden autenticarse y acceder a múltiples aplicaciones a través de un único punto de entrada. Para ello utiliza el servicio de autenticación CAS (*Central Authentication Service*) desarrollado por la Universidad de Yale, con el que una aplicación puede participar en un entorno *single sign on* a nivel de toda la empresa. Ya no es necesario que cada aplicación tenga su propia base de datos de autenticación, ni tampoco existe la restricción de que sólo se pueda utilizar dentro del

mismo servidor de aplicaciones. Otras características avanzadas que proporciona son soporte para proxy y refrescamiento forzado de logins. Tiene completa integración con Spring, utiliza los propios mecanismos de configuración de Spring y asegura la seguridad a nivel de instancia de objetos del dominio. En muchas aplicaciones es deseable definir listas de control de acceso (*Access Control Lists* o *ACL*s) para instancias de objetos del dominio individuales. *Spring Security* proporciona un completo paquete ACL con características que incluyen máscaras de bits, herencia de permisos, un repositorio utilizando JDBC, caché y un diseño *pluggable* utilizando interfaces.

Spring Security también implementa la protección de las peticiones HTTP. Ya no es necesario depender de restricciones de seguridad definidas en el fichero web.xml. Las peticiones HTTP pueden ser protegidas por una serie de expresiones regulares que definen expresiones de caminos de acceso, así como para la autenticación, autorización y gestores de reemplazo de credenciales para la ejecución como otro usuario, todo ello totalmente configurable.

Además, el sistema de seguridad puede redirigir automáticamente las peticiones a un canal de transmisión adecuado. Comúnmente esto se aplica para asegurar que las páginas seguras estarán sólo disponibles sobre HTTPS, y las páginas públicas sobre HTTP, aunque es suficientemente flexible para soportar cualquier tipo de requisitos de "canal". (JBoss, 2009)

1.4.5 Generadores de informes

BIRT Report

BIRT (*Business Intelligence and Reporting Tools***)** es un proyecto de software de código abierto que provee las funcionalidades de informes e inteligencia de negocios para aplicaciones web, especialmente aquellas basadas en Java y J2EE. BIRT es uno de los proyectos principales de la *Eclipse Foundation*.

Las metas del proyecto son las de abarcar una amplia gama de necesidades de informes que incluyan desde los típicos informes de una aplicación empresarial de gestión de

datos hasta un procesamiento analítico multidimensional en línea (OLAP). El proyecto cuenta con una amplia comunidad de usuarios y desarrolladores.

BIRT tiene dos componentes fundamentales: un diseñador visual de informes incluido en el IDE Eclipse y un componente en tiempo de ejecución para generar informes que pueden ser ejecutados en cualquier ambiente Java. (Wikipedia, 2010)

Jasper Reports

Es una herramienta de creación de informes que puede exportar el contenido a la impresora o a ficheros PDF, HTML, XLS, CSV y XML. Está escrito completamente en Java y puede ser usado en gran variedad de aplicaciones, incluyendo J2EE o aplicaciones web, para generar contenido dinámico. Su propósito principal es ayudar a crear documentos de tipo páginas, preparados para imprimir en una forma simple y flexible. JasperReports se usa comúnmente con *iReport*, una interfaz visual de código abierto para la edición de informes.

1.5 Selección de marcos de trabajo

Después de hacer un análisis detallado de los marcos de trabajo disponibles en la red de redes, tomando en cuenta las características y sobre todo, la integración entre ellos se ha decidido utilizar para el desarrollo de la aplicación las siguientes herramientas:

- *Richfaces* y *Facelets* para la capa de presentación.
- *Spring* para la capa de negocio.
- *Hibernate* para la capa de datos.
- *Spring Security* para gestionar la seguridad de la aplicación.
- *BIRT Report* para la generación de informes.

CAPÍTULO 2. ANÁLISIS Y DISEÑO DEL PROCESO DE CONTROL DE PORTADORES ENERGÉTICOS

El control de la utilización de los portadores energéticos en Cuba se ha convertido en una importante prioridad para el gobierno y todas las instituciones del Estado. La Empresa Nacional del Software (Desoft) ha venido desarrollando una solución informática que automatiza parte del proceso cubierto por los departamentos energéticos de las entidades nacionales, a la vez que proporciona una fuerte gestión contable a partir de las informaciones primarias generadas por el sistema. El producto *Celador S2C*, software comercial ya utilizado en el país se valora por tener cinco años de explotación, siendo utilizado por gran parte de los clientes de Desoft en lo que se refiere al control del consumo de combustibles por tarjeta magnética.

2.1 Algunos conceptos importantes

Para la explicación de los procesos que a continuación se describen será necesario esclarecer el uso de algunas palabras que forman parte del proceso.

- Tarjeta magnética: Dispositivo en forma de tarjeta que contiene un número de identificación nacional único de 16 dígitos y que se utiliza como monedero electrónico por parte de las entidades cubanas para efectuar el proceso de carga-consumo del combustible para los vehículos automotores.
- Suministrador: Entidad que vende el combustible a las entidades y que es la encargada de efectuar el proceso de carga de la tarjeta.
- Portador energético natural: Son aquellos "provistos por la naturaleza", ya sea en forma directa, como la energía hidráulica, eólica y solar, o después de atravesar un proceso minero, como el petróleo, el gas natural, el carbón mineral, los minerales fisionables y la geotermia, o a través de la fotosíntesis, como es el caso de la leña y los otros combustibles vegetales y de origen animal. Los portadores naturales que se producen en Cuba y de los cuales se dispone de información

estadística recopilada y sistemática son: Petróleo, Gas natural, Hidroenergía, Leña, Productos de la caña (en lo fundamental bagazo)

- Portador energético secundario: Son los productos resultantes de las transformaciones o elaboración a partir de portadores energéticos naturales o en determinados casos a partir de otro portador ya elaborado. Son portadores energéticos elaborados la electricidad, toda la amplia gama de derivados del petróleo, el carbón vegetal, el alcohol desnaturalizado y el gas manufacturado (o gas de ciudad). El grupo de los derivados del petróleo incluye una amplia variedad de productos energéticos útiles que se obtienen a partir del procesamiento del petróleo en las refinerías, entre los cuales se encuentran las gasolinas, los turbo combustibles y los combustibles diesel (gasóleos) de extraordinaria demanda universal. Los principales productos que se obtienen en Cuba de la refinación del petróleo son los siguientes: Asfalto de petróleo, Diesel, Fuel oil, Gas licuado de petróleo, Gasolinas, Naftas, Queroseno, Solventes
- Portador energético: Se le llama a todos aquellos que estén dentro de las dos categorías anteriores.
- Centros y unidades de costo: Son unidades básicas dentro de la entidad que se subdividen a conveniencia de las labores productivas de la empresa en cuestión Una unidad de costo es parte de un centro de costo.
- Chip de consumo: Son los tickets o comprobantes que se reciben en el servicentro una vez que se consume el combustible.
- Cierre contable: Proceso mediante el que se efectúa el cierre de las operaciones, para un período dado, en el área contable de la empresa.
- Comprobantes: Documentos emitidos por el área contable de una entidad que contiene los movimientos efectuados en ella.
- Documento de liquidación: Es un resumen elaborado a partir de la suma de los tickets de consumo para un período dado.

- Facturación de compra de combustible: Incluye las facturas de compra de combustible a través de Tarjetas magnéticas, definiendo el suministrador, el número de la factura, y la fecha en que se emitió. En los detalles de la factura se especifican las tarjetas involucradas en la factura con el saldo a aumentar. En este proceso los saldos actuales de las tarjetas se actualizarán añadiéndole el saldo definido en la factura.
- Niveles de actividad: Son las actividades que se definen para cada empresa u organismo que se corresponden con las funciones propias de cada una y que se asocian usualmente a centros y unidades de costo.
- Saldo de la tarjeta: Dinero que posee la tarjeta magnética para el consumo de combustible de los vehículos de la empresa.

2.2 Descripción del proceso de trabajo con la Tarjeta Magnética

A continuación se pasan a detallar los procesos necesarios para el trabajo con la tarjeta magnética.

2.2.1 Comprar tarjetas

Es el proceso mediante el que se hace la compra de la tarjeta. La persona autorizada en la empresa a hacer este tipo de trámite va hacia uno de los proveedores de tarjeta existentes en el país y compra una tarjeta. Según la Resolución 23 del 2008 debe haber como mínimo 1,5 tarjetas por carro, su límite superior puede extenderse. La cantidad de tarjetas a comprar debe estar en correspondencia con la cantidad de autos que posea la empresa. Cuando el comprador adquiere su tarjeta se genera una factura que pasa a Economía que contabilizará la compra de la nueva tarjeta. A su vez, la entidad proveedora registra el nuevo número y la entidad que compró la tarjeta bajo ese número único de identificación. Cada tarjeta tiene un tipo de portador asignado. En las empresas, las tarjetas deben ser controladas por el área económica que debe tener un listado de todas las tarjetas que posee la empresa. Por regla general, en las dependencias locales de una entidad nacional no se ejecuta este proceso, pues su entidad matriz se

encarga de hacerlo y luego hace el proceso de asignación de estas tarjetas entre sus diferentes dependencias.

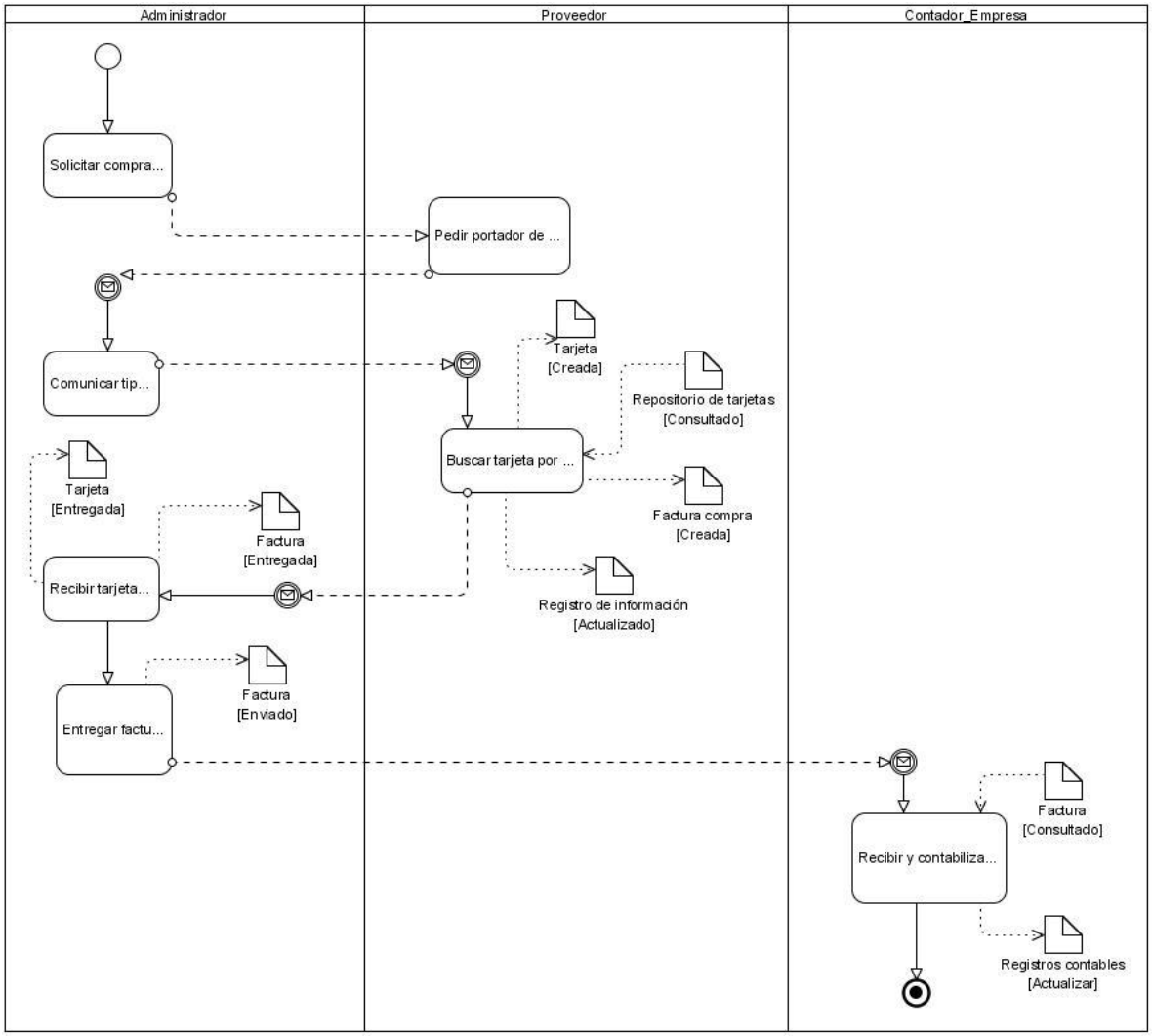

Figura 2.1. Diagrama del proceso de compra de tarjetas

2.2.2 Facturar combustible

Es el proceso mediante el cual las entidades adquieren el combustible asignado. Los Ministerios de forma vertical asignan cifras de combustible necesarias para el funcionamiento interno de sus dependencias. De forma ramificada y jerárquica cada estructura distribuye los recursos y asigna cifras de combustible a consumir para cada una de sus dependencias. De esta manera se constituye lo que se denomina Plan de consumo. Los organismos que hacen la asignación notifican a sus dependencias y a los

proveedores de tarjetas (Fincimex, Cubalse) la cifra, de manera que ambas lleven un registro de la disponibilidad que tienen. Un responsable en representación de una entidad se dirige hacia las oficinas de una entidad suministradora y requiere la compra de cierta cantidad de combustible. El suministrador verifica que la entidad tiene cifra de asignación de combustible disponible. Si aún le queda saldo por facturar, se pasa al proceso de venta de combustible y rebaja del saldo actual de la entidad. La factura de venta que se genera pasa al departamento contable de la empresa y se genera un comprobante contable de pago anticipado

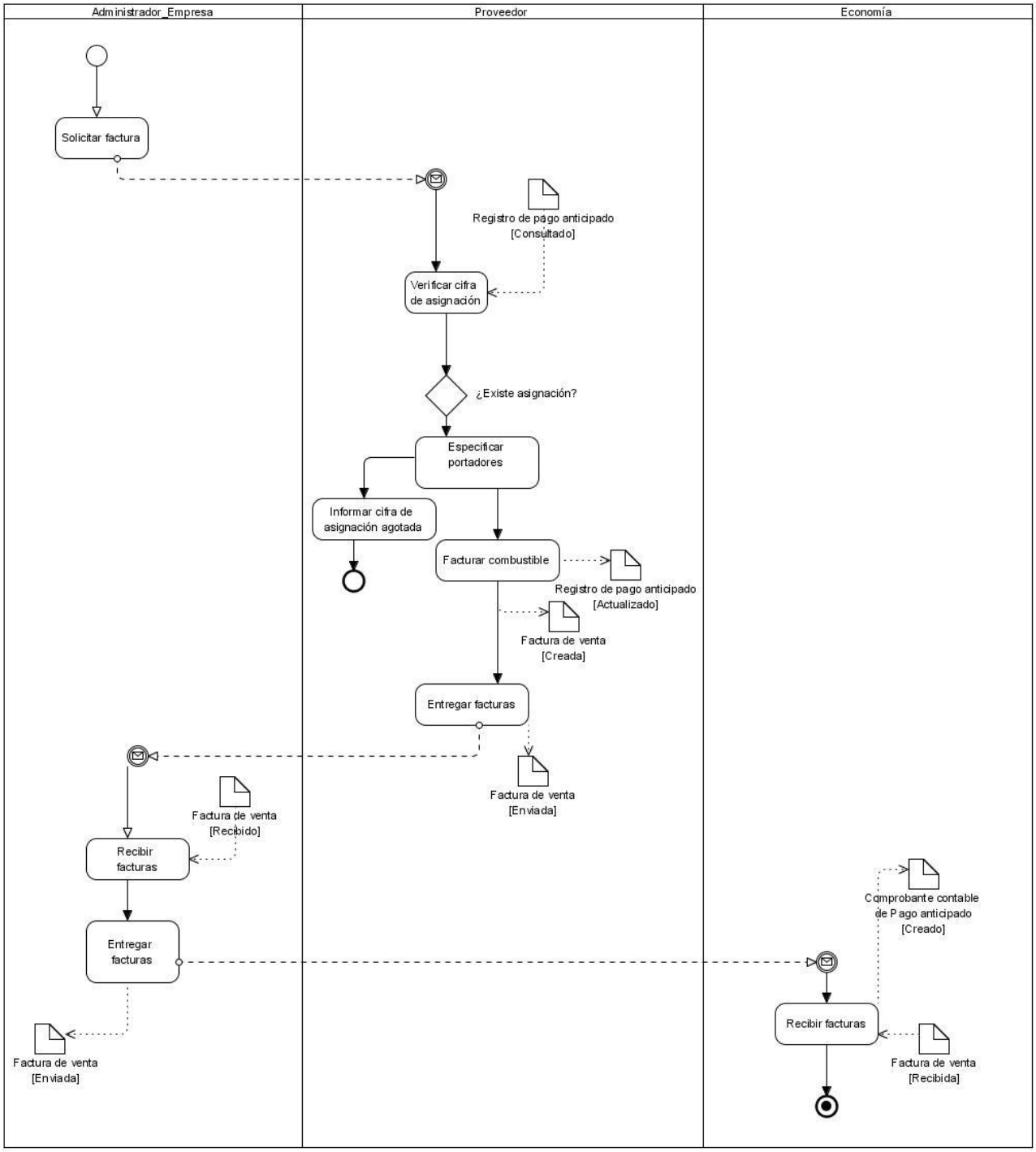

Figura 2.2. Diagrama del proceso de facturación de combustible

2.2.3 Cargar tarjetas

A partir de la facturación del combustible comienza el proceso de carga de las tarjetas. Estas actúan como una especie de monedero electrónico al igual que una tarjeta propia de teléfono, o una tarjeta de banco. Luego de ser cargada, ese dinero puede ser consumido. Siempre hay una correspondencia cantidad en litros – monto de dinero. De manera que por ejemplo, si existe gasolina regular con un precio de 0.60 centavos y la tarjeta tiene $30.00 de saldo, entonces se divide 30/0.60 lo que corresponde exactamente a 50 litros de gasolina. Las entidades deben determinar a partir de la facturación previamente hecha a cuáles tarjetas deben cargar con saldo. Tómese como ejemplo la factura previamente hecha por una entidad por la cantidad de $780.00 de Gasolina Regular de 0.60 centavos. Esta entidad luego debe distribuir esa cifra internamente entre las diferentes tarjetas que posee. Por ejemplo:

Tarjeta	**Monto en dinero**	**Equivalencia en litros**
201345673210	$ 180.00	300
124564321369	$ 360.00	600
458900123478	$ 240.00	400
Total	**$ 780.00**	**1300**

La actividad de carga se ejecuta por parte del suministrador y, por lo general, se realiza acto seguido a la actividad de facturación de combustible. El suministrador entrega la(s) tarjetas(s) y los comprobantes de carga a la entidad que solicita el servicio. Luego la entidad debe proceder a actualizar el submayor de tarjetas y generar el comprobante de liquidación de pago anticipado.

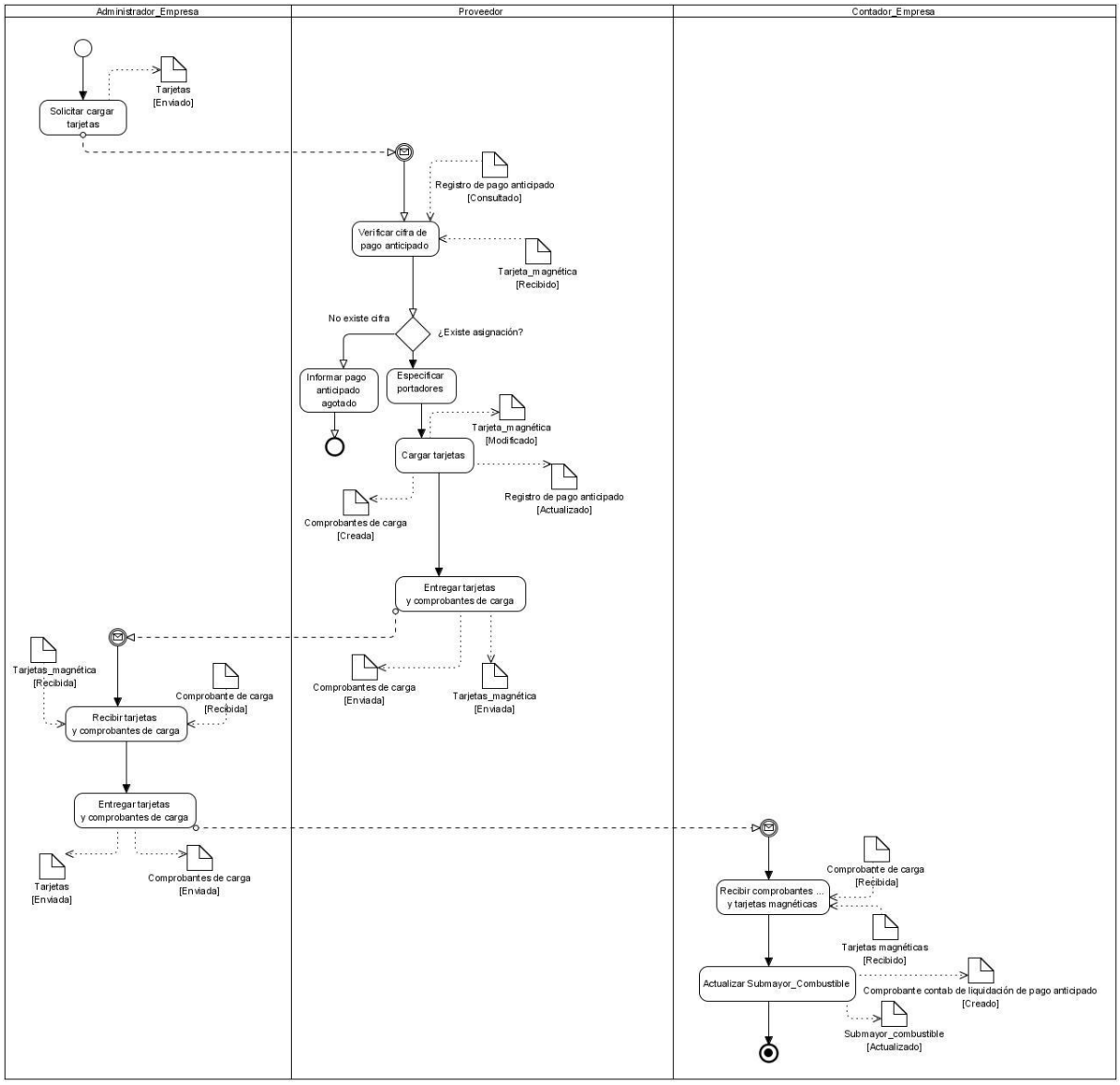

Figura 2.3. Diagrama del proceso de carga de tarjetas

2.2.4 Entrega y devolución de tarjetas

A partir del año 2009, comenzó a aplicarse una nueva Resolución, la número 60 refiriéndose a un nuevo proceso dentro del flujo del negocio del control de las Tarjetas Magnéticas de Combustible en Cuba. Se trata de la obligatoriedad de entregar y devolver la tarjeta a la Caja, depósito físico que tienen las empresas cubanas en el que se asegura el dinero y otros bienes a custodiar. Así mismo, el control contable que esto genera para tener ubicado en cada momento la situación real de cada tarjeta. Se trata de introducir en el sistema los modelos de Anticipo y Liquidación de tarjetas que permitirá llevar el movimiento diario de efectivo en Caja.

Cuando se realiza la entrega de la tarjeta al funcionario autorizado, el cajero debe registrar los datos necesarios como la fecha de entrega y el saldo de la tarjeta en el momento de la entrega. En el momento de la liquidación, después de introducidos los tickets, el cajero debe completar la parte del modelo que corresponde a la liquidación del anticipo (fecha de devolución de la tarjeta, y el saldo final). El saldo final introducido en este modelo debe coincidir con el saldo final de las tarjetas que resulte de los tickets de consumo introducidos.

Si como resultado de la liquidación se observa que existen tarjetas que no han sido liquidadas completamente, o sea que tienen saldo al finalizar el periodo, estos valores deberán ser incorporados al movimiento diario de efectivo en caja.

Guiándose por los modelos de Anticipo y liquidación de gastos de combustible y por los vales de consumo introducidos se obtiene un asiento contable.

Vale aclarar que este proceso del negocio es completamente nuevo y no está presente en el *Celador S2C*.

2.2.5 Consumir combustible

Con la tarjeta con saldo y la necesidad de consumir ese combustible previamente cargado en la tarjeta comienza el proceso de consumo, acción que se ejecuta directamente en un servicentro, a través de los mecanismos implementados para ello. El operador servicia al auto y rebaja el saldo de la tarjeta con la cual se consume el combustible. El servicentro emite un comprobante (chip o vale de consumo) con el que las entidades justifican el combustible consumido. Los servicentros guardan una copia de estos datos y le entregan al una copia. El departamento de contabilidad procesa este consumo y rebaja del saldo de la tarjeta la cifra consumida y procede a generar los comprobantes contables necesarios para actualizar la situación actual de la tarjeta.

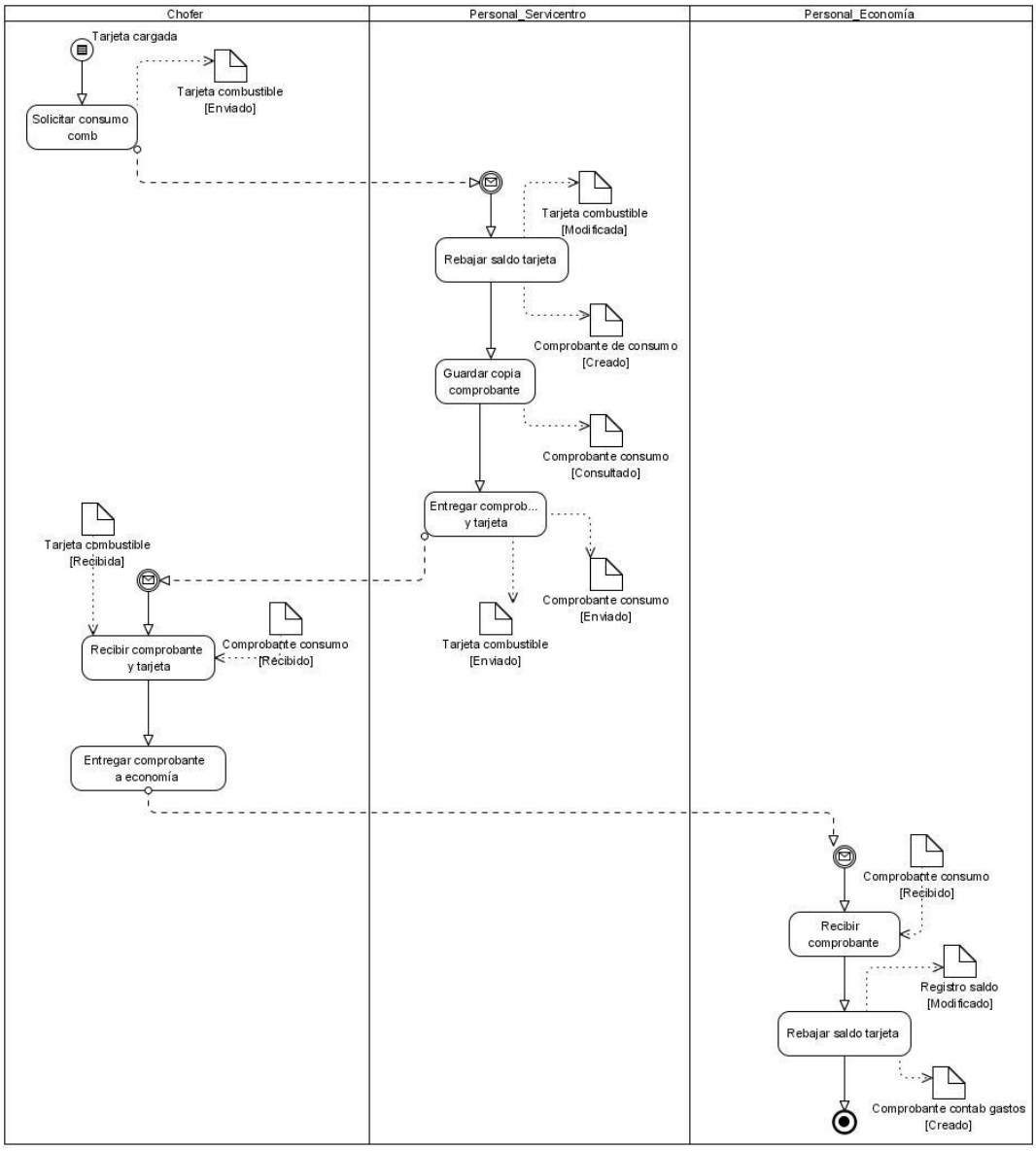

Figura 2.4. Diagrama del proceso de consumo de combustible.

2.2.6 Movimientos internos de la tarjeta

En su proceso natural de carga – consumo, la tarjeta puede tener cuatro tipos de movimientos que alteran su contenido. Estos son:

- Transferencia de combustible: Es el proceso que se verifica de una entidad a otra. Está referido por una necesidad muy puntual entre entidades de la misma organización. Por orden de las personas que puedan autorizar esa transferencia, se procede a enviar físicamente la tarjeta hacia la otra entidad. En la entidad

receptora a la tarjeta debe dársele de alta económicamente hablando a través de una cuenta de transferencias entre dependencias, mientras que la tarjeta se recibe de forma física. Esta tarjeta pasa a ser, a partir de ese momento, parte del conjunto de tarjetas que posee la entidad. En la entidad que envía económicamente se refleja en una cuenta de transferencia entre dependencias como salida. La tarjeta se lleva a saldo cero y se justifica su movimiento de saldo a través del comprobante en la citada cuenta de transferencia. Para el caso de la entidad que recibe la tarjeta se recibe con un saldo y se comienza a utilizar. Como las tarjetas pueden ser utilizadas en todo el país en las redes de las entidades proveedoras de tarjetas, la tarjeta transferida puede ser usada sin problemas aun cuando la transferencia hay sido hecha entre entidades que estén en diferentes localidades del país.

- Traspaso de saldo: El proceso se genera en colegio con la entidad suministradora para el caso en que una tarjeta se averíe y se necesite traspasar el saldo de una tarjeta en mal estado hacia una en buen estado técnico. La persona autorizada se persona en las oficinas del suministrador y solicita un traspaso de saldo. La entidad suministradora realiza el cambio de saldo desde la tarjeta inservible hacia la(s) tarjeta(s) a donde se distribuirá la cifra. La entidad suministradora entrega un certificado de traspaso de saldo que contiene los datos de las transacciones realizadas. Los traspasos pueden ser
    - De forma completa, de una tarjeta a otra tarjeta.
    - De forma parcial, de una tarjeta a varias tarjetas, siempre del mismo portador.
- Ajuste de saldo: Se realiza de forma interna en la entidad y se hace cuando sea necesario ajustar las tarjetas por algún motivo de descuadre entre el saldo real de la tarjeta (la que se puede obtener físicamente a través de un equipo que lea la tarjeta) y el saldo que tiene el departamento contable. Los descuadres suelen ocurrir cuando se extravía algún comprobante de consumo o no se introduce en

el sistema por algún motivo. Al realizar el cierre contable, los saldos entre la tarjeta y el submayor de tarjetas no coinciden. Es entonces cuando a través del sistema computarizado se reajusta el saldo para llevarlo al saldo real de la tarjeta. Este ajuste puede ser de aumento o de disminución. Al final del proceso los saldos deben coincidir. Este ajuste debe justificarse en Economía a través de un documento que avale las razones del mismo.

- Cambio de portador: Cuando por algún motivo se necesite cambiar el portador de la tarjeta y se desea seguir utilizando una misma tarjeta, es decir no comprar una nueva, se cambia, de conjunto con el suministrador de tarjetas, el portador de la misma. La persona autorizada se persona en el lugar, manifiesta su necesidad de hacer el cambio. En la entidad suministradora hace los ajustes necesarios para hacer el cambio, guardan una copia de la información en la tarjeta a través de un equipo y dejan constancia en sus oficinas del nuevo portador de esa tarjeta. A su vez, en la empresa este cambio debe reflejarse en el submayor de tarjetas adicionando una nueva fila que diga Cambio de portador, la fecha en que se cambió y cuál es ese nuevo portador. Es importante destacar que un cambio de precio en un portador determinado genera una acción de cambio de portador aun cuando el combustible empleado sea el mismo.

2.2.7 Informes de control

Se refiere a toda la generación de informes o documentos de salida que son los resultados que necesita la entidad como consecuencia de su acción de controlar todo lo referente al proceso de la tarjeta magnética. Los informes son:

- Submayor de tarjetas.
- Comprobante de entrada
- Comprobante de gastos.
- Balance contable.
- Existencia en tarjetas.
- Listado de comprobantes de consumos.
- Listado de ajustes realizados.
- Listado de cargas a la tarjeta.
- Consumo de combustible en un período dado (por tarjeta, que puede ser listado por centro de costo, vehículo y por la chapa del equipo).
- Informe de plan vs consumo.

2.3 Otros procesos detectados

En el análisis de la lógica de negocio se detectaron otras necesidades vinculadas al proceso de control de las Tarjetas Magnéticas. Se trata del control de Hoja de Ruta con el objetivo de lograr conocer el consumo real de Combustible por viajes y llevar el kilometraje recorrido para poder establecer los índices de consumo por vehículos. También, la necesidad de generar comprobantes contables que puedan ser usados en los módulos correspondientes de las aplicaciones informáticas para la gestión económica existentes en las entidades. En este sentido, importante el rol que podría jugar el sistema al dar la posibilidad de exportar datos hacia estos sistemas y a su vez recibir de ellos.

2.4 Requisitos funcionales

A continuación se describen los requisitos funcionales detectados y que se asocian al trabajo con las Tarjetas. El resto de los requisitos se completan en el Anexo A de este documento.

- Introducir datos de nueva tarjeta.
- Registrar los datos de la facturación de combustibles.
- Registrar las cargas de combustible.
- Captar los consumos de combustible.
- Registrar movimiento de datos entre la caja y los usuarios.
- Personalizar el trabajo con la tarjeta magnética
    - Incluye las acciones de Cambio de Portador, Ajustes de la Tarjeta, Transferencias entre dependencias, Traspaso de Saldo, Activar y desactivar tarjeta
- Exportar los datos a un sistema contable.
- Obtener submayor de tarjetas.
- Obtener comprobantes.
- Obtener balance contable.
- Exportar comprobantes.
- Informar vencimiento de tarjetas.
- Listar consumo de combustibles.
- Controlar tarjetas vencidas.
- Registrar cuentas contables con sus niveles y especificidades

2.5 Determinación de nuevas necesidades funcionales

Partiendo de la existencia de un sistema anterior al realizar el análisis de requisitos del nuevo sistema se detectan algunas funcionalidades que son mejoradas con respecto al sistema anterior y otras que son completamente nuevas. Entre las nuevas necesidades están:

- La incorporación de un registro donde se pueda consignar las actividades y niveles de actividad de la empresa de manera que se puedan asociar los elementos de gastos de los portadores energéticos a acciones tangibles ejecutadas por departamentos y por niveles de actividad.
- Proceder al registro de operaciones entre el usuario y la Caja para automatizar el proceso anticipo-consumo según Resolución 60 / 2009

Dentro de las necesidades mejoradas con respecto a la versión anterior de la aplicación están:

- Administración del clasificador de cuentas de las entidades flexibilizando la elaboración de los comprobantes contables.
- Interfaz visual del nomenclador de Portadores Energéticos que en la aplicación existente se compone de cuatro ventanas diferentes para poder introducir los datos en el sistema.

Después de haber hecho un análisis detallado de las funcionalidades actuales y las deseables para esta nueva versión, se procede a definir los actores y casos de uso del sistema que servirán de base para la implementación de las funcionalidades del nuevo sistema informático.

2.6 Actores del sistema

Luego de analizar los nuevos requisitos funcionales y la problemática planteada se definen los siguientes actores:

- Administrador
- Contador
- Transporte

2.7 Casos de uso del sistema

Un caso de uso constituye una técnica utilizada para describir el comportamiento del sistema, a través de un documento narrativo que define la secuencia de acciones que

obtienen resultados de valor para un actor que utiliza un sistema para completar un proceso, sin importar los detalles de la implementación. Ningún sistema se encuentra aislado. Cualquier sistema interesante interactúa con actores humanos, mecánicos u otros sistemas, que lo utilizan con algún objetivo y que esperan que el sistema funcione de forma predecible. Un caso de uso especifica el comportamiento de un sistema o de una parte del mismo, y es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor. Los casos de uso se emplean para capturar el comportamiento deseado del sistema en desarrollo, sin tener que especificar cómo se implementa ese comportamiento. Los casos de uso proporcionan un medio para que los desarrolladores, los usuarios finales del sistema y los expertos del dominio lleguen a una comprensión común del sistema. A continuación se presenta el modelo de casos de uso de sistema para el actor Contador.

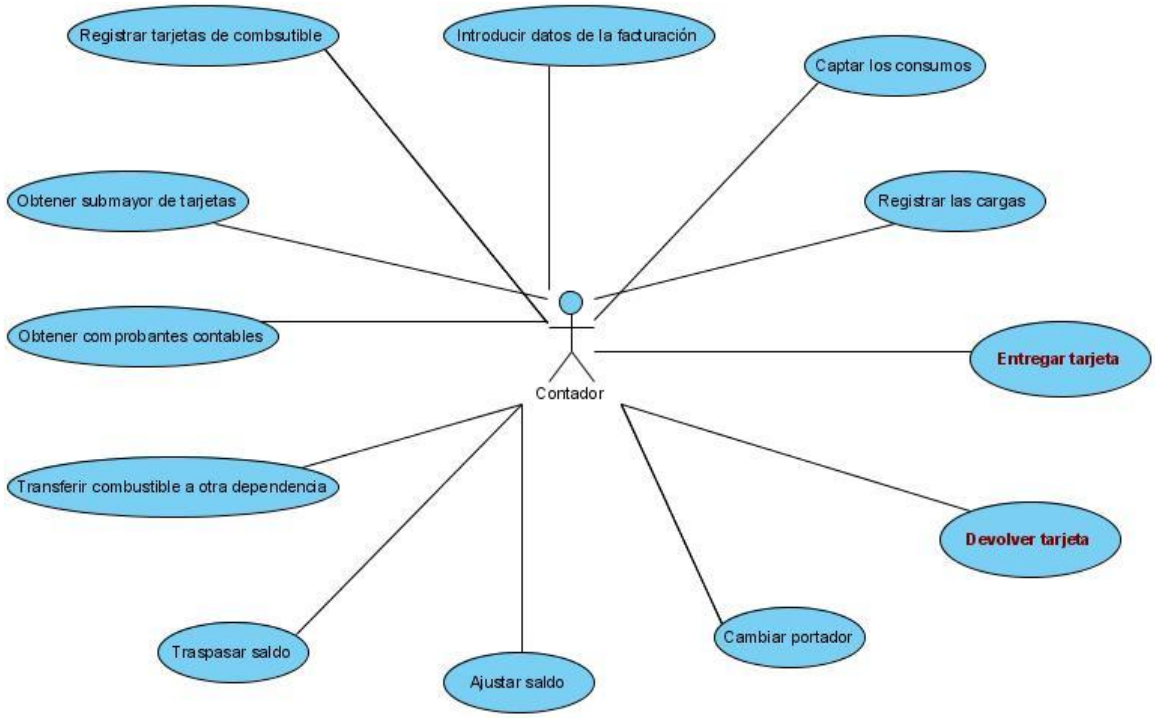

Figura 2.5. Modelo de los principales casos de uso de sistema del actor Contador asociados al trabajo con la Tarjeta Magnética.

Y los casos de uso del actor Transporte.

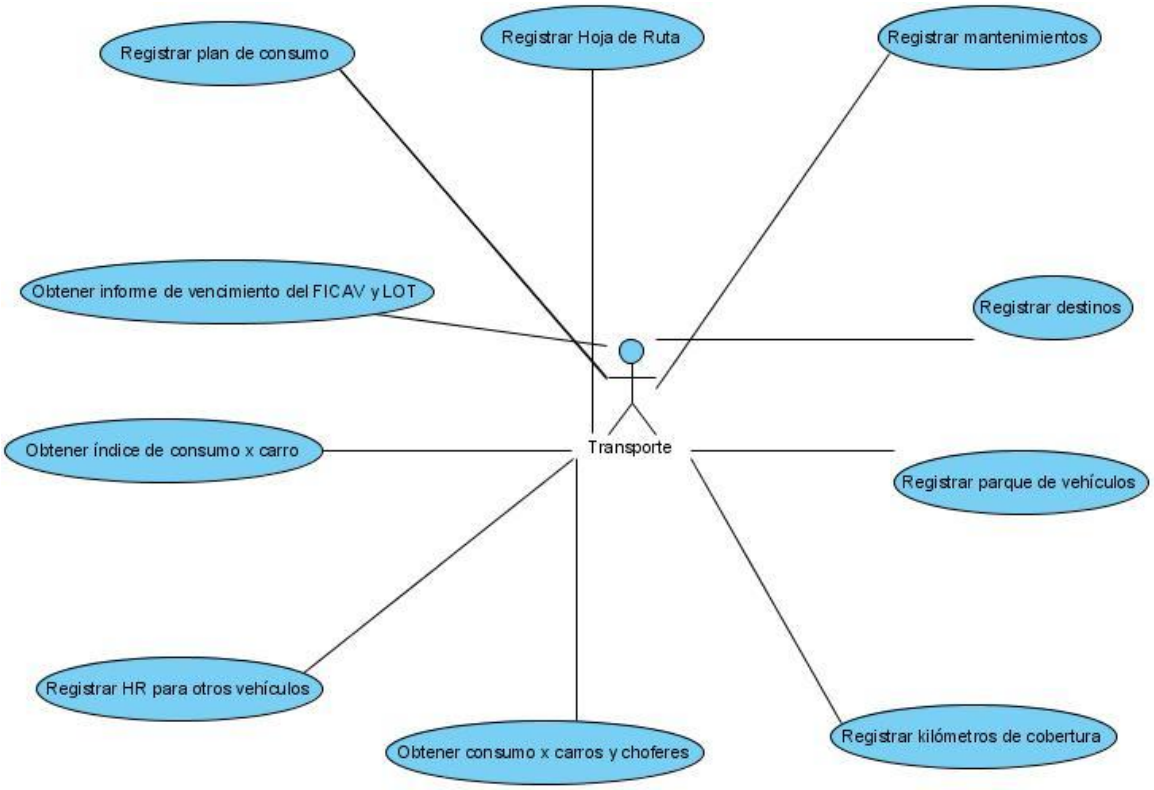

Figura 2.6. Modelo de los principales casos de uso de sistema del actor Transporte.

Los casos de uso asociados al actor Administrador están vinculados al trabajo con la seguridad de la aplicación en acciones tales como: adicionar usuarios y grupos de usuarios, definir sus permisos, ver la bitácora del sistema, eliminar datos históricos de la bitácora y datos de la aplicación.

2.8 Especificación de casos de uso del sistema

A continuación se describen tres casos de uso del sistema. Para ver el listado completo remitirse al Anexo B.

2.8.1 Registrar empresa

Para el caso de los nomencladores se puede definir un caso de uso típico con el siguiente diagrama.

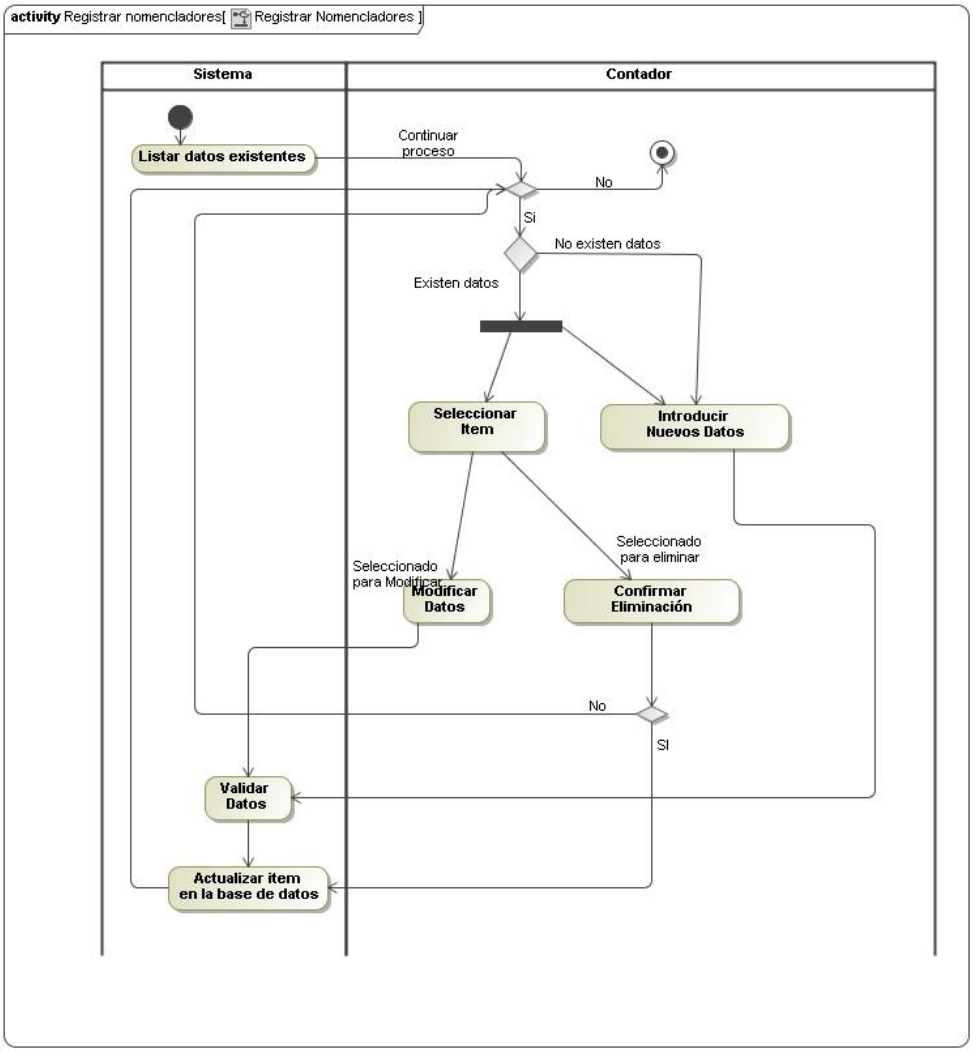

Figura 2.7. Diagrama de actividad Registrar Nomencladores.

Su especificación permite conocer las diferentes acciones a ejecutar en todo momento de su tiempo de vida.

Caso de Uso	Registrar empresa.
Actores	Contador
Propósito	Registrar, modificar y eliminar los datos de una empresa
Tipo	
Descripción	El CU se inicia cuando el actor decide registrar, modificar y eliminar datos de una empresa y finaliza con la actualización de los datos de ésta en la base de datos.
Referencias	
Requisitos Funcionales	Registrar las empresas
Precondición	

Flujo Básico	
Acción del Actor	**Respuesta del Sistema**
1. El contador accede en el menú principal a la opción Gestionar empresas.	2. El sistema muestra la ventana de registro de empresa, verifica si existen empresas en la tabla y las muestra en un grid.
3. El actor decide: • Agregar una nueva empresa(es el flujo básico). • Modificar una empresa previamente creada (ver sección	

Modificar). • Eliminar una empresa previamente creada (ver sección Eliminar).	
4. El actor acciona el botón Insertar.	5. El sistema: Carga los organismos en un select. Muestra el contenido de los campos: Código, Siglas, Descripción, Logotipo, Consolidadora (Si/No) y los botones Aceptar y Cancelar activos.
6. El actor introduce el código REUP de la empresa, introduce el resto de los datos. Presiona el botón Aceptar	7. El sistema verifica que el código de la empresa no se haya registrado con anterioridad, y procede a registrar los datos de la empresa en la tabla de la base de datos. El sistema no cierra la ventana de insertar hasta que el usuario dé click en cerrar o cancelar, de esta forma podrán adicionarse varias empresas en el momento.
	8. Muestra un mensaje de información: "Los datos de la empresa han sido insertados con éxito". El sistema muestra el grid actualizado.

Flujo Alternativo	
Acción del Actor	**Respuesta del Sistema**
Línea 8: Si la empresa ya fue previamente creada el sistema el sistema muestra un mensaje de error "Ya existe esta empresa"	
Sección: Modificar datos de la empresa	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona una empresa en el grid.	2. Activa el botón Modificar.
3. El actor presiona el botón "Modificar".	4. El sistema muestra la interfaz de usuario con el campo Código REUP, deshabilitado y los campos Siglas, Descripción, Logotipo, Consolidadora (Si/No) con la información disponible para ser cambiada Los botones Aceptar y Cancelar se mostraran activos.
5. El actor modifica los datos y acciona el botón aceptar	6. El sistema guarda los cambios en la tabla y actualiza el grid con la nueva información guardada.
7. El actor selecciona una tarjeta en el grid.	8. Activa el botón Modificar.

Flujo Alternativo	
Acción del Actor	**Respuesta del Sistema**
Línea 3: Si el actor acciona el botón cancelar no se modificaran los datos.	
Sección: Eliminar datos de tarjeta	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona una empresa en el grid.	2. El sistema activa el botón Eliminar
3. El actor acciona el botón "Eliminar"	4. El sistema muestra un mensaje de alerta: "Esta seguro de querer eliminar el registro seleccionado"
5. El actor presiona el botón "Aceptar"	6. El sistema elimina la empresa seleccionada de la tabla y actualiza el grid en la interfaz de usuario.
Poscondiciones	Quedan actualizados los datos de la empresa

2.8.2 Cargar tarjetas

Caso de Uso	Cargar tarjetas.
Actores	Contador
Propósito	Registrar, modificar y eliminar los datos de

	una carga
Tipo	
Descripción	El CU se inicia cuando el actor decide registrar y modificar datos de una carga y finaliza con la actualización de los datos de esta en la base de datos.
Referencias	
Requisitos Funcionales	Registrar las cargas de combustible.
Precondición	

Flujo Básico	
Acción del Actor	**Respuesta del Sistema**
9. El contador accede en el menú principal a la opción Cargar tarjetas	10. El sistema muestra la interfaz, verifica si existen cargas registradas en la tabla y los muestra en un grid. La interfaz se muestra con el botón "Guardar" activo.
11. El actor decide: • Agregar una nueva carga (es el flujo básico). • Modificar una carga (ver sección Modificar).	
12. El actor acciona el botón Distribuir saldo.	13. El sistema muestra, en la ventana, los botones Aceptar y Cancelar activos.

14. El actor procede al desglose de la factura señalada y se especifica la carga a la tarjeta por cada tipo de combustible.	15. El sistema verifica que la carga puede ser rebajada a partir de la cantidad facturada con anterioridad. De estar correcto inserta la nueva cifra de carga en la tabla de la base de datos.
	16. Muestra un texto con el saldo restante de la factura después de la carga
	17. Muestra el mensaje "La carga ha sido insertada con éxito". El sistema muestra el grid actualizado.
Poscondiciones	Quedan actualizadas las cargas y los saldos de las facturas.

2.8.3 Introducir tickets de consumo

Caso de Uso	Introducir tickets de consumo.
Actores	Contador
Propósito	Registrar, modificar y eliminar los tickets de consumo
Tipo	
Descripción	El CU se inicia cuando el actor decide registrar, modificar y eliminar datos de un

	consumo y finaliza con la actualización de los datos de esta en la base de datos.
Referencias	
Requisitos Funcionales	Captar los consumos de combustible.
Precondición	

Flujo Básico	
Acción del Actor	**Respuesta del Sistema**
18. El contador accede en el menú principal a la opción Tickets de consumo.	19. El sistema muestra la interfaz asociada a la apatación de los consumos de combustible, verifica si existen tickets registrados en la tabla y los muestra en un grid. La interfaz se muestra con el botón "Guardar" activo.
20. El actor decide: • Agregar un consumo (es el flujo básico). • Modificar un consumo (ver sección Modificar). • Eliminar un consumo (ver sección Eliminar).	
21. El actor acciona el botón Insertar.	22. El sistema muestra la ventana, con los botones Aceptar y Cancelar activos.

	Además los campos Nro Terminal, Nro. Ticket, Fecha, Suministrador, Nro. Tarjeta, Centro y Unidad de Costo a cargar el consumo, Chofer, Chapa, Observaciones, Cantidad a consumir
23. El actor introduce la cifra de consumo que desea insertar y acciona el botón Aceptar.	24. El sistema rebaja de la tarjeta el saldo disponible y procede a registrar el consumo en la tabla correspondiente.
	25. Muestra un mensaje: "El consumo ha sido insertado con éxito". El sistema muestra el grid actualizado.
Flujo Alternativo	
Acción del Actor	**Respuesta del Sistema**
Línea 7: Si al efectuar la entrada del consumo, el saldo disponible de la tarjeta es cero o menor, el sistema muestra un mensaje de error: "No se puede consumir esta cantidad"	
Sección: Modificar consumo	
Acción del Actor	**Respuesta del Sistema**
1. El actor selecciona del grid el	2. El sistema activa el botón Modificar.

suministrador que desea modificar.	
3. El actor acciona el botón Modificar.	4. El sistema muestra la interfaz de usuario con la denominación del consumo por el actor. La interfaz se muestra con el botón Aceptar y Cancelar activos.
5. El actor modifica la denominación y presiona el botón Aceptar.	6. El sistema envía un mensaje al actor confirmándole si desea modificar el registro: "Está seguro que desea modificar el registro seleccionado"
7. El actor acciona el botón Aceptar.	8. El sistema verifica que el consumo no se encuentre registrado, procede a registrar el cambio en la tabla de la base de datos.
	9. Envía un mensaje: "Ha sido modificado el registro seleccionado". El sistema muestra el grid actualizado.
Sección: Eliminar consumo	
Acción del Actor	**Respuesta del Sistema**
7. El actor selecciona del grid el consumo que desea Eliminar.	8. El sistema activa el botón Eliminar.
9. El actor acciona el botón Eliminar.	10. El sistema muestra un mensaje de alerta: "Está seguro de querer eliminar el registro seleccionado"

11. El actor acciona el botón Aceptar.	12. El sistema elimina de la tabla el consumo seleccionado.
Poscondiciones	Quedan actualizados los consumos y el saldo de las tarjetas magnéticas.

2.9 Arquitectura del sistema

La arquitectura del sistema se planifica a partir de los marcos de trabajo ya seleccionados y será consecuente con la descripción del empaquetado y componentes de la aplicación que se definirá en el siguiente capítulo. Se definen cinco paquetes que estarán presentes en las clases con código en Java. Para facilidad de uso, se han dividido muchos de esos paquetes en cuatro subpaquetes comunes: *interface*, donde residirán las clases que definen la implementación de la clase que estará en *implementation*. El subpaquete *config* contendrá clases necesarias para la configuración del paquete padre y por último un paquete *útil* donde residirán todas aquellas clases utilitarias o que sirvan de soporte para el resto de las clases del paquete principal.

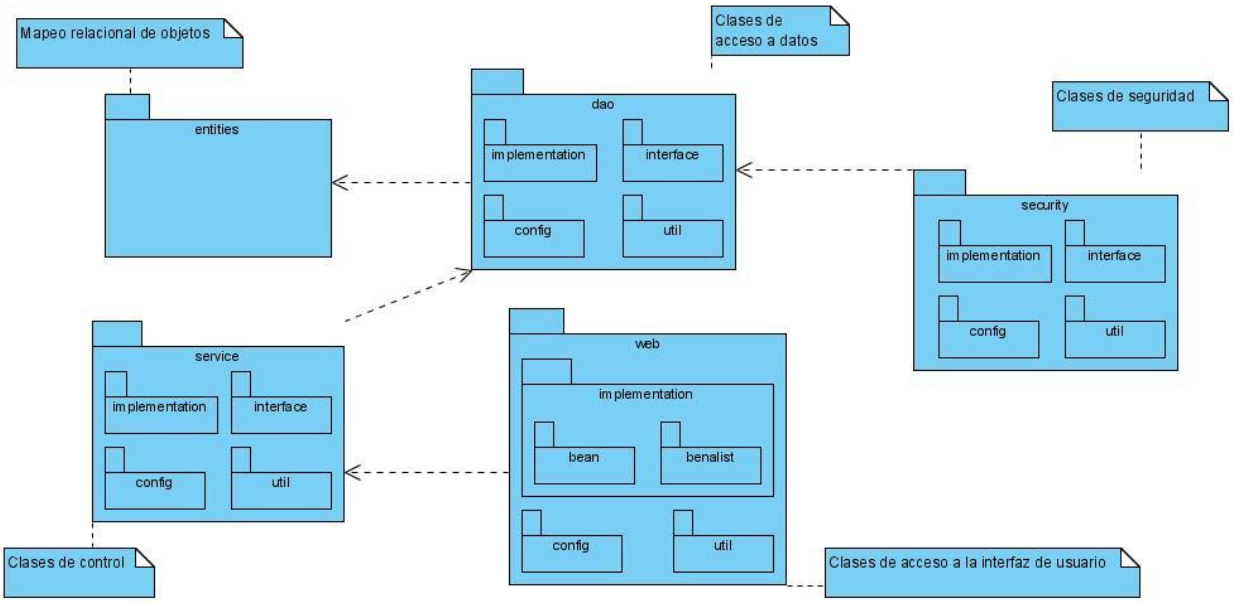

Figura 2.8. Diagrama de paquetes

y sus interacciones e interrelaciones.

2.10 Diseño de la base de datos

La base de datos del sistema, compuesta por más de cuarenta tablas, ha sido diseñada siguiendo los patrones del modelo Entidad – Interrelación. Se muestran sólo los diagramas principales. Los procedimientos almacenados así como las tablas comenzarán con el prefijo identificativo de 3 caracteres que identifica a la parte asociada (internamente se dividió el trabajo en submódulos). Ejemplo: nom_ListarPortador, tra_ListarHojaRuta

Los caracteres identificativos son:

- nom – Nomencladores
- tar – Tarjeta magnética
- tra - Transporte
- his – Histórico
- seg – Seguridad del sistema

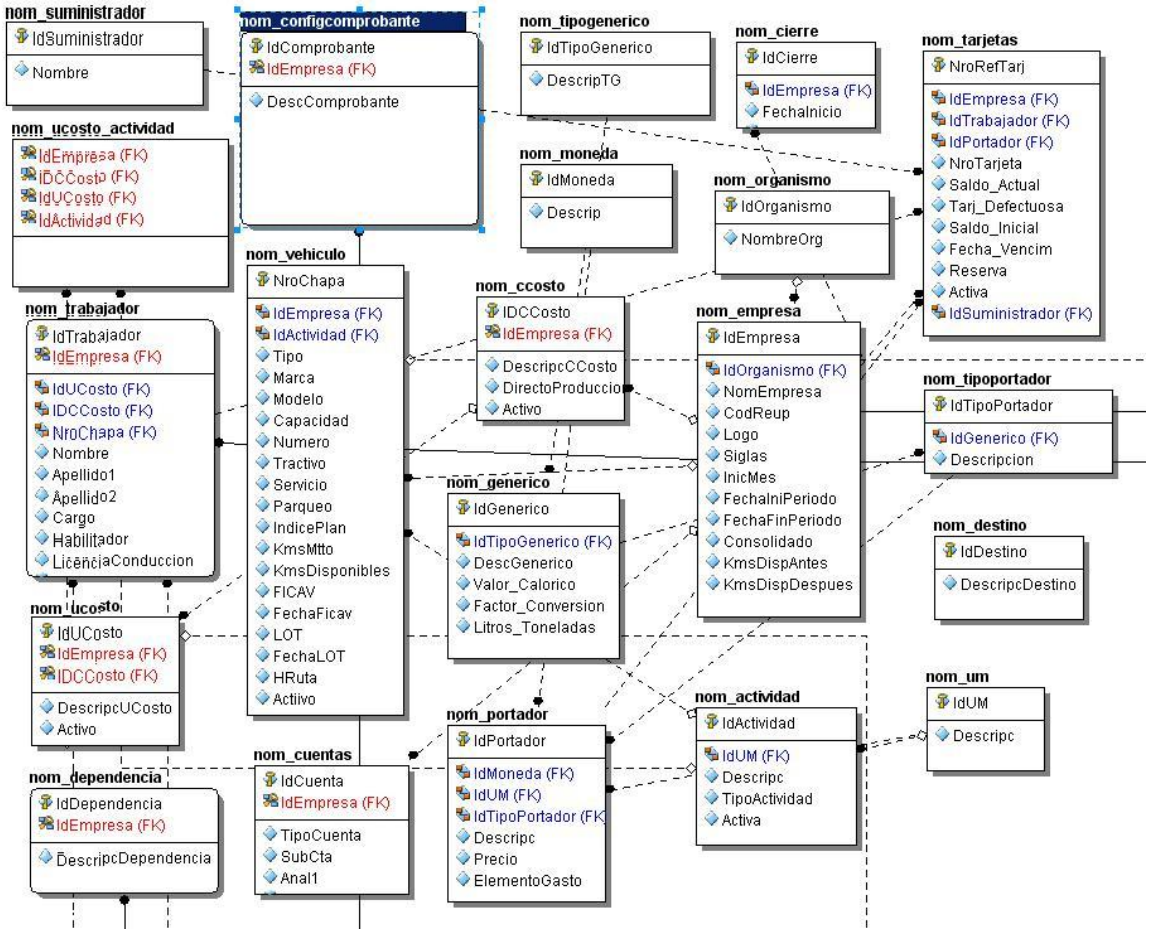

Figura 2.9. Diagrama Entidad-Interrelación asociado al trabajo con los Nomencladores.

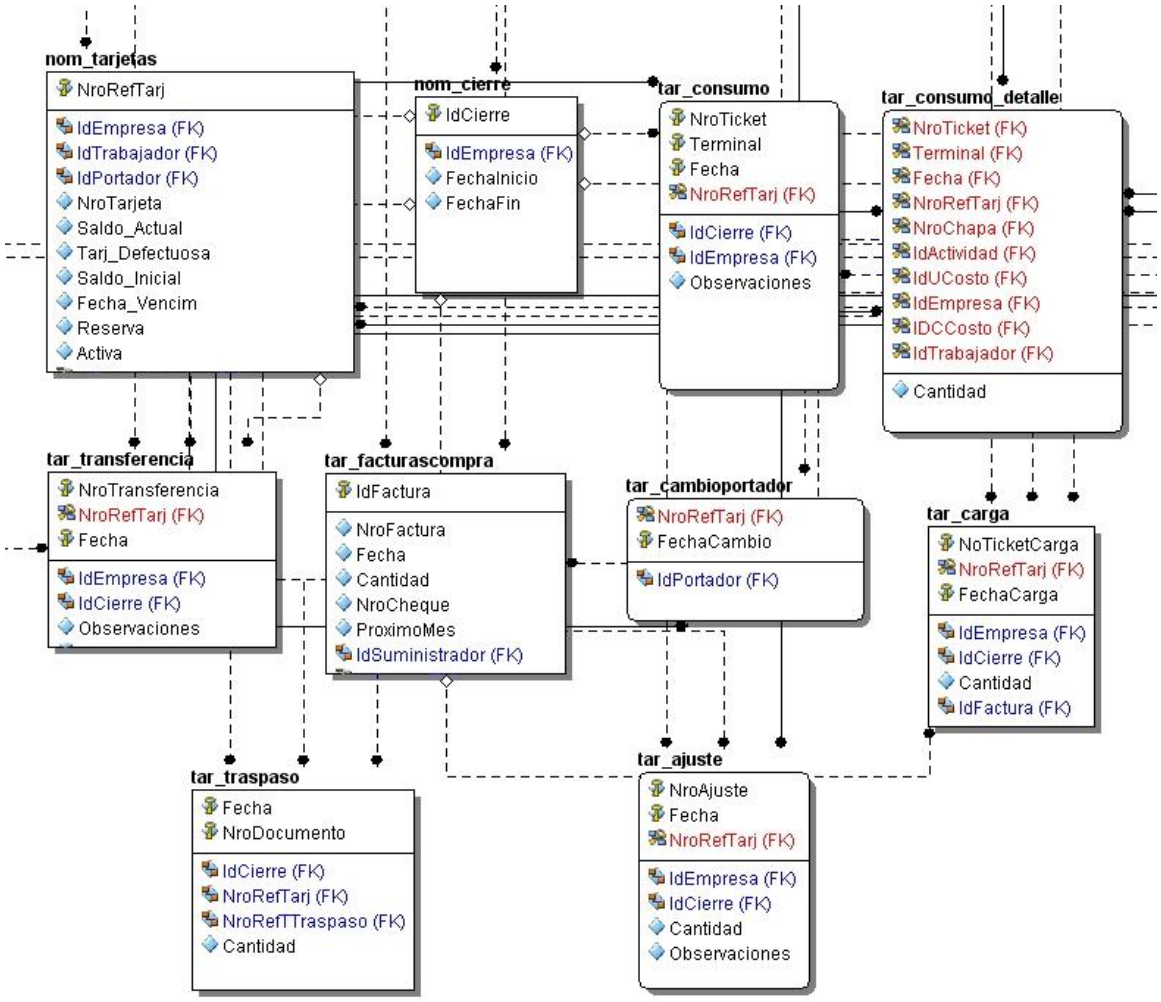

Figura 2.10. Diagrama Entidad-Interrelación asociado al trabajo con la Tarjeta Magnética.

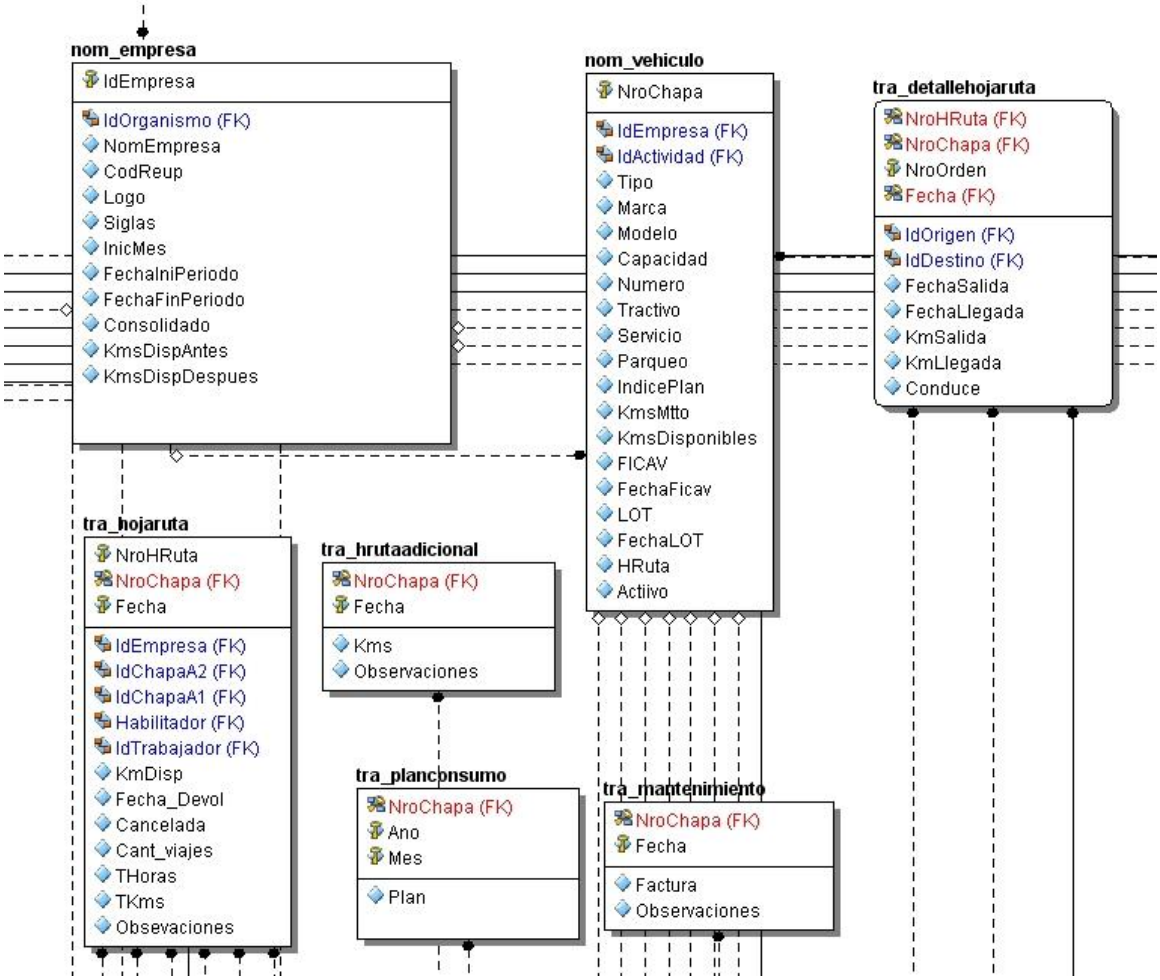

Figura 2.11. Diagrama Entidad-Interrelación asociado al trabajo con Transporte.

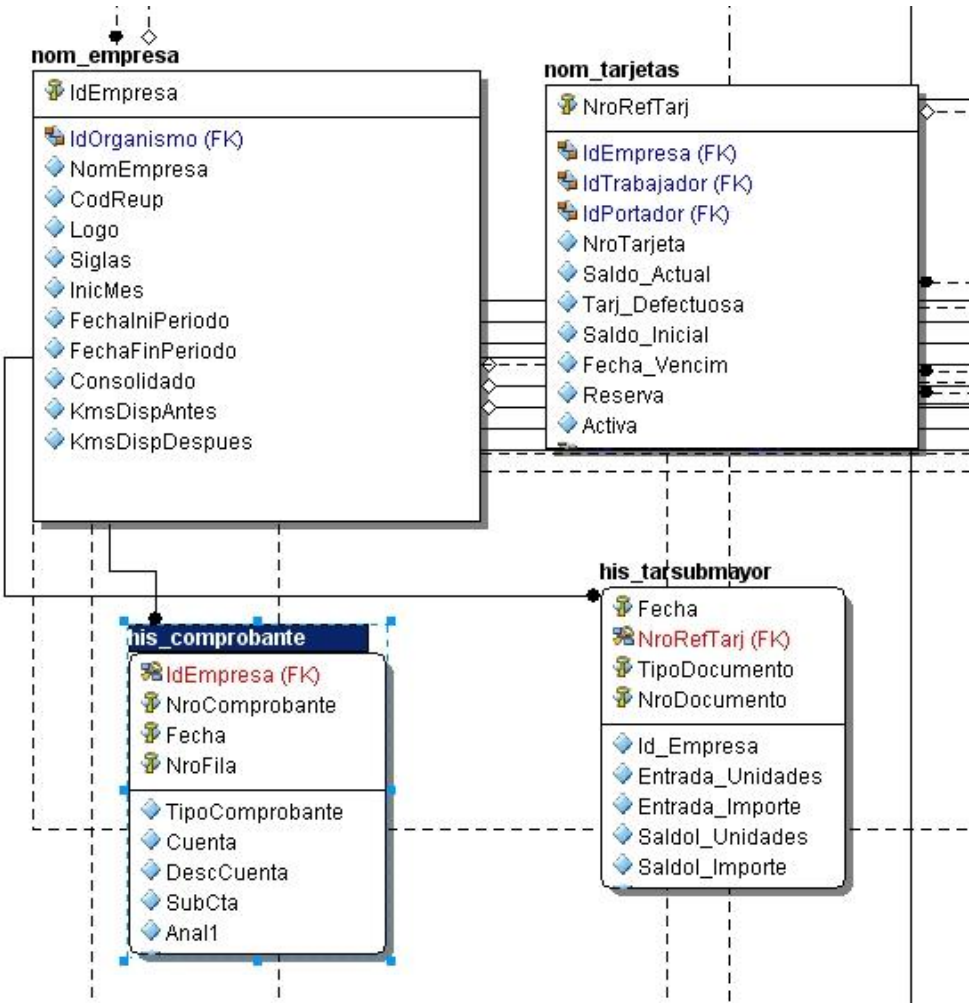

Figura 2.12. Diagrama Entidad-Interrelación
asociado al trabajo con el Histórico.

CAPÍTULO 3. IMPLEMENTACIÓN DE *ENERGUX*

El software libre ha devenido punta de lanza de un gran grupo de personas alrededor del mundo entero que forman comunidades de desarrollo que agrupan a disímiles culturas, nacionalidades e idiomas todos unidos en un solo interés: decir no al software propietario y dar una alternativa viable a compartir conocimiento para toda la humanidad. *Energux* surge como una herramienta de código abierto desarrollada con estándares de código abierto y de desarrollo sobre web, un medio que cada vez más se convierte en la principal plataforma de trabajo de la empresa moderna.

3.1 Código abierto

En inglés, *open source*, es el término con el que se conoce al software distribuido y desarrollado libremente. Fue utilizado por primera vez en 1998 por algunos usuarios de la comunidad del software libre, tratando de usarlo como reemplazo al ambiguo nombre original en inglés del software libre (*free software*). No obstante, el término continúa siendo ambivalente, puesto que se usa en la actualidad por parte de programadores que no ofrecen software libre pero, en cambio, sí ofrecen las fuentes o código de los programas para su revisión o modificación previamente autorizada por parte de sus pares académicos.

Dada la anterior ambivalencia, se prefiere el uso del término “software libre” para referirse a programas que se ofrecen con total libertad de modificación, uso y distribución bajo la regla implícita de no modificar dichas libertades hacia el futuro.

Free en inglés significa dos cosas distintas dependiendo del contexto: gratuidad y libertad. Lo cual implica, para el caso que nos ocupa, "software por el que no hay que pagar" (software gratuito) y, además, software libre, según la acepción española de libertad.

El término para algunos no resultó apropiado como reemplazo para el ya tradicional *free software*, pues eliminaba la idea de libertad, confundida usualmente con la simple gratuidad.

Desde el punto de vista de una "traducción estrictamente literal", el significado textual de "código abierto" es que "se puede examinar el código fuente", por lo que puede ser interpretado como un término más débil y flexible que el del software libre. Sobre esta base, se argumenta que un programa de código abierto puede ser software libre, pero también puede ser semilibre o incluso completamente no libre. Sin embargo, por lo general, un programa de código abierto puede ser y de hecho es software libre, como igualmente un programa Software Libre es o*pen source*. Esto ocurre dado que ambos movimientos reconocen el mismo conjunto de licencias y mantienen principios equivalentes.

Hay que diferenciar los programas *open source*, que dan a los usuarios la libertad de mejorarlos, de los programas que simplemente tienen el código fuente disponible, posiblemente con fuertes restricciones sobre el uso de dicho código fuente. Mucha gente cree que cualquier software que tenga el código fuente disponible es *open source*, puesto que lo pueden manipular. Sin embargo, mucho de este software no da a sus usuarios la libertad de distribuir sus modificaciones, restringe el uso comercial, o en general restringe los derechos de los usuarios.

La idea que está detrás del *open source* es sencilla: cuando los programadores en Internet pueden leer, modificar y redistribuir el código fuente de un programa, éste evoluciona, se desarrolla y mejora. Los usuarios lo adaptan a sus necesidades, corrigen sus errores a una velocidad impresionante, mayor a la aplicada en el desarrollo de software convencional o cerrado, dando como resultado la producción de un mejor software.

3.2 Implementación

Para la elaboración de la aplicación informática que da solución al problema planteado se han utilizado las siguientes herramientas:

- Java™ 2 SDK, Standard Edition 1.6 Sun Microsystems.
- *Visual Paradigm* para la realización de los diagramas UML.
- *Embarcadero ER/Studio* para definir el diseño lógico de la base de datos.

- *Eclipse IDE SDK* como IDE de trabajo con los siguientes *plugins*:
    - *Spring IDE* para la integración del marco de trabajo *Spring* en el IDE
    - *Subclipse*, para trabajar con SVN y el versionamiento del proyecto.
    - *Web Tool Platform* (WTP), para funcionalidades básicas para trabajar con proyectos Web.
    - *JBoss Tools*, como herramienta para el diseño y programación de aplicaciones web con la tecnología *JSF* y *Richfaces*, y que incluye *plugins* para el desarrollo de servicios web, modelado de procesos de negocio y el *Hibernate Tools*.
- Servidor de aplicaciones Web *Apache Tomcat*
- *Adobe Dreamweaver CS3* como IDE de diseño de páginas Web.
- Navegador de Internet, *Microsoft Internet Explorer* 6.0 o superior, *Mozilla Firefox* 3.0 o superior, *Opera* 9.0 o superior, *Google Chrome* 1.0 o superior.

Programación por capas

La solución informática desde el punto de vista de su desarrollo está compuesta por tres capas de programación, a saber, capa de modelo, controladora y de presentación. A continuación se describen las configuraciones y programación necesarias en cada una de ellas para el correcto funcionamiento de todos los marcos de trabajo que actúan dentro de la aplicación. Sirva este material de valor metodológico que puede resultar de gran utilidad para futuros proyectos que deseen basar su programación sobre las herramientas descritas.

3.2.1 Estructura del proyecto

En primer lugar es importante definir la estructura del proyecto de manera que la programación resulte sencilla y fácil de mantener. La decisión fue de estructurarlo de una forma similar al modelo de capas que se emplea en la propia aplicación. La estructura de carpetas se divide en siete carpetas de nivel superior (Figuras 3.1 y 3.2):

- Capa de datos o de Hibernate (*dao*) donde residen las clases de abstracción del modelo de datos, así como las configuraciones necesarias para la conexión con la base de datos. En estas clases de Java es donde se programa el trabajo con *Hibernate*, el marco de trabajo de acceso a datos.
- Capa controladora (s*ervice*) donde residen las clases controladoras de *Spring*. Aquí radican además los archivos de configuración para lograr la integración entre *Spring* y *Hibernate* así como la inyección de dependencia.
- Clases del modelo de base de datos (*entities*) donde se describen las tablas de la base de datos.
- Control de la seguridad (*security*) donde están los archivos necesarios para asegurar la integridad del uso del sistema.
- Espacio reservado para las pruebas de unidad (*test*)
- Capa de beans del Java (*web*). Es la capa donde se programan los beans que van a servir de puente entre la interfaz web (capa de presentación) y *Spring*, el marco de la trabajo de la capa controladora. En este espacio se implementan los beans de respaldo de las páginas, así como las clases controladoras de la colección de beans (*beanlist*).
- Por último, la carpeta *WebContent*, donde se almacena, en todo proyecto web sobre Java, las páginas en JSP que forman la capa de presentación de la aplicación.

Figura 3.1. Vista de la estructura del proyecto en el IDE Eclipse.

3.2.2 Clases de mapeo relacional de objetos

Uno de los elementos más importantes del uso de *Hibernate* como marco de trabajo para la abstracción de datos es el rol jugado a la hora de representar los datos y trabajar con ellos. Utilizando la técnica del mapeo relacional de objetos se logra abstraer el nivel de datos y hacer independiente a la aplicación del sistema gestor de base de datos usado en el sistema. Esto quiere decir sencillamente que si en algún momento se decide utilizar cualquier otro sistema gestor de base de datos solo basta con cambiar la configuración de la cadena de conexión en un fichero de tipo XML donde se proporcionan estos datos y no habría que realizar ningún otro tipo de cambio en el resto de la programación de los módulos.

Para lograr el mapeo de objetos, *Hibernate* utiliza clases en memoria que contienen una referencia a los campos de la tabla y especifica los detalles importantes a tomar en cuenta de los datos físicos mediante un sistema propio de anotaciones. En el sistema este mapeo se encuentra ubicado en la carpeta *entities* de la raíz del proyecto web.

```
package entities;

import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;

@Entity
@Table(name = "nom_moneda", schema = "public", uniqueConstraints =
@UniqueConstraint(columnNames = "descrip"))
public class NomMoneda implements java.io.Serializable {
        private String idmoneda;
```

```
private String descrip;
private Set<NomPortador> nomPortadors = new HashSet<NomPortador>(0);

public NomMoneda() {
}

public NomMoneda(String idmoneda, String descrip) {
        this.idmoneda = idmoneda;
        this.descrip = descrip;
}

public NomMoneda(String idmoneda, String descrip,
                Set<NomPortador> nomPortadors) {
        this.idmoneda = idmoneda;
        this.descrip = descrip;
        this.nomPortadors = nomPortadors;
}
@Id
@Column(name = "idmoneda", unique = true, nullable = false, length = 5)

public String getIdmoneda() {
        return this.idmoneda;
}
public void setIdmoneda(String idmoneda) {
        this.idmoneda = idmoneda;
}

@Column(name = "descrip", unique = true, nullable = false, length = 20)
public String getDescrip() {
        return this.descrip;
}

public void setDescrip(String descrip) {
        this.descrip = descrip;
}
```

```
	@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY,
	mappedBy = "nomMoneda")
	public Set<NomPortador> getNomPortadors() {
		return this.nomPortadors;
	}

	public void setNomPortadors(Set<NomPortador> nomPortadors) {
		this.nomPortadors = nomPortadors;
	}
}
```

3.2.3 Clases de acceso a datos

Para apoyar las clases de mapeo de las tablas físicas se elaboran las clases DAO (*Data Access Object*) que en primera instancia son las que se comunican al más bajo nivel con las clases de mapeo del Hibernate. Estas clases DAO hacen uso de los métodos y eventos que proporciona *Hibernate* para interactuar con los datos, que incluye HQL (*Hibernate Query language*), un modelo orientado a objeto para hacer consultas a la base de datos con el que no hay necesidad de un conocimiento profundo de SQL ni de sus sentencias. Usando la clase *Criteria* del objeto de sesión de *Hibernate* se pueden definir los filtros deseados sobre los datos de la base de datos.

```
package dao.impl;

import java.io.Serializable;
import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Restrictions;
import dao.interf.SegGruposUsuariosDAOInt;
import entities.SegGruposUsuarios;

public class SegGruposUsuariosDAOImp extends BaseDAOImpl<SegGruposUsuarios>
implements SegGruposUsuariosDAOInt{
```

```
	public List<SegGruposUsuarios> findAll() {
		Criteria c = getSession().createCriteria(SegGruposUsuarios.class);
		c.addOrder(Order.asc("idgrupo"));
		return c.list(); }

	@Override
	public SegGruposUsuarios findById(Serializable ... id) {
	return (SegGruposUsuarios)
	getHibernateTemplate().get(SegGruposUsuarios.class, id[0]);
	}

	public List<SegGruposUsuarios> findByNombre(String nombre){
		Criteria c = getSession().createCriteria(SegGruposUsuarios.class);
		c.add(Restrictions.eq("id.idusuario", nombre));
		return c.list();
	}

	public SegGruposUsuarios findByNombreEmpresa(String nombre, int empresa){
		Criteria c = getSession().createCriteria(SegGruposUsuarios.class);
		c.add(Restrictions.eq("id.idusuario", nombre));
		c.add(Restrictions.eq("id.idempresa", empresa));
		return (SegGruposUsuarios) c.uniqueResult();
	}
}
```

3.2.4 Clases de servicio

Las clases de servicio implementan funcionalidades a un nivel superior a las clases de acceso a datos. Estas clases hacen llamada a las clases DAO logrando una verdadera separación entre las capas de la aplicación. Corresponde a las clases controladoras del modelo de tres capas.

```
package service.impl;
```

```
import java.io.Serializable;
import java.util.List;
import dao.interf.SegGruposUsuariosDAOInt;
import entities.SegGruposUsuarios;
import entities.SegUsuarios;
import service.interf.SegGruposUsuariosServiceInt;

public class SegGruposUsuariosServiceImp implements SegGruposUsuariosServiceInt{
	private SegGruposUsuariosDAOInt segGruposUsuariosDAO;
	public SegGruposUsuariosDAOInt getSegGruposUsuariosDAO() {
		return segGruposUsuariosDAO;
	}

	public void setSegGruposUsuariosDAO(SegGruposUsuariosDAOInt
	 segGruposUsuariosDAO) {
		this.segGruposUsuariosDAO = segGruposUsuariosDAO;
	}

	public List<SegGruposUsuarios> getAll() {
		return  segGruposUsuariosDAO.findAll();
	}

	public SegGruposUsuarios getById(Serializable ... id) {
		return segGruposUsuariosDAO.findById(id[0]);
	}

	public List<SegGruposUsuarios> getByName(String nombre) {
		return segGruposUsuariosDAO.findByNombre(nombre);
	}

	public SegGruposUsuarios getByNameEmp(String nombre, int empresa) {
		return segGruposUsuariosDAO.findByNombreEmpresa(nombre, empresa);
	};

	public void createOrUpdate(SegGruposUsuarios obj) {
```

```
                segGruposUsuariosDAO.saveOrUpdate(obj);
        }

        public void remove(SegGruposUsuarios obj) {
                segGruposUsuariosDAO.delete(obj);
        }
}
```

3.2.5 *Beans* de respaldo

Las clases que forman parte de esta parte de la aplicación tienen como función principal lograr el enlace entre la interfaz visual de la aplicación y la capa controladora. Son las clases que representan en memoria los datos de la página que se muestra al usuario y realizan manipulaciones al nivel más alto de la aplicación. Es en estas clases donde se hace el manejo de todo lo que se presenta al usuario en pantalla, ya sea una lista, la creación de un elemento en tipo de ejecución, varias de las funcionalidades a nivel de cliente que se necesitan ejecutar.

```
package web.impl.bean;

import javax.faces.event.ActionEvent;
import entities.NomUm;
import web.util.FacesUtils;

public class UmedidaBean extends BaseBean {
        private int idum =0 ;
        private String descripc="";
        private Boolean canDelete = false;

        public int getIdum() {
                return idum;
        }

        public void setIdum(int idum) {
```

```
		this.idum = idum;
	}

	public String getDescripc() {
		return descripc;
	}

	public void setDescripc(String descripc) {
		this.descripc = descripc;
	}

	public Boolean getCanDelete() {
		return canDelete;
	}

	public void setCanDelete(Boolean canDelete) {
		this.canDelete = canDelete;
	}

	public UmedidaBean() {
	}

	public void save() {
		try {
			serviceLocator.getUmedidaService().createOrUpdate(toObject(this));
			descripc="";
			idum = 0;
			FacesUtils.setParameterInSession("Id_Umedida", idum);
		} catch (Exception e) {
			FacesUtils.addInfoMessage(e.getMessage());
		}
	}

	public void change(ActionEvent ev){
		try {
```

```
                    FacesUtils.setParameterInSession("Id_Umedida", idum);
                } catch (Exception e) {
                    FacesUtils.addErrorMessage(e.getLocalizedMessage());
                }
        }

    public static UmedidaBean toBean(NomUm obj){
                UmedidaBean bean = new UmedidaBean();
                bean.setIdum(obj.getIdum());
                bean.setDescripc(obj.getDescripc());
                return bean;
        }

        public static NomUm toObject (UmedidaBean bean){
                NomUm obj = new NomUm();
                obj.setIdum(bean.getIdum());
                obj.setDescripc(bean.getDescripc());
                return obj;
        }

        protected void init() {
                try {
                int ID = Integer.parseInt(
FacesUtils.getParameterInSession("Id_Umedida").toString());
                    if (ID>0) {
                        idum = ID;
                        UmedidaBean bean =
 toBean(serviceLocator.getUmedidaService().getById(idum));
                        populate(bean);
                        }
                    } catch (Exception e) {
                        FacesUtils.addErrorMessage(e.getLocalizedMessage());
                    }
        }
```

```
        public void populate(UmedidaBean bean) {
                this.idum = bean.getIdum();
                this.descripc = bean.getDescripc();

        }

        public void cancel() {
            try {
                descripc = "";
                idum = 0;
                FacesUtils.removeParameterInSession("Id_Umedida");
            } catch (Exception e) {
                FacesUtils.addInfoMessage(e.getMessage());
            }
        }

}
```

En nuestro modelo también se ha implementado una clase de lista asociada a cada *bean* para facilitar las operaciones con listas dentro de las páginas que se muestran en la interfaz visual.

3.2.6 La capa de presentación

Al nivel más cercano al usuario se encuentran las páginas Web, utilizando el marco de trabajo *Richfaces* para representar los componentes visuales.

```
<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:a4j="http://richfaces.org/a4j"
        xmlns:rich="http://richfaces.org/rich"
        template="../home/nomenclador.xhtml">
```

```
            <ui:define name="body">
    <div class="main-content">
      <h1 class="pagetitle">Registro de monedas</h1>

<f:view>
<!-- Grid para mostrar datos -->
<div id="contentDiv">
<h:form>
            <f:facet name="header">Visualización de datos</f:facet>
            <rich:dataTable id="table" cellpadding="0" cellspacing="0" rows="10"
                    width="100%" onRowMouseOver="
                    this.style.backgroundColor='#F1F1F1'"
                    headerClass="tableHeader"
                    onRowMouseOut="this.style.backgroundColor=
                    '#{a4jSkin.tableBackgroundColor}'"
                    border="0" value="#{monedaList.listBean}" var="monedaBean">
                    <f:facet name="header">
          <rich:columnGroup this.style.backgroundColor="#F1F1F1" >
            <rich:column styleClass="bkg" colspan="3" >
              <h:outputText value="Listado de monedas"/>
            </rich:column>
            <rich:column styleClass="bkg" breakBefore="true">
              <h:outputText value=""/>
            </rich:column>
            <rich:column >
              <h:outputText align="left" value="Identificador"/>
            </rich:column>
            <rich:column >
              <h:outputText align="left" value="Descripción"/>
            </rich:column>
          </rich:columnGroup>
        </f:facet>

        <rich:column styleClass="cbcolumn" width="3%">
          <f:facet name="header">
```

```
                    <h:outputText value=" " ></h:outputText>
                  </f:facet>
                  <h:selectBooleanCheckbox value="#{monedaBean.canDelete}" >
                  <a4j:support event="onchange"
                   action="#{monedaBean.cancel}" reRender="idmoneda,desc" >  </a4j:support>
                  </h:selectBooleanCheckbox>
                </rich:column>

                <rich:column sortable="true" filterBy="#{monedaBean.idMoneda}"
                  filterEvent="onkeyup" width="15%">
                 <h:outputText value="#{monedaBean.idMoneda}" />
                </rich:column>

                <rich:column sortable="true" sortBy="#{monedaBean.descrip}"
                   filterBy="#{monedaBean.descrip}" filterEvent="onkeyup" >
                  <h:commandLink value="#{monedaBean.descrip}"
                   actionListener="#{monedaBean.change}" />
                </rich:column>

                <f:facet name="footer">
                  <rich:datascroller styleClass="scroller" align="center" for="table"
                   maxPages="10" id="sc1" renderIfSinglePage="false"> </rich:datascroller>
                </f:facet>
        </rich:dataTable>
            <h:panelGrid columns="1">
                  <a4j:commandButton accesskey="E" styleClass="button" value="Eliminar"
                   action="#{monedaList.delete}" reRender="table"/>
            </h:panelGrid>

                  </h:form>
        </div>
                  <br/><br/>

        <!-- Panel donde se actualizan los datos -->
                  <h:form>
```

```
                <rich:panel style="padding:0" headerClass="outpanelHeader">
                <f:facet name="header">Datos</f:facet>
              <h:panelGrid columns="3">
                <h:outputText value="Identificador del tipo de moneda " />
                        <h:inputText id="idmoneda" styleClass="field"
               value="#{monedaBean.idMoneda}" maxlength="5" required="true"
              requiredMessage="Debe introducir un identificador para la moneda"
               disabled="#{monedaList.noModify}"/>
                        <br />
                        <h:outputText value="Descripción del tipo de moneda " />
                        <h:inputText id="desc" styleClass="field" value="#{monedaBean.descrip}"
                         maxlength="20" required="true" requiredMessage="Debe introducir el
                         tipo de moneda" />
              </h:panelGrid>
              </rich:panel>
              <h:panelGrid columns="3">
                        <a4j:commandButton accesskey="S" styleClass="button"
               action="#{monedaBean.save}" value="Salvar" reRender="idmoneda,desc,table" />
              <a4j:commandButton accesskey="C" styleClass="button"
               action="#{monedaBean.cancel}" value="Cancelar" reRender="idmoneda,desc,table" />
                        <rich:messages showDetail="true" showSummary="false" styleClass="error" />
              </h:panelGrid>
              </h:form>
        </f:view>
          </div>
        </ui:define>
        </ui:composition>
```

3.2.7 Los archivos de configuración

El trabajo con los marcos de trabajo en Java exige la presencia de varios ficheros de configuración que le indiquen al *Spring*, al *Hibernate* y al JSF cómo hacer las cosas, a quién dirigir sus llamadas y como resolver la búsqueda de dependencias. En el caso de las aplicaciones Web con Java todos estos ficheros se escriben son en formato XML. Son los siguientes:

- JavaSource/applicationContext-configuration.xml

  Proporciona la configuración de la conexión a la fuente de datos, el listado de las clases de mapeo de datos y otras configuraciones importantes

- JavaSource/hibernate.cfg.xml

  Proporciona la configuración del *Hibernate* para el acceso a la base de datos.

- JavaSource/dao/applicationContext-dao.xml

  Relaciona el nombre de las clases DAO con la clase física que implementa el objeto de la forma

```
<bean id="trabajadorDAO"
 class="dao.impl.TrabajadorDAOImp">
<property name="sessionFactory" ref="sessionFactory" />
</bean>.
```

- JavaSource/service/ applicationContext-service.xml

  Relaciona la clase controladora con el DAO correspondiente utilizando la inyección de dependencia de *Spring*.

```
<bean id="trabajadorService" parent="proxyTemplate">
        <property name="target">
         <bean class="service.impl.TrabajadorServiceImp">
                <property name="trabajadorDAO"
                        ref="trabajadorDAO" />
         </bean>
        </property>
</bean>
```

- WebContent/WEB-INF/faces-config.xml

  Define el alcance de los *beans* de respaldo, que especifica que nivel de ejecución y de importancia tiene cada página dentro del sistema

```
 <managed-bean>
 <managed-bean-name>trabajadorBean</managed-bean-name>
 <managed-bean-class>
web.impl.bean.TrabajadorBean</managed-bean-class>
 <managed-bean-scope>request</managed-bean-scope>
 <managed-property>
```

```
    <property-name>serviceLocator</property-name>
    <value>#{serviceLocator}</value>
   </managed-property>
  </managed-bean>
```

- WebContent/WEB-INF/web.xml

  Importante archivo que define muchas de las configuraciones generales comunes a toda la aplicación.

3.2.8 La seguridad de la aplicación

En la carpeta *security* del proyecto se definen las clases relativas a la seguridad del sistema, además de tener la presencia del fichero applicationContext-spring-security.xml ubicado en /JavaSource/security/

En las clases que componen el paquete se ejecuta el procesamiento de la autenticación, la seguridad de acceso a las páginas de la aplicación, el acceso a las mediante la definición de los roles de usuario y es el propio *Spring Security* quien le entrega a la aplicación la pertenencia del rol de seguridad en dependencia del usuario autenticado.

```
public class AuthenticationDAOImpl extends BaseDAOImpl<SegUsuarios> implements
        AuthenticationDAO {

    public UserDetails loadUserByUsername(String name)
            throws UsernameNotFoundException, DataAccessException {
        try {
        int emp =
        Integer.parseInt(FacesUtils.getParameterInSession("IdEmp").toString());

        SegUsuarios objUser = getUsuario(name.toLowerCase());
        SegGruposUsuarios objGU = getDatosUsuario(name.toLowerCase(), emp);

        //Set Application variables
        Navitas.setFechaActual(showDateFullStyle(new Locale("es","CU")));
        Navitas.setNombreUsuario(objUser.getNombre() + " (" + name + ")");
        Navitas.setIdNivelAcceso(objGU.getSegGrupos().getIdgrupo());
        Navitas.setNivelAcceso(objGU.getSegGrupos().getRol());
```

```
                Navitas.setUsuario(name);
                Navitas.setLogin(true);
                Navitas.setEmpresa(objGU.getNomEmpresa().getNomempresa());
                Navitas.setIdEmpresa(emp);
                Navitas.setMenuNivel1("Energux");
                Navitas.setMenuNivel2("Acceso rápido");

                //Get Data from NomEmpresa
                NomEmpresa e_act=
                 ServiceLocator.instance().getEmpresaService().getById(Navitas.idEmpresa);
                Navitas.setFechaFinPeriodo(e_act.getFechafinperiodo());
                Navitas.setFechaInicioPeriodo(e_act.getFechainiperiodo());
                Navitas.setInicMes(e_act.getInicmes());
                Navitas.setKmsCoberturaAntes(e_act.getKmsdispantes());
                Navitas.setKmsCoberturaDespues(e_act.getKmsdispdespues());

                //Security
                GrantedAuthority[] auths = new GrantedAuthorityImpl[1];
                auths[0] = new GrantedAuthorityImpl(objGU.getSegGrupos().getRol());
                return new User(name, objUser.getContrasena(), true, true, true,
                                true, auths);
                } catch (UsernameNotFoundException e) {
                throw new UsernameNotFoundException(
                "Usuario o contraseña incorrectos. Por favor, intente nuevamente.");
                }
        }
```

CAPÍTULO 4. *ENERGUX*: CONTROL DE PORTADORES ENERGÉTICOS.

EnerguX (*Energ* de Energía y *uX*, prefijo que distingue en la división provincial de la Empresa Nacional del Software a las aplicaciones desarrolladas en entorno de código abierto) se presenta ante el usuario con una interfaz desarrollada completamente en ambiente Web, agradable a la vista y con un funcionamiento bien sencillo. Sin pretender ser extensos ni hacer de este capítulo un manual de usuario, a continuación se detallarán algunos puntos interesantes de la interfaz de usuario de la aplicación lo que servirá de base para las especificidades técnicas desarrolladas en el siguiente capítulo.

Para la ejecución del software se requiere:

- Un servidor de aplicaciones web (Apache Tomcat)
- Sistema gestor de base de datos relacional PostgreSQL 8.0
- Un navegador web (Microsoft Internet Explorer, Mozilla Firefox, Google Chrome, Opera, etc...)

4.1 Autenticación

El sistema se presenta ante el usuario con una pantalla de autenticación a través de la que el usuario podrá introducir su nombre de usuario, contraseña y la empresa a la que pertenece. Si en este momento no está definido el servidor de base de datos sobre el que se trabajará se podrá seleccionar accediendo a un hipervínculo habilitado en esta pantalla que muestra una pantalla donde se podrán definir los parámetros de conexión.

Figura 4.1. Autenticación de EnerguX.

4.2 Pantalla principal del sistema

La pantalla principal de la aplicación está compuesta de varias partes y muestra de una sola vez todo el entorno gráfico disponible de manera que el usuario solo se moverá dentro de la aplicación a partir de los elementos que visualiza en esta ventana principal. Esta compuesta de una parte de identificación del producto, donde se exhibe el logo de la solución informática, el nombre y el número de la versión actual.

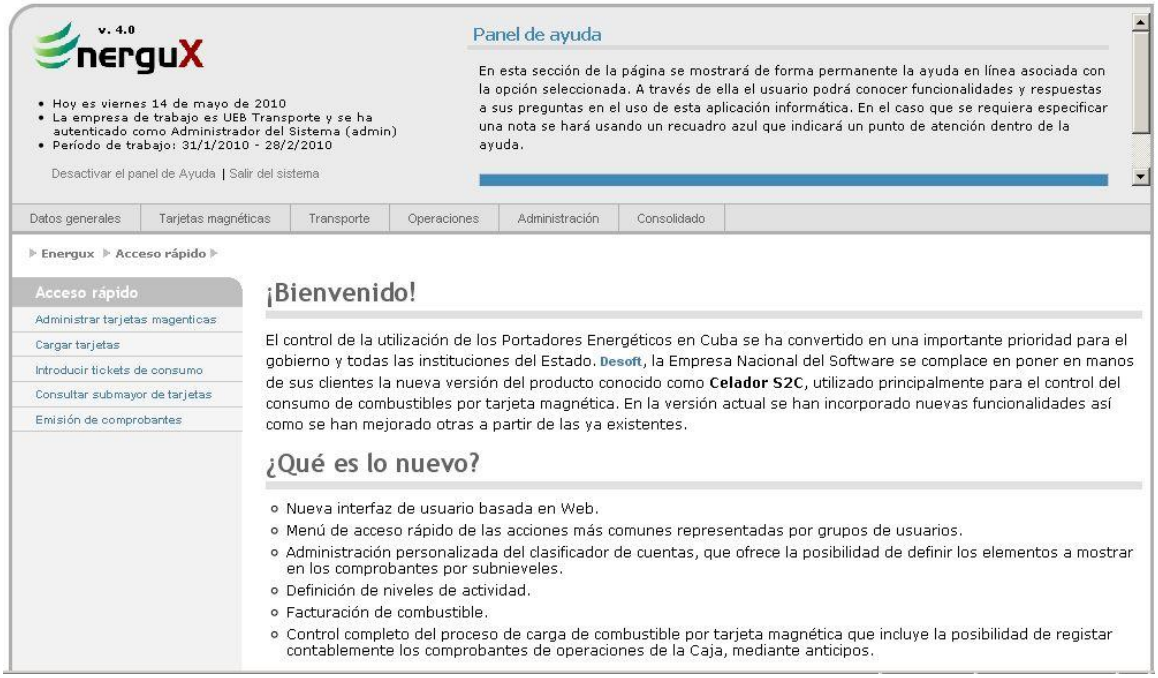

Figura 4.2. Pantalla principal de EnerguX.

Una sección superior con un panel informativo le proporcionará información rápida al usuario, sobre el día, la empresa activa y los datos de autenticación. De igual forma el período de trabajo actual definido en el sistema. Al lado de este sección se podrá encontrar el *Panel de Ayuda* que podrá ser ocultado o mostrado de acuerdo a la conveniencia del usuario

Figura 4.3. Vista ampliada y reducida del *Panel de Ayuda* de *EnerguX.*

A continuación un menú horizontal permitirá la activación de las distintas funcionalidades de nivel superior que contiene el sistema lo que activará, a su vez, un menú que se muestra debajo a la izquierda de la pantalla y que cambiará en dependencia de la opción seleccionada.

Figura 4.4. Menú principal de EnerguX.

Este submenú tiene la particularidad de estar agrupados por pequeñas secciones de trabajo ordenadas por subcategorías o conjunto de acciones. Además de dar orden al trabajo, hace que el uso del sistema sea más intuitivo. En el caso de la ventana principal del sistema, se muestra, en este menú, un listado de las acciones más frecuentes que el usuario suele realizar con el sistema de manera que pueda acceder a ellas de forma rápida sin tener que entrar a las diferentes opciones de menú y submenúes del sistema

Figura 4.5. Menú de opciones de EnerguX.

4.3 Una pantalla típica para introducción de datos

La pantalla típica de introducción de datos que acompañará al usuario en el trabajo con el sistema es resultado del diseño de una interfaz sencilla compuesta por los siguientes elementos:

- *Funcionalidad seleccionada*, que encabeza la página que se presenta.
- *Panel de visualización de datos*, que incluye una lista de los datos presentes en la tabla actual, con la posibilidad de que los usuarios marquen las filas de esta lista, de que visualicen el contenido de la misma y puedan hacer ordenamiento o filtrar los datos de forma rápida y sencilla. Dentro de este panel, un botón Eliminar que actúa con el objetivo de remover de los datos uno o varios de los elementos marcados con anterioridad en la lista.
- *Panel de Datos*, que se utiliza para interactuar con los datos de la tabla. En él se brinda la posibilidad de adicionar un nuevo elemento o modificar el contenido de uno existente. Adicionalmente, cancelar la edición de los datos del elemento actual.

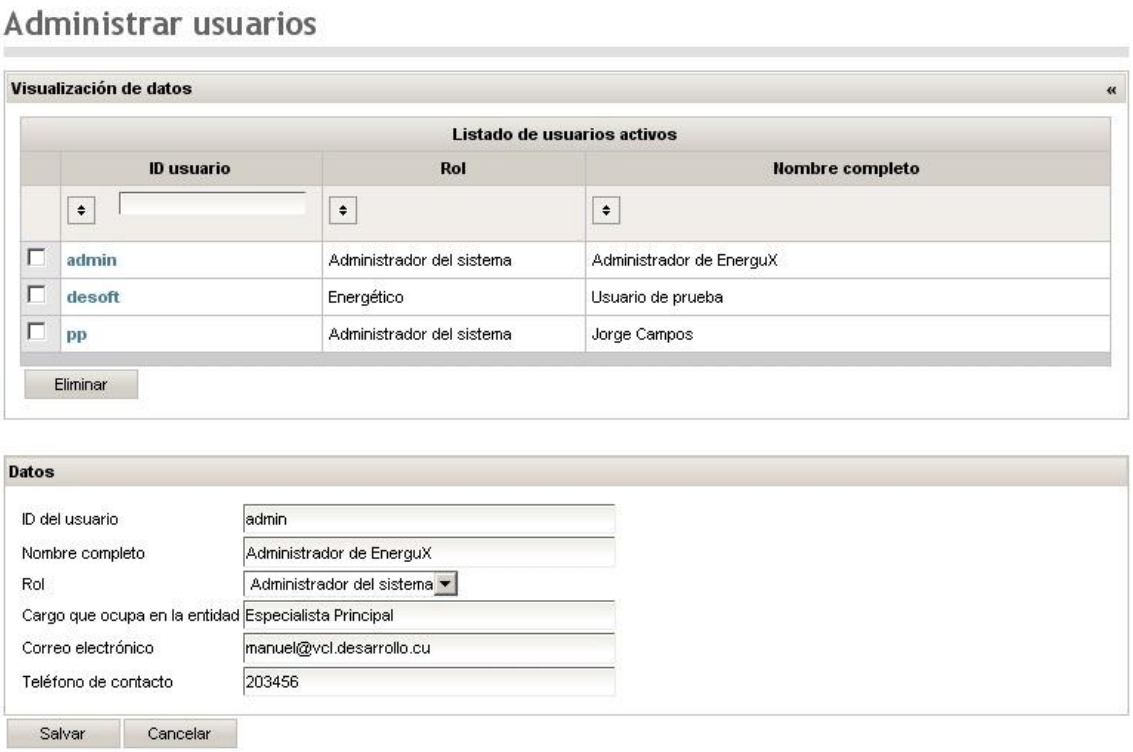

Figura 4.6. Una pantalla típica de introducción de datos.

4.4 Introducción de datos de forma jerárquica

Una de las funcionalidades que más ha mejorado desde el punto de vista de la experiencia del usuario ante la interfaz visual es la ostensible mejora de la entrada de datos en nomencladores con información jerárquica. En el *Celador S2C*, la introducción de datos en forma de árbol de dependencia se tornaba una tarea engorrosa y agotadora, puesto que había pantallas independientes definidas para lograr cada una de estas funcionalidades. En *EnerguX* dos de las funcionalidades típicas que tienen este comportamiento es la introducción de Centros y Unidades de Costos y la de los propios portadores energéticos.

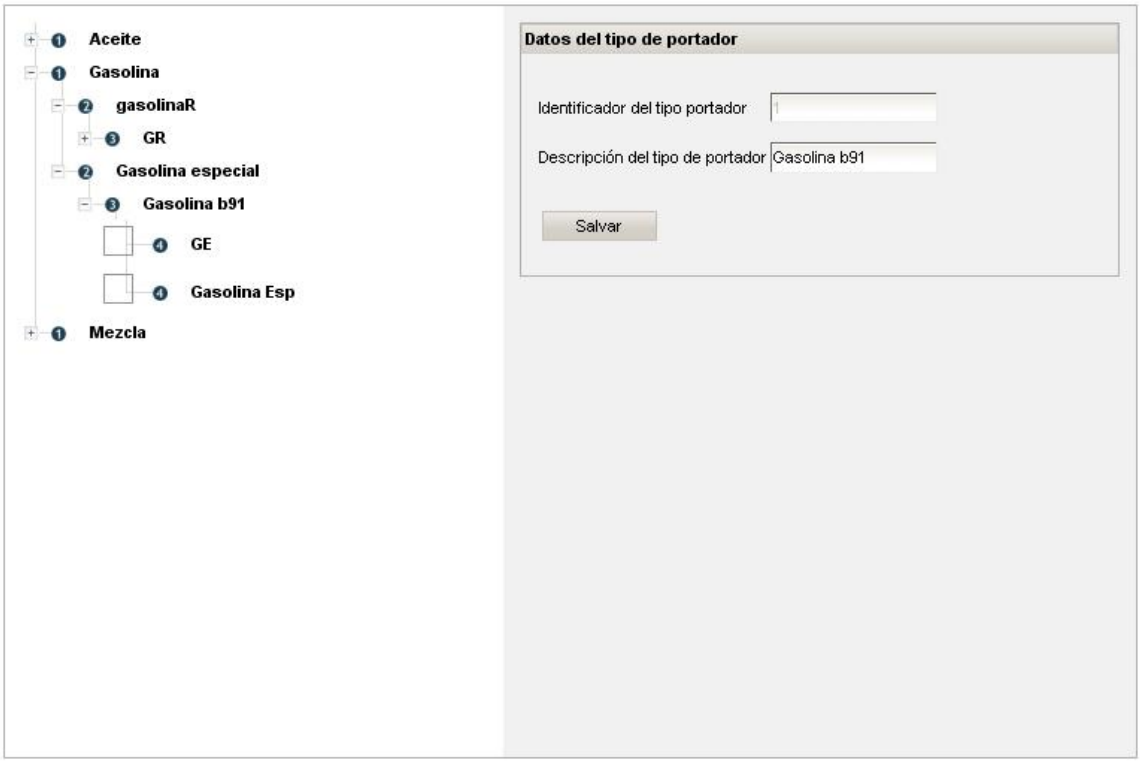

Figura 4.7. Introducción de datos en forma jerárquica.

4.5 Introducción de datos relacionados entre sí en una misma pantalla

Otra de las características que distinguen a la interfaz de usuario es la posibilidad que se brinda de introducir datos relacionados sin tener que abandonar la ventana de procesamiento. De forma notable, en la funcionalidad de Hoja de Ruta se identifica este comportamiento y se le ofrece al usuario la posibilidad de navegación a la introducción de más datos sin tener que abandonar la visualización del panel donde se muestran la información almacenada en la base de datos.

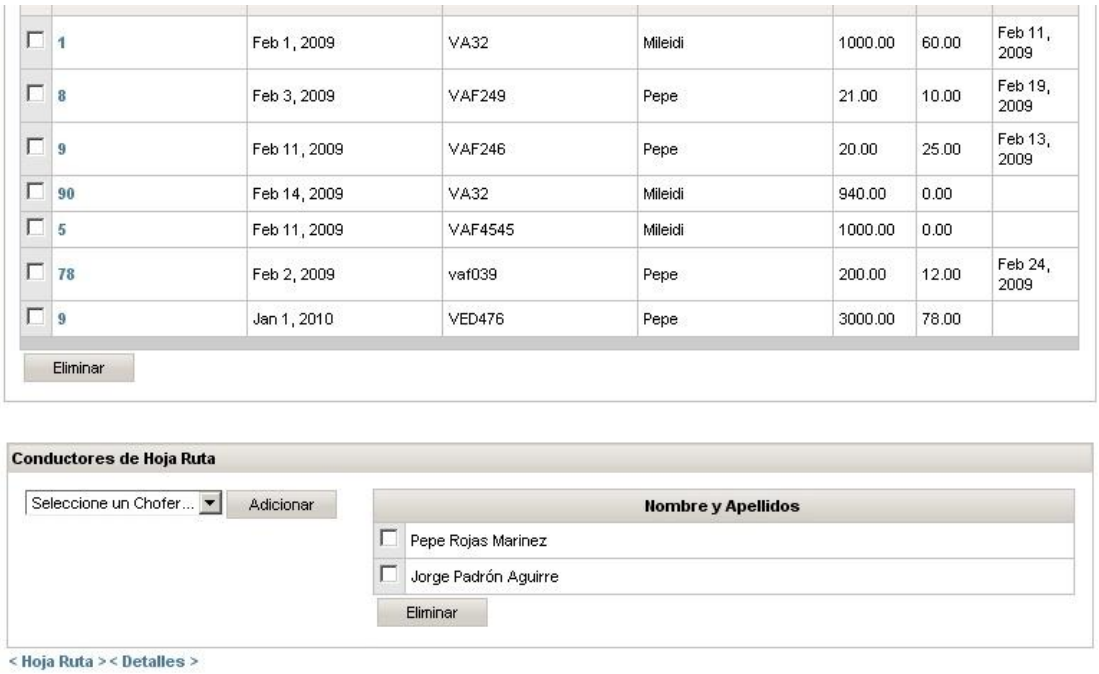

Figura 4.8. Introducción de datos relacionados en una misma pantalla.

4.6 Retroalimentación visual de las operaciones realizadas

En algunas de las funcionalidades que requieran de que el usuario sea informado detalladamente de los errores de una operación se ha puesto a disposición de la interfaz el panel de errores que le proporciona al usuario referencia al instante y de forma totalmente visual de las operaciones realizadas y del resultado de las mismas.

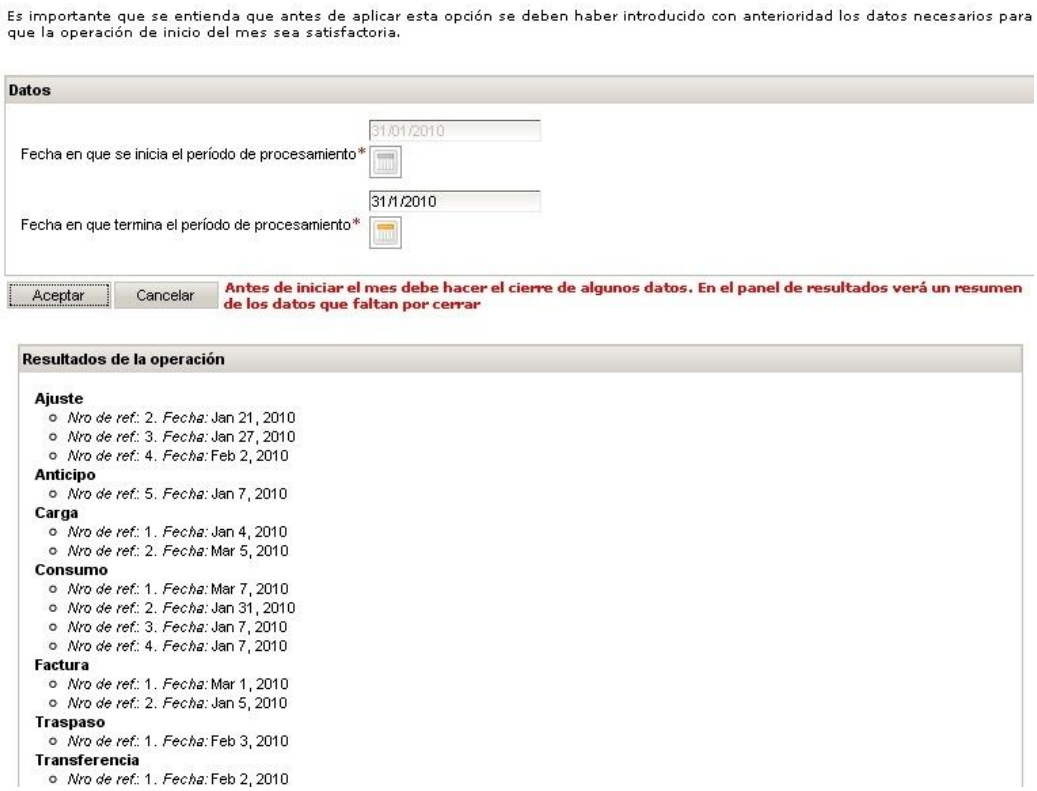

Figura 4.9. Notificación visual del estado de las operaciones.

4.7 Informes

Los informes se muestran a través del panel de Visor del informe disponible en las pantallas donde el usuario tiene acceso a ver, guardar o imprimir el resultado del procesamiento de los datos ya introducidos en el sistema. Antes de poder visualizar un informe con un conjunto de datos, se le pedirá al usuario la entrada de ciertos parámetros que servirán de filtro para seleccionar los datos que realmente se desean. Luego de seleccionados el usuario podrá visualizar el contenido de los datos a partir de los criterios seleccionando el botón *Ver informe*.

Figura 4.10. Selección de criterios para visualizar un informe.

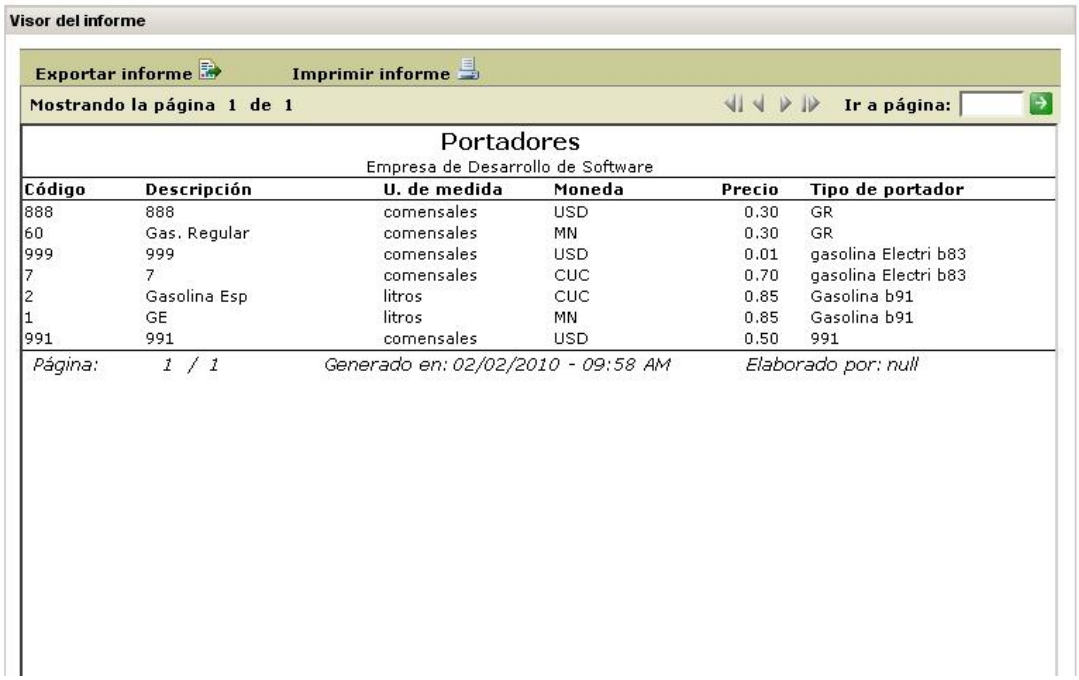

Figura 4.11. Visor del informe.

CONCLUSIONES

Como resultado de este trabajo se concluye:

1. Se realizó un estudio de los marcos de trabajo en Java y se procedió a la selección de *Richfaces* y *Facelets* para la capa de presentación, *Spring* para la capa de negocio, *Hibernate* para la capa de datos, *Spring Security* para gestionar la seguridad y *BIRT Report* para la generación de informes, por ser estos los más adecuados para solucionar los objetivos propuestos.

2. Se definieron los procesos de negocio de compra de tarjetas, facturación de combustible, carga de tarjetas, entrega y devolución, consumo de combustible y los movimientos internos que modifican el saldo y el estado de las tarjetas Magnética de Combustible.

3. Se realizó un análisis de las funcionalidades existentes en el *Celador S2C* y se determinó que existían características a mejorar en la nueva implementación como la mejora de la administración del Clasificador de Cuentas y la interfaz visual de la ventana de administración de los Portadores Energéticos. De igual manera, se detectaron funcionalidades a adicionar como las de introducir el registro de actividades y niveles de actividad y proceder a la automatización del registro de operaciones descrito por la Resolución 60 del 2009.

4. Se definió la arquitectura de la aplicación basada en tres capas: datos, negocio y presentación lo que permitió facilidad de implementación y separación del código dando la posibilidad de hacer una aplicación con componentes totalmente reusables y extensibles.

5. Se implementó la aplicación utilizando marcos de trabajo en Java y se comprobó la efectividad de la puesta en práctica de un método de trabajo y el diseño de una arquitectura de desarrollo que redundó en la elaboración de una aplicación Web con tecnologías modernas. Se sugiere, a la vez, la metodología que permita realizar futuros proyectos con herramientas similares.

RECOMENDACIONES

1. Implementar el control de plan y consumo para el resto de los portadores energéticos más usados en el mundo empresarial cubano tales como Electricidad (que incluye la generada por los grupos electrógenos), Gas Licuado, Diesel, Fuel Oil, Leña, entre otros, lo que redundará en un nuevo valor agregado para esta herramienta. De igual forma, llegar a obtener el Índice de Intensidad Energética (IIE), valor final que indica la eficiencia lograda desde el punto de vista energético.
2. Implementar la exportación de datos desde el sistema informático hacia cualquier fuente externa de datos usando la tecnología de *web services*. De igual manera, la importación ofreciendo la posibilidad de tomar datos a partir de un *web service* o de un archivo de datos.

BIBLIOGRAFÍA

- Agüero, M., 2007. *Introducción a Spring Framework*. Universidad de Palermo. Facultad de Ingeniería.
- Albin, S., 2003. *The Art of Software Architecture: Design methods and techniques*. New York: Wiley.
- Alur, D., Crupi, J. & Malks, D., 2003. *Core J2EE Patterns, Best Practices and Design Strategies*. 2nd ed. Prentice Hall.
- Apache, 2010. *Ajax4JSF*. [En línea] Disponible en: http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=Ajax4Jsf [Consultado 14 de enero de 2010].
- Apache, 2010. *My Faces*. [En línea] Disponible en: http://myfaces.apache.org/index.html [Consultado 31 de enero de 2010].
- Bachmann, F..e.a., 2000. *Technical concepts of component-based software engineering*. 2nd ed. Software Engineering Institute, Carnegie Mellon University.
- Bagüés, R., 2009. *Proactiva Calidad - Framework Spring*.
- Bass, L., Clements, P. & Kazman, R., 1998. *Software Architecture in Practice*. Addison-Wesley.
- Booch, G., Rumbaugh, J. & Jacobson, I., 1999. *The UML Modeling Language User Guide*. Adisson-Wesley.
- Bosch, J., 2000. *Design and use of Software Architecture*. Addison-Wesley.
- Brey, G.A.e.a., 2005. *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas*. Buenos Aires: Universidad Tecnológica Nacional. Facultad Regional de Buenos Aires.
- Buschmann, F..e.a., 1996. *Pattern Oriented Software Architecture: A System of Patterns*. London: John Wiley & Sons.

- Camacho, E.C.F.y.N.G., 2004. *Arquitecturas de Software. Guía de Estudio*.
- Campo, M.Á., 1999. *UBU - Guía de Iniciación al Lenguaje Java*. Universidad de Burgos.
- Fielding, R. & Taylor, R., 2002. *Principled design of the modern Web architecture*. ACM Transactions on Internet Technologies.
- Froufe, A., 1997. *Tutorial de Java*. [En línea] Universidad de las Palmas de gran Canarias Disponible en: http://www.ulpgc.es/otros/tutoriales/java/Intro/tabla.html [Consultado 5 de abril de 2010].
- García, S., 2009. *Hibernate*. Universidad Complutense de Madrid.
- Gómez, O.S., 2007. *Evaluando Arquitecturas de Software*. México DF: Brainworx S.A.
- JBoss, 2009. *Manual de referencia Spring*. JBoss. Disponible en: *http:*//docs.jboss.org/hibernate/core/3.5/reference/es/ [Consultado 25 de diciembre de 2009]
- Kruchten, P.1., 1999. *The Rational Unified Process*. Addison Wesley Longman, Inc.
- Marañón, G., 2008. *¿Qué es Java?* [En línea] Disponible en: http://www.iec.csic.es/CRIPTONOMICON/java/funcionamiento.html [Consultado 13 de diciembre de 2009].
- Microsystem, S., 2008. *Java en castellano*. [En línea] Disponible en: http://www.programacion.net/java/tutorial/patrones2/0/ [Consultado 21 de enero de 2010].
- Nuñez, J.M., 2003. *Investigación de la plataforma J2EE y su aplicacion practica*. Chile: Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas, Departamento de Ciencias de la Computación.

- O'Regan, G., 2004. *Introduction to Aspect-Oriented Programming*. [En línea] Disponible en: http://www.onjava.com/pub/a/onjava/2004/01/14/aop.html.
- Pérez, A., 2009. *Aplicaciones Web con Java*. [En línea] Disponible en: http://www.monografias.com//trabajos71/aplicaciones-web-java/aplicaciones-web-java.shtml [Consultado 24 de enero de 2010].
- Pressman, R.S., 2002. *Ingeniería de Software. Un enfoque practico*. España: McGraw Hil.
- Sanchez, C., 2004. *ONess: un proyecto open source para el negocio textil mayorista desarrollado con tecnologías open source innovadoras*. Coruña, España: Universidade da Coruña.
- Singh, I., Stearns, B. & Johnson, M., 2002. *Designing Enterprise Applications with the J2EE Platform*. 2nd ed. Addison Wesley.
- Weiss, M., 2009. *Data Structures and Problem Solving Using Java*. Addison Wesley.
- Wikipedia, 2007. *Mapeo objeto-relacional*. [En línea] Disponible en: http://es.wikipedia.org/wiki/Mapeo_objeto-relacional [Consultado 7 de abril de 2010].
- Wikipedia, 2010. *JSP*. [En línea] Disponible en: http://es.wikipedia.org/wiki/JSP [Consultado 2 de febrero de 2010].
- Wikipedia, 2010. *Reporting tools*. [En línea] Disponible en: http://en.wikipedia.org/wiki/Reporting_Tools [Consultado 23 de marzo de 2010].
- Wikipedia, 2010. *Spring*. [En línea] Disponible en: http://es.wikipedia.org/wiki/Spring [Consultado 2 de febrero de 2010].
- William, C., 2004. *Data Structures and the Java Collections Framework*. McGraw-Hill Osborne.

ANEXO A – LISTADO DE REQUISITOS FUNCIONALES

1. Realizar una identificación de los usuarios que tendrán acceso al sistema y determinar los privilegios que le corresponden.
2. Cambiar las contraseñas de acceso al sistema
3. Implementar una bitácora de sucesos que permita editar, eliminar y visualizar los registros de las acciones realizadas en el sistema.
4. Registrar los datos de la entidad y sus dependencias
5. Registrar parque de vehículos de la empresa.
6. Registrar lugares de destino para los vehículos de transporte.
7. Registrar suministradores de combustible.
8. Registrar clasificador de actividades y nivel de actividad de la empresa.
9. Registrar centros y unidades de costos.
10. Registrar portadores energéticos y sus especificidades.
11. Configuración de los clasificadores de cuentas y los comprobantes contables.
12. Registrar el listado de trabajadores de la empresa.
13. Registrar las monedas y unidades de medida utilizadas para el trabajo con el sistema.
14. Obtener listados informativos de los principales registros del sistema
15. Captar datos de nueva tarjeta.
16. Efectuar la facturación de la cifra de combustible asignada a la entidad.

17. Captar datos de la carga de combustible.
18. Captar datos de consumo de combustible.
19. Personalizar el trabajo con la tarjeta magnética

    Incluye las acciones de Cambio de Portador, Ajustes de la Tarjeta, Transferencias entre dependencias, Traspaso de Saldo, Activar y desactivar tarjeta
20. Registrar movimiento de datos entre la caja y los usuarios.
21. Exportar comprobantes a un sistema contable.
22. Importar datos desde un sistema contable
23. Obtener submayor de tarjetas.
24. Obtener comprobantes contables.
25. Obtener balance contable.
26. Obtener listado de facturas emitidas
27. Informar vencimiento de tarjetas.
28. Listar consumo de combustibles.
29. Efectuar operaciones contables.

    Incluye las acciones de Iniciar el período de proceso, Cambiar día de cierre, Efectuar cierre contable, Deshacer último cierre, Exportar comprobantes.

30. Registrar datos de la hoja de ruta.
31. Registrar planes de consumo de combustible.
32. Calcular índice de consumo.
33. Informar de la necesidad de mantenimiento de un vehículo.
34. Registrar los mantenimientos de un equipo.
35. Calcular kilómetros disponibles.
36. Calcular kilómetros recorridos.
37. Obtener informes de vencimiento del FICAV y LOT.
38. Consolidar datos de dependencias.
39. Registrar datos de la Hoja de Ruta para vehículos no automotores
40. Obtener listado de Hojas de Ruta
41. Cuadrar contablemente los anticipos y los consumos.

ANEXO B – LISTADO DE CASOS DE USO POR ACTORES

Entre paréntesis los requerimientos funcionales que cubre casa caso de uso y se muestran en negritas aquellos casos de uso que son nuevos o mejoran en la aplicación descrita en estas páginas.

Sistema o Administrador

- Autenticar usuarios (RF1)
- Administrar usuarios (RF1)
- Administrar grupos (RF1)
- Administrar permisos de grupos (RF1)
- Administrar permisos de usuarios (RF1)
- Administrar acceso a módulos (RF1)
- Ver bitácora de sucesos (RF3)
- Cambiar contraseña (RF2)

Contador

- Registrar empresas (RF4)
- **Registrar actividades y nivel de actividad** (RF8)
- Registrar organismos (RF4)
- Registrar Tipos de Monedas (RF13)
- Registrar Unidades de medida (RF4)
- **Registrar Portadores** (RF10)
- Registrar Centros y Unidades de Costo (RF9)
- Registrar dependencias (RF4)

- **Administrar clasificador de cuentas** (RF11)
- **Registrar tipos de comprobantes** (RF11)
- Registrar trabajadores (RF12)
- Obtener listado de las actividades (RF14)
- Obtener listado de los portadores definidos (RF14)
- Obtener listado de Centros y Unidades de Costo (RF14)
- Registrar nueva tarjeta magnética (RF15)
- Facturar combustible (RF16)
- Cargar tarjetas (RF12)
- **Entregar tarjetas** (RF20)
- Introducir tickets de consumo (RF18)
- **Devolver tarjeta** (RF20)
- Cambiar Portador a una tarjeta (RF19)
- Ajustar el saldo a una tarjeta (RF19)
- Transferir combustible e/ dependencias (RF19)
- Traspasar saldo de una tarjeta a otras(s) (RF19)
- Activar y desactivar tarjeta magnética (RF19)
- Obtener submayor de tarjetas (RF23)
- Obtener comprobantes (RF24)
- Obtener consumos según criterios (RF28)
- Obtener resumen de facturas emitidas (RF26)
- Obtener listado de tarjetas vencidas (RF27)
- Obtener balance contable (RF25)

- **Obtener informes de cuadre de anticipos vs consumos**
- Importar datos desde sistema contable (RF22)
- Exportar comprobantes a sistema contable (RF21)
- Deshacer el último cierre contable (RF29)
- Efectuar cierre contable (RF29)
- Iniciar el procesamiento de un mes (RF29)
- Cambiar día de cierre del período (RF29)
- Consolidar datos de dependencias (RF38)
- Obtener informe de cuadre de Anticipos vs Consumos (RF41)

Transporte

- Registrar parque de vehículos (RF5)
- Registrar Suministradores (RF7)
- Registrar destinos (RF6)
- Registrar kilómetros de cobertura (RF5)
- Obtener listado del parque de vehículos (RF14)
- Registrar Hoja de Ruta (RF30)
- Registrar hoja de ruta para otros vehículos (RF39)
- Registrar Mantenimientos (RF34)
- Registrar Plan de Consumo por vehículos (RF31)
- Obtener listado de mantenimientos (RF33)
- Obtener listado de control de entrega de Hojas de Ruta (RF40)
- Obtener informe de kilómetros recorridos (RF36)

- Obtener informe de vencimiento de FICAV (RF37)
- Obtener informe de vencimiento de LOT (RF37)
- Obtener índices de consumo (RF32)
- Obtener informe de Plan vs Real del Combustible (RF32, RF35, RF36)