UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

PROYECTO FIN DE CARRERA

Integración de un captador de datos biométricos en una intranet corporativa para el control y gestión de horarios

Pablo Quintana Quintana
Las Palmas de Gran Canaria, 2014

Proyecto fin de carrera de la Escuela de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria presentado por el alumno:

Pablo Juan Quintana Quintana

Título del Proyecto**:** Integración de un captador de datos biométricos en una intranet corporativa para el control y gestión de horarios

Tutor: Rodríguez Rodríguez Abraham
Cotutor: Vera Gómez Juan Alberto

A mis abuelos Antonio y Pura.

ÍNDICE

1. Introducción

Una joven empresa canaria, Edosoft Factory S.L, en adelante Edosoft, enmarcada en sector de la informática y las telecomunicaciones, en su afán por mejorar las condiciones laborales de sus empleados ha flexibilizado su jornada laboral. Así mismo, Edosoft dispone de una Intranet corporativa para la gestión de empleados y proyectos. Partiendo de la necesidad de realizar una gestión de horarios en el marco de esta nueva herramienta de uso corporativo se propone incorporar a la misma la funcionalidad necesaria para realizar las tareas de control y gestión de horarios. Como elemento hardware de captación de datos para el control se empleará un dispositivo biométrico.

Los captadores biométricos son sistemas diseñados para autentificar usuarios basándose en características únicas e inalterables de cada individuo (Huella Dactilar, la Retina, etc.) con el fin de autorizar su paso o simplemente, como en el caso que nos ocupa, registrar la hora de llegada o salida.

En el marco de esta memoria se explica todas las etapas del proceso de desarrollo del software de este módulo de gestión y control de horarios, comenzando por la selección de paradigma de desarrollo del software.

Al tratarse de un añadido, un conjunto de funcionalidades agregadas a un software existente, la intranet corporativa, se realizará un estudio previo de la misma a todos los niveles. Este estudio se realiza con el fin de plantear las diferentes etapas en función del software del que se parte, permitiendo siempre una óptima integración, anticipación de problemas y dificultades, y un diseño y arquitectura global coherente.

Cabe destacar como punto relevante y diferenciador con un desarrollo software puro que corra sobre una determinada plataforma, la necesidad del dispositivo hardware para la captación de datos biométricos y su interconexión con el mismo. La selección de la tecnología de captación de entre todas las existentes, y la elección de una tipología y modelo de terminal será otro de los punto clave en el desarrollo de este proyecto. Elección que vendrá marcada fundamentalmente por las necesidades del proyecto y de la empresa.

2. Gestión de proyecto

2.1 Selección del modelo de desarrollo del software

Un proceso para el desarrollo de software, también denominado ciclo de vida del desarrollo de software es una estructura aplicada al desarrollo de un producto de software. Hay varios modelos a seguir para el establecimiento de un proceso para el desarrollo de software, cada uno de los cuales describe un enfoque diferente para diferentes actividades que tienen lugar durante el proceso.

El modelo que más se ajusta a las necesidades de desarrollo es una adaptación del modelo en cascada. El modelo de cascada muestra un proceso donde los desarrolladores han de seguir las siguientes fases de forma sucesiva:

- Análisis de requisitos
- Diseño del software
- Implementación del software
- Integración
- Pruebas (o validación)
- Documentación
- Despliegue (o instalación)
- Mantenimiento

En el caso de este desarrollo, al ser un conjunto de funcionalidades nuevas que se añaden a una aplicación ya existente, la fase de integración formaría parte transversalmente de las siguientes etapas que componen el modelo: Análisis, Diseño e Implementación. Como parte del análisis comprendería una subetapa que se ha denominado Selección de modalidad de viabilidad de integración.

Se ha decido dividir el análisis en una serie de subetapas secuenciales debido a la fuerte componente de análisis que requiere el proyecto. De esta manera se diferenciará las tareas que corresponderían al Análisis de Requisitos propios del desarrollo en curso de otros análisis complementarios necesarios para completar exitosamente esta fase.

El despliegue comprende el paso a producción de la aplicación realizada, instalación y puesta en marcha en su entorno final de funcionamiento. En este caso al tratarse de un módulo de una aplicación existente, el despliegue consistirá en realizar la

entrega de código desde la rama de desarrollo del módulo a la rama de desarrollo de proyecto, validación mediante los test unitarios propios de la Intranet más los tests desarrollados para la nueva funcionalidad, etiquetado de la versión y finalmente actualización en el servidor de producción con la nueva versión de la Intranet.

Así mismo el despliegue también comprende la instalación física del terminal en instalaciones de Edosoft, dotándole tanto de tensión eléctrica como de conectividad de datos a la LAN de la oficina. Conforme a estas necesidades se escogerá el mejor lugar para su ubicación, el cual debe resultar también cómodo y accesible para los usuarios.

Todo lo correspondiente al control de la configuración que adelanta este capítulo para explicar la fase de despliegue se expone con mayor detalle en su el capítulo. 2.4 Gestión de la configuración.

La fase de mantenimiento como tal, al ser este un proyecto finito, objeto del PFC, queda fuera del alcance de este desarrollo enmarcándose en el mantenimiento de la Intranet. En vez de la inclusión de esta etapa se dedicará un capítulo en esta memoria a desarrollos futuros.

Así pues el modelo en cascada de nuestro desarrollo se ajustaría a las siguientes etapas:

- Análisis
    - Análisis previos (Estudio de la intranet corporativa y Estudio de los sensores biométricos)
    - Análisis de requisitos.
- Diseño del software
- Implementación del software
- Pruebas
- Despliegue
- Documentación

La elección de este modelo se justifica desde dos puntos fundamentales. El primero de estos es un análisis escrupuloso con requisitos cerrados. Al ser un desarrollo interno, los requisitos para la primera versión del software que corresponde a este proyecto serán captados por un Analista a partir de la información que darán otros

analistas conocedores de los procedimientos empresariales actuales y la Intranet. Unos requisitos en función de la herramienta ya existente de la que se parte, la nueva funcionalidad que se desea incluir, sujetos a la adquisición exitosa del terminal biométrico adecuado, y validados mediante el correspondiente análisis de viabilidad de integración a todos los niveles de la aplicación.

El segundo es que los recursos asignados al mismo es de tan solo una persona por lo que no tiene sentido el solapamiento de tareas, desarrollos agiles, etc.

2.2 Planificación e hitos

Tras la selección y adaptación del modelo de desarrollo la planificación realizada en la propuesta de proyecto ha sufrido algunas variaciones poco significativas, orientadas a una mejor gestión de proyecto según el paradigma seleccionado.

T1.- Gestión de proyecto. (40 h.)

Elección de un modelo de ciclo de vida del software y justificación.

Planificación de tareas, costes, establecimiento de hitos, identificación de riesgos y planes de contingencia.

T2.- Estudios previos. (200 h.)

Fase de adquisición de conocimientos necesarios para abordar el resto del proyecto con suficientes garantías. Esta tarea se descompone de las siguientes subtareas:

T2.1.- Estudio de la intranet corporativa.

Incluye el estudio en las tecnologías utilizadas en su desarrollo y la generación de un documento de diseño si no los hubiera.

T2.2.- Estudio de los sensores biométricos

T3.- Análisis de requisitos (50 h.)

T3.1.- Análisis de los requisitos de usuario y del software.

Incluye la generación del documento de requisitos del software.
T3.2.- Análisis de requisitos del hardware.

T3.3.- Análisis de viabilidad de integración

T4.- Selección y justificación del terminal (24h)

T5.- Diseño (80 h)

Realizar el documento de diseño a partir del documento redactado en la T3 incluyendo todos los diagramas UML que fueran necesarios para resultado óptimo.

T6.- Estudio y prueba de la interfaz para el manejo del Sensor biométrico seleccionado (120 h.)

Adquisición de todos los conocimientos necesarios en el manejo de la interfaz del dispositivo biométrico comprado. El entregable de esta tarea constara de los siguientes apartados:

Descripción y funcionalidad principal

Pruebas y ejemplos de uso

Dificultades halladas si las hubiera

T7.- Implementación (180 h)

Codificación del diseño realizado en los lenguajes de programación pertinentes.

T8.- Pruebas (200 h.)

Se analizarán, diseñarán e implementarán las pruebas, intentando abarcar al menos la implementación de los casos de prueba de mayor prioridad, en función su relevancia (posibilidad de fallo, impacto del fallo).

T9.- Documentación (40 h.)

Cada tarea tiene incluido en su dedicación la generación de documentación, por lo que esta tarea está concebida para unificar los contenidos en la Memoria de Proyecto, completarla y/o corregirla en el caso de que sea necesario.

Además se incluye como actividad en esta tarea la confección de la presentación de la lectura del proyecto.

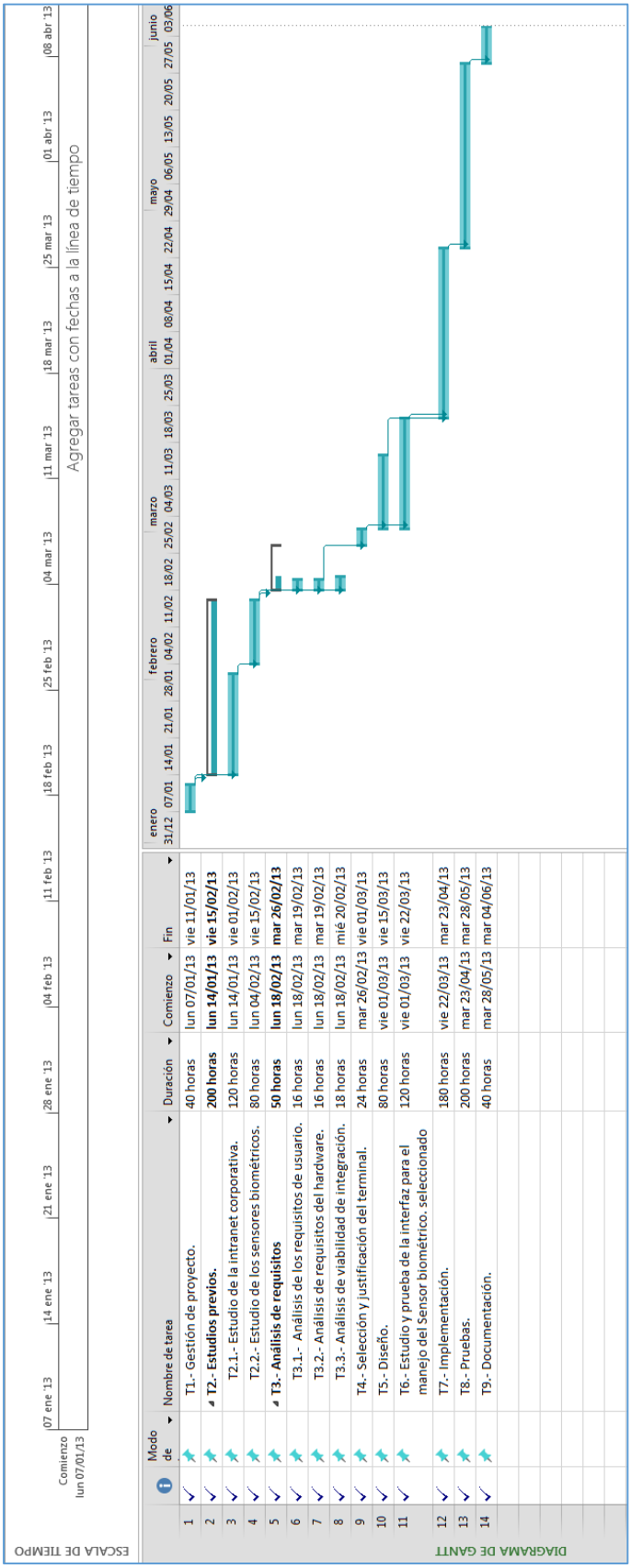

Figura 1. Diagrama de Gantt con la planificación del proyecto.

2.3 Gestión de riesgos

Los objetivos de la gestión de riesgos son identificar, controlar o eliminar las fuentes de riesgo antes de que empiecen a afectar al cumplimiento de los objetivos del proyecto.

Se han identificado los siguientes riesgos para el desarrollo del proyecto para los cuales se ha ideado uno o varias medidas de contingencia.

Retraso en la adquisición del terminal.

La compra del terminal biométrico, tal y como indica la planificación, debe efectuarse una vez concluida T4 y debe ser un hecho al comienzo de la T6. El riego más plausible es que los trámites de compra retrasen el poder disponer del terminal más del tiempo máximo que implica esta dependencia, retrasando el resto de las etapas y la consecución del proyecto. En este riesgo también se incluye la posibilidad de retraso debido a que una primera compra no cumpla las expectativas deseadas.

Para minimizar este riesgo se seleccionará un proveedor nacional y de garantías. En el caso que aun así se retrasara se podría ir abordando de forma adelantada la T7 implementando las capas menos relacionadas con la interfaz del terminal, GUI, modelo de datos. Pudiendo llegar a solicitar el manual de instrucciones del terminal para completar detalles del desarrollo dependientes del terminal biométrico adquirido, como los parámetros de configuración del mismo que mantendrá el modelo de datos.

Este riesgo, al verse influido por factores externos, no se elimina, pero con las medidas adoptadas si se reduce tanto la posibilidad que ocurra, como en su impacto en el proyecto en forma de retrasos temporales.

Impactos por modificaciones o fallos en la Intranet

Como ya se ha expuesto el desarrollo es dependiente de un software ya existente, que está sujeto a continuos cambios y posibles fallos que pueden impactar negativamente retrasando el desarrollo del proyecto.

Para minimizar esta dependencia se partirá de una versión estable de la intranet que no se modificará con nueva funcionalidad al margen de la del proyecto, a lo

largo de toda la vida del proyecto, excepto para incluir la solución a posibles fallos encontrados. Una vez concluido el desarrollo se planeará la entrega de la nueva funcionalidad en la versión en curso de la intranet. El equipo de desarrollo de la intranet tendrá en cuenta que existe un desarrollo paralelo evitando cambios que lo impacten y perjudiquen la entrega. Y el equipo de desarrollo del módulo de control y gestión de horarios se ceñirá a la planificación para no perjudicar el desarrollo de la intranet debido a esta dependencia.

En este caso el riesgo queda neutralizado por las medidas adoptadas.

2.4 Gestión de la configuración

Se denomina Gestión de la Configuración al conjunto de procesos destinados a asegurar la calidad de todo producto obtenido durante cualquiera de las etapas del desarrollo, a través del estricto control de los cambios realizados sobre los mismos y de la disponibilidad constante de una versión estable de cada elemento para toda persona involucrada en el citado desarrollo.

Como se ha adelantado en la gestión de riesgos el desarrollo de la nueva funcionalidad en la etapa de implementación se realizará en una rama de desarrollo del Sistema de control de versiones (CVS). Esta rama será exclusiva para este proyecto, partiendo de una versión de la rama principal de desarrollo, con contenido estable y probado de la intranet.

Tan solo la solución de fallos que impacten a la rama de desarrollo del módulo serán traídos desde la rama principal.

Finalmente se realizará un merge de entrega desde la rama de desarrollo a la rama principal.

El diagrama de la Figura 2 ilustra los procedimientos para el control de cambios a nivel de proyecto general que constituye el desarrollo de la intranet.

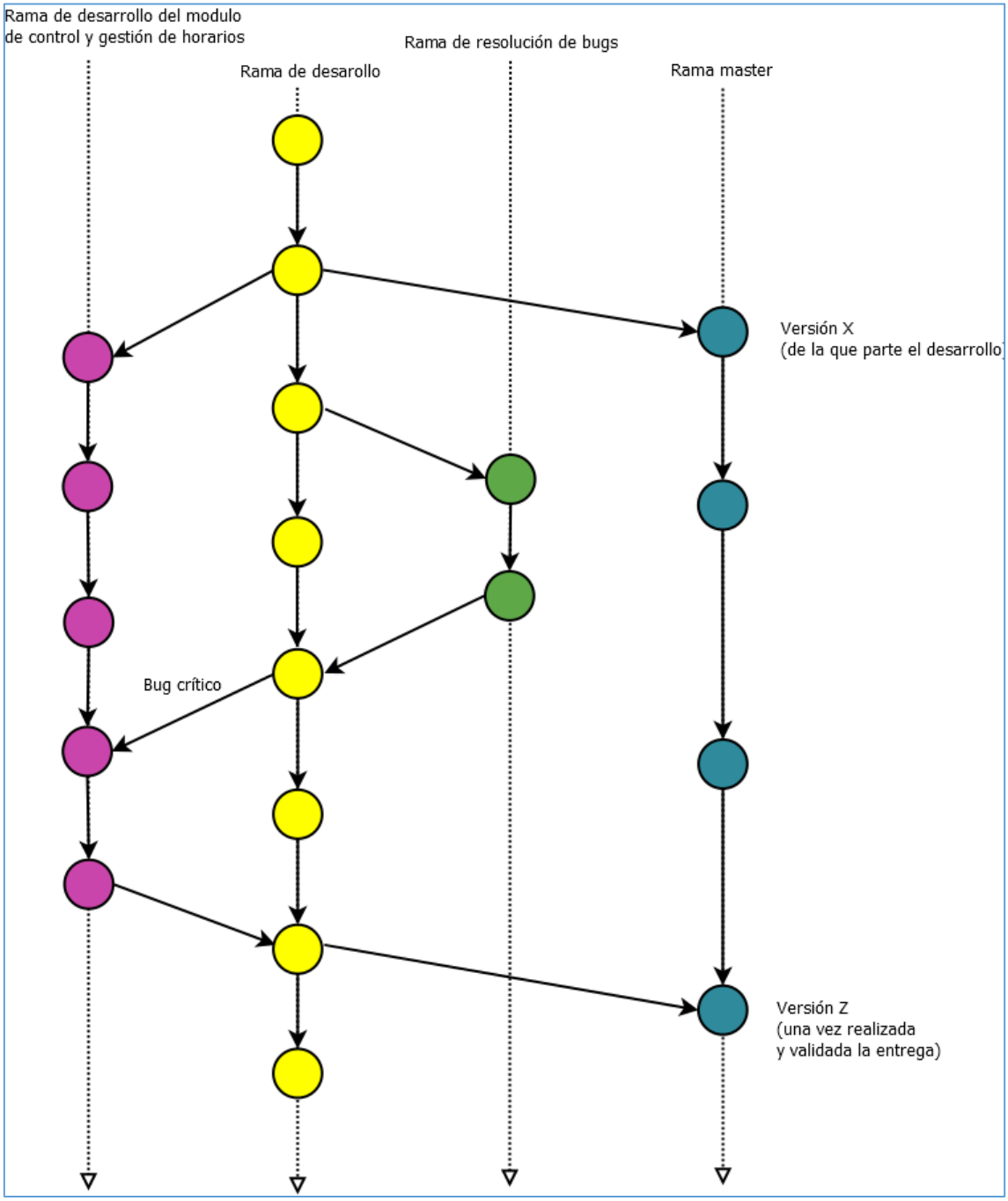

Figura 2. Diagrama de control de cambios.

3. Estudios previos

3.1 Estudio de la intranet corporativa

3.1.1 Introducción

La intranet corporativa de Edosoft Factory (ICEF) es una herramienta cliente servidor a media que permite la gestión integral de la empresa: empleados, proyectos, proveedores, clientes… En el estudio de la herramienta se describen las tecnologías y lenguajes empleados en las diferentes partes que componen la aplicación (Servidor: Modelo de negocio y Cliente: Interfaz gráfica de usuario). Además se aísla y expone el diseño, mediantes sus correspondientes diagramas, de todo los elementos relevantes para el desarrollo del proyecto. Se atienden a detalles de bajo nivel tales como la estructura de datos complejos almacenados en la base de datos que sean necesarios manejar para una correcta integración de la nueva funcionalidad, así como los detalles de implementación más significativos. Destacar por último que en este capítulo también se trata el estado actual de la Interfaz Gráfica de Usuario desde el punto de vista de su aspecto, con el fin de determinar a posteriori donde se ubicarán los nuevos elementos visuales para las diferentes vistas en función de roles (Administrador, Usuario, etc.)

3.1.2 Estudio de modelo de negocio

El modelo de negocio comprende el núcleo de la aplicación, el backend de la misma. Está formado por el modelo de datos que mantiene la información a ser mantenida y manejada. Y la lógica de negocio mediante la cual se realiza las diferentes operaciones sobre los datos.

3.1.2.1 Tecnología del modelo de negocio

El modelo de negocio de la herramienta está implementado en la plataforma de programación J2EE en su versión 1.4. Esta plataforma permite desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones. Java EE tiene varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc

y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), JavaServer Pages y varias tecnologías de servicios web.

Se lista a continuación las APIs de J2EE empleadas en la intranet, con una breve descripción de cada una de ellas:

- Enterprise JavaBeans: También conocidos por sus siglas EJB son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE. Los EJB proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJB 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.
  Existen tres tipos de EJBs:
  EJB de Entidad (Entity EJBs): su objetivo es encapsular los objetos del lado del servidor que almacena los datos.
  EJB de Sesión (Session EJBs): gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos:
  - Con estado (stateful): En un bean de sesión con estado, las variables de instancia del bean almacenan datos específicos obtenidos durante la conexión con el cliente. Cada bean de sesión con estado, por tanto, almacena el estado conversacional de un cliente que interactúa con el bean. Este estado conversacional se modifica conforme el cliente va realizando llamadas a los métodos de negocio del bean. El estado conversacional no se guarda cuando el cliente termina la sesión.
  - Sin estado (stateless). Los beans de sesión sin estado son objetos distribuidos que carecen de estado asociado permitiendo por tanto que se acceda concurrentemente. No se garantiza que los contenidos de las variables de instancia se conserven entre llamadas al método.

- EJB dirigidos por mensajes (Message-driven EJBs): son los únicos beans con funcionamiento asíncrono. Usando el Java Messaging System (JMS), se suscriben a un tema (topic) o a una cola (queue) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

- JavaMail: implementa el protocolo SMTP, así como los distintos tipos de conexión con servidores de correo -TLS, SSL, autentificación con usuario y password, etc.
- JAXP es el acrónimo de Java Api for XML Processing. API Java (definido por Sun Microsystems) sirve para la manipulación y el tratamiento de archivos XML.

Los beans de entidad se emplean para el manejo de la capa de datos. Estos beans abstraen y ocultan la base de datos subyacente, en este caso es una MySql versión 6.0. Cada bean de entidad corresponde a una entidad del modelo de datos: Empleado, Roll, Proyecto, etc.

Un único bean de sesión sin estado es empleado como fachada de la parte servidora de la aplicación. Este contiene todas las operaciones necesarias sobre el modelo de negocio, manejando para ello los beans de entidad, que son invocados por la parte cliente, la Interfaz gráfica de usuario. Se emplea un bean sin estado ya que el cliente web es una página web cuyo único contenido es una película Adobe Flash y se emplea para la comunicación el framework Granite Data Services (GraniteDS). Con el empleo de esta solución de integración no aplica el mantenimiento de la sesión por parte del servidor.

JavaMail es empleado para el envío automatizado de emails a los empleados para notificaciones de diferente índole: aprobación de SVPs (Solicitud de vacaciones y permisos) y PAs (Partes de actividad)

JAXP se emplea para el tratamiento de datos en formato XML empleados para el almacenamiento de información estructurada y para la comunicación cliente servidor.

Servidor de aplicaciones

La capa correspondiente al par servidor de la intranet se ejecuta, como es lógico, en el contexto de un servidor de aplicaciones.

Servidor de aplicaciones es un servidor en una red de computadores que ejecuta ciertas aplicaciones. Usualmente se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. [7]

En este caso el servidor de aplicaciones es el JBoss en su versión 4.2.2.GA.

Base de datos

Los datos almacenados mediante el empleo de EJB de entidad en el contexto del Servidor de aplicaciones son persistidos, mediante el uso de la librería Hibernate, en un servidor de base de datos. La base de datos, tablas, atributos, etc. quedan definidos a partir de la propia estructura que conforman estas entidades y sus anotaciones de Hibernate. El "data source", desplegado en el directorio deploy del Servidor de aplicaciones, y referenciado mediante JNDI por el archivo de configuración ubicado en el directorio META-INF de la aplicación, contiene el resto de datos para la adecuada configuración de Hibernate, para una correcta interacción con la base de datos. Esta es la forma habitual de configurar y estructurar los proyectos que emplean J2EE y la librería en cuestión.

En este caso el Servidor de Base de datos empleada es MySQL Server 5.1 y la librerías de Hibernate empleadas son las incluidas por defecto en el Servidor JBoss que se utiliza.

Seguridad

La autenticación de usuarios en la intranet se realiza mediante JAAS (Java Authentication and Authorization Service). JAAS es una interfaz que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso. Apareció como paquete opcional en la versión 1.3 y lleva siendo parte del estándar desde la versión 1.4. La autorización de accesos a diferentes elementos y procesos en función de los roles asociado a los usuarios no está implementada mediante JAAS.

3.1.2.2 Diseño del modelo de negocio

Este subcapítulo está compuesto por la información relativa al diseño de la Intranet, partiendo del estado en el momento de comienzo del desarrollo del proyecto. La mayoría de esta información, compuesta por diagramas y la explicación de los mismo, o era inexistente o no estaba actualizada, por lo que ha requerido un importante trabajo de reingeniería.

Se muestra en la Figura 3 el subconjunto del modelo de datos de la intranet, que serán necesario conocer para el desarrollo del nuevo módulo, mediante el correspondiente diagrama de Clases. En este se especifican tanto las clases del modelo que tendrán alguna relación con las nuevas clases a añadir o aquellas que son necesario manejar para la ejecución de la nueva funcionalidad.

Se incluyen vacías las clases que no son relevantes en el estudio que nos ocupa, a partir de las cuales se extiende el modelo de datos total original. Y la clase Empleado solo contiene los atributos más representativos y las relaciones relevantes.

En este se puede observar como existen dos subconjuntos de clases que parten de la clase Usuario, que mantiene todos los usuarios de la intranet con su login y password para el acceso.

Por un lado las clases (Rol, Rol_has_Accion, Acciones y Modulos) que, como ya se explicará más adelante son necesarias conocer y modificar su contenido para la gestión correcta de la inclusión del nuevo módulo en la GUI.

Y por otra parte las clases: Empleado, ParteVacacionesPermisos, ParteVacacacionesPermisosEstado y Calendario, con otras que las relacionan, que son necesario conocer para la realización de los nuevos procesos de la lógica de negocio. Sobre todo los encaminados a determina el carácter de los días (hábil, no hábil, festivo, vacaciones. etc) para cada empleado.

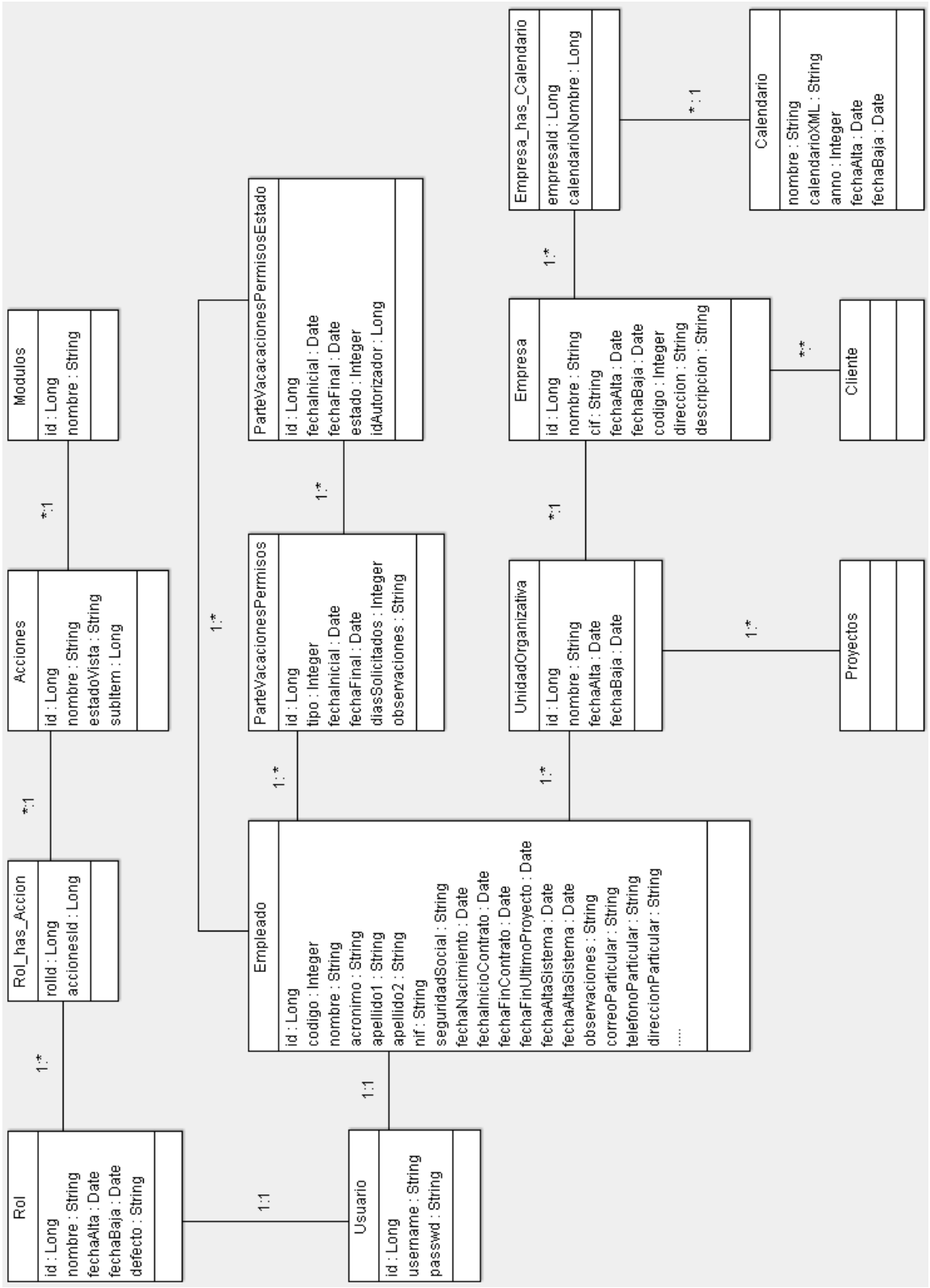

Figura 3. Diagrama de clases del modelo de datos de la Intranet corporativa.

3.1.2.3 Configuración del Servidor de aplicaciones y la base de datos.

El Servidor de aplicaciones JBoss 4.2.2.GA mantiene su configuración por defecto, la única modificación es la inclusión de algunas librerías de Java en el directorio `jboss-4.2.2.GA-icef\server\default\lib` para la realización de diferentes funciones:

- jasypt-1.3.jar: Para la encriptación y desencriptación de datos en la base de datos.
- backport-util-concurrent.jar, cfgatewayadapter.jar, commons-codec-1.3.jar, commons-httpclient-3.0.1.jar, commons-logging.jar, concurrent.jar, flex-ejb-factory.jar, flex-messaging-common.jar, flex-messaging-core.jar, flex-messaging-opt.jar, flex-messaging-proxy.jar, flex-messaging-remoting.jar, xalan.jar: Para la comunicación con el par cliente desarrollado en Flex, mediante el empleo de GraniteDS, y descargadas como parte de esta solución desde la página oficial.

Para la configuración de acceso a la base de datos se ha configurado 2 ficheros. El primero de ellos es `/META-INF/Persistemce.xml` donde se especifica, entre otras cosas, mediante un identificador JNDI el "data source" desplegado:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
       <persistence-unit name="IntranetEF" transaction-type="JTA">
              <jta-data-source>java:/MySqlDS</jta-data-source>
              <properties>
                     <property name="hibernate.hbm2ddl.auto" value="update"/>
               </properties>
       </persistence-unit>
</persistence>
```

Por otro lado el `jboss-4.2.2.GA-icef\server\default\deploy\ mysql-ds.xml` donde se almacenan datos para la conexión a la base de datos. Entre estos datos destacan, ubicación del servidor de base de datos, que incluye el nombre del servidor,

puerto y nombre de la base de datos, driver empleado, usuario y pasword de autorización al servidor de base de datos y nombre de la base de datos.

```
<datasources>
<local-tx-datasource>
    <jndi-name>MySqlDS</jndi-name>
    <connection-url>jdbc:mysql://localhost:3306/icef</connection-url>
    <driver-class>com.mysql.jdbc.Driver</driver-class>
    <user-name>root</user-name>
    <password>mysql5613</password>
    <exception-sorter-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLExceptionSorter</exception-
sorter-class-name>
    <!-- should only be used on drivers after 3.22.1 with "ping" support
    <valid-connection-checker-class-
name>org.jboss.resource.adapter.jdbc.vendor.MySQLValidConnectionChecker</valid-
connection-checker-class-name>
    -->
    <!-- sql to call when connection is created
    <new-connection-sql>some arbitrary sql</new-connection-sql>
      -->
    <!-- sql to call on an existing pooled connection when it is obtained from
pool - MySQLValidConnectionChecker is preferred for newer drivers
    <check-valid-connection-sql>some arbitrary sql</check-valid-connection-sql>
      -->

    <!-- corresponding type-mapping in the standardjbosscmp-jdbc.xml -->
    <metadata>
       <type-mapping>mySQL</type-mapping>
    </metadata>
  </local-tx-datasource>
</datasources>
```

Este archivo corresponde a un despliegue para desarrollo con un servidor de base de datos local, base de datos llamada icef y con el usuario root con la contraseña mysql5613, con todos los permisos para el acceso para la base de datos icef desde localhost. En función de otros despliegues, preproducción, producción este fichero deberá ser modificado.

3.1.2.4 Compilación y despliegue.

La compilación y despliegue de la aplicación se realiza mediante la ejecución de las diferentes task configuradas en el archivo build.xml mediante la herramienta Ant, en su versión 1.6.2. Entre estas task destacan:

- compile: Compila por completo la aplicación servidora generando un archivo jar.

- deploy: Realiza el despliegue del archivo jar y el "data source" en el directorio `deploy` del servidor de Aplicaciones ubicado en `C:\proyectos\as\jboss-4.2.2.GA-icef`. El jar estará incluido dentro del directorio `intranetEF.ear`. Esta tarea tiene dependencia con la anterior.

Tal y como se puede observar el `build.xml` esta solamente preparado para realizar un despliegue en un JBoss local en un entorno de desarrollo. Se identifica en este punto una mejora. La inclusión en este archivo de tareas de despliegue en servidores remotos que incluyan la modificación automática del fichero "data source" para facilitar el despliegue a otros hosts (producción, preproducción, demos, etc.)

3.1.2.5 Detalles de implementación, estructura de datos complejos y valores estáticos en persistencia

De las operaciones de fachada (`UsuarioFacade`) actualmente existentes en la lógica de negocio no será necesario la utilización de ninguna de ellas. De otra funcionalidad de apoyo se empleará la clase `EmailReport` y `ReportException` para el envío de emails y algunos métodos de la clase `Util`.

Cabe destacar que las entidades módulos, acciones y rol_has_action configuran dinámicamente los diferentes módulos que componen la interfaz de usuario, determinando los elementos en la barra de menús, y los elementos(acciones) que componen a su vez cada uno de esos menús en función del rol de usuario. A su vez cada opción de menú está asociado con un estado de vista asociado, que determina que componentes visuales se muestran. Será necesario añadir nuevos valores a los ya existentes para la creación dinámica en la GUI del acceso al módulo de control y gestión de horarios.

Se incluyen en este apartado aspectos de la implementación importantes tales como el manejo de logs y de excepciones, para realizar un uso adecuado y homogéneo del mismo. Para los logs se emplea la clase `ICEFLogger` que encapsula el manejo del apache.log4j con algunas particularidades propias del desarrollo.

Con respecto a las excepciones tan solo destacar la manera de proceder, la captura de las mismas en los métodos de fachada para que no pasen al cliente, y envío de código

de error al efecto. Es decir que por defecto las excepciones de servidor son tratadas en el propio servidor.

A continuación se incluyen los tipos de datos complejos (XML) que mantiene la aplicación y que serán necesario conocer:

Atributo calendarioXML de la clase Calendario

```
<calendario>
  <meses>
    <enero label="Enero">
      <festivos label="Festivos">5,12,19,26,6,13,20,27</festivos>
      <intensivo label="Intensivos"></intensivo>
      <noHabiles label="No habiles"></noHabiles>
    </enero>
      ……
    <diciembre label="Diciembre">
      <festivos label="Festivos">8,25,6,13,20,27,7,14,21,28</festivos>
      <intensivo label="Intensivos"></intensivo>
      <noHabiles label="No habiles">24,31,26</noHabiles>
    </diciembre>
  </meses>
</calendario>
```

Necesario para calcular si un día es hábil. El atributo *festivo* de cada mes corresponde a los fines de semana y el atributo *noHabil* a las festividades.

En cuanto a los valores estáticos almacenados que deben ser tenidos en cuenta, son los de la clase roll, mostrados en la Tabla 1, con el fin de poder añadir las acciones asociadas a cada roll que se pueden realizar en el nuevo módulo.

Id	Rol
1	Usuario
2	Gestor
3	Jefe UOI
4	Administrador
5	Gerente
6	Comercial

Tabla 1

Se concluye de este análisis del diseño e implementación que el nuevo módulo de control y gestión de horarios poseerá un grado mínimo de integración con la funcionalidad actual en el modelo de negocio.

3.1.3 Estudios de la interfaz

La interfaz de usuario consta de una serie de vistas, las cuales muestran información obtenida del servidor de aplicaciones, es decir, en el inicio de la aplicación se realizan peticiones a la lógica de negocio y devuelve la información necesaria para mostrarlas en las distintas vistas de la aplicación. Del mismo modo, el usuario puede realizar nuevas peticiones y si esa información no la contiene en su modelo de datos la peticiona nuevamente al servido

3.1.3.1 Tecnologías de la interfaz

La interfaz gráfica está desarrollada con Adobe Flex 3.0. Flex es un kit de desarrollo de software que agrupa una serie de tecnologías que permite construir aplicaciones visualmente rica, intuitiva y fácilmente configurables y facilita la comunicación con el modelo de negocio de la aplicación a través del patrón Modelo, Vista, Controlador.

Flex utiliza MXML para definir el diseño de interfaz de usuario y otros aspectos estáticos no visuales, ActionScript para hacer frente a los aspectos dinámicos y como código subyacente. La aplicación resultante puede ser ejecutada en Adobe AIR o, en el caso que nos ocupa, mediante Flash Player en el contexto de un navegador web.

Se ha combinado Flex con el empleo de Cairgorm. Cairngorm es uno de los principales framework de código abierto para la arquitectura de la aplicación en Adobe Flex.

Cairngorm se basa en el modelo MVC, que se divide en tres partes:

- El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones. Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea visible (típicamente a un usuario). Las

peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'12 .

- El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información. El 'controlador' hace de intermediario entre la 'vista' y el 'modelo'.
- La Vista: Presenta el 'modelo' en un formato adecuado para interactuar (usualmente la interfaz de usuario) por tanto requiere de dicho 'modelo' la información que debe representar como salida.

Se ha diseñado específicamente para facilitar la sincronización de estado y de datos complejos entre el cliente y el servidor, mientras se mantiene la programación de la capa de la vista separada de la aplicación de datos. Como la utilización de este framework define el diseño de la aplicación se expondrá el mismo en el siguiente subcapítulo. Para la inclusión de nueva funcionalidad en la vista será imprescindible dominar en profundidad el empleo de este framework.

Como se ha apuntado previamente la interfaz gráfica recurre a GraniteDS para la comunicación con el servidor. GraniteDS es la solución de integración para crear aplicaciones RIA Flex / JavaEE, GraniteDS es de código abierto y liberado bajo la licencia LGPL v2.license y la alternativa a Adobe LiveCycle (Flex 2) de pago en los momentos iniciales de desarrollo de la intranet.

Para la implantación del GraniteDS se emplea a su vez Gas3, herramientas de generación de código que ayudan a replicar Java beans y servicios de la entidad del lado del servidor en su equivalente ActionScript3 del lado del cliente. Estas herramientas están disponibles como Eclipse plug-in o una tarea Ant, que es la que se utiliza en este caso.

3.1.3.2 Diseño

Como ya se ha indicado el diseño de la arquitectura de vista está basado en el framework Cairgorm que implementa el patrón MVC. En el diagrama de la Figura 4 muestra el esquema de funcionamiento.

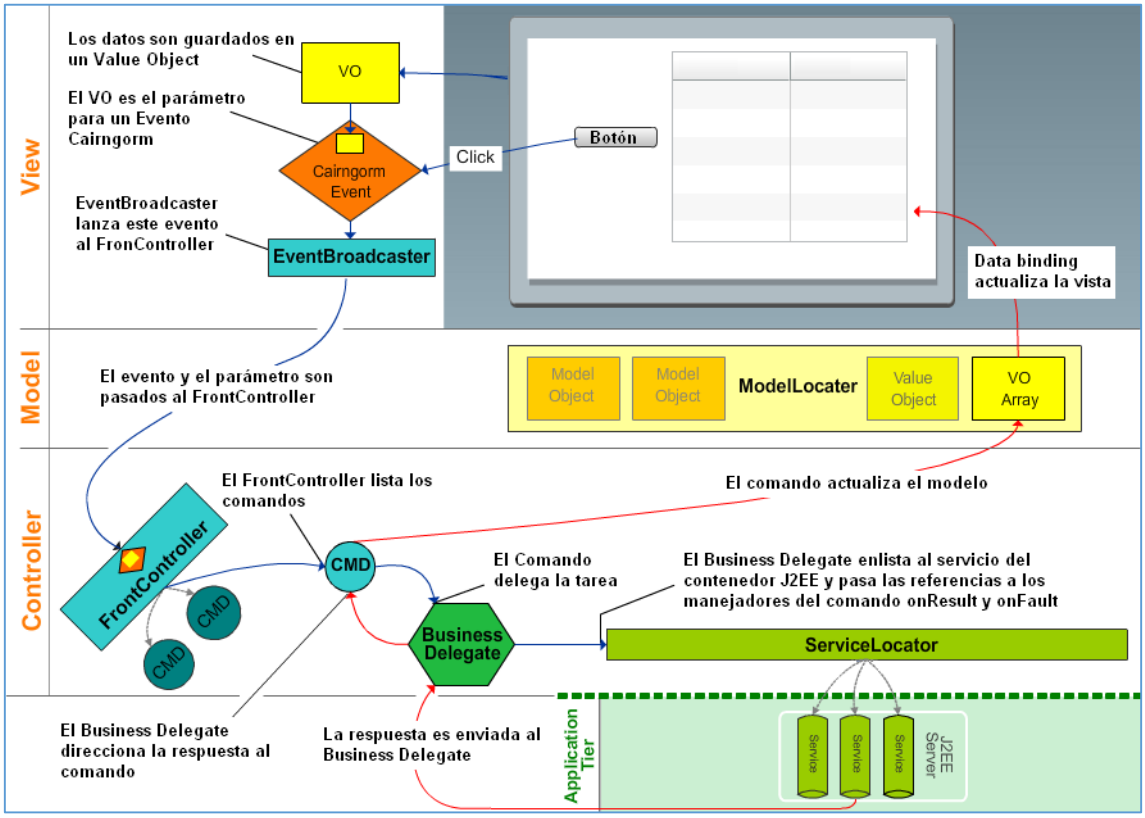

Figura 4. Diagrama de funcionamiento del Cairngorm.

El papel de la capa de la vista en una aplicación de Cairngorm es lanzar eventos y enlazarse a los datos almacenados en el modelo. Componentes de la vista pueden unirse a objetos de valor y otras propiedades de la capa de modelo (datos).

En un modelo de Cairngorm, los datos relacionados se almacenan en objetos de valor (VOs), mientras que las variables simples pueden ser almacenadas como propiedades directas de la clase `ModelLocator`. Una referencia estática a la instancia singleton `ModelLocator` es utilizado por las capas para localizar los datos requeridos.

El controlador es la parte más sofisticada de la arquitectura Cairngorm. La capa de controlador se implementa como un `FrontController` singleton. La instancia `FrontController`, que recibe todos los eventos generados, y despacha de los eventos a la clase de comando asignado según el tipo declarado del evento.

La clase de comandos a continuación, procesa el evento mediante la ejecución de método `execute` de la clase `Command`, que es un método de interfaz `ICommand`. El objeto de evento puede incluir información adicional si lo requiere el desarrollador. El método `execute` puede actualizar el modelo central, así como invocar una clase de servicio que

normalmente implica una comunicación con un servidor remoto, a través del Business Delegate. La interfaz `IResponder`, que también está implementada por la clase de comandos, incluye los métodos `onResult` y `onFault` para manejar las respuestas devueltas por el servicio remoto invocado.

3.1.2.4 Compilación y despliegue.

La compilación y despliegue de la interfaz gráfica se realiza con el propio entorno de desarrollo, incluyendo como parámetros de compilación:

```
-services" C:\proyectos\as\jboss-4.2.2.GAicef\server\default\
deploy\intranetEF.ear\intranet.war\WEB-INF\flex\services-config.xml"
-context-root "."  -locale en_US
```

El directorio `WEB-INF\flex` contiene los archivos necesarios para la correcta comunicación con el servidor empleando GraniteDS.

Y como salida de la misma:

```
jboss-4.2.2.GA-icef\server\default\deploy\intranetEF.ear\intranet.war
```

Estas configuraciones forman parte de las propiedades del proyecto Flex que constituye la Gui y se muestran ya configuradas tras la importar el proyecto con Flex Builder. El directorio flex con su contenido se puede obtener de la web oficial de GraniteDS. Los directorios `intranet.war\WEB-INF` deberán ser creados, tras el primer despliegue de la parte del servidor en el servidor de aplicaciones.

3.2 Sensores biométricos (Estado del arte)

3.2.1 Introducción

La biometría consiste en la identificación de un individuo a través del análisis de sus características físicas o rasgos conductuales. El término se deriva de las palabras griegas "bios" de vida y "métron" de medida. [1]

Los captadores biométricos son sistemas diseñados para conceder acceso a usuarios basándose en características únicas e inalterables de cada individuo que difícilmente pueden ser duplicadas, perdidas, olvidadas o robadas, pretendiendo

reconocer quien es el individuo, opuestamente a los sistemas de acceso más comunes basados en objetos (llaves, tarjetas RFID…) o códigos que pueden ser perdidos, olvidados, robados o descifrados. [2][3][4]

Son precisamente estas características físicas o conductivas de cada individuo lo que ha impulsado el desarrollo de distintos tipos de sistemas para su identificación; desde el punto de vista conductivo es posible analizar la caligrafía o la voz de un individuo, siendo esta última donde más esfuerzos se han aplicado desde una perspectiva tecnológica. Pero sin duda alguna el desarrollo de los sistemas biométricos, centra sus esfuerzos en la identificación de las características físicas como son, por ejemplo, la Huella Dactilar, la Retina, el Iris, la Geometría de la Mano o la Cara.[3][5]

2.2.2 Funcionamiento y rendimiento

El funcionamiento habitual en un Sistema Biométrico, consta de un proceso previo de registro, donde se adquieren uno o varios de los parámetros biométricos morfológicos o fisiológicos del usuario, se procesan mediante algún algoritmo numérico para finalmente ser almacenados en una base de datos con la información única para cada individuo. Posteriormente en el proceso de autentificación o identificación de un usuario, el sistema obtiene y procesa las características biométricas, las compara con las almacenadas en la base de datos y si concuerda, valida la operación.

Alguno de los parámetros definidos para medir el rendimiento de un sistema biométrico son; [3][4][6]

Tasa de Falso Positivo (False Acceptance Rate o FAR); es la probabilidad de que un usuario no registrado sea aceptado en cualquier caso. Mide la frecuencia con que un usuario no registrado es admitido por equívoco, actualmente el margen esta entre el 0.0001% y el 0.1%. [5] Y asume intentos pasivos del usuario. A partir del FAR se puede obtener una medida de la seguridad del sistema de verificación biométrica, según la ecuación, seguridad = (1 – FAR).

Tasa de Falso Reconocimiento (False Match Rate o FMR). Es la tasa con que los usuarios no registrados son confundidos durante un proceso de comparación de características.

Tasa de Falso Rechazo (False Rejection Rate o FRR); es la probabilidad de que un usuario este registrado pero sea rechazado, Mide la frecuencia con que un usuario registrado es rehusado. Comercialmente su valor varía entre el 0.00066% y el 1%.[5]

Conociendo FRR, se puede precisar la conveniencia de un sistema biométrico, de acuerdo con la ecuación, conveniencia = (1 – FRR). Conforme aumenta FRR menos conveniente resulta el sistema por el alto número acceso denegado de forma errónea.

Tasa de Falso Negativo (False NonMatch Rate o FNMR). Es la tasa con que los usuarios registrados no son identificados durante un proceso de comparación de características. Es común confundir los conceptos FAR con FMR y FRR con FNMR, sin embargo la principal diferencia consiste en la no contabilización de los rechazos producidos debidos a la baja calidad de la imagen o imposibilidad de adquirirla cuando se trata de FMR respecto a FAR o de FNMR hacia FRR.

Tasa de Fallo de Registro (Failure-to-enroll Rate, FTR o FER); incapacidad del sistema biométrico de generar un registro fiable de un usuario.

Tasa de Error Igual (Equal Error Rate, EER); es la intersección entre FAR y FRR siendo una de las medidas más usada para determinar el umbral de calidad del sistema biométrico, cuando los errores de aceptación y rechazo son iguales, el umbral se establece en el 50%. Cuanto más bajo es EER el sistema es más fiable. Estos umbrales deben ser ajustados de acuerdo con la garantía requerida.

3.2.3 Procesos de Autentificación e Identificación Biométrica

En el control de accesos mediante biometría existe una fase de verificación, durante la cual, el dato biométrico o plantilla, ya adquirido por el sistema, es cotejado con el que está siendo sometido a análisis, intentando confirmar la identidad declarada de un individuo, al confrontar la muestra suministrada con una o más plantillas registradas con anterioridad siguiendo uno de los dos procesos fundamentales, el de Autentificación o el de Identificación. [3][4]

El Proceso de Autentificación (o Verificación), también conocido como Uno-Para-Uno (1:1), se basa en la comparación de los rasgos biométricos con los de un patrón ya guardado. El objetivo es confirmar la identidad del individuo.

Los sistemas automáticos para verificación se denominan AFAS.

El proceso de Identificación o Uno-Para-Muchos (1:N), compara los rasgos biométricos con los de un conjunto de plantillas residentes en la base de datos. Pretende identificar al individuo.

Cuando se conoce que el individuo a identificar está dentro de la base de datos, se dice que la identificación es “de grupo cerrado”.

En identificación "de grupo abierto”, también llamada “lista de vigilancia”, no existe garantía de que el individuo sea parte de la base de datos. El sistema debe determinar si el individuo es parte de la base de datos y en caso afirmativo proporcionar su identidad (AFIS).

Debido al número de comparaciones y procesamiento que debe llevarse a cabo en el proceso de Identificación, resulta más rápido el proceso de Autentificación, especialmente conforme va aumentando N en la Identificación.

3.2.4 Modalidades biométricas

No es el propósito de esta memoria hacer un recorrido exhaustivo por todas las modalidades biométricas existentes. El alcance de este apartado se limitará a recoger en una tabla comparativa (Tabla 2), [5] las características más importantes a valorar para los sistemas biométricos más comunes, que permitirá la selección de una modalidad en función de las necesidades del proyecto.

	Ojo(Iris)	**Ojo(Retina)**	**Huellas Dactilares**	**Geometría de la mano**	**Escritura y Firma**	**Vista**	**Cara**
Fiabilidad	Muy Alta	Muy Alta	Alta	Alta	Media	Alta	Alta
Facilidad de Uso	Media	Baja	Alta	Alta	Alta	Alta	Alta
Prevención de Ataques	Muy Alta	Muy Alta	Alta	Alta	Media	Media	Media
Aceptación	Media	Baja	Alta	Alta	Muy Alta	Alta	Muy Alta
Estabilidad	Alta	Alta	Alta	Media	Baja	Media	Media
Costo	Muy Alto	Alto	Bajo	Bajo	Alto	Medio	Medio
Incidencias	Enfermedades Oculares, Luz	Enfermedades Oculares, Gafas	Ausencia de miembros	Edad, Ausencia Miembros	Edad, analfabetismo, cambios	Ruido, Afecciones	Edad, pelo
Tamaño Plantilla (byte)	512	96	250-1000	20	11000-3000	10000-20000	84-1300

Tabla 2

4. Análisis de requisitos

4.1 Requisitos de usuario

Se describe a continuación los requisitos de usuario obtenidos de entrevistas con diferentes usuarios de la intranet y Jefes de Departamentos.

El módulo de control de horarios será parte de la intranet corporativa y será accesible desde la página principal de esta para los roles Usuario, Jefe de UOI y Administrador.

Usuarios con privilegios (roles Jefe de UOI y Administrador) podrán cursar el alta, la baja y modificación de los empleados registrados en la intranet en el módulo de control y gestión de horarios.

Para dar de alta a un empelado será necesario asignarle una jornada laboral que también podrá ser creada por usuarios con privilegios.

En la creación de una jornada laboral se permitirá definir hasta dos horarios diferentes a lo largo de un año, normal e intensivo, que podrán ser asignados de manera única a los diferentes meses que componen el año. Para cada uno de los horarios de define una hora límite máximo de inicio y una hora límite mínimo de finalización de la jornada para cada día de la semana, y un número mínimo de horas semanales.

El alta de un empleado permitirá la asignación de un pin de manera opcional, que se solicitara en los procesos de identificación de entrada a la oficina.

El alta de un empleado permitirá la asignación de un responsable directo para la notificación de incidencias por incumplimiento de horario.

El alta de un empleado permitirá la asignación de una excepción de incidencias. Al empleado en cuestión no se le asignaran incidencias provocadas por el incumplimiento de horario.

El alta de un empleado permitirá la asignación de una excepción de notificación. Al empleado en cuestión se le asignaran incidencias provocadas por el incumplimiento de horario, pero no se notificaran a su responsable directo.

La captura inicial de huellas para el alta de usuarios en el módulo se realizará en el mismo terminal que los marcajes diarios. Un marcaje es la identificación en el terminal

mediante huella dactilar, de un determinado usuario, realizada a una determinada hora. La información del marcaje será almacenada en el sistema para su control y gestión.

Los usuarios podrán realizar sus marcajes diarios en las diferentes estancias que componente las oficinas de Edosoft, aunque en primera instancia se asume que se realizará solo en la oficina principal.

Los marcajes diarios podrán ser de entrada o de salida, indicando de este modo si se está comenzando o finalizando la jornada laboral y pudiendo pedir pin de acceso en caso de que sea un marcaje de entrada.

Los marcajes de entrada solo podrán realizarse en una franja horaria comprendida entre las 6:30 y las 24:00. Si fuera posible los marcajes producidos fuera de esta franja horaria deberían ser descartados por el terminal directamente. Si no existiera esa posibilidad se crearía una incidencia con los marcajes anteriores a esa hora.

La posibilidad de realizar marcajes de entrada y la salida puede provocar que el orden y/o el número de marcajes no sean los correctos. El número ideal de marcajes por día son dos, primero uno de entrada y por ultimo uno de salida. La Tabla 3 muestra el resto de casos contemplados, si se considera un día con marcajes válidos, y en este caso que marcaje de entrada y de salida son utilizados para el cómputo de horas del día. El superíndice n representa agrupaciones de marcajes de salida o de entrada desde 1 hasta n. Los puntos suspensivos indican cualquier sucesión de marcajes de entrada y salida.

Orden de marcajes	**Marcajes validos**	**Entra y salida seleccionada**
Salida^{n}, Entrada^{n}	No	-
Entrada^{n}, Salida^{n}	Si	Primer marcaje de entrada y primero de salida.
Salida^{n}, Entrada^{n}, Salida^{n}	Si	Primer marcaje de entrada y primero de salida después de una entrada.
Entrada^{n}, Salida^{n}, Entrada^{n}	Si	Primer marcaje de entrada y primero de salida.
Salida^{n}, Entrada^{n}, Salida^{n},…	Si	Primer marcaje de entrada y primero de salida después de una entrada.
Entrada^{n}, Salida^{n}, Entrada^{n},…	Si	Primer marcaje de entrada y primero de salida.

Tabla 3

Diariamente se analizara los marcajes de cada empelado registrando una incidencia y enviando una notificación por correo según los diferentes casos previstos:

- No existe ningún marcaje, el día es hábil según el calendario laboral en vigor y el empleado no tiene aprobado vacaciones o permiso para ese día.
- Existe un marcaje de entrada pero no de salida. Las horas computadas ese día serán 0.

- El marcaje de entrada tiene hora posterior al límite de entrada especificado por la jornada laboral para el mes en cuestión.
- El marcaje de salida tiene hora anterior al límite de salida especificado por la jornada laboral para el mes en cuestión.
- El marcaje de entrada tiene hora posterior al límite de entrada especificado por la jornada laboral y el marcaje de salida tiene hora anterior al límite de salida especificado por la jornada laboral.
- Existe marcaje de entrada y/o de salida y el día no es hábil según el calendario laboral en vigor o el empleado tiene aprobado vacaciones o permiso para ese día. En este caso el número de horas será computado

Como ya se ha advertido la incidencia se creara si el usuario no tiene excepción de control asignada y se enviara una notificación por correo electrónico a su supervisor si no tiene una excepción de notificación en su perfil de usuario.

El último día de la semana se analizaran las horas computadas a lo largo de toda la semana creando incidencias y enviando notificaciones si las horas totales son menores que las especificadas en la jornada laboral para el mes en cuestión.

El número máximo de horas de una jornada laboral de jornada continua es de 7 horas. Para el cálculo del número de horas de cada día se restará una hora automáticamente correspondiente a la hora de comer si el número de horas de dicho día es mayor de 7 horas y 15 minutos. Esta resta no se realizará durante el periodo en que la jornada laboral intensiva está en vigor, aunque se supere el tiempo indicado.

Los usuarios con privilegios podrán acceder a la información de marcajes, cálculo de horas semanales, e incidencias de todos los empleados del sistema. La información se deberá mostrar de manera apropiada incluyendo graficas que faciliten la comprensión y análisis de los datos.

Los usuarios sin privilegios (rol Usuario) podrán acceder únicamente a información relativa a sus propios marcajes e incidencias.

Los usuarios con privilegios dispondrán de una operación de verificación de la existencia de inconsistencia entre los datos almacenados por la aplicación y los almacenados por cada uno de los terminales biométricos. Entre las diferentes inconsistencias buscadas en este proceso se encuentran:

- Usuario dado de alta en el sistema pero no existente en uno o varios terminales.
- Usuario de baja en el sistema pero existente en uno o varios terminales.
- Usuario dado de alta tanto en el sistema como en todos los terminales, pero con datos diferentes de PIN.

Los usuarios con privilegios dispondrán de una operación de subsanación para reparar las inconsistencias encontradas durante el proceso de verificación.

Los usuarios con privilegios dispondrán de una operación de inserción masiva de los usuarios del sistema en un terminal. Pensado para la inclusión de terminales nuevos en el sistema.

Los usuarios con privilegios podrán acceder a la información de configuración de los terminales (MAC, dirección IP, mascara de red, DHCP activado) y de operatividad (número de marcajes totales que admite y número de marcajes almacenados)

Los usuarios con privilegios podrán modificar determinados parámetros de configuración de los terminales de alta en el sistema.

Los usuarios con privilegios podrán realizar algunas operaciones sobre los terminales:

- Verificación de la conexión con los terminales.
- Borrado de los marcajes almacenados en los terminales.
- Obtención de número de marcajes almacenados en los terminales.

4.2 Requisitos del software

El módulo de control de horarios como parte de la intranet corporativa y será accesible desde la barra principal de menús mediante un nuevo elemento y desde la botonera, mediante la inclusión de un nuevo icono.

Podrá mantener un mínimo de 500 usuarios dados de alta y sin límites de marcajes.

Mantendrá y gestionará la conexión con múltiples terminales. Este manejo de la comunicación incluye conexión inicial, reintento de conexión tras pérdidas de la misma, y sincronización de información de marcajes tras la recuperación.

Los terminales podrán funcionar de manera autónoma (modo off-line). Esto permite al terminal realizar autenticaciones aunque no esté conectado al servidor, pero será necesario realizar de manera automática la sincronización de marcajes entre el servidor y los terminales al volver a modo on-line.

A continuación se exponen los requisitos funcionales mediante los diagramas de casos de uso para cada uno de los roles definidos en la intranet.

Se muestra en la Figura 5 el diagrama UML los casos de uso para un usuario sin privilegios del sistema que corresponde al rol Usuario.

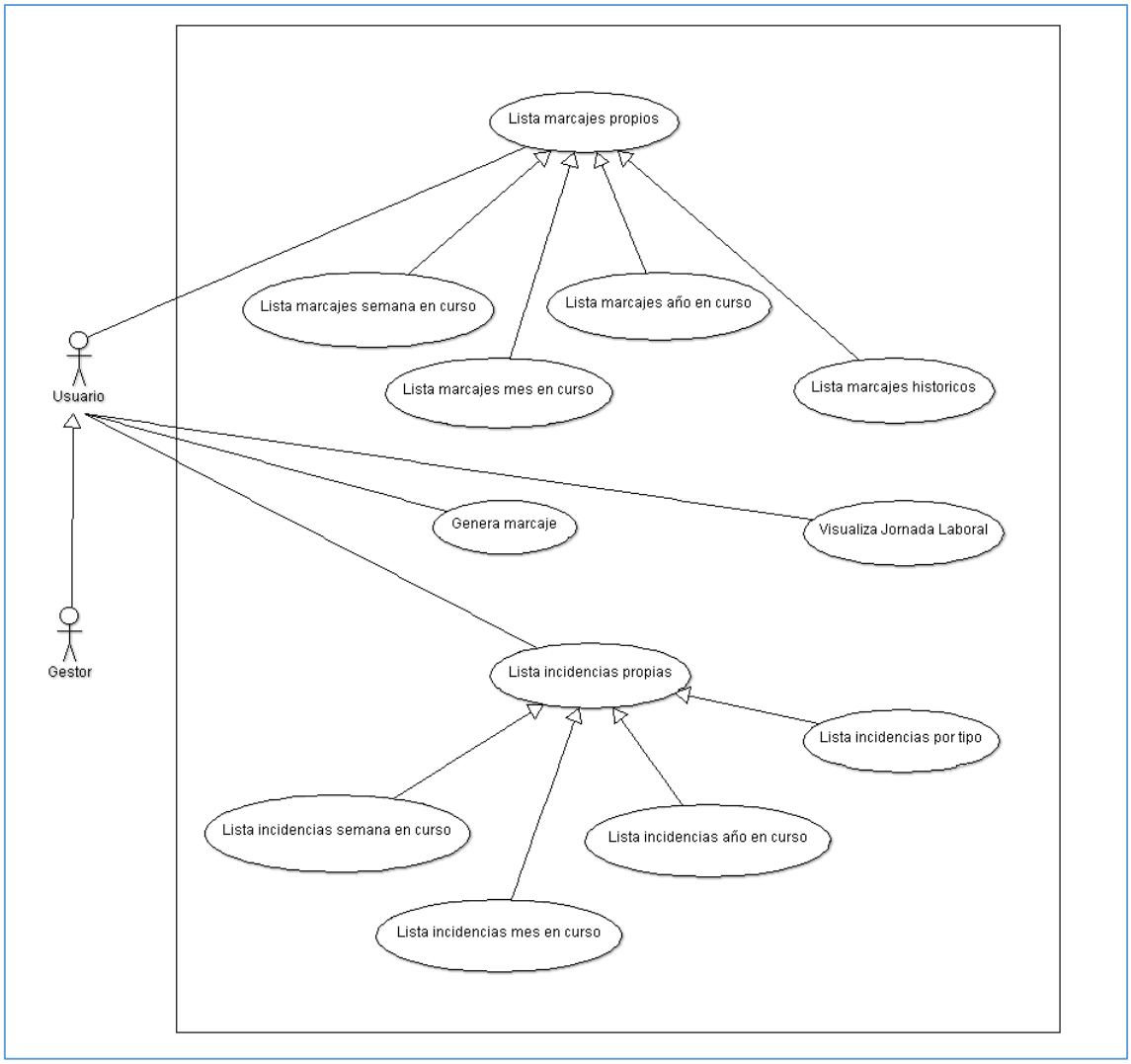

Figura 5. Diagrama de casos de uso usuario genérico.

En la Figura 6 los casos de uso de un Gestor que comprende los rolles: Gestor, Jefe de UOI y Administrador.

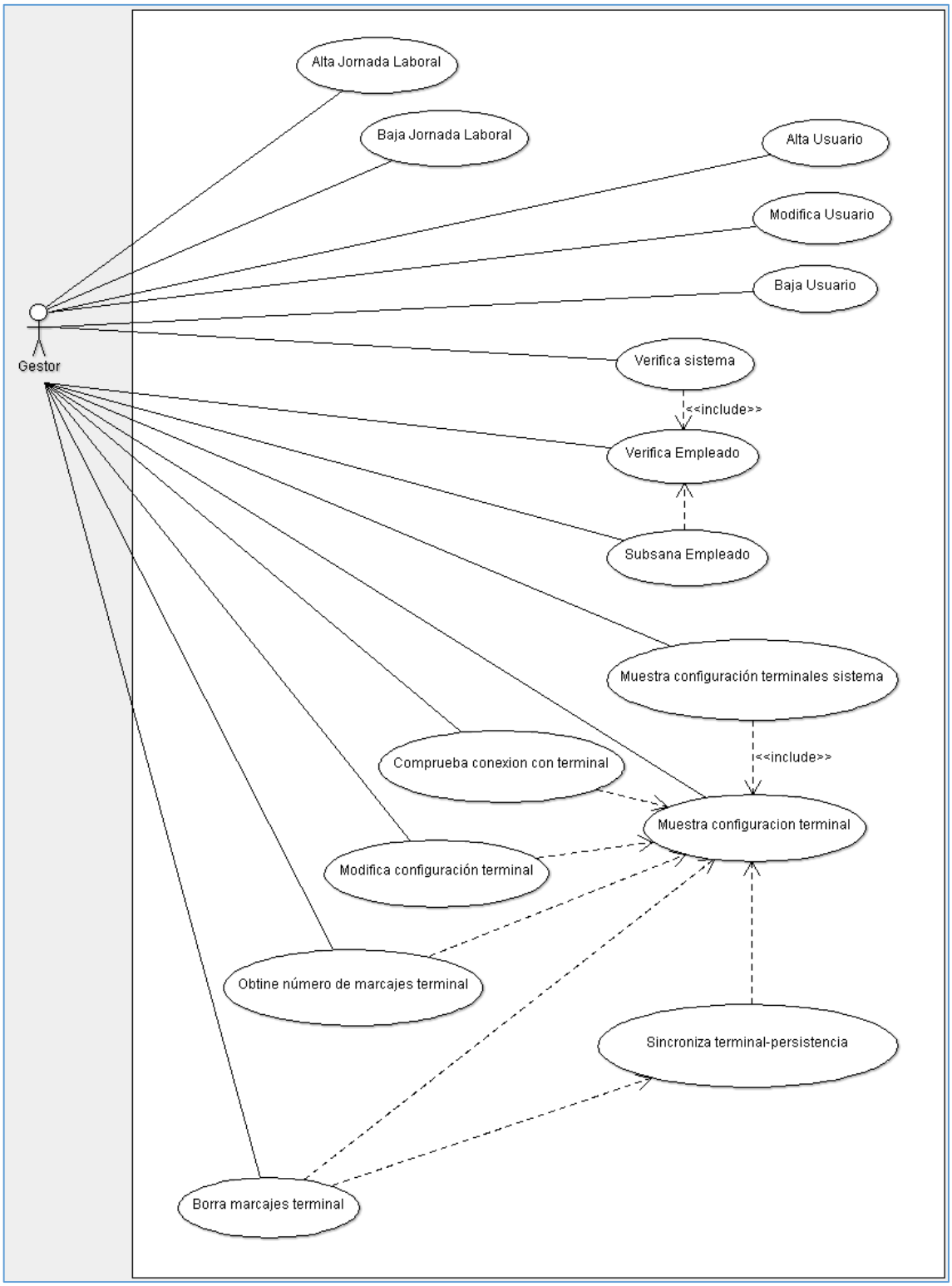

Figura 6. Diagrama de casos de uso de usuario con privilegios.

4.3 Requisitos del hardware

Los requisitos del hardware engloban las características que debe poseer el terminal biométrico para cumplir tanto con los requisitos del software como para mantener unos determinados parámetros operativos en su funcionamiento.

Los requisitos del hardware identificados son los siguientes:

- Capacidad de mantener un mínimo de 500 usuarios.
- Capacidad de funcionamiento off-line.
- Buena tasa de identificación: FAR: 1/100.000 y FRR: 1/1.000
- Capacidad de comunicación Ethernet para comunicación con el servidor a través de la LAN de la oficina.
- Posibilidad de asociar un PIN de acceso a usuarios del sistema, que deberá ser introducido en el proceso de autentificación.
- Posibilidad de identificar un marcaje como una entrada o una salida, con el fin de solo solicitar dicho PIN solo en el caso de una entrada.
- Que cuente con un conjunto de relés para conectar actuadores tal como una pestillera eléctrica. De esta manera se podrá abrir automáticamente la puerta de la oficina tras un marcaje de entrada correcto. (Futuros desarrollos)
- Las incidencias biométricas que este sistema presenta deberán poder ser resueltas mediante otro tipo de autentificación dentro del mismo sistema, código de acceso con PIN, o tarjeta RFID. (Futuros desarrollos)

4.4 Estudio de viabilidad de integración

En base al análisis de las tecnologías, diseño e implementación de la intranet y el análisis de los requisitos, se realizara el estudio de viabilidad de integración de la funcionalidad del nuevo módulo tanto para el modelo de negocio incluyendo la interfaz de comunicación con el terminal, así como la interfaz de usuario.

4.4.1 Viabilidad de integración en el modelo de negocio

Dado el escaso nivel de acoplamiento de la nueva funcionalidad con la ya existente y en empleo de J2EE se puede prever una integración satisfactoria y con un alto grado de modularidad. La ampliación del modelo de negocio con el módulo de gestión de horario consistirá en el añadido de varios beans de entidad que manejen los datos, con posibles relaciones a los beans ya existentes, y la ampliación del bean de sesión actual con las nuevas operaciones, o la creación de una nueva fachada que mantenga toda la lógica de funcionamiento del módulo.

El sistema deberá contar con algunos planificadores que ejecuten rutinas para el control de las horas computadas por los usuarios, con periodicidad diaria y semanalmente. J2EE permite la realización de estas tareas mediante la configuración de schedulers.

Con respecto a la integración del terminal biométrico se pueden plantear varias opciones. La primera de estas consiste en la creación de un programa externo al servidor de aplicaciones que mantenga permanentemente la comunicación con el terminal mediante un socket para el envío de tramas de órdenes, sobre el que actuará como servidor. La comunicación del modelo de negocio, con un funcionamiento asíncrono y una doble implementación, se podría realizar mediante CORBA u otro socket en sentido Core-Servidor de comunicación, para las peticiones al terminal, y la invocación de un bean remoto que actuará como fachada en sentido Servidor de comunicación-Core, para las respuestas.

La segunda de las opciones de conexión modelo de negocio terminal biométrico es el desarrollo de un Java EE Connector Architecture (JCA) que es una solución tecnológica basada en el Lenguaje de programación Java para conectar servidores de aplicaciones y sistemas de información empresarial (EIS) como parte de soluciones de integración de aplicación de empresa (EAI).

Se puede concluir, después de un análisis concienzudo, que desde el punto de vista del modelo de negocio la integración de la nueva lógica, incluida la que permita la interconexión con el terminal biométrico, es absolutamente factible y con ningún aspecto que a priori pudiera suponer alguna dificultad, existiendo una solución para las necesidades mediante las tecnologías que han sido empleadas en el desarrollo del proyecto.

4.4.3 Viabilidad de integración en la interfaz gráfica

En el análisis del diseño de la GUI se ha comentado el empleo del framework Cairgorm para el desarrollo de la interfaz gráfica. Este permite añadir nueva funcionalidad de forma sencilla, organizada y estandarizada siguiendo el patrón MVC, sin afectar y con muy poco conocimiento de la funcionalidad ya existente. Por otro lado los componentes visuales que a priori puede requerir el nuevo módulo: formularios, tablas, ventanas emergentes, etc, no distan mucho de los que ya se emplean en el resto de la intranet mediante la utilización del Flex.

Es por esto que desde el punto de vista de la GUI queda asegurada también la integración del módulo de control y gestión de horarios.

5. Proceso de selección del terminal biométrico

En el presente capítulo se presenta el terminal escogido, previa selección de una determinada modalidad biométrica de entre todas las presentadas en el capítulo 3.2.4 Modalidades biométricas. Se justifica tanto la selección de la modalidad biométrica, como la del propio terminal atendiendo a los diferentes parámetros relevantes identificados.

5.1 Selección de una modalidad biométrica

Atendiendo a los datos mostrados en la sección anterior, de entre todos las modalidades biométricas, los sensores de huella dactilar son los que presentan un menor costo junto con buena aceptación del usuario y un óptimo rendimiento biométrico

Se trata de un sistema de captura en vivo que puede fabricarse bajo multitud de variantes tecnológicas aunque se pueden agrupar en una de estas familias: Ópticos, Estado Sólido, Ultrasónicos y Sin contactos. Independientemente de la familia tecnológica a la que pertenece el sensor, en el mercado existen dos líneas de productos, los sensores de huella dactilar estáticos y los sensores de huella dactilar por desplazamiento. [3][6]

Mientras que los estáticos llevan mucho tiempo en el mercado, actualmente están surgiendo diferentes tipos de los llamados sensores de huella dactilar por desplazamiento.

Los sensores estáticos hacen uso de una ventana de captura de imagen que tiene el tamaño requerido para la huella dactilar, apoyando el dedo sobre esta ventana durante el tiempo necesario para la captura de la imagen. Este método tiene la ventaja de capturar la imagen en una única operación. Entre sus desventajas destacan el gran tamaño del dispositivo, incrementando su coste y dificultando la portabilidad, además la posibilidad de que permanezca una huella latente en la ventana de captura la convierte en un poco menos segura.

Mientras que los dinámicos consisten en el uso de una ventana rectangular con una anchura capaz de cubrir el ancho del dedo y unos pocos pixeles de altura, por lo que para obtener la huella dactilar completa, el sensor debe capturar imágenes continuamente mientras el dedo se va desplazando por la superficie.

Están basados en la tecnología del silicio, resultan más baratos de fabricar, requieren poco mantenimiento, prestan gran precisión, un bajo consumo y portabilidad.

Permite reducir el área del dispositivo en un factor de 5, así puede ser usado en un amplio número de sistemas empotrados, incluyendo sistemas de control de acceso, PCs, PDAs… usándose en cualquier situación o lugar siendo rápido, fiable y preciso, tanto para Identificación como Autentificación.

Puede considerarse como "autolimpiable", pues el propio desplazamiento del dedo sobre la superficie del sensor imposibilita que permanezcan huellas latentes aumentando la seguridad. Seguridad que se ve reforzada por la dificultad de obtener imágenes fiables habiendo forzado a un individuo a usar el dispositivo, debido a movimientos erráticos o dedos sudorosos y ante la dificultad de desplazar un dedo artificial con suficiente sensibilidad como para poder reconstruir una huella.

Siempre que el desplazamiento del dedo sobre la superficie del sensor se realice a una velocidad apropiada, el solapamiento de sucesivos fotogramas, debe permitir la reconstrucción de la huella dactilar, a través de la información de la velocidad de desplazamiento proporcionada por el sensor o estimada por la información del solapamiento de capturas consecutivas.

En este caso atendiendo al costo y puesto que la seguridad no es prioritaria en la elección, la opción del sensor estático es la más que se adecúa a las necesidades del proyecto.

Existen también multitud de tecnologías desarrolladas para los sensores de huellas dactilar y es un campo en continuo avance. De todas ellas, tal vez, las más extendidas son las ópticas y las de estado sólido.

La elección de una determinada tecnología de sensor sería un aspecto a tener en cuenta para diseñar e implementar un Sistema biométrico. En este caso la adquisición de un sistema biométrico, una solución comercial, permite la abstracción de la tecnología de fabricación, debiendo atender exclusivamente a los parámetros de rendimiento y otras especificaciones aportadas por el fabricante.

5.2 Selección y análisis del terminal modelo Kreta 3 de Kimaldi

La selección del terminal biométrico se ha realizado atendiendo varios factores. El primero de ellos, y fundamental, es que sea cumpla los requisitos del hardware.

De entre todas las opciones posibles se tendrá en cuenta una serie de aspectos. Se primará que sea ofertado por un distribuidor nacional de garantía, con servicio técnico para posibles reparaciones y soporte técnico para las cuestiones de diferente índole, funcionamiento, configuración, que surjan en el transcurso del proyecto.

Dicho distribuidor debería ser también fabricante o ensamblador de sus propias soluciones, con un amplio conocimiento de los productos que ofrece. Con una línea de productos completa y coherente, en continua evolución, pero que ofertando nuevos modelos de terminales no pierda la compatibilidad hacia atrás. Esto nos permitirá ampliar en futuro el sistema de control y gestión de horarios mediante la adquisición de nuevos equipos sin que haya que realizar enormes modificaciones en interfaces y funcionamiento debido a la adquisición de un nuevo modelo, al estar el anterior descatalogado.

Se valorará la calidad de la calidad de la documentación del sistema biométrico, manual de instrucciones, guía de instalación, etc. Además de la aportación de proyectos de ejempló que ilustren de forma practica el manejo del terminal, así como programas para una simple configuración inicial.

Kimaldi es un referente en el mercado de en equipos de identificación y control de personas que cumple todos los requisitos exigidos para proveer el sistema biométrico del desarrollo que nos ocupa.

El momento del proceso de selección y adquisición del terminal Kimadi ofertaba diferentes líneas de productos optando finalmente por los terminales Kreta 3.

Los terminales Kreta 3 permiten una cierta customización para adaptarse a las necesidades del cliente, pudiendo seleccionar entre diferentes capacidades de usuarios, diversos modelos de carcasas, varias interfaces de comunicación, etc.

El terminal de Ref 01KRT3ESB11F1X1 (Figura 8) dispone de un sensor biométrico de huella dactilar y un lector de proximidad de tarjetas RFID, con vistas a futuros desarrollos, y presenta las siguientes características:

- Biometría mediante el sensor FIM20:
    a) Identificación del usuario por reconocimiento de huella digital 1:N hasta 4.000 huellas y 15.000 autenticaciones. También permite realizar la verificación 1:1 y 1:Subgrupo, combinando la lectura biométrica con la introducción de un password o lectura RFID. Esto reduce el tiempo de verificación, ya que con sólo poner el dedo y pulsar una tecla simplemente busca en un subgrupo total de huellas almacenadas.
    b) Tasa de identificación muy elevada: FAR: 1/100.000 y FRR: 1/1.000
    c) Asignación de hasta 2 dedos por usuario.
    d) Múltiples modos de enrolamiento de usuario: local inmediato, local diferido o remoto.
    e) Módulo biométrico de alta calidad, con un potente algoritmo de matching y un sensor óptico de elevada dureza (7 Moh).
- RFID: Identificación del usuario mediante radiofrecuencia a través de un lector de proximidad de baja frecuencia 125kHz (EM, Unique, Q5)
- Conectividad
    a) Ethernet: Puerto Ethernet TCP/IP,UDP
    b) RS-232: Puerto RS-232 para conectividad de host.
- Otras características:
    a) Función Auto on: detecta el dedo en el sensor.
    b) Permite el funcionamiento de terminal en modo off-line al mantener en el interior del dispositivo todos los datos necesarios para realizar la autenticación de un usuario. Los datos de los procesos de autenticación que tengan lugar mientras el terminal esté operando en modo off-line podrán ser accedidos al recuperar la conectividad.
    c) Caja office con teclado matricial de 4 filas x 4 columnas para la inserción de PIN y Display LCD, 20x2 caracteres para la muestra de mensajes.
    d) 2 años de garantía.
- Documentación:
    a) Manual Módulo Kreta3 V 1.01: manual de especificaciones y funcionamiento del Módulo Kreta3, dedicado a Control de Presencia y Control de Acceso

b) Guía rápida de instalación V 1.0: Muestra los primeros pasos con el terminal Kreta 3. Instalación y configuración inicial básica con la aplicación Servicio de localización. Además incluye un breve manual sobre el empleo del programa de ejemplo Demo kreta 3.

c) Manual control SLK: Manual operativo del Servicio de Localización Kimaldi.

- Programas complementarios:
  a) Demo kreta3: Aplicación de ejemplo de funcionamiento del terminal biométrico, con un alto componente didáctico. Para cada operación sobre el terminal muestra el contenido de la trama de datos que se envía al mismo.
  b) Servicio de localización: Aplicación que permite escanear una subred, en busca de dispositivos Kimaldi basados en la plataforma ID32 (por ejemplo, Kreta3 o BioMax2) conectados a ella. Una vez conectada permite la configuración de sus parámetros de conexión (IP, mascara de red, DHCP activado, etc).

El material adquirido se compone del terminal biométrico Kreta 3 (Figura 8), un adaptador de corriente, un par de tarjetas RFID para pruebas y un CD-Rom con Manuales, herramientas y software Demo (Figura 7)

Figura 8. Foto del terminal biometrico Kreta 3

Figura 7. Complementos adjuntos al terminal.

6. Diseño

La definición del diseño está subdividida en dos capas fundamentales correspondientes a una aplicación cliente servidor, tal y como se ha comentado en otros capítulos de esta memoria. Así pues, por una parte se encuentra el diseño del modelo de negocio, donde se amplía el diagrama de clases correspondiente al modelo de datos para mantener las nuevas clases necesarias. Además se incluye un pequeño diagrama de componentes para mostrar la composición del modelo de negocios, y su relación con el terminal biométrico. Y por último una seria de diagramas de secuencia, para los casos de uso más significativos, en donde se muestra el conjunto de llamadas a métodos de clases que tienen lugar.

El diseño de la interfaz gráfica viene claramente determinado por la utilización del framework Cairgorm en la misma. Es por eso que para la definición del mismo será suficiente con mostrar el modelo de datos y un diagrama de secuencia que muestre el conjunto de llamadas que se producen entre las diferentes capas (Modelo, Vista, Controlador) para un caso de uso representativo, mediante la utilización de Eventos, Comandos y Bussines Delegates, tal y como define el framework. Se trata pues de la especificación para un caso particular del diagrama expuesto en la Figura 4.

6.1 Diseño del nuevo módulo en la lógica de negocio

Apoyados en diferentes diagramas UML y el desarrollo de los mismos mediantes las pertinentes explicaciones se expone en este capítulo el diseño de la lógica de negocio para la nueva funcionalidad a implementar.

Primeramente se muestra en la Figura 9 el modelo de datos para el componente de Control de horario. Este solo mantiene la clase Empleado como enlace al modelo de datos original del diagrama de la Figura 3.

La clase EmpleadoTerminalK3 representa a la clase de los empleados dados de alta en el sistema de control y gestión de horarios, que mediante una relación 1 a 1 está asociado con la clase Empleado del modelo original. De esta clase parten relaciones con el resto de las clases del modelo. De los atributos pertenecientes a esta clase se listan y

explican tan solo aquellos que no resultan triviales de entender a tenor de su nombre y el análisis previo:

- códigoSemana: Hace referencia al código que identifica un horario semanal que tendrá asignado un empleado. Por defecto solo habrá almacenado un único horario semanal en todos los terminales, que permite fichar a cualquier hora de la semana, que será el utilizado por todos los empleados del sistema.
- codigoIntegridad: Mantiene el estado del empleado en cada uno de los terminales del sistema mediante un código determinado. Empleado para las operaciones de verificación de la integridad y subsanación de la misma.
- minucias: Mantiene los datos de la minucia de huellas para el empleado en cuestión. Dos minucias consecutivas son almacenadas en 400 bytes de datos.
- estadoOperativo: Mantiene los estados del empleado en el sistema tras un alta o baja (INSERTADO o ELIMINADO). Este campo también puede mantener estados intermedios de un proceso de alta o baja sin finalizar, por ejemplo por un error en la captura de la huella, un fallo en la inserción o eliminación de información de dichos empleados en los terminales, etc. Se mantiene este estado intermedio con el fin de poder continuar con posterioridad con dicha operación.
- estadoOperación: Mantiene el estado de una operación (Modificación, Verificación o Subsanación) sobre el empleado en cuestión, con propósito de comprobación, verificación y muestra de logs.

Figura 9. Diagrama de clases del modelo de datos del modulo de control y gestión de horarios.

La clase JornadaLaboral contiene especificación de la jornada laboral que se le aplica a un empleado dado de alta en el sistema de control y gestión de horarios. El atributo jornadaLaboralXML mantiene dicha especificación mediante lenguaje XML. Se muestra a continuación un ejemplo del mismo.

Atributo jornadaLaboralXML de la clase JornadaLaboral

```
<jornadaLaboral>
      <jornadaOrdinaria>
            <meses>Enero;Febrero;Marzo;Abril;Mayo;Junio;Septiembre;
            Octubre;Noviembre;Diciembre</meses>
           <lunes>0900;1500</lunes>
           <martes>0900;1500</martes>
           <miercoles>0900;1500</miercoles>
           <jueves>0900;1500</jueves>
           <viernes>0900;1500</viernes>
           <sabado></sabado>
           <domingo></domingo>
            <horas>41</horas>
      </jornadaOrdinaria>
      <jornadaExtraordinaria>
           <meses>Julio;Agosto</meses>
           <lunes>0900;1400</lunes>
           <martes>0900;1400</martes>
           <miercoles>0900;1400</miercoles>
           <jueves>0900;1400</jueves>
           <viernes>0900;1330</viernes>
           <sabado></sabado>
           <domingo></domingo>
            <horas>35</horas>
        </jornadaExtraordinaria>
</jornadaLaboral>
```

La clase Incidencia mantiene información sobre él incumplimiento de la jornada laboral, la falta de marcajes o cualquier otro tipo de eventualidad, para un empleado. El atributo codigoIncidencia almacena el tipo de incidencia que se ha producido con un código al efecto.

Código	Descripción
001	Excedida la hora de entrada máxima fijada
002	Incumplida la hora de salida mínima fijada
003	Incumplida la hora de entrada mínima fijada
004	Excedida hora de salida máxima fijada
005	Fichaje de entrada no debido (Fichaje en un día festivo, vacaciones o definido como no laborable)
006	Sin fichaje de salida (Solo presente el fichaje de entrada)
007	Sin fichaje (Falta fichaje de entrada y salida en un día laborable)
008	Incumplido el cómputo de horas semanales definidas.

La clase Marcaje mantiene información sobre cada uno de los fichajes que realiza un empleado dado de alta en el sistema. De los atributos que tiene esta clase se exponen con su pertinente definición, todos aquellos que no resultan triviales de deducir mediante su nombre y el análisis previo:

- codigoEvento: Determina la validez de un marcaje, siendo el valor de marcaje correcto el "33" para los fichajes de entrada y el valor"31" para las incidencias de salida. La siguiente tabla muestra códigos para marcajes denegados:

Código	Descripción
38	EVENTO ACCESO PRESENCIA DENEGADO POR HORARIO
36	EVENTO ACCESO PRESENCIA DENEGADO POR PIN
35	EVENTO ACCESO PRESENCIA DENEGADO POR ACCESO

- incidencia: Determina si es un marcaje de entrada con el valor "01" o de salida con el valor "02"
- valido: Determina si un marcaje de entrada o salida en considerado valido y será utilizado para el computo de horas de un determinado día. La validez dependerá del código de evento, de la incidencia y del orden de los marcajes (ver Tabla 3)

La clase TerminalKreta3 mantiene información de los terminales conectados al sistema. Esta clase está relacionada con la clase EmpleadoTerminalK3 a través de la clase AltaEnT3 para conformar una relación m a m. De esta manera un empeadoTK3 se

encuentra dado de alta en todos los terminales conectados y en un determinado terminal estarán todos los empleados dados de alta en el sistema. De los atributos que tiene esta clase se explican aquellos que no resultan simples de entender con el dato de su nombre y mediante el análisis previo:

- terminalBloqueado: Mantiene información cuando el terminal está bloqueado debido a alguna operación en curso. Cuando el terminal está bloqueado su valor será 'true', y no se puede iniciar nunca otra operación mientras tanto, en caso contrario su valor será 'false'.
- modoOnlineActivado: Determina el modo de funcionamiento del terminal. Con modo online activo, los marcajes serán enviados por el terminal hacia la intranet cuando tienen lugar, en caso contrario tan solo se almacenan en el terminal. Por defecto y el modo común de funcionamiento será con el modo online activado.
- configuraciónHoraria: Determina el rango de horas en un día que el terminal permite marcajes. Su valor será fijo, y permitirá marcajes a cualquier hora del día
- configuraciónSemanal: Determina los días de la semana que el terminal permite marcajes dentro de la configuración horaria. Su valor fijo será de lunes a domingo. Con la configuración horaria se admite la captura de marcajes a cualquier hora y fecha, quedando la valoración de su validez bajo en funcionamiento de la lógica de negocio.

Una vez concluido con el modelo de datos del servidor, se muestra un diagrama de componentes (Figura 10) donde quedan diferenciados los componentes principales de la lógica de negocio.

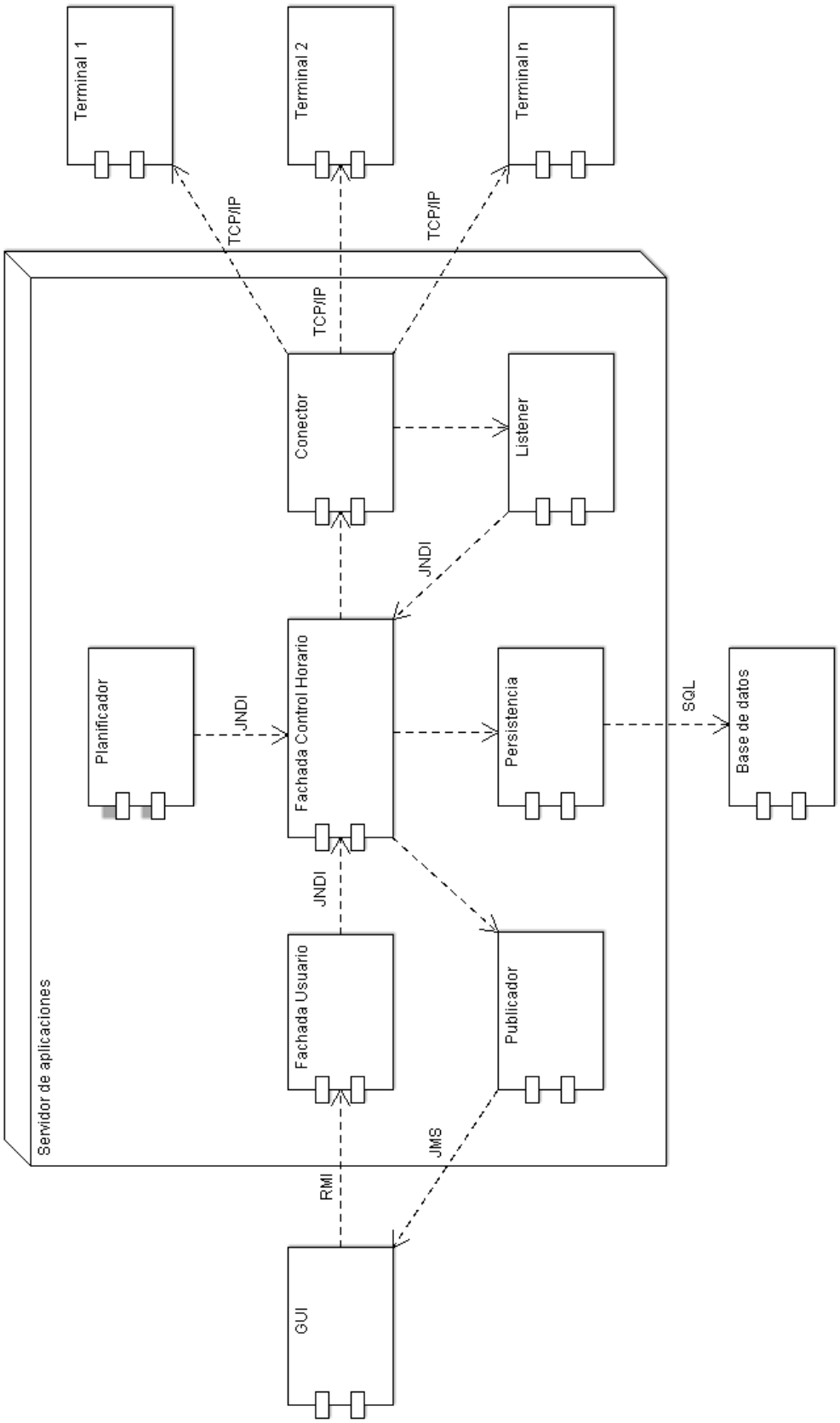

Figura 10. Diagrama de componentes de la lógica de negocio.

La Fachada de Usuario constituye el punto de acceso al sistema y es empleada por la GUI para realizar las diferentes operaciones. Esta fachada ya existe en el software original y solo se han añadido los nuevos métodos correspondientes a la nueva funcionalidad, que en la totalidad de los casos es una llamada directa a los métodos homónimas de la fachada Control Horario tras escasas operaciones de verificación y transformación de datos. La fachada Control Horario es un componente de nueva creación que contiene todos los métodos correspondientes a la funcionalidad de gestión y control horaria.

El Conector es un elemento clave diferenciado del resto de lógica de la lógica de negocio. Este conector constituye una capa de abstracción que oculta la comunicación con los terminales al resto de la aplicación. Tanto desde el punto de vista del control de flujo de datos, como de la conformación e interpretación de las tramas de datos enviadas y recibidas respectivamente. Es utilizado por la fachada de Control Horario para el envío de órdenes a los terminales conectados al sistema.

El componente Persistencia engloba el almacenamiento de datos, está definida mediante el diagrama de la Figura 9 y constituye una capa de abstracción con la base de datos con la cual se comunica. Es empleado por la fachada de Control Horario para el almacenamiento y la recuperación de datos.

El Listener es un componente que se invoca desde el Conector cuando se recibe la respuesta asíncrona a alguna de las peticiones realizadas. Desde el mismo se invoca a métodos de la fachada Control Horario para completar operaciones en curso con los resultados obtenidos desde los terminales.

El Publicador es utilizado desde la fachada de Control Horario para informar a la vista del resultado total o parcial de operaciones asíncronas en curso.

El Planificador es un Scheduler que corre en el contexto del servidor de aplicaciones y se ejecuta cada día a las doce de noche, e invocando a métodos de la fachada Control Horario para realizar la comprobación de incidencias y cursar las correspondientes notificaciones por correo si procede. El cómputo de horas semanales y la incidencia con respecto al no cumplimiento de las mismas se realiza el domingo de cada semana.

Y por último cabe destacar la inclusión la GUI, los terminales y la base de datos como elementos externos al modelo de negocio.

Se expone por último los diagramas de secuencia de los casos de uso más significativos. En la figuras de la 11 a la 14 se muestra el correspondiente al caso de uso Alta Usuario. En este conjunto de diagramas de secuencia se incluyen solo las llamadas y clases fundamentales del flujo de ejecución

La Figura 11 presenta la parte síncrona del caso de uso, desde la invocación al método de la fachada de Usuario hasta el envío del comando de captura de huella al terminal, retornando finalmente del método de fachada invocado.

Continúa el flujo de ejecución del caso en la Figura 12 desde que se recibe la respuesta asíncrona con la captura de la minucia dactilar por el `Socket` hasta que se invoca al bean activado por mensajes `MDBListernerTerminalKreta3`. Con posterioridad se muestra la Figura 13, que se inicia donde termino el flujo del diagrama anterior, y finaliza con las altas diferidas del empleado en cada uno de los terminales biométricos conectados al sistema mediante el comando correspondiente.

El flujo final se explica en la Figura 14, desde que se recibe la respuesta asíncrona a los comandos de alta provenientes del terminal biométrico, hasta que se envían los mensajes de notificaciones a la GUI, que indican la finalización de las altas en los diferentes terminales. El diagrama muestra tan solo una de las respuestas, de las cuales habrá genéricamente una por cada comando de alta enviado hacia cada terminal.

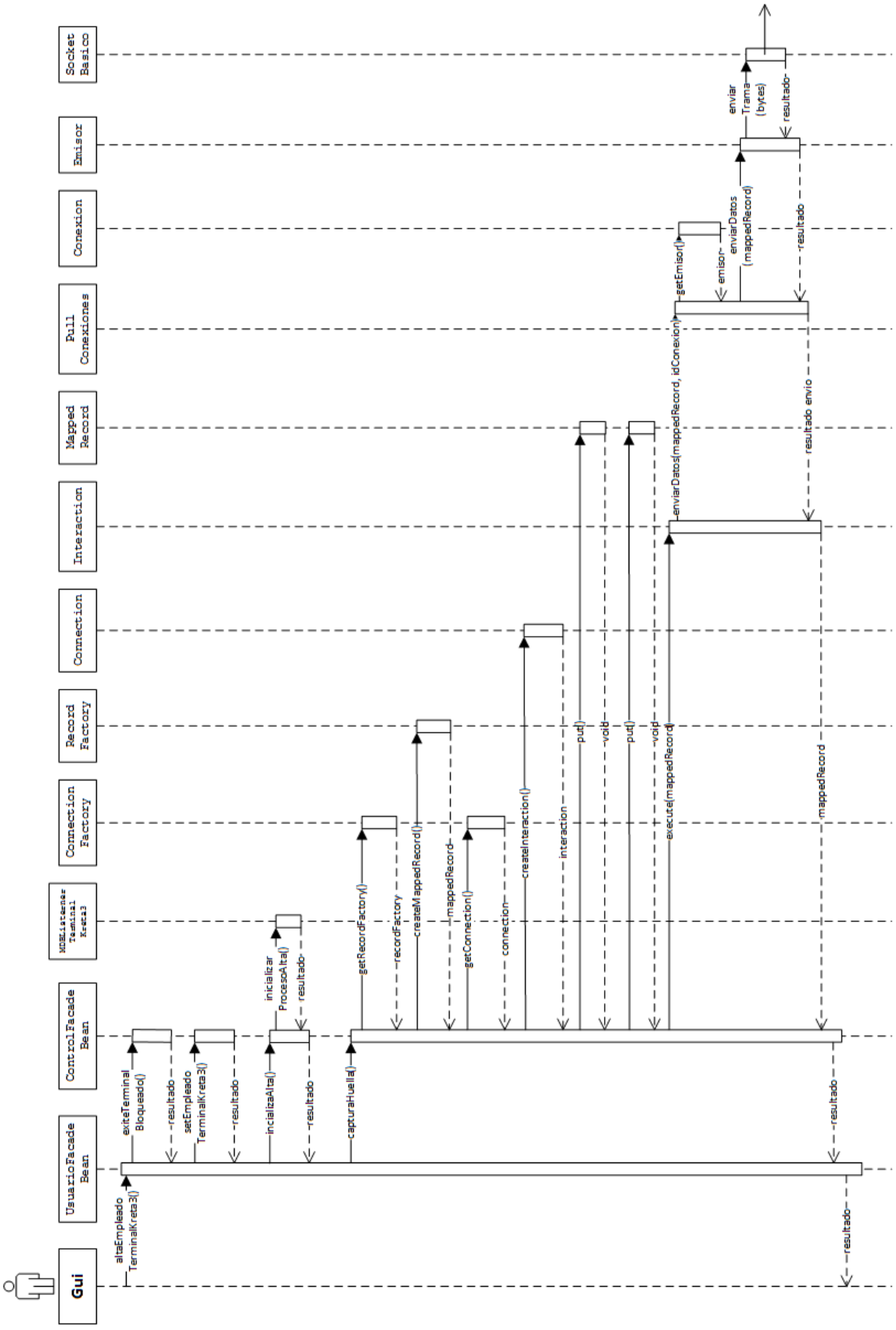

Figura 11. Diagrama de secuencia Alta usuario. Flujo síncrono.

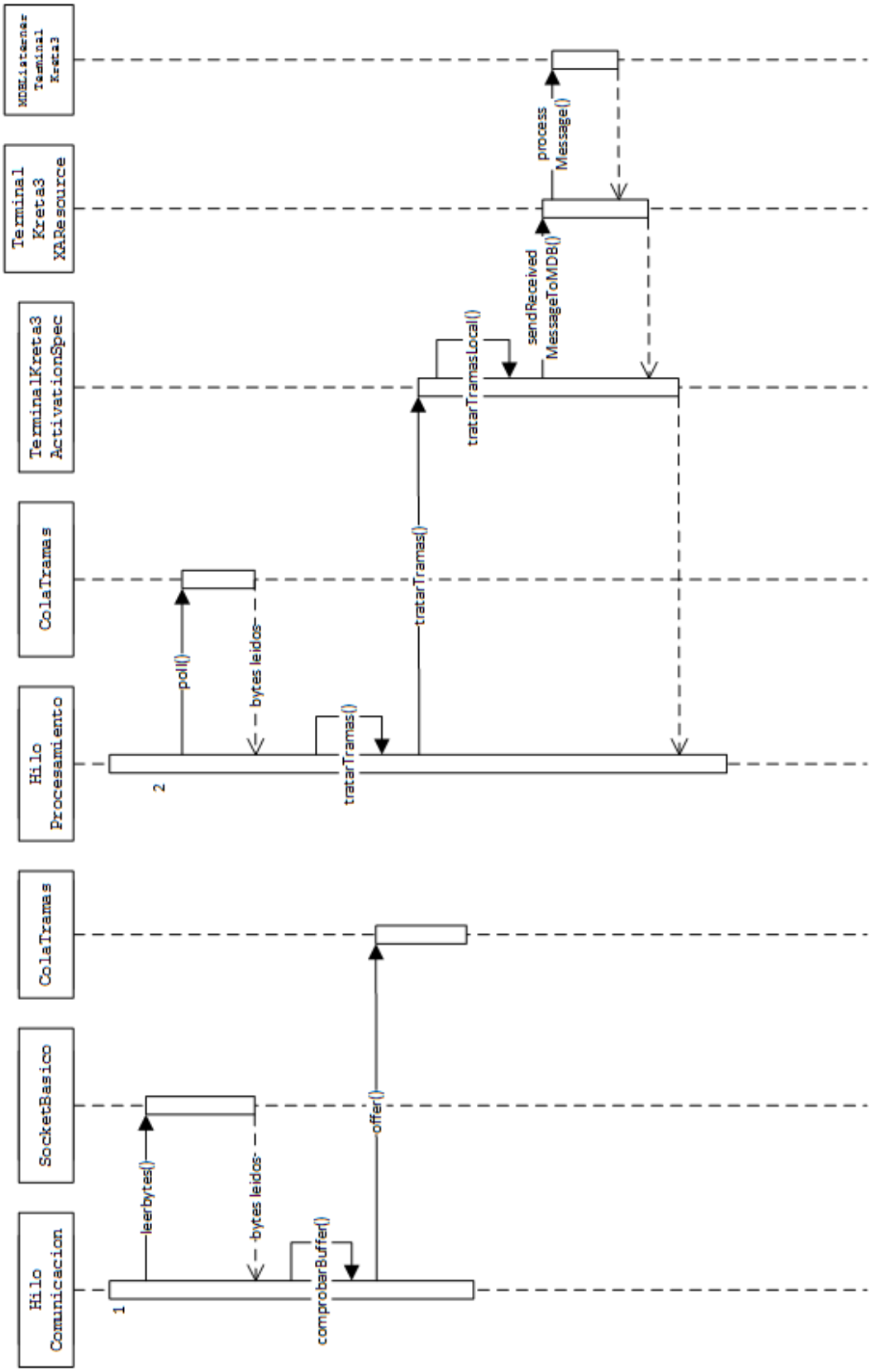

Figura 12. Diagrama de secuencia Alta usuario. Flujo asíncrono (1ª parte)

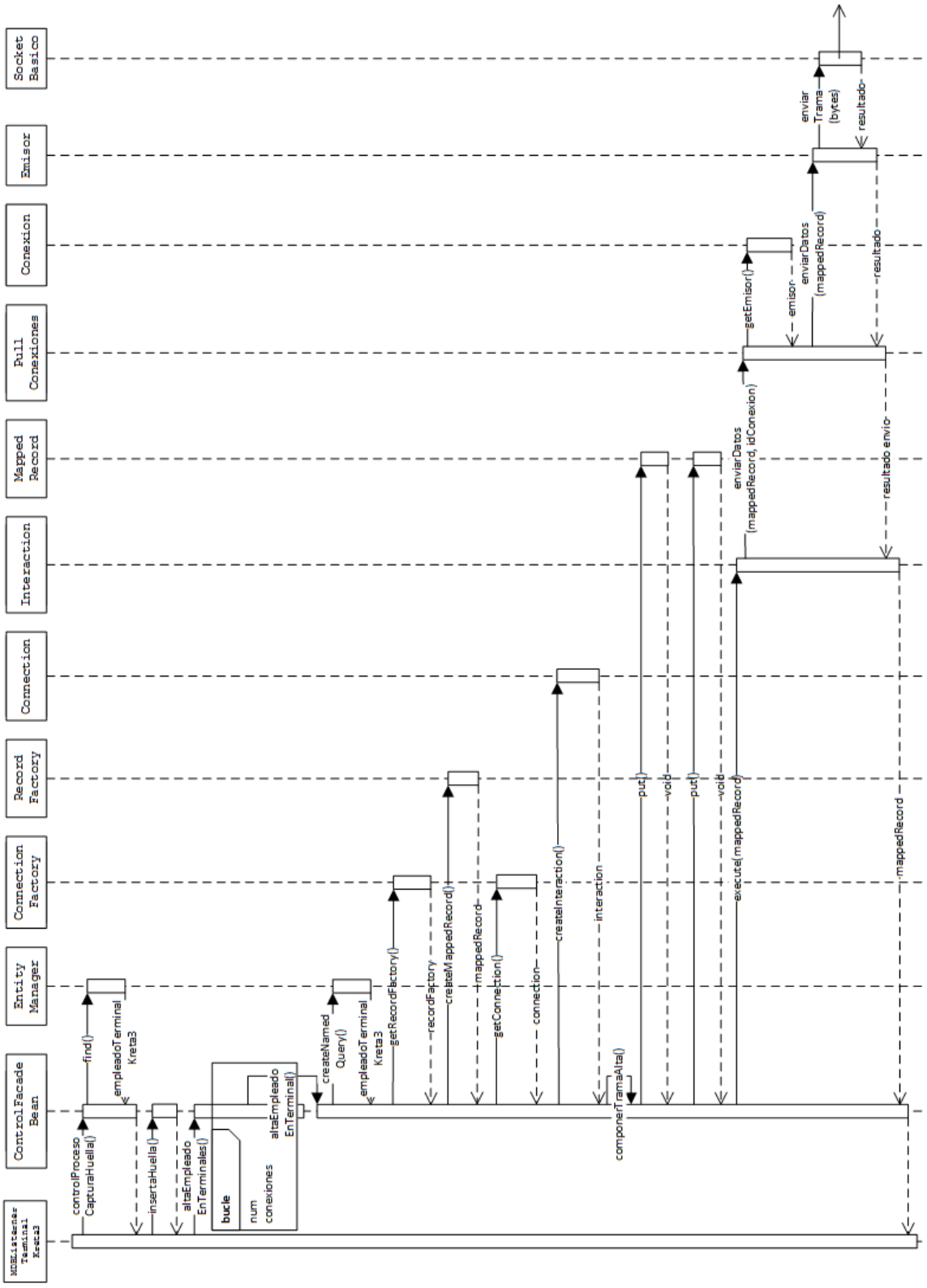

Figura 13. Diagrama de secuencia Alta usuario. Flujo asíncrono (2ª parte)

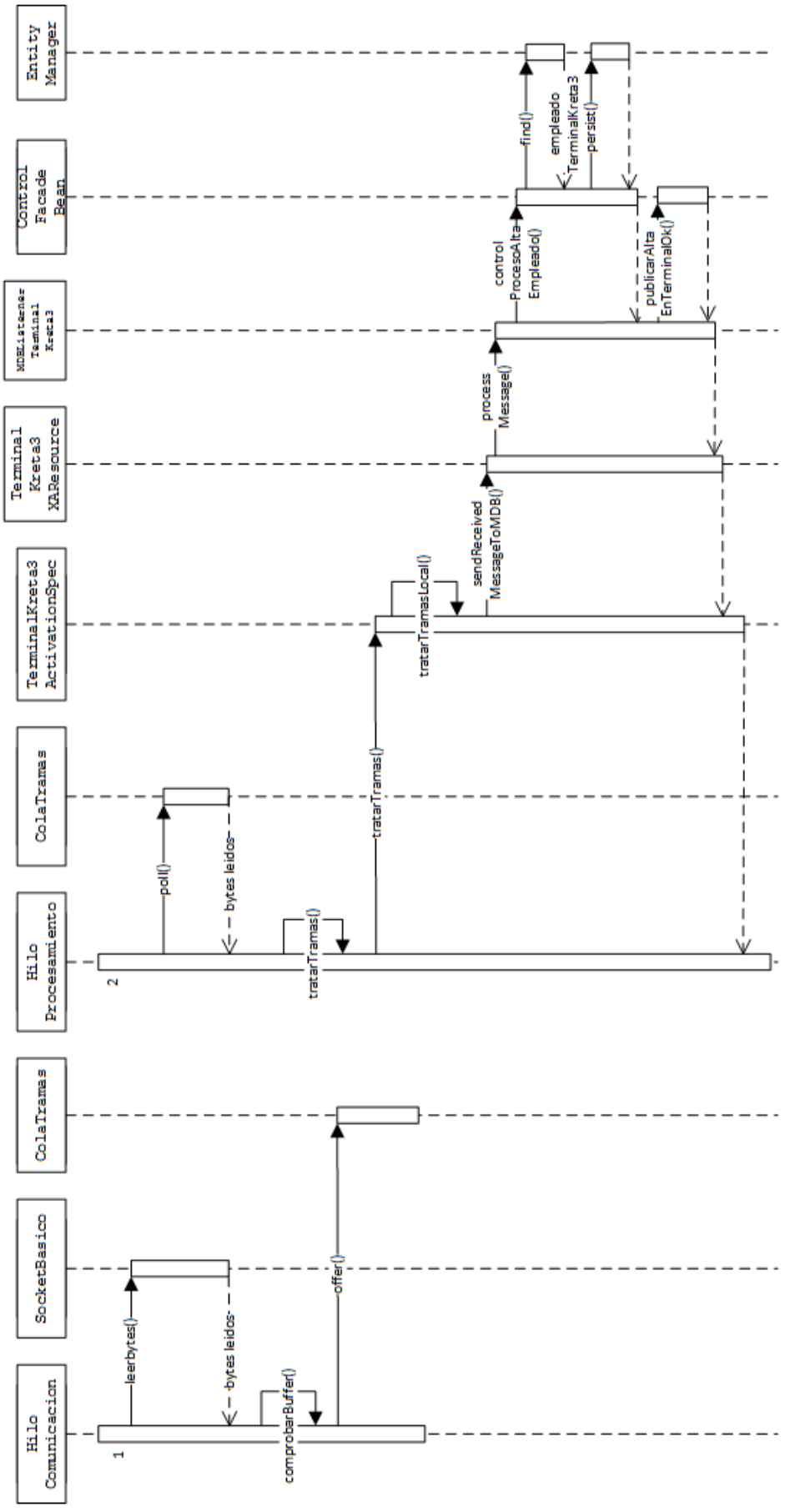

Figura 14. Diagrama de secuencia Alta usuario. Flujo asíncrono (3ª parte)

En la Figura 15 y Figura 16 se expone el diagrama de secuencia correspondiente a la lectura inicial de los datos de configuración de terminal, tras el arranque del servidor de aplicaciones.

La Figura 16 es la continuación de la Figura 15 por el punto 5. La trama enviada por el handshake es una operación de ACK cuya respuesta iniciará el proceso de configuración.

Los bucles del diagrama de la Figura 15 y su tratamiento mediante la llamada a `processMassage()` se repiten mientras queden pendientes peticiones de los diferentes parámetros de configuración. En la Figura 16, en el punto 5, comienza el tratamiento de la respuesta del ACK la cual provoca la petición del primer parámetro mediante el método `getConfiguración()`, es decir es la primera invocación del método `processMassage()` en el proceso de configuración inicial. A partir del punto 6 representa el tratamiento de la segunda llamada y subsiguientes, en donde se persiste el dato pedido en la iteración anterior y se solicita el siguiente a través del método `getConfiguración()`. En el procesamiento del último parámetro de configuración, no se realizará ninguna petición más.

Lista de los diferentes parámetros solicitados en el proceso de arranque del servidor:

- Dirección MAC
- IP Terminal
- IP Gateway
- Máscara de red
- DHCP
- IP Remota
- Puerto remoto

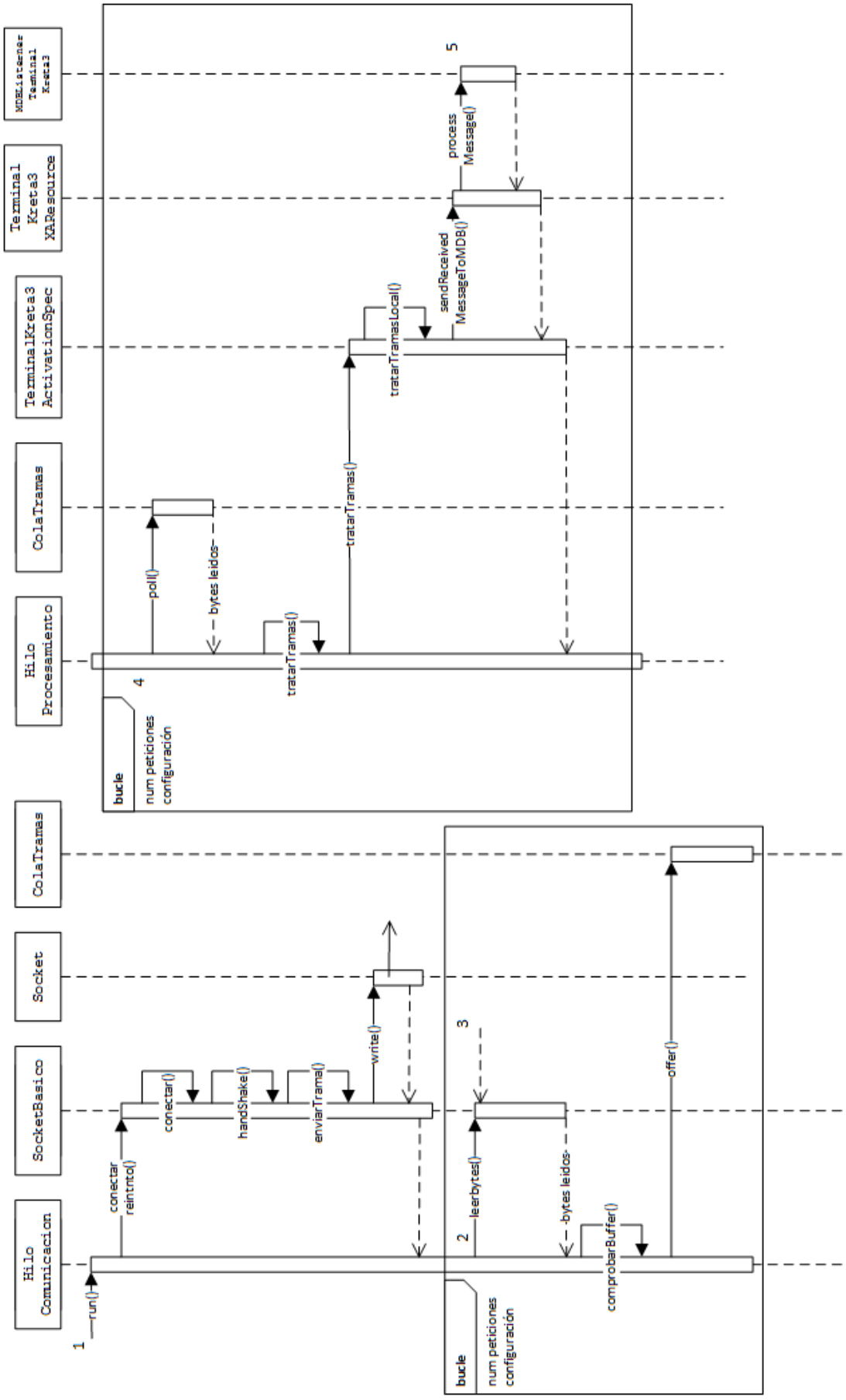

Figura 15. Diagrama de secuencia lectura/escritura de configuración del terminal durante el aranque del servidor (1º parte)

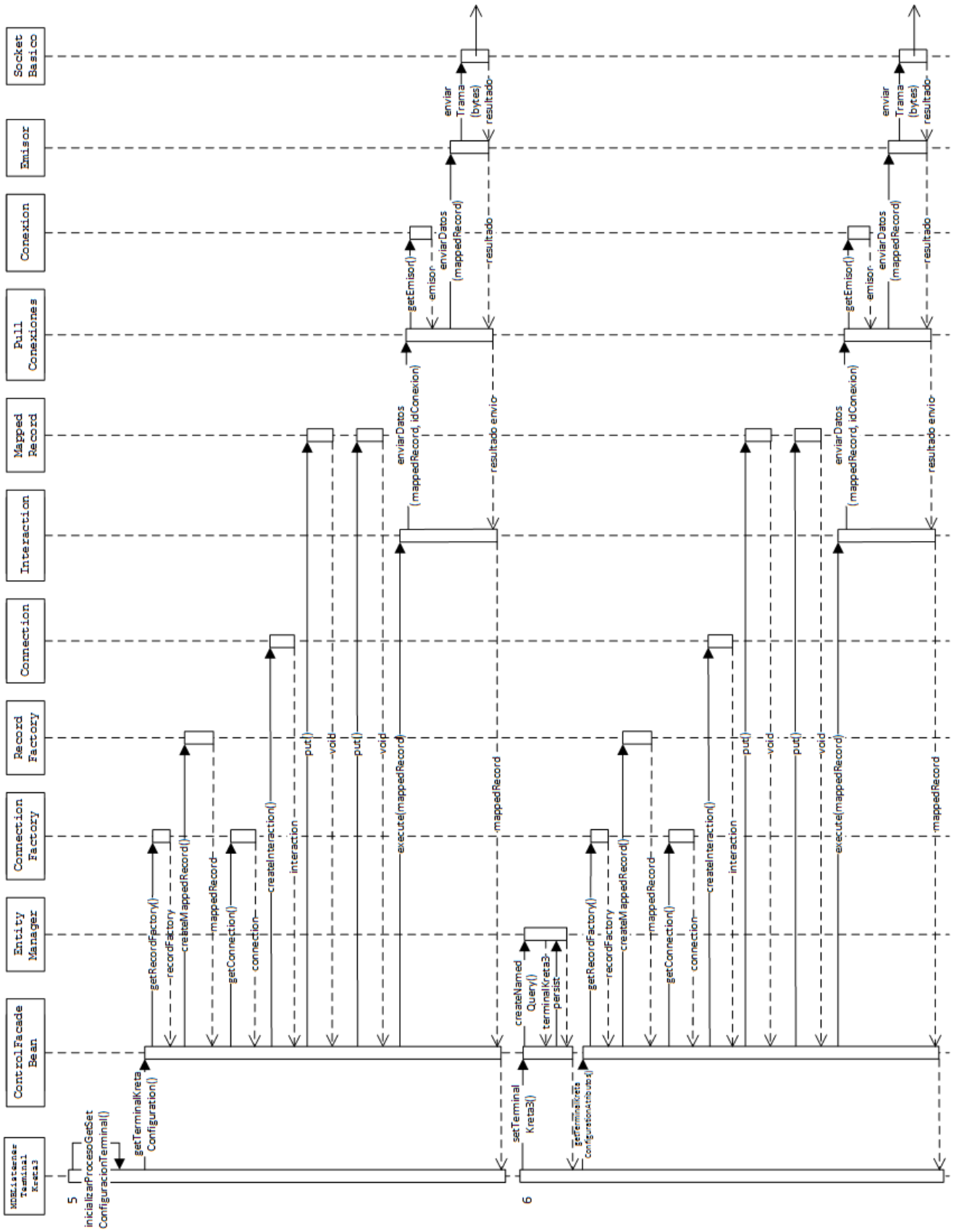

Figura 16. Diagrama de secuencia lectura/escritura de configuración del terminal durante el aranque del servidor (2ª parte)

6.2 Diseño del nuevo componente de interfaz

El modelo de datos mantenido mediante el ModelLocator está formado por clases VO de contenido muy similar al modelo de datos de la lógica de negocio. Este modelo está orientado a la representación visual de la información en la Interfaz Gráfica por lo que contendrá además de un subconjunto de datos provenientes del servidor datos calculados a partir de estos. Se muestra en la Figura 18 el diagrama de clases que constituye el modelo de datos para un usuario.

En la Figura 16 se exponen un diagrama de secuencia de uno de los casos más significativos, Alta de usuario. De esta manera junto con el diagrama de Alta de usuario del subcapítulo anterior se tendrá una visión completa de la secuencia de ejecución desde que el Administrador del sistema comienza con esta operación hasta que finaliza.

El diagrama comienza con la activación del evento click del botón Aceptar de la ventana de altas de empleados en el sistema. En ese momento se desencadena la sucesión de llamadas que tiene como resultado la llamada al método de fachada del servidor correspondiente. Con el tratamiento del resultado de esta llamada desde el comando que la invoca, mostrando una ventana con el resultado inicial de la operación. Aquí concluye la parte síncrona del proceso.

A continuación el punto 2 del diagrama indica la llegada y tratamiento del evento de alta en terminal que llega por mensajería al topic de JMS, informando al usuario de la evolución del proceso mediante la ventana lanzada durante la secuencia que parte del punto 1. El punto 2 se ejecutará tantas veces como terminales biométricos conectados al sistema haya.

Finalmente a partir del punto 3 se muestra la secuencia de llamadas para el tratamiento del último evento recibido mediante el sistema de mensajería y por el cual se da por concluido el proceso de Alta de usuario mostrando el resultado pertinente mediante la ventana lanzada durante la secuencia que parte del punto 1.

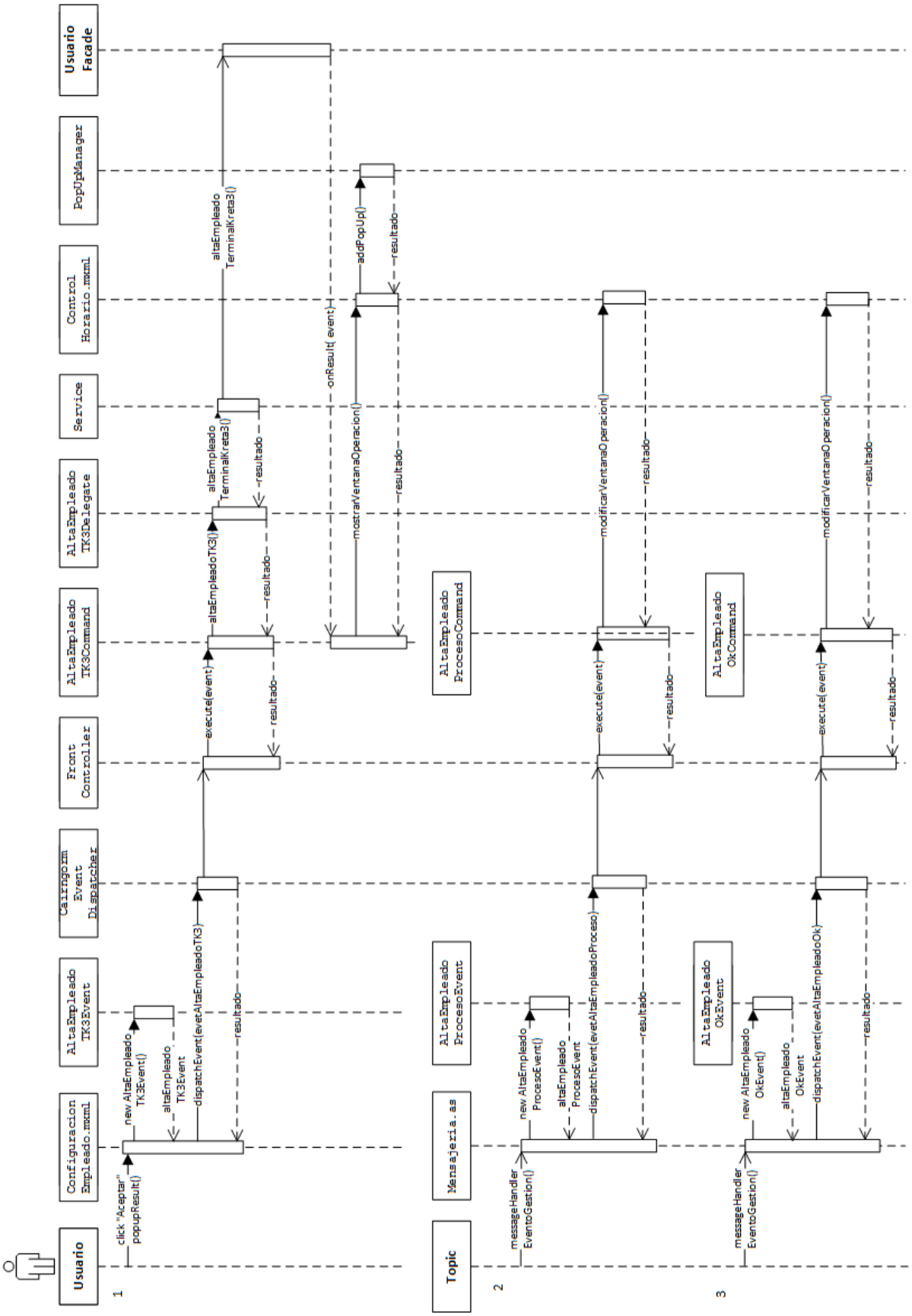

Figura 17. Diagrama de secuencia Alta usuario.

Figura 18. Diagrama de clases del modelos de datos mantenido en el cliente gráfico.

Finalmente se exponen algunos prototipos de las diferentes pantallas que compone la interfaz gráfica indicando donde se ubican los diferentes elementos tanto para la vista de gestor como para la de usuario sin privilegios.

Prototipos para el roll gestor.

La Figura 19 contiene un esquemático de la GUI de la intranet corporativa de Edosoft y en particular el prototipo del contenido del área de visualización para el módulo de control y gestión de horarios, que se encuentra bajo la barra de menús. La cual está compuesta por una botonera contextual, un TabNavigator con diferentes pestañas y un área de visualización de dados en la parte derecha. En este prototipo se muestra el contenido de la pestaña Empleados, un DataGrid donde aparecen todos los empleados dados de alta en la intranet, y la información relativa al control de horario de cada uno, si han sido dados de alta en el sistema. Además se muestra la ventana emergente empleada para el Alta y Modificación de datos de empleados en módulo.

La Figura 20 muestra la misma pantalla que la figura anterior, pero con varios usuarios dados de alta en el sistema de gestión y control horaria. En el dataGrid aparece la información del perfil de usuario, y en la parte derecha un Canvas con un TabNavigator que contendrá en cada pestaña información de Marcajes e incidencias para cada usuario seleccionado en el Datagrid principal. La Figura 21 muestra el contenido de cada una de estas pestañas.

La Figura 22 y Figura 23 expone los prototipos para la gestión de Jornadas laborales, la ventana emergente para su creación o modificación, la botonera contextual, el contenido de la pestaña Jornadas Laborales, compuesto por la visualización de los datos que constituyen una jornada laboral el parte derecha de visualización y un DataGrid en la parte izquierda que contiene todas las Jornadas Laborales creadas en el sistema.

Por último la Figura 24 presenta el contenido de la pestaña Terminales Kreta 3 que contiene un DataGrid con información de todos los terminales conectados al sistema, así como su botonera relacionada para realizar distintas operaciones sobre los diferentes terminales.

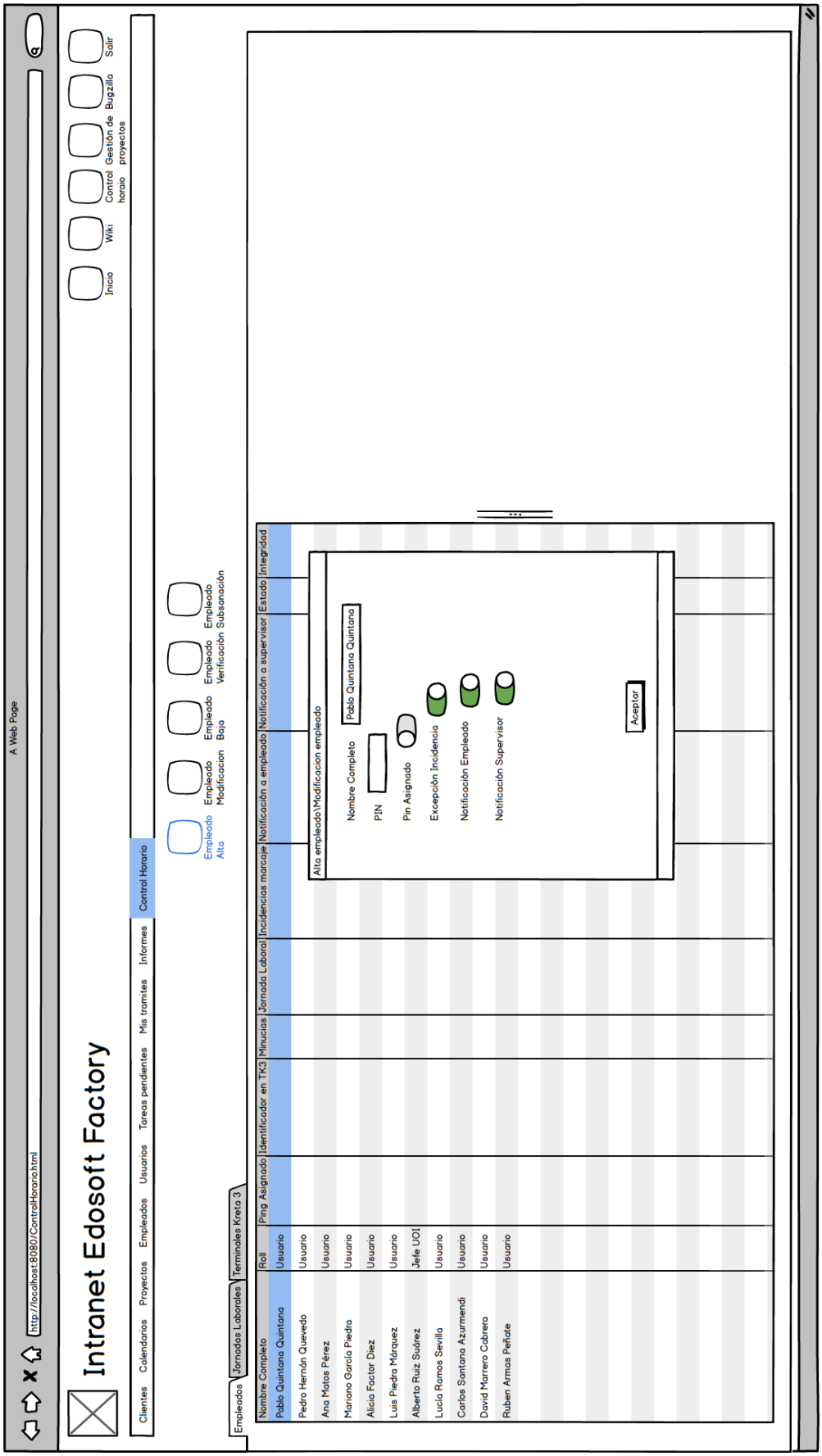

Figura 19. Prototipo de interfaz gráfica del modulo de control y gestión de horarios I

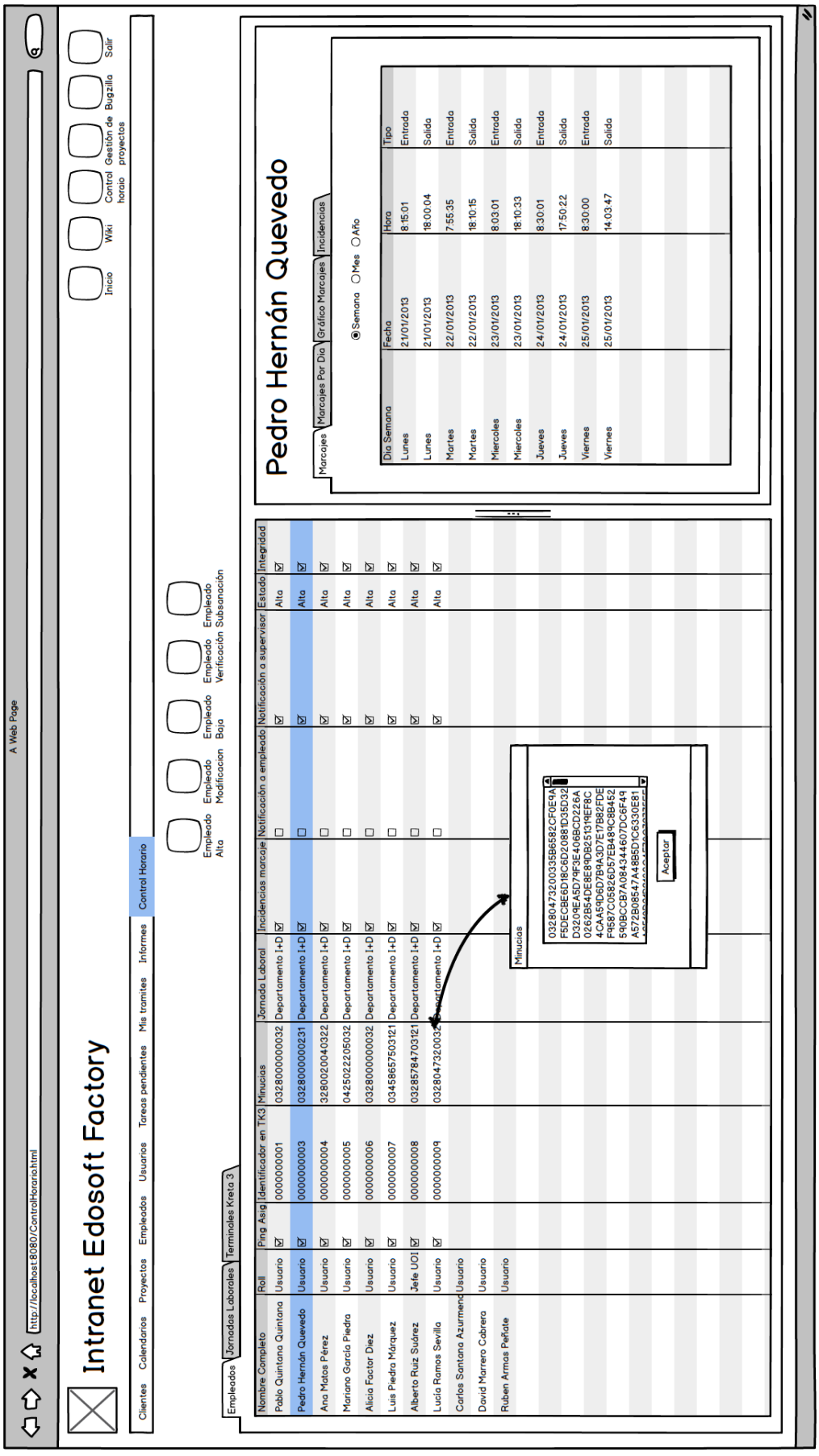

Figura 20. . Prototipo de interfaz gráfica del modulo de control y gestión de horarios II

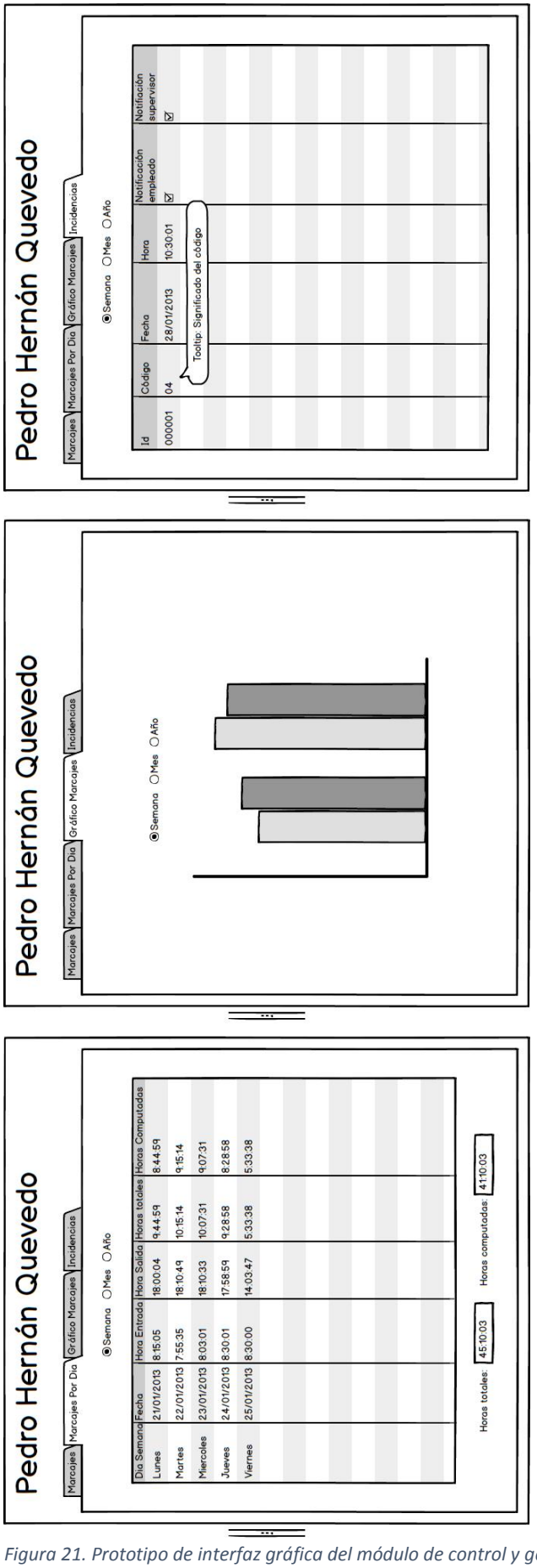

Figura 21. Prototipo de interfaz gráfica del módulo de control y gestión de horarios III

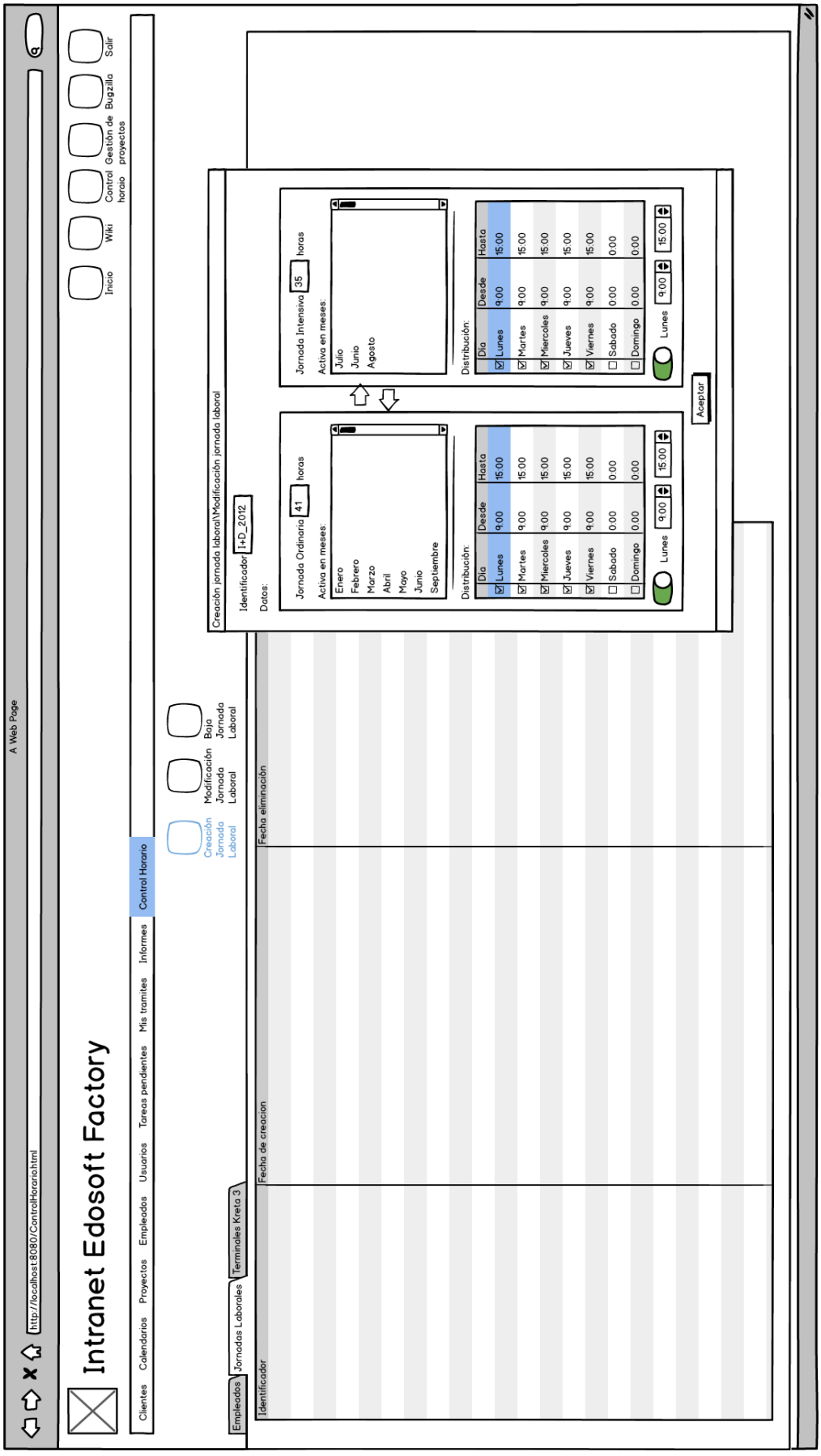

Figura 22. Prototipo de interfaz gráfica del módulo de control y gestión de horarios IV

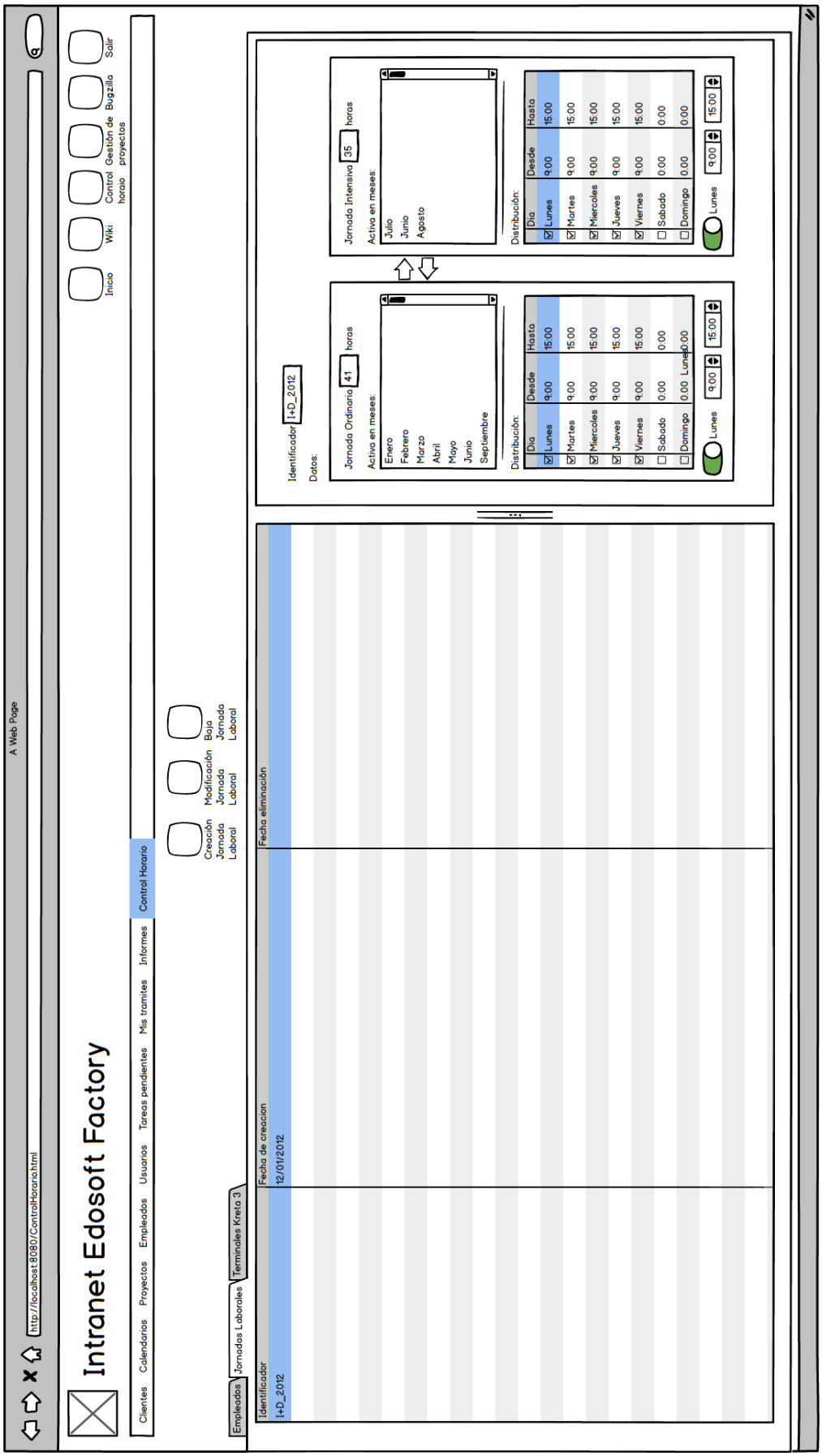

Figura 23. Prototipo de interfaz gráfica del modulo de control y gestión de horarios V

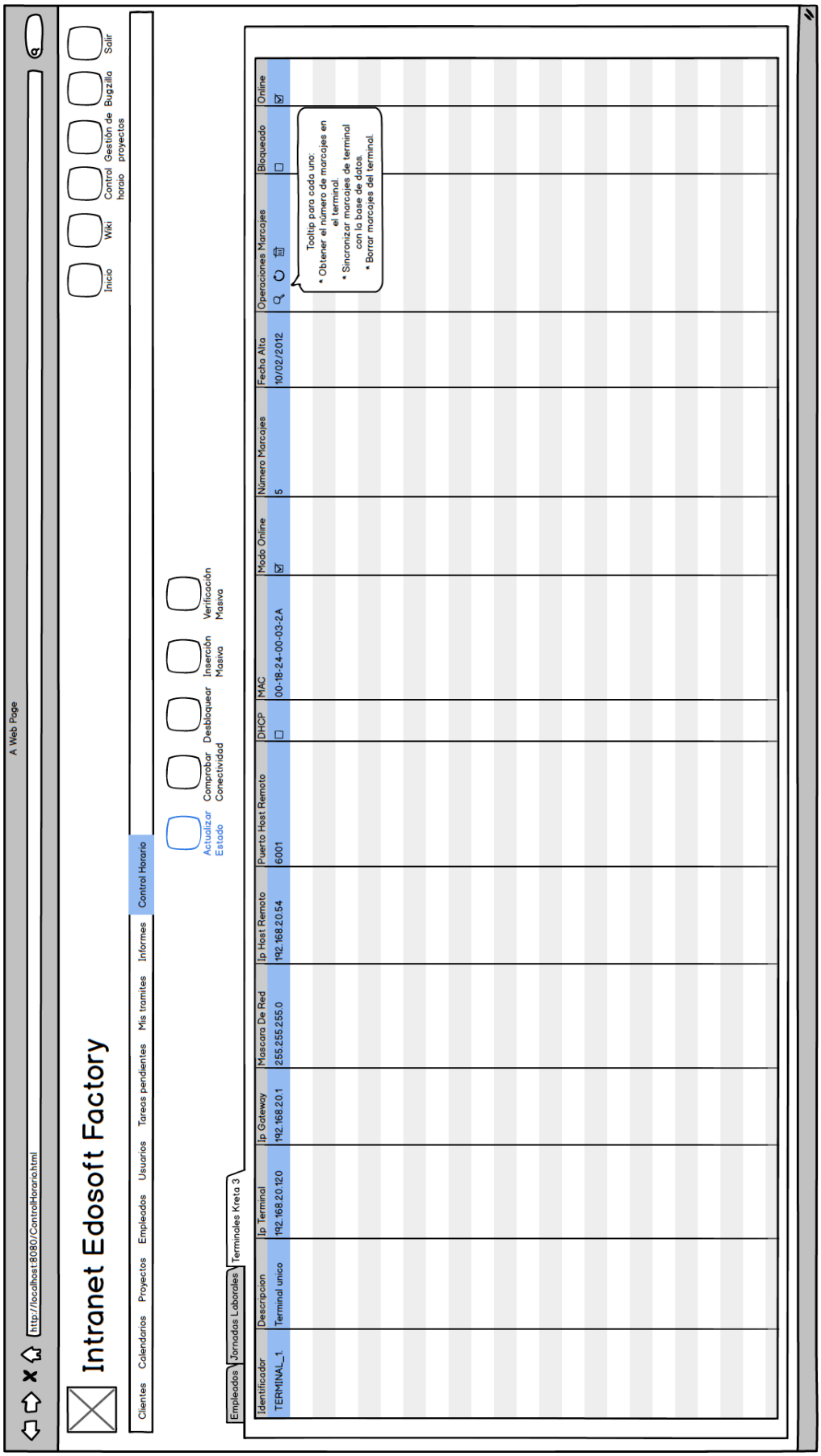

Figura 24. Prototipo de interfaz gráfica del modulo de control y gestión de horarios VI

Prototipos para el roll usuario sin privilegios.

La Figura 25 contiene el prototipo de pantalla principal para un usuario sin privilegios, en el que su área de visualización solo se muestra el TabNavigator referente a marcajes e incidencias del propio usuario. La información contenida en el resto de pestaña es la misma que las vistas en la Figura 21 con anterioridad

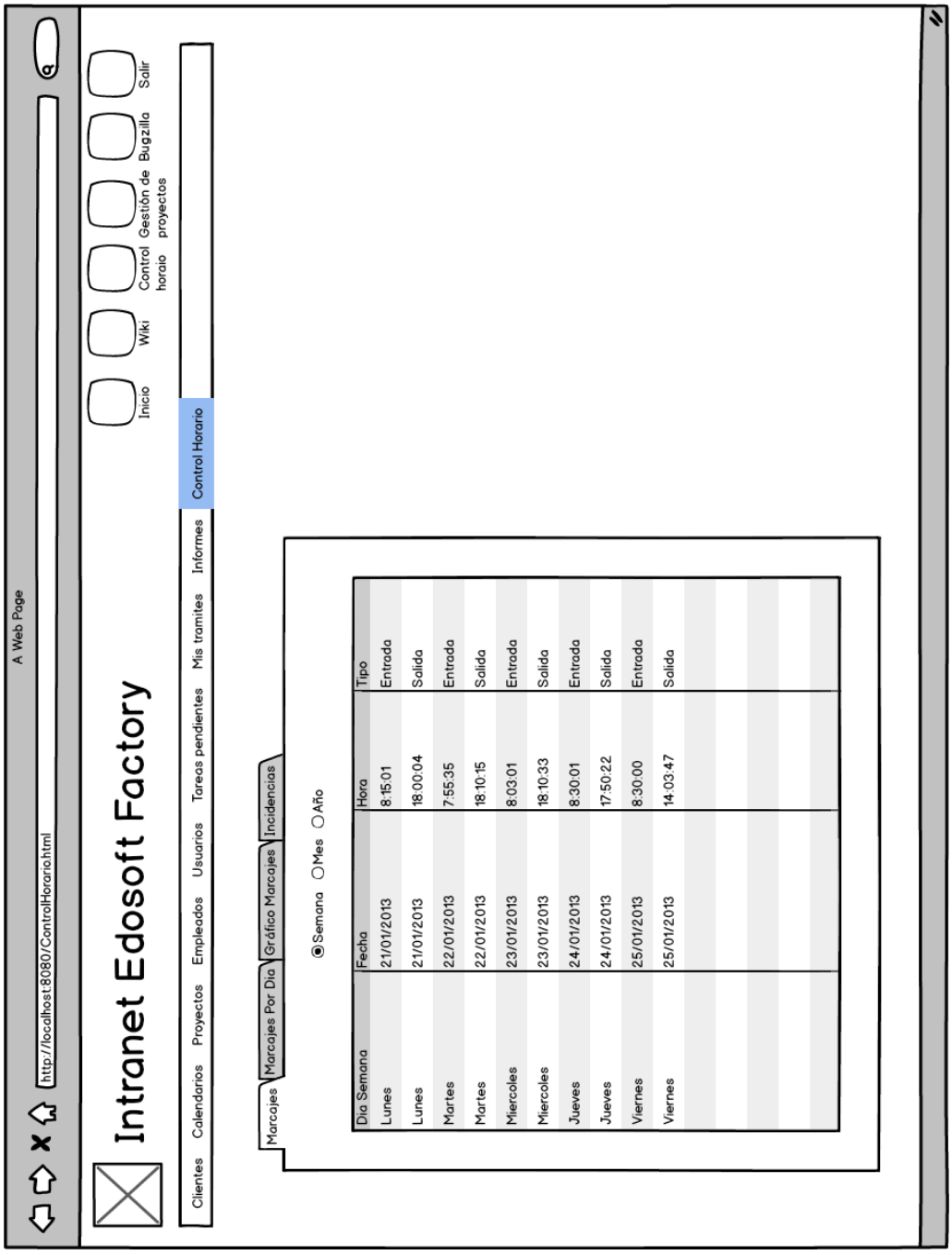

Figura 25. Prototipo de interfaz gráfica del modulo de control y gestión de horarios VII

6.3 Diseño del conector

Java EE Connector Architecture (JCA) es una solución tecnológica basada en el Lenguaje de programación Java para conectar servidores de aplicaciones y sistemas de información empresarial (EIS) como parte de soluciones de integración de aplicación de empresa (EAI). Mientras JDBC se usa específicamente para conectar aplicaciones Java a Bases de datos, JCA es una arquitectura más genérica para conectarse a sistemas heredados (incluyendo bases de datos). JCA fue desarrollado bajo el Java Community Process como JSR 16 (JCA 1.0), JSR 112 (JCA 1.5) y JSR 322 (JCA 1.6). Dada la versión de J2EE que se emplea la versión adecuada de JCA es 1.5

JCA especifica una serie de interfaces a implementar que definen casi por completo el diseño del de este componente.

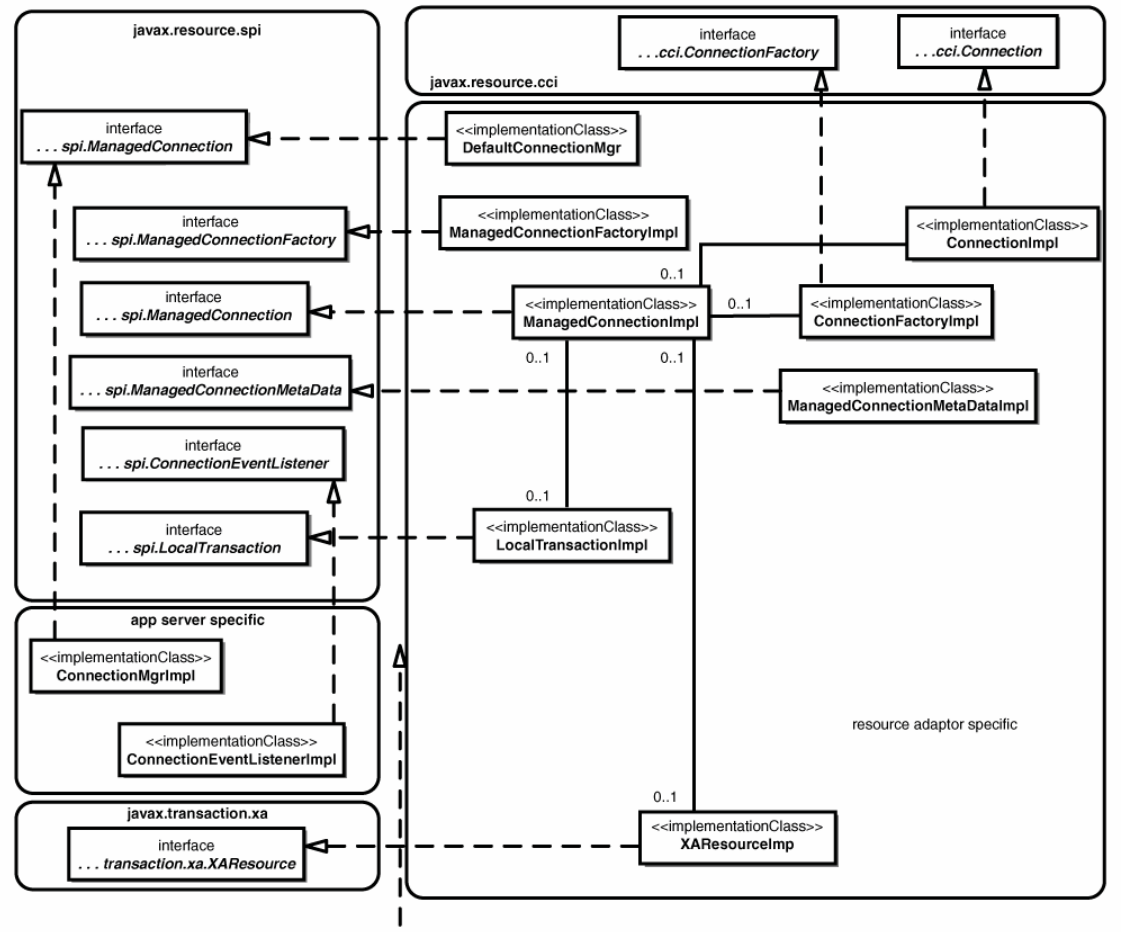

Figura 26. Diagrama de clases de la especificación del conector JCA

El diagrama de la Figura 26 muestra las interfaces, las clases que las implementan y las relaciones de dependencia entre entras clases.

Cabe destacar que hay una pequeña parte del diseño, de bajo nivel, no definido por la estructura inherente al conector. Se trata de la manera en que se definen y mantienen las conexiones con los terminales en el ámbito del conector. Es implementación desde el punto de vista del conector pero que también requiere la realización de un diseño previo, que se muestra en el diagrama de clases de la Figura 27.

Se trata pues de un pool de conexiones que se establecen con los terminales mediantes sockets. Este pool centraliza toda la operatividad de comunicación, pudiendo realizarse desde esta clase las conexiones iniciales, recepción de datos y el envío de datos.

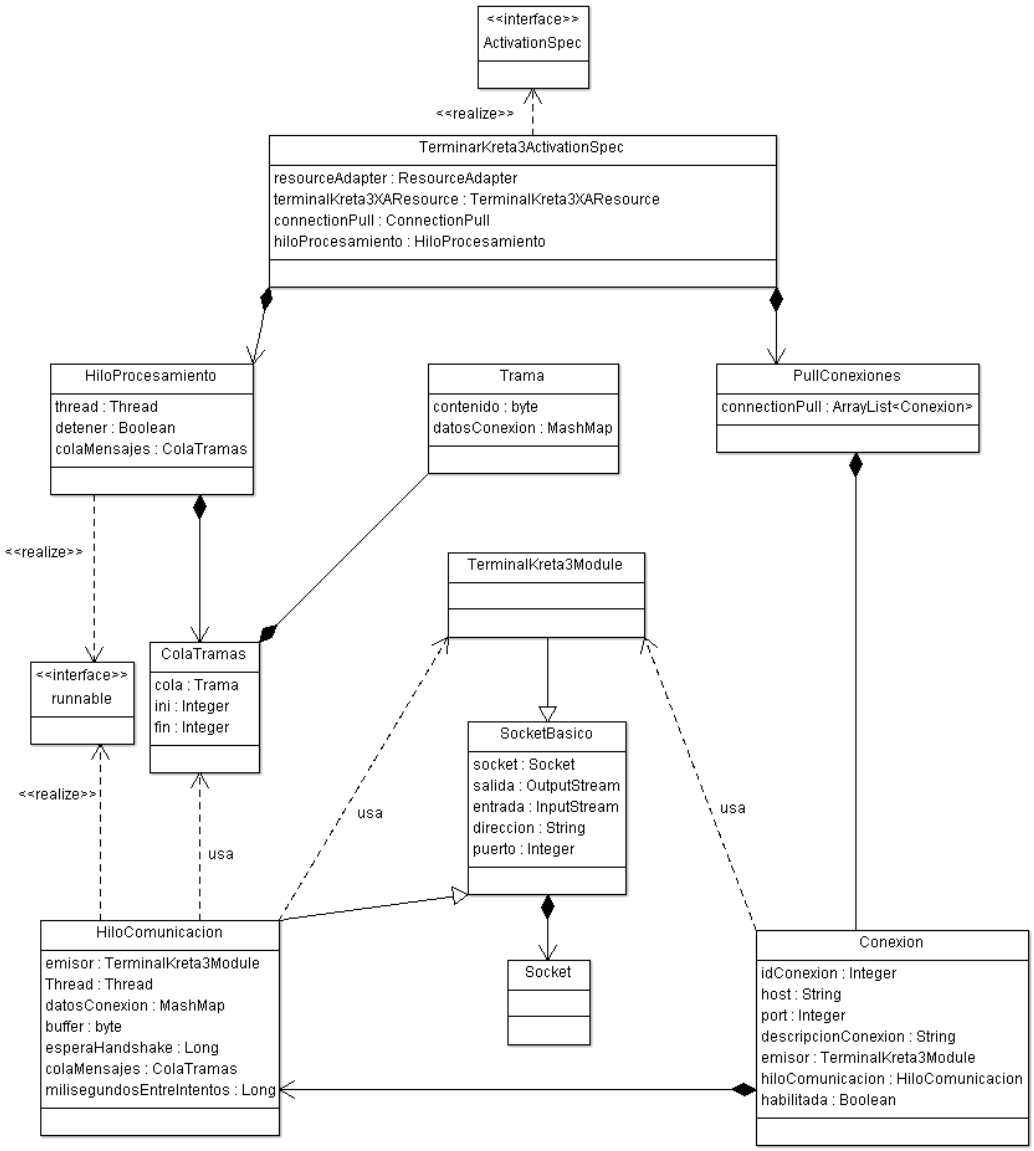

Figura 27. Diagrama de clases de la implementación del conector.

Por último se exponen un breves nociones de como interactúa el pool de conexiones en el ámbito del conector, haciendo un repaso a los métodos más significativos de las implementaciones a realizar.

Clase TerminalKreta3XAResource

El método `sendReceivedMessageToMDB` es el encargado de invocar invocar al método `processMessage`, definido en la interfaz `TerminalKreta3Listener` y desarrollado en la correspondiente implementación de la misma.

Clase TerminalKreta3ResourceAdapter

Mantiene los datos de conexión (direcciones IP, puertos) y descripción para todos los terminales configurados en el terminal en el archivo `META-INF/ra.xml`

El método `endpointActivation` instancia los objectos de las clases `TerminalKreta3Activation SpecTerminalKreta3XAResource` que son mantenidos por esta clase. Para la primera asigna el `Endpoint`, que no es más que el listener para que pueda ser invocado el método `processMessage`. Con respecto a la segunda llama al método `connectToTerminalKreta3` para realizar las conexiones a los terminales, previamente asigna el propio objeto de clase `TerminalKreta3ResourceAdapter` en esta, para poder acceder a los datos necesario para las conexiones.

Clase TerminalKreta3ActivationSpec

Tal y como muestra la Figura 27 mantiene el pool de conexiones a los terminales.

El método `connectToTerminalKreta3` crea un pool de conexiones, una por cada terminal configurado. Una vez establecida cada conexión envía un handshake consistente en una trama de askForReady.

Cada conexión tiene asociado un hilo de comunicación (clase `HiloComunicación`) donde se realiza la conexión mediante un Socket TCP/IP (clase `SocketBasico`) y donde se leen las tramas recibidas por el mismo insertándolas en una cola. Y un hilo procesamiento (clase `HiloProcesamiento`) que lee de esta cola, que se instancia en esta clase con una implementación inline del método abstracto `tratarTrama`, en donde se hace una llamada al método `tratarTramaLocal` de esta clase. De esta manera el hilo procesamiento llama a `tratarTrama` para cada trama recibida, que llamara a su

vez al `tratarTramaLocal`, que como se verá llama a `sendReceivedMessageToMDB` de `terminalKreta3XAResource`, que finalmente invoca al listener. La conexión tiene asociada también un emisor de la clase `TerminalKreta3Module`, encargada del envío de tramas. Esta clase le será asignada un socket activo una vez se haya producido la conexión en el contexto del hilo de comunicación.

También se asigna el objeto de la clase `TerminalKreta3XAResource` a partir del objeto `resourceAdapter` seteado previamente. Esto permitirá llamar al método `sendReceivedMessageToMDB`

Finalmente inicia los hilos de procesamientos asociados a cada conexión encargados de procesar las tramas recibidas, para que comiencen la espera activa por tramas en el socket.

El método `tratarTramaLocal` invoca a `sendReceivedMessageToMDB` de `terminalKreta3XAResource` encargado de invocar al listener.

Clase TerminalKreta3Module

Clase que hereda de `SocketBasico` con lo que permitirá setear un socket con conexión ya establecida con el método `addActiveSocket`, permitiendo el envío de tramas mediante el método `enviar_datos`. Este objeto es mantenido por cada conexión activa en el pool

Clase TerminalKreta3ManagedConnectionFactory

Mantiene una instancia de `TerminalKreta3ResourceAdapter` como punto de entrada para el envío de tramas a través de las conexiones.

Clase TerminalKreta3ConnectionFactory

La clase permite obtener objetos de clases que implementen la interfaz `Connection`, es decir, `TerminalKreta3Connection` en este caso, que se crea y se le asigna el pool de conexiones a partir del objeto de la clase `TerminalKreta3ManagedConnectionFactory` pasado en el constructor.

El método `getConnection` devuelve el objeto de la clase `TerminalKreta3Connection` que mantiene el pool de conexiones.

El método `getRecordFactory` devuelve el objeto de la clase `RecordFactory` que permite la creación de `MappedRecord`.

Este objeto será accesible desde el código mediante jndi y es el primer punto de acceso al conector.

Clase TerminalKreta3Connection

Mantiene el pool de conexiones que es seteado mediante el método `setConnectionPull`.

El método `createInteraction`, instancia y retorna un objeto de la clase `TerminalKreta3Interaction` con el pool de conexiones pasado en el constructor, que será utilizado para el envío de tramas desde el código.

Clase TerminalKreta3Interaction

Es el último punto de entrada al conector desde el código

El método `execute` recibe los datos mediante un objeto de la clase `Record`, que contiene información sobre la trama a enviar y sobre la conexión a utilizar, y se encarga de enviarlo empleando el pool.

Clase TerminalKreta3MappedRecord

Clase utilizada para contener la información de un determinado envío de datos mediante el conector. Además contiene información para la composición de tramas de órdenes, cabeceras, terminación de bloque y tramas especiales.

Clase TerminalKreta3RecordFactory

Factoría de `MappedRecord`, mediante `createMappedRecord` se obtiene un objeto de la clase `TerminalKreta3MappedRecord`.

7. Estudio y prueba de la interfaz para el manejo del terminal biométrico Kreta3

7.1 Descripción general y funciones principales

El Módulo Kreta3 se presenta como un terminal de altas prestaciones y adaptable, mediante configuración, a muy diversas combinaciones de lectores, actuadores e interfaces de usuario. En este apartado se describirán solo los aspectos del terminal que son necesarios para el desarrollo de este proyecto:

Control de presencia (Aunque existe también el control de acceso, este no será empleado)

Base de datos para 15.000 marcajes.

Conectividad a Host: Ethernet.

Soporte de biometría:

- Hasta 2 sensores biométricos de identificación por huella.

- Hasta 4.000 usuarios (identificación offline)

- Modos de identificación: 1:1, 1:N, 1:n

Las interfaces de usuario que dispone el terminal se dividen en elementos de entrada y salida. Entre las de entrada las más importantes son:

Tecla verde (“Entrar”) Indica la Incidencia 01 (“Entrada”).

Tecla roja (“Salir”) Indica la Incidencia 02 (“Salida”).

Teclas 0..9 Permiten la introducción de un código numérico (típicamente el de PIN).

Entre los elementos que componen la interfaz de salida cabe señalar:

Display LCD Dispone de un display de 20 caracteres x 2 líneas. En la primera línea se indica la fecha y la hora, mientras que la segunda se utiliza para guiar al usuario en su interacción con el equipo.

Zumbador (Buzzer) Señaliza la correcta operación del teclado, así como el éxito o fracaso del proceso de identificación del usuario.

El módulo Kreta3 comprende diversos métodos de identificación de personas, pero para este desarrollo solo se empleara la identificación biométrica en modo 1:N

7.2 Comunicación IP

Para acceder a los servicios de comunicación IP, hay que configurar adecuadamente los parámetros de red. La configuración se puede realizar a través del protocolo Kreta Classic, vía RS-232, aunque por simplicidad se ha empleado el Servicio de Localización Kimaldi que se detalla a continuación. Una vez la configurado el equipo, se dispone de dos protocolos distintos a través de Sockets, TCP o UDP. Para este proyecto se empleará comunicación TCP.

El servicio de Localización Kimaldi hace posible la detección de los módulos Kreta3 que estén conectados a nuestra red de área local. Una vez localizado el equipo a partir de su dirección MAC es posible configurar sus parámetros IP y reiniciar el equipo. La petición de la localización desde el Host, hacia el módulo Kreta3 se realiza a través del puerto 2000. Y la recepción de tramas en el Host. Procedentes del módulo Kreta3 se realizan a través del puerto 2001.

Cabe destacar que existe una DLL proporcionada por Kimaldi que permite la integración de este Servicio en cualquier aplicación software. La integración de éste funcionalidad en la intranet no será llevada a cabo, pero puedo ser una mejora para desarrollos futuros, y como tal será incluido en este capítulo de la memoria.

La comunión tal y como se ha comentado con anterioridad se realiza a través de Sockets TCP. El protocolo utilizado es el Kreta-Classic disponible para este tipo de comunicación. El módulo Kreta3 se encuentra en modo servidor. Por tanto, puede aceptar tramas desde cualquier Host (ver SLK-Seguridad en Apartado 10.1.6.) El socket de conexión siempre será iniciado por el Host. Kreta3 sólo generará eventos TCP mientras dicho socket esté activo. La transmisión de comandos desde el Host hacia el módulo Kreta3 se realiza a través de un puerto arbitrario del Host (Local Port del Host). La recepción de tramas en el módulo Kreta3 se realiza a través del puerto 1001 (Remote Port desde el Host). Las tramas serie consistirán en valores ASCII-Hex según el formato de

trama descrito en el próximo capítulo. Se incluirán los caracteres delimitadores <STX>, <ETX>.

Es interesante subrayar que aunque el módulo Kreta3 dispone de una dirección lógica para estar integrado en una red KPS, el direccionamiento del mismo se realiza a partir de su dirección IP. De entre los datos de configuración relativos a la comunicación del terminal destacan dos: la dirección MAC y la dirección IP.

La dirección MAC de cada módulo Kreta3 es única y asignada por el fabricante. Está etiquetada encima de la electrónica en formato hexadecimal. Es un parámetro de solo lectura.

La dirección IP de cada módulo Kreta3 será asignada por el usuario en función de las características del área local a la que se conecte. También puede ser asignada por el servidor de la LAN, si se tiene activado el protocolo DHCP. En general, se trata de un campo en la configuración, modificable por el usuario a través del Servicio de Localización Kimaldi o de cualquier servicio de conexión al Host.

Por ultimo hablar sobre el formato de transmisión de datos hacia el lector. En el caso de manejar lectores biométricos tipo FIM2030, surgirá la necesidad de enviar información "a través" del módulo Kreta3, para algunas operaciones. Estas tramas no serán interpretadas por el módulo Kreta3, y por tanto deberán ser manejadas de un modo distinto a las tramas habituales. Por los motivos expuestos, las tramas dirigidas directamente a los lectores se denominarán "Extra Data", y se transmitirán a continuación de un carácter separador de bloques (carácter "End Transmission Block" o <ETB>, ASCII $17). El formato "Extra Data" a través de TSP es: [Longitud][Datos][CRC].

- [Longitud]: Se trata de 4 bytes en formato ASCII-Hex. La longitud mayor soportada es 912 ($0390).

- [Datos]: Cadena de 2*[Longitud] bytes. Contiene la información binaria expresada en ASCII Hex. Por ejemplo, un byte de valor $3F se codificará como '3F', que son los bytes [$33 $46].

- [CRC]: Se trata de 2 bytes en formato ASCII-Hex. Constituye la suma de todos los bytes ASCII-Hex de [Longitud] y [Datos] en módulo 255, y luego con vertido este valor a ASCII-Hex.

Recordar que “Extra Data” vendrá precedido del carácter <ETB> (End Transmission Block, ASCII $17), y se cerrará la trama con el carácter <ETX> (ASCII $03).

7.3 Presentación general del funcionamiento del módulo Kreta3

El equipo Kreta3 integra en un solo equipo un "Control de Presencia" y un "Control de Accesos". Estas funciones se pueden desempeñar separadamente o conjuntamente según sea la configuración del equipo. Tres bytes de configuración, CFG_Lista, CFG_Presencia y CFG_Accesos permiten las distintas posibilidades de funcionamiento que se muestran en el siguiente cuadro:

CFG_Lista	CFG_Presencia	CFG_Accesos	Funcionamiento
Blanca o Libre	TRUE	FALSE	Sólo control de presencia.
Blanca o Libre	FALSE	TRUE	Sólo control de accesos.
Blanca o Libre	TRUE	TRUE	Control de presencia y control de accesos.
Blanca o Libre	FALSE	FALSE	Automático: Control de presencia o control de accesos
Negra	X	X	Terminal bloqueado.
Verde	X	X	Accesos abiertos.

En todos los casos, se generará un único marcaje con la información de todos los procesos realizados.

La configuración para todos los terminales se realizará por defecto y de manera fija para que realicen funciones de control de presencia y acceso, para lo cual se configurará los tres bytes mencionados tal y como indica la tabla para este caso. Con el parámetro CFG_Presencia a "TRUE", el equipo Kreta3 registrará un marcaje por cada lectura procedente de su lector principal. Si además el byte de configuración CFG_LectorAuxiliar está a "TRUE", también se registrarán marcajes a partir de las lecturas capturadas por el lector auxiliar. Como no se emplea el lector auxiliar CFG_LectorAuxiliar estará asignado a “FALSE” de forma general y fija para todos los terminales.

Cada marcaje recoge el código de la tarjeta o de usuario, la fecha y la hora de su generación, y un código de incidencia que indica el significado del evento.

El equipo Kreta3 dispone de dos algoritmos distintos para generar marcajes por el lector principal. El byte de configuración CFG_FicharMasivo permite seleccionar el algoritmo deseado. Con CFG_FicharMasivo a "FALSE", el equipo Kreta3 interrogará a cada usuario el código de incidencia a registrar. Para minimizar las pulsaciones del teclado en caso de afluencia masiva, configure CFG_FicharMasivo a "TRUE". De este modo el equipo asociará automáticamente un código de incidencia, sin necesidad de teclear.

Aunque no se espera una afluencia masiva de empleados se habilitará el fichaje masivo asignando al parámetro CFG_FicharMasivo el valor “TRUE”, con el fin de ahorrar un paso a los usuarios del sistema. Con este modo activado se emplea la “Adquisición del código de incidencia en Marcaje Masivo”. En este modo el equipo Kreta3 exhibirá igualmente la fecha y la hora en la primera línea de la pantalla y en la segunda se mostrará uno de los dos mensajes de modo de marcaje masivo, por ejemplo, "MODO ENTRADA" y "MODO SALIDA". El modo puede seleccionarse pulsando las teclas de Entrada/Salida. Si se registra una lectura mientras se exhibe el mensaje "MODO ENTRADA" se asociará el código de incidencia 01, mientras que si se muestra "MODO SALIDA" la incidencia asociada será la 02.

Estos valores por defecto de las incidencias de “MODO ENTRADA” y “MODO SALIDA” se encuentran codificados en el parámetro CFG-UI_Incid_FicharMasivo de las respectivas UIs, y pueden ser modificados si es necesario.

Una vez obtenida la huella dactilar asociada a la incidencia establecida, se comprobará su existencia en la base de datos. Si no existe se indicará también por pantalla y se generará un marcaje indicando el tipo de error.

Si existe dicho usuario y es una ENTRADA se pasa a continuación a control de acceso. Si fuera una SALIDA se visualizará el texto asociado por la pantalla y se generará un evento con la incidencia obtenida. El evento se guardará en la base de datos del equipo Kreta3 para su posterior lectura por parte del Host. Adicionalmente, el evento se puede transmitir de forma inmediata al Host, en caso que el parámetro CFG_TrazaMarcaje valga “TRUE”.

El algoritmo de control de acceso realiza diferentes comprobaciones en función de la configuración del terminal. Con la configuración que se empleada solo se control de horario y verificación del PIN, en caso de estar activo.

Para ello consultará en primer lugar el registro de permisos del usuario en cuestión, se tomará de él el código semanal a utilizar para posteriormente leer el registro semanal y extraer de él el código horario a aplicar.

La comprobación previa a este primer paso, del estado presente/ausente en el registro de permisos, para en caso de estar ya presente en el interior del recinto impedir el acceso y se generar un evento de error, se obviará al tener asignado el parámetro CFG_Antipassback el valor “FALSE”.

Una vez obtenido el código del horario por uno de los dos métodos, se obtiene el registro horario y se comprueba si la hora actual pertenece a alguno de los tres intervalos del registro. En caso afirmativo se continúa con la comprobación del PIN. En caso contrario se generará un evento de error.

En el siguiente paso se pregunta el PIN de acceso, si así consta en el registro de permisos. Si se falla el PIN se genera un evento de error. En caso de PIN coincidente o que no esté activado habrá concluido exitosamente el acceso. Se mostrará su texto asociado por la pantalla y se generará un evento de acceso con la incidencia obtenida.

Como ocurre con las SALIDAS el evento se guardará en la base de datos del equipo Kreta3 para su posterior lectura por parte del Host. Adicionalmente, el evento se puede transmitir de forma inmediata al Host, en caso que el parámetro CFG_TrazaMarcaje valga “TRUE”.

Cuando el proceso finaliza con el control de presencia se denomina evento de marcaje, y cuando está activo el control de acceso se denomina evento de acceso, aunque ambos tipos son almacenados en la tabla marcajes, distinguidos por su código de evento. Por lo que según esta configuración las entradas son eventos de control de acceso y las salidas eventos de marcajes. Esta distinción es realiza las propias especificaciones del terminal, pero en general hacemos referencia a ambos como marcajes de entrada y salida.

7.4 Modelo de programación

El modelo de programación es el conjunto de estructuras de datos e instrucciones que dispone el programador para acceder a ellas.

7.4.1 Configuración

La configuración del equipo consta de numerosas estructuras. Se muestra a continuación tan solo aquellas que son significativas desde el punto de vista de configuración y manejo del terminal en el ámbito del módulo de control y gestión de horarios. Así mismo, dentro de cada estructura solo se hace referencia a los parámetros o elementos relevantes desde el mismo punto de vista.

Array de parámetros, Kreta3-DB

El array de parámetros en el módulo Kreta3-DB consta de 42 posiciones de un byte. Se numeran de la 1 a la 42 (en hexadecimal de la $01 a la $2A) y se expresan mediante dos dígitos hexadecimales. Cuando se represente un valor booleano se tomará como "FALSE" el valor $00 y como "TRUE" cualquier otro.

Los parámetros de configuración que se emplean en este desarrollo se muestran a continuación, con su posición, nombre y significado:

- $01 - CFG_Presencia: Es un valor Booleano. A "TRUE" obliga a realizar siempre las funciones de control de presencia del equipo.

- $02 - CFG_Acceso: Es un valor Booleano. A "TRUE" obliga a realizar siempre las funciones de control de accesos del equipo.

- $04 - CFG_FicharMasivo: Es un valor Booleano. A "TRUE" habilita los modos de "Entrada masiva" y "Salida Masiva" evitando así que deba teclearse el código de incidencia para cada marcaje.

- $16 - CFG_Lista: Configura el modo de funcionamiento del control de acceso. El Valor $00 significa Lista Blanca, $01 Lista Libre y $03 Lista Verde. Por último, $02 ó $FF corresponden a Lista Negra.

- $1A - CFG_TrazaMarcaje: A "TRUE" genera una traza hacia el Host en el momento en que se realiza un marcaje.

- $03 - CFG_Antipassback: Es un valor Boolano. A "TRUE" activa la característica antipassback. Sólo se manifiesta si el equipo está configurado como lector de accesos. De esta forma solo se permitirá la entrada si en el registro de Permisos, este usuario consta como ausente.

Array de días

Las denominaciones de los días de la semana que se muestran en la pantalla son configurables y está almacenadas en un array de 7 posiciones de 3 caracteres ASCII imprimibles cada una. El array posee 7 nombres. Con índice $01 consta el Lunes y el Domingo con índice $07.

Array de meses

Las denominaciones de los meses del año que se muestran en la pantalla son configurables y están almacenadas en un array de 12 posiciones de 3 caracteres ASCII imprimibles cada una. El array posee 12 nombres. Con índice $01 consta el mes de Enero y el mes de Diciembre con índice $0C

Configuración IP - Sockets TCP

Los parámetros de configuración TCP/IP que se emplean en este desarrollo se muestran a continuación, con su posición, nombre y significado:

- $01 - IP-Client: Son ocho caracteres ASCII-Hex, indicando la dirección del terminal Kreta3 que se está configurando.

- $02 - IP-Gateway: Son ocho caracteres ASCII-Hex, indicando la dirección de la puerta de enlace de nuestra red IP.

- $03 - IP-NetMask: Son ocho caracteres ASCII-Hex que configuran la máscara de subred de nuestra red IP.

- $04 - IP-Server: Son ocho caracteres ASCII-Hex, indicando la dirección del servidor DHCP. Este parámetro suele configurarse automáticamente si DHCP está activado.

- $05 - IP-RemoteHost: Son ocho caracteres ASCII-Hex, indicando la dirección del Host desde el cual se establece comunicación con el terminal Kreta3 a través de un Socket UDP. Para comunicarse a través del Socket TCP/IP, no es necesario especificar este parámetro, si bien es recomendable (aumenta la seguridad del equipo).

- $06 - IP-Context: Obsoleto. Dejar al mismo valor que la IP-Netmask.

- $07 - Port-RemoteHost: Son cuatro dígitos BCD que indican el puerto del Host por el que se va a establecer la comunicación con el Kreta3 (Protocolo Kreta-Classic vía UDP). Su valor por defecto es 5001. Si su valor es 0000, el módulo Kreta3 responderá automáticamente hacia el puerto del Host que haya originado la trama.

- $08 - SLK-Seguridad: si el bit más significativo está a 1, sólo se permite la comunicación TCP/IP desde la IP declarada como IP-RemoteHost. Si el bit más significativo está a 0, la comunicación TCP/IP se puede establecer desde cualquier host. Por motivos de seguridad, es recomendable filtrar la IP del Host, de modo que el valor por defecto de este parámetro será $FF. Los siete bits menos significativos están reservados para uso futuro.

- $09 - DHCP: Si está a "TRUE", es el servidor quien se encarga de asignar la dirección IP al terminal Kreta3. Si está a "FALSE", la dirección IP será fija, y determinada por nosotros.

- $0A - MAC-Address: Son doce caracteres ASCII-Hex que especifican los 6 bytes de la dirección MAC. Este valor solamente se puede acceder en lectura, no en escritura.

- $0D - Descripción: Es una cadena de texto de hasta 40 caracteres que nos permite identificar el terminal en lenguaje humano (por ejemplo: "Terminal Puerta 1").

Todos los parámetros listados son manejados desde la aplicación, algunos son leídos y escritos y otros leídos solamente. La mayoría de estos en el proceso de configuración inicial.

7.4.2 Base de Datos

La base de datos del equipo se almacena en SRAM y consta de una serie de tablas. Se describen a continuación aquellas que son manejadas por la aplicación.

Tabla de Reloj/Calendario

Esta tabla contiene un único registro con la fecha, la hora y el día de la semana. Su valor se actualiza automáticamente sin necesidad de apagar la alimentación. Este registro tiene el formato AAMMDDhhmmssnn dónde AAMMDD es la fecha, hhmmss corresponde a la hora, y nn es el día de la semana contando a partir de 01 para el lunes.

Ejemplo:

02050912453004

Corresponde al día 9 de Mayo del 2002 a las 12:45:30 siendo Jueves.

Tabla de Horarios

La tabla de horarios tiene capacidad para 255 registros numerados del $01 al $FF, cada uno de los cuales está formado por tres campos de igual formato que representan los intervalos horarios con acceso permitido. Cada intervalo se define por su hora de inicio y su hora de fin en formato hhmm. Para que una hora pertenezca al intervalo deberá ser mayor o igual que la hora de inicio del intervalo y estrictamente menor que la hora de fin. Para poder generar intervalos de 24 horas se admite como hora de fin de intervalo las 24:00

Ejemplo:

Supóngase el registro:

090010001200130015001600

O sea:

- Campo_Intervalo1: 09001000
- Campo_Intervalo2: 12001300
- Campo_Intervalo3: 15001600

Está definiendo tres intervalos, el primero va de las nueve de la mañana a las diez. El segundo de las doce del mediodía a la una de la tarde. Y el tercero de las tres a las cuatro de la tarde. Dentro de estos periodos el acceso está permitido.

Si nuestra aplicación no necesita alguno de los intervalos del registro pueden dejarse a 00000000. Téngase presente que un intervalo con Hora Fin = Hora Inicio no contiene ningún valor de hora, por lo que este intervalo no contiene las cero horas.

Tabla Semanal

La tabla semanal tiene capacidad para 255 registros numerados del $01 al $FF. Cada registro contiene siete campos, uno para cada día de la semana (de lunes a domingo). Cada campo tiene dos dígitos hexadecimales que especifican el número de registro horario que hay que aplicar cada día de la semana. Se puede especificar el valor de horario 00 que se tomará como aquel cuyos tres intervalos son nulos.

Ejemplo:

El registro $01 contiene:

01010101010200

O sea:

- Campo_Horario_Lunes: 01
- Campo_Horario_Martes: 01
- Campo_Horario_Miercoles: 01
- Campo_Horario_Jueves 01
- Campo_Horario_Viernes: 01
- Campo_Horario_Sábado: 02
- Campo_Horario_Domingo: 00

Este registro especifica el empleo del horario número 01 de lunes a viernes. El sábado debe usarse el horario 02 y el domingo no tiene horario asignado.

Tabla de Permisos

La tabla de permisos tiene capacidad para 7.500 registros. Cada registro contiene cinco campos. Veamos su significado con un ejemplo:

Tomemos el registro:

01234567890112340100

Tenemos:

• Campo_Código_Usuario: 0123456789

Especifica el código hexadecimal de diez dígitos correspondientes al código de usuario. Para el motor de la base de datos éste es un campo clave único, de manera que solamente existe un solo registro por usuario.

• Campo_Usar_Pin: 01

Este campo de dos dígitos hexadecimales representa un valor booleano. Si es cero se tomará como falso y en caso contrario será verdadero. Indica si el control de acceso deberá preguntar el PIN a este usuario.

• Campo_PIN: 1234

Este campo contiene cuatro dígitos decimales que representan el PIN que el usuario deberá teclear para poder acceder.

• Campo_Semanal: 01

Los dos dígitos hexadecimales de este campo representan el número de registro semanal que se usará por defecto para este permiso en caso que no se use la excepción de tipo $20 que se explicará más adelante.

• Campo_Presente: 00

Este campo representa un valor booleano. Su significado es el de usuario presente en el interior del recinto.

Tabla de marcajes

La tabla de marcajes es una FIFO con capacidad para 15000 registros. El módulo irá almacenando los eventos por estricto orden de generación y el programador podrá recuperarlos en el mismo orden.

Veamos un registro de marcajes.

012345678902050318102538 21

• Identificador de usuario: 0123456789

Especifica el código hexadecimal de diez dígitos correspondientes al código de usuario.

• Campo_Fecha_Hora: 020503181025

Contiene la fecha y la hora de generación del evento en formato AAMMDDhhmmss

• Campo_Código_Evento: 31

Contiene dos dígitos hexadecimales que informan sobre el tipo de evento.

El primer dígito indica lo siguiente:

- 0: Evento generado por el lector principal funcionando en modo automático.

- 1: Evento generado por el lector principal funcionando como presencia.

- 2: Evento generado por el lector principal funcionando como accesos.

- 3: Evento generado por el lector principal funcionando como presencia+accesos.

- 4: Evento generado por el lector auxiliar.

- A: Evento generado por el monitor del estado de la batería.

- B: Evento generado por el avisador de cambio de turno.

- E: Extensión del código de incidencia

- F: Evento generado por el gestor de alarma.

Dada la configuración estática de los terminales para funcionar en modo control de presencia más control acceso el valor de este digito será siempre 3.

El segundo dígito tiene el siguiente significado:

- 0: Evento de acceso correcto
- 1: Evento de marcaje correcto
- 2: Evento de acceso correcto por excepción
- 3: Evento de acceso correcto por horarios
- 4: Evento de marcaje erróneo por incidencia
- 5: Evento de acceso denegado por permiso
- 6: Evento de acceso denegado por PIN
- 7: Evento de acceso denegado por excepción
- 8: Evento de acceso denegado por horario
- 9: Evento de acceso denegado por antipassback
- A: Evento de acceso denegado por festivo
- B: Evento de acceso denegado por exceso de aforo
- D: Evento de enrolamiento biométrico diferido
- E: Evento de enrolamiento biométrico erróneo
- F: Evento de activación Online (accesos); Evento de alarma o timbre

El segundo dígito contendrá el valor 3 indicando un marcaje correcto por horarios o los valores 5, 6, 7, en caso de que

• Campo_Incidencia: 21

Para eventos de Presencia/Acceso el campo incidencia contiene dos dígitos hexadecimales con el número de incidencia tecleado por el usuario si éste es el caso. En nuestro caso para un marcaje de entrada tendrá el valor “01” y de salida con el valor “02”.

7.4.2 Descripción de las instrucciones

Las instrucciones que admite el módulo Kreta3 así como las respuestas que emite tienen el siguiente formato:

{Grupo}{Objeto}{Operación}{Datos}

Donde:

- {Grupo} es un dígito decimal ASCII.

- {Objeto} es un dígito decimal ASCII.

- {Operación} es un dígito decimal ASCII.

- {Datos} es una cadena de caracteres ASCII.

Se muestra a continuación el conjunto de operaciones utilizadas para la configuración y operatividad necesaria del terminal.

Operaciones de configuración

Las instrucciones de configuración pertenecen al {Grupo}='3' y sus respuestas al {Grupo}='4'.

Los objetos definidos empleados son:

- Registro de Parámetros: {Objeto}='1'

- Configuración IP-UDP Socket: {Objeto}='5'

- Estructura de la Base de Datos: {Objeto}='6'

Y las operaciones que aceptan que se han empleado son:

- Write {Operación}='1': Permite almacenar datos en la memoria del equipo.

Formato instrucción:

- {Objeto}: 1,5

- {Operación}='1'

- {Datos}:

• Objeto ‘1’: se especificará mediante dos dígitos hexadecimales el número de parámetro a escribir ($01 al $2A) y dos dígitos más con el valor deseado para el parámetro.

• Objeto ‘5’: se especificará mediante ocho dígitos hexadecimales la dirección IP. En caso de valores booleanos (parámetro $09), se usarán dos dígitos hexadecimales. En caso de Descripción (parámetro $0D), se usarán hasta 40 caracteres ASCII.

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.

- {Operación}: el que aparece en la instrucción.

- {Datos}:

• Objetos ‘1’ y ‘5’: operación correcta indicada por 11.

- Read {Operación}='2': Permite leer los valores almacenados en la memoria del equipo.

Formato instrucción:

- {Objeto}: 1,5 y 6.

- {Operación}: 2

- {Datos}: Se especificará mediante dos dígitos hexadecimales la posición del array que se desea consultar.

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.

- {Operación}: el que aparece en la instrucción.

- {Datos}: el contenido de la posición del array referenciado.

• Objeto ‘6’: se usa para verificar la estructura de la base de datos (por ejemplo el número de permisos, que tenemos disponibles). Se especificará un valor entre 01 y 09, correspondiente a los objetos de la base de datos. Observar que este objeto es Read-Only.

Operaciones sobre la base de datos

Las instrucciones sobre la base de datos pertenecen al {Grupo}='5' y sus respuestas al {Grupo}='6'.

Los objetos definidos empleados son:

- Reloj/Calendario: {Objeto}='0'
- Tabla Horarios: {Objeto}='2'
- Tabla Semanal: {Objeto}='3'
- Tabla Permisos: {Objeto}='6'
- Tabla Marcajes: {Objeto}='8'

Y las operaciones que aceptan que se han empleado son:

- DeleteRecord {Operación}='2': Permite borrar un registro de la base de datos.

  Formato instrucción:

  - {Objeto}: 2,3,6,8
  - {Operación}: 2
  - {Datos}: Para las tablas 2,3 se especificará el número de registro a borrar mediante cuatro dígitos hexadecimales.

    Para las tabla 6 se especificará el código de usuario a borrar mediante 10 dígitos hexadecimales

    Para la tabla 8 no se especificará argumento porque siempre se borrará el registro más antiguo (no se podrá borrar un marcaje si no se ha leído previamente a través de la instrucción ‘587’)

  Formato Respuesta:

  - {Objeto}: el que aparece en la instrucción.
  - {Operación}: el que aparece en la instrucción.
  - {Datos}: 11 para indicar operación correcta.

- Store {Operación}='4': Permite guardar/modificar un registro de una tabla.

Formato instrucción:

- {Objeto}: 0,2,3,6

- {Operación}: 4

- {Datos}: Para la tabla 0 se especificara la fecha y hora en formato AAMMDDhhmmssnn dónde AAMMDD es la fecha, hhmmss corresponde a la hora, y nn es el día de la semana contando a partir de 01 para el lunes.

Para las tablas 2 y 3 se especificará el número de registro a guardar/actualizar mediante cuatro dígitos hexadecimales seguido del valor del registro.

Para la tabla 6 se especificará directamente el valor del registro a guardar/actualizar. En caso de estar dando de alta un nuevo usuario y usar módulos FIM2030, deberá añadirse la huella biométrica.

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.

- {Operación}: el que aparece en la instrucción.

- {Datos}: 11 para indicar operación correcta.

- Size {Operación}='5': Devuelve el número de registros activos que contiene una tabla.

Formato instrucción:

- {Objeto}: 8

- {Operación}: 5

- {Datos}: Ninguno.

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.

- {Operación}: el que aparece en la instrucción.

- {Datos}: cuatro dígitos hexadecimales indicando el número de registros activos de la tabla.

- Size&Begin {Operación}='6': Devuelve el número de registros activos que contiene una tabla y a demás sitúa al principio de la tabla el puntero de lectura secuencial de dicha tabla.

- Retrieve {Operación}='7': Devuelve el contenido de un registro de una tabla.

Formato instrucción:

- {Objeto}: 0,2,3,6,8

- {Operación}: 7

- {Datos}: Para las tablas 0,8 no hay argumentos

Para las tablas 2,3 se especificará el número de registro a consultar mediante cuatro dígitos hexadecimales.

Para las tablas 6 se especificará el código de usuario. Para la tabla 6, existe la opción de recuperar la huella biométrica. Ver siguiente subapartado.

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.

- {Operación}: el que aparece en la instrucción.

- {Datos}: el contenido del registro indicado.

- RetrieveNext {Operación}='8': Devuelve el contenido del primer registro activo encontrado después de la posición apuntada por el puntero de lectura secuencial de esa tabla.

Formato instrucción:

- {Objeto}: 6
- {Operación}: 8
- {Datos}: No hay argumentos

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.
- {Operación}: el que aparece en la instrucción.
- {Datos}: el contenido del registro indicado si existe. En caso contrario se devolverá 00.

Operaciones sobre periféricos (FIM2030)

Las instrucciones sobre algunos periféricos del módulo Kreta3 pertenecen al{Grupo}='7' y sus respuestas al {Grupo}='8'.

El único objeto definido que hemos empleado es:

- FIM2030 actuando como Lector Principal: {Objeto}='1'

Y única operaciones utilizada para este objeto es:

- Scan FP {Operación}='3': Permite escanear una o dos veces la huella de un usuario. Esta función es Online, de modo que desde el Host se puede gestionar el proceso de enrolamiento de un nuevo usuario.

Formato instrucción:

- {Objeto}: 1
- {Operación}: 3

- {Datos}: Se especificará mediante dos dígitos hexadecimales el número de veces que se escanea la huella (sólo es posible $01 ó $02).

Formato Respuesta:

- {Objeto}: el que aparece en la instrucción.

- {Operación}: el que aparece en la instrucción.

- {Datos}: '11' seguido de la cadena en formato Extra Data conteniendo una o dos huellas.

7.5 Pruebas y ejemplos de uso

Se muestra a continuación el formato de algunas de las tramas empleadas y ejemplos de tramas utilizas tanto desde la aplicación de prueba suministrada por el fabricante como en un pequeño cliente de comunicaciones desarrollado previo al desarrollo del módulo. En primer lugar las referentes a la gestión de usuarios.

Alta de usuario

El alta de usuario se realiza a través de la operación '564'. El formato de trama será el siguiente:

<STX>564[CódigoUsuario|PIN|Semanal|Presencia]3<ETB>[Extra_Data]<ETX>

Es decir, justo al final del comando usado habitualmente para dar de alta, se añade:

- '3': Dirección a la que vamos a mandar los datos biométricos. '1' correspondería al FIM Principal, '2' al FIM Auxiliar y '3' a los dos anteriores.

- <ETB> (End Transmission Block, ASCII $17): Termina el bloque de comando y va a empezar el bloque de Extra Data.

- [Extra Data]: Las dos muestras de la huella biométrica son transferidas en un formato binario.

Ejemplo de trama alta del usuario 0000000001 con PIN activo 1234, código semanal 05, no presente y su correspondiente huella:

```
{0x02, 0x35, 0x36, 0x34, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x31, 0x31, 0x31, 0x31, 0x32, 0x33, 0x34, 0x30, 0x35, 0z30, 0x30,
0x33, 0x17, [400 bytes], 0x03}
```

Modificación del permiso de usuario

Una vez dado de alta un usuario, podemos modificar su código semanal o su PIN sin necesidad de adjuntar de nuevo su huella biométrica. Para ello, usaremos la misma operación '564' en su formato habitual:

<STX>564[CódigoUsuario|PIN|Semanal|Presencia]<ETX>

Ejemplo de trama de modificación del usuario 0000000001 deshabilitando PIN.

```
{0x02, 0x35, 0x36, 0x34, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x31, 0x30, 0x30, 0x31, 0x32, 0x33, 0x34, 0x30, 0x35, 0z30, 0x30,
0x03}
```

Baja de usuario

El proceso de baja de usuario se realiza normalmente a partir del código de usuario. El módulo Kreta3 procederá automáticamente a borrar la memoria del módulo biométrico que funcione como lector principal, y eventualmente del auxiliar. No se requiere manejar información biométrica.

<STX>562[CódigoUsuario]<ETX>

Ejemplo de trama de eliminación del usuario 0000000001:

```
{0x02, 0x35, 0x36, 0x33, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x31, 0x03}
```

Lectura del permiso de usuario

La operación '567' (ver Apartado 11.3.7.) nos permite recuperar el registro de la tabla de permisos correspondiente a un código determinado de usuario:

<STX>567[CódigoUsuario]<ETX>

Si además queremos recuperar la información biométrica del usuario, deberemos añadir la dirección del módulo biométrico al final del comando:

<STX>567[CódigoUsuario]0<ETX>

Es decir, justo al final del comando usado habitualmente para realizar la lectura, se añade:

- '0': Dirección de la que vamos a leer los datos biométricos. '0' significa "indistinto" y nos devuelve la lectura del primer FIM que la tenga disponible (normalmente, el FIM Principal). '1' correspondería exclusivamente al FIM Principal, mientras que'2'forzaría la obtención de la huella procedente del FIM Auxiliar.

La respuesta correcta será evidente. No obstante, en caso de error, éstos son los códigos que aparecen:

- '667A0': ha fallado la lectura en el FIM Principal, y no existe FIM Secundario.

- '667B0': ha fallado la lectura en el FIM Secundario.

- '667C0': ha fallado la lectura en los dos FIM.

Ejemplo de trama de obtención del usuario 0000000001 en cualquiera de los dos FIM:

```
{0x02, 0x35, 0x36, 0x37, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x31, 0x30, 0x03}
```

Obtención de datos biométricos del usuario

Previo al proceso de alta, será necesario obtener los datos biométricos del usuario que vamos a registrar. En aplicaciones de un solo módulo Kreta3, o bien para simplificar al máximo el software de aplicación, es posible escanear las huellas de usuario a través del propio módulo Kreta3. En esta circunstancia, el módulo Kreta3 está operando Online, puesto que se limita a mandar al Host los datos obtenidos.

Veamos con un poco de detalle cómo funciona esta segunda opción.

Para dar de alta a un nuevo usuario, necesitamos dos lecturas de la misma huella dactilar. Dado que el formato "Extra Data" requiere de un CRC, lo más sencillo será pedir

directamente un escaneo doble, de modo que los datos retornados se puedan "copiar y pegar" directamente para realizar el proceso de alta:

Instrucción Scan FP:

<STX>71302<ETX>

- Respuesta:

<STX>81311<ETB>[Extra_Data]<ETX>

- La cadena se puede adjuntar directamente al comando Store:

<STX>564[CódigoUsuario|PIN|Semanal|Presencia]3<ETB>[Extra_Data]<ETX>

Ejemplo de trama de bytes para la obtención de datos biométricos:

```
{0x02, 0x37, 0x31, 0x33, 0x30, 0x32, 0x03}
```

Ejemplo de trama de bytes de respuesta:

```
{0x02, 0x38, 0x31, 0x33, 0x31, 0x31, 0x17,[400 bytes], 0x03}
```

Cada huella dactilar ocupa unos 400 bytes de memoria y reside físicamente en la memoria no volátil del módulo biométrico FIM2030. La escritura y recuperación de los datos biométricos de cada usuario se realiza de forma integrada con la Tabla de Permisos, a través de las operaciones descritas en este apartado.

Por último mostraremos algunas referentes a la configuración del terminal.

Obtención de diversos parámetros de configuración

La obtención de los parámetros de configuración se realiza a través de la operación '352'. El formato de trama será el siguiente:

<STX>352|NumParametro<ETX>

Ejemplo de trama de obtención del parámetro IP del Terminal:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x31, 0x03};
```

Ejemplo de trama de obtención del parámetro IP del Gateway:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x32, 0x03};
```

Ejemplo de trama de obtención del parámetro Mascara de red:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x33, 0x03};
```

Ejemplo de trama de obtención del parámetro IP del Host remoto:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x35, 0x03};
```

Ejemplo de trama de obtención del parámetro Puerto del Host Remoto:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x37, 0x03};
```

Ejemplo de trama de obtención del parámetro DHCP:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x39, 0x03};
```

Ejemplo de trama de obtención del parámetro Dirección MAC:

```
{0x02, 0x33, 0x35, 0x32, 0x30, 0x41, 0x03};
```

Ejemplo de trama de respuesta del parámetro IP del Terminal:

```
{0x02, 0x34, 0x35, 0x32,[0x67, 0x48, 0x65, 0x56, 0x48, 0x49, 0x48, 0x65:
IP] ,0x33};
```

Configuración de la tabla horarios y tabla semanal

La configuración de la tabla horarios se realiza a través de la operación ‘524’. El formato de trama será el siguiente:

<STX>524| CodigoRegistro|Periodo1|Perido2|Periodo3 <ETX>

Ejemplo de trama para la inserción de un registro con código 0001 en la tabla horarios, con un único intervalo de 06:30 a 24:00:

```
{0x02, 0x35, 0x32, 0x34, 0x30, 0x30, 0x30, 0x31, 0x30, 0x36, 0x33,
0x30, 0x32, 0x34, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,
0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30, 0x30,0x03};
```

Ejemplo de trama de bytes de respuesta:

```
{0x02, 0x36, 0x32, 0x34};
```

La configuración de la tabla semanal se realiza a través de la operación ‘534’. El formato de trama será el siguiente:

<STX>534| CodigoResgistro|HorarioLunes|HorarioMartes|..|HorarioDomingo<ETX>

Ejemplo de trama para la configuración de tabla semanal con asignación del horario anteriormente descrito para todos los días de la semana:

```
{0x02, 0x35, 0x33, 0x34, 0x30, 0x30, 0x30, 0x35, 0x30, 0x31, 0x30,
0x31, 0x30, 0x31, 0x30, 0x31, 0x30, 0x31, 0x30, 0x31, 0x30, 0x31,
0x03};
```

Ejemplo de trama de bytes de respuesta:

```
{0x02, 0x36, 0x33, 0x34};
```

7.6 Dificultades encontradas

Durante la fase de pruebas iniciales del terminal, entiéndase pruebas iniciales como toma de contacto, o incluso durante la implementación del módulo de control y gestión de horarios, se han encontrado los siguientes problemas a los que se le han dado adecuadas soluciones.

Control de flujo en la configuración inicial.

Tras la conexión inicial se inicia la configuración del terminal y la lectura. En una implementación inicial las órdenes de este proceso que se enviaban al terminal eran despachadas de forma secuencial y sin ningún control de flujo. Esto producía que a partir de la cuarta orden enviada el terminal no atendía a la mayoría de las órdenes posteriores, no llegando respuesta alguna para estas.

Para solucionar este problema se implementó un control de flujo de tal manera que la siguiente orden es envía solo cuando se recibe la respuesta de la anterior. De esta manera se aseguró el hecho de no desbordar la capacidad de respuesta del terminal. Estas peticiones iniciales no llegan a 10 por terminal, por lo que el tiempo invertido en la configuración del mismo mediante este procedimiento sigue siendo adecuado.

Tamaño de trama limitado

Durante la fase de pruebas del terminal se descubrió una limitación en el comportamiento, que para las tramas de respuestas a órdenes de gran tamaño, las que contienen ‘ExtraData’, que eran recibidas, lo hacían en dos tramas de comunicaciones diferenciadas. El fin de flujo del InputStream asociado al Socket alcanzaba el fin de flujo

antes que concluyera la recepción de la totalidad de la trama. Posteriormente llegaba otra trama que completaba la respuesta.

La solución adoptada tanto para el programa inicial de pruebas y para el desarrollo final es tener en cuenta esta eventualidad y proporcionar una solución basada en los caracteres de inicio de bloque <STX>, y final de bloque <ETX> que se encuentran al principio y al final de toda respuesta envía desde el terminal. De esta manera si se recibe una respuesta sin carácter de final de bloque, se almacena en un buffer hasta la llegada de la trama que finalice con este, tras la cual se conforma la trama completa y se procesa como una trama standard que hubiera llegado de una solo vez.

Perdida de la configuración por defecto del terminal

Tras un periodo de inactividad prolongado el terminal biométrico perdió su configuración de fábrica por completo. El terminal biométrico posee numerosos y complejos parámetros de configuración de bajo nivel. Estos parámetros son a nivel de montaje, puertos internos, velocidad de comunicación de estos, etc. Conocimiento lejos del nivel de programación en el que se ha movido el proyecto durante su trascurso.

El motivo de esta pérdida de configuración fue que la pila se agotó durante un periodo prolongado en que el terminal estuvo desconectado.

Tras la consulta al Servicio técnico de Kimaldi proporcionaron una aplicación que con unos sencillos pasos retornaba al terminal a su configuración inicial. Aunque la configuración del terminal podía haber sido llevado a cabo siguiendo los manuales de instrucciones, esto hubiera llevado un tiempo considerable, que con la solución aportada por el fabricante se ahorró.

Fuente de alimentación estropeada

La fuente de alimentación del terminal se estropeó, posiblemente por una sobre tensión. La solución fue adquirir otra de las mismas características y evitar la utilización del terminal sin la protección de una UPS en el entorno de desarrollo. Para su instalación final esta medida ya estaba contemplada.

8. Implementación

Una vez realizado el diseño de los distintos componentes que conforman el módulo de control y gestión de horarios para la ICEF se ha realizado la implementación de los mismos, en los lenguajes y con las tecnologías ya seleccionadas. El conocimiento del repertorio de órdenes necesarias para configurar y operar con el terminal facilitó la implementación de la comunicación con el terminal.

Este capítulo se limitará a describir los proyectos software que componen el módulo y la estructuración de estos. Así mismo se aportarán detalles de la compilación, los ejecutables resultantes de la misma y su despliegue en el servidor de aplicaciones.

En el Apéndice II. JavaDoc se incluye el JavaDoc de los métodos de la fachada `UsuarioFacade` relacionados con el control y la gestión horaria. En el CD-Rom adjunto a esta memoria se incluyen los ficheros fuentes de estos proyectos con el JavaDoc de las partes más significativas de los proyectos Java, así como otros contenidos de interés. Ver Apéndice IV. Contenido CD-Rom

El código entregado en los proyectos adjuntos contienen tan solo el código fuente desarrollado para este proyecto de fin de carrera, y el propio de la ICEF que es estrictamente necesarios para la compilación y su correcta ejecución, modificado incluso para la eliminación de la referencias a código suprimido. Se han eliminado por tanto todos los módulos de la ICEF desarrollados por Edosoft, permaneciendo únicamente el módulo de control y gestión de horarios.

8.1 Implementación del cliente (GUI)

El proyecto que conforma la implementación de la GUI corresponde a un proyecto Flex que puede ser editado y compilado con la herramienta Adobe Flex Builder 3. El proyecto de nombre `ControlHorario` contiene los siguientes elementos:

- `asserst`: Contiene los diferentes elementos gráficos utilizados en la GUI tales como iconos, botones, fondos, etc
- `lib`: Directorio de librerías, que contiene como único elemento la librería del framework Cairgorm, llamada Cairgorm.swc.
- `src`: Contiene el código fuente estructurado en paquetes.

- `bin-debug`: Directorio creado por defecto en los proyectos Flex. Utilizado para ejecuciones en modo debug desde el entorno de desarrollo.
- `html-template`: Directorio creado por defecto de los proyectos Flex. Este contiene la plantilla html sobre la que se montará la aplicación final.
- `controlHorario.mxml`: Componente gráfico principal de la aplicación

Se muestra a continuación la estructuración del código fuente en el directorio src. Esta estructuración viene definida por los diferentes elementos del framework Cairgorm empleado.

- `com.edosoftfactory.intranet.business`: Contiene las clases ActionScript que implementan los BussinesDelegate Cairgorm.
- `com.edosoftfactory.intranet.commands`: Contiene las clases ActionScript que implementan los Comandos Cairgorm.
- `com.edosoftfactory.intranet.constantes`: Contiene la clase `Constantes`.
- `com.edosoftfactory.intranet.control`: Contiene la clase `Controller` del Cairgorm.
- `com.edosoftfactory.intranet.events`: Contiene las clases ActionScript que implementan los Eventos Cairgorm.
- `com.edosoftfactory.intranet.mensajeria`: Contiene la clase `Mensajeria` que implementa un Topic para la recepción de mensajes desde el servidor.
- `com.edosoftfactory.intranet.model`: Contiene la clase ActionScript `ModelLocator` de Cairgorm.
- `com.edosoftfactory.intranet.model.vo`: Contiene todos los VOs que definen el modelo de datos en la interfaz gráfica. Sobre las instancias de estos objetos en el ModelLocator se realiza el bindaje con los elementos visuales.
- `com.edosoftfactory.intranet.pojos`: Contiene los pojos utilizados para recibir información desde el servidor.
- `com.edosoftfactory.intranet.utils`: Contiene la clase `Utils` que agrupa diferentes métodos estáticos que se utilizan de forma generalizada en el código.
- `com.edosoftfactory.intranet.view`: Contiene la definición de componentes gráficos de conforman la vista. Archivos en formato mxml.

La compilación se realiza en la versión del SDK 3.2 con los parámetros `-services "C:\proyectos\as\jboss-4.2.2.GA-icef\server\default\deploy\`

`intranetEF.ear\intranet.war\WEB-INF\flex\services-config.xml" -context-root "." -locale en_US` necesario para la utilización de la BlazeDS. BlazeDS es una tecnología de mensajería asincrónica para componentes de servidor desarrollados en Java que permite a los desarrolladores conectar aplicaciones Adobe Flex y Adobe AIR con entornos empresariales Java.

El resultado de la compilación se despliega directamente en el directorio `.war` ubicada en el `.ear` del directorio `deploy` del servidor de aplicaciones del entorno de desarrollo.

```
jboss-4.2.2.GA-icef\server\default\deploy\intranetEF.ear\intranet.war
```

De esta manera queda perfectamente ubicado para su ejecución en el servidor de aplicaciones como parte web de la aplicación ICEF.

8.2 Implementación del servidor

Se muestra es este subcapítulo todos los proyectos software que en conjunto implementan el servidor del módulo. Para estos se lista su contenido, y se da algunos detalles acerca de su compilación, dependencias y despliegue en entorno de desarrollo.

8.2.1 Implementación de la lógica de negocio

El proyecto que conforma la implementación de la parte servidora de la aplicación corresponde a un proyecto Java Empresarial que puede ser editado y compilado con la herramienta Eclipse. El proyecto de nombre IntranetEF contiene los siguientes elementos:

- `src`: Contiene el código fuente estructurado en paquetes.
- `docs`: Contiene los JavaDocs de las clases más significativas.
- `pojos`: Directorio de destino de los pojos de la interfaz gráfica que se generan de forma automática a partir de sus homónimos en la aplicación de servidor mediante la herramienta Gas3. Para realizar este proceso se dispone de una tarea al efecto en el bulid.xml para ser ejecutada mediante la herramienta Ant. Una vez creado son movidos manualmente al directorio de pojos de la GUI.
- `classes`: Compilados de la aplicación.

- `build.xml`: Fichero de entrada para la ejecución de diferentes tareas mediante Ant
- `build.properties`: Fichero que define propiedades empleado por build.xml

Se muestra a continuación la estructuración del código fuente en el directorio src:

- `com.edosoftfactory.intranet.ejb.entities`: Contiene beans de entidad.
- com.edosoftfactory.intranet.ejb.listener: Contiene el Message-driven beans, que es empleado como listener del conector.
- `com.edosoftfactory.intranet.ejb.sessions`: Contiene los beans de sesión.
- `com.edosoftfactory.intranet.ejb.encrypt`: Contiene utilidades para la encriptación de los datos en la base de datos.
- `com.edosoftfactory.intranet.pojos.controlhorario`: Contiene los pojos para el envío de información al cliente.
- `com.edosoftfactory.intranet.report`: Contiene las clases relacionadas con el envío de correos electrónicos.
- `com.edosoftfactory.intranet.util.controlhorario`: Contiene la clase Utils que agrupa diferentes métodos estáticos que se utilizan de forma generalizada en el código.
- `META-INF`: Directorio que se incluye en la generación del jar con diferente información.

La compilación se realiza en la versión del JDK 1.6 mediante la correspondiente tarea definida en `build.xml` y ejecutada mediante Ant. El resultado de la compilación se empaqueta en el correspondiente archivo `.jar` y se copia en el directorio `.ear` del directorio `deploy` del servidor de aplicaciones del entorno de desarrollo, mediante otra tarea al efecto.

Este proyecto presenta una dependencia con el proyecto del conector relativo a la interfaz TerminalKreta3Listener.

8.2.2 Implementación del conector

El proyecto que conforma la implementación del conector corresponde a un proyecto Java que puede ser editado y compilado con la herramienta Eclipse. El proyecto de nombre kreta3TerminalConnector contiene los siguientes elementos:

- `src`: Contiene el código fuente estructurado en paquetes.
- `build/clases`: Compilados de la aplicación.
- `deploy`: Directorio destino donde se copia el archivo jar empaquetado a partir de las clases compiladas, mediante la tarea al efecto en el `build.xml` para ser ejecutada mediante Ant. También contiene el archivo data source `kreta3Terminal-ds.xml`. Contiene los dos elementos necesarios para realizar el despliegue del conector en el servidor de aplicaciones.
- `build.xml`: Fichero de entrada para la ejecución de diferentes tareas mediante la herramienta Ant.

Se muestra a continuación la estructuración del código fuente en el directorio `src`:

- `com.edosoftfactory.intranet.terminalKreta3.comunicacion`: Contiene todas las clases relativas a la comunicación a nivel de sockets con los terminales.
- `com.edosoftfactory.intranet.terminalKreta3.jca.cci`: Contiene la implementación de las clases del conector relacionadas con la ejecución del mismo desde el código que lo emplea.
- `com.edosoftfactory.intranet.terminalKreta3.jca.inbound`: Contiene la interfaz que implementa el message-driven bean que actúa de Listener.
- `com.edosoftfactory.intranet.terminalKreta3.spi`: Contiene la implementación de las clases del conector relacionadas con la comunicación.
- `com.edosoftfactory.intranet.terminalKreta3.util`: Contiene la clase `Constantes`, `UtilesCadenas` y `ConnectorLogger`.

La compilación se realiza en la versión del JDK 1.6 mediante la correspondiente tarea definida en build.xml. Y el resultado de la compilación se empaqueta en el correspondiente archivo rar y se copia junto con el "data source" en el directorio `deploy` del proyecto y del servidor de aplicaciones del entorno de desarrollo, mediante la correspondiente tarea de la utilidad Ant.

8.2.3 Implementación de los planificadores

El proyecto que conforma la implementación de los planificadores corresponde a un proyecto Java que puede ser editado y compilado con la herramienta Eclipse. El proyecto de nombre Scheduler contiene los siguientes elementos:

- `srcControlDairio`: Contiene el código fuente del planificador Control Diario estructurado en paquetes.
- `buildControlDairio/clases`: Compilados del planificador Control Diario.
- `srcCambioModo`: Contiene el código fuente del planificador Control Diario estructurado en paquetes.
- `buildCambioModo/clases`: Compilados del planificador Control Diario.
- `deploy`: Directorio destino donde se copia los archivos sar, uno por cada planificador, empaquetado a partir de las clases compiladas, mediante la tarea al efecto en el `build.xml` para ser ejecutada mediante Ant. Contiene el elemento necesario para realizar el despliegue de los planificadores en el servidor de aplicaciones.
- `build.xml`: Fichero de entrada para la ejecución de diferentes tareas mediante la herramienta Ant.

Se muestra a continuación la estructuración del código fuente de los directorios srcControlDiario y srcCambioModo:

- `com.edosoftfactory.intranet.controlDiario.schedulerControlDiario`: Contiene la clase del planificador Control Diario.
- `com.edosoftfactory.intranet.cambioModo.schedulerCambioModo`: Contiene la clase del planificador Cambio Modo.

La compilación se realiza en la versión del JDK 1.6 mediante la correspondiente tarea definida en el archivo `build.xml`. Y el resultado de la compilación se empaqueta en los correspondientes archivos sar y se copian en el directorio `deploy` del proyecto y del servidor de aplicaciones del entorno de desarrollo, mediante la correspondiente tarea de Ant.

Este proyecto presenta una dependencia con el proyecto `IntenetEF` relativo a la interfaz remota del bean de sesión `ControlHorarioFacade`.

9. Pruebas

Las pruebas funcionales automatizadas diseñadas para el módulo de control y gestión de horarios abarca la mayor parte de la funcionalidad del servidor. Las pruebas relativas al cliente se han limitado a la especificación de una lista de comprobación que contempla la totalidad de los casos de uso, la cual se ha realizado de forma manual.

9.1 Pruebas GUI

Existen algunos frameworks de pruebas poco evolucionados para las pruebas automatizadas de aplicaciones desarrolladas en Flex. Por este motivo y dado el reducido número de casos de uso el test funcional de la interfaz gráfica del módulo corresponderá a una serie de casos de prueba propuestos que se realizara de forma manual.

Condiciones iniciales del test funcional

1. El terminal biométrico debe estar completamente vacío. Se puede comprobar su estado, y proceder al borrado de las diferentes tablas mediante la aplicación demo proporcionada por el fabricante.
2. Cambio de la fecha del sistema al día 09-27-2013, para la correcta visualización de la información de los datos de prueba.
3. El servidor de aplicaciones debe estar arrancado con el despliegue de la intranet realizado en el directorio `deploy` de este.
4. Ejecución del script de población al efecto adjunto en el proyecto de pruebas, en la ruta: "/recursos/icef_prod_recreate_20110709.sql", que inserta en la base de datos información sobre los empleados que se emplean para realizar los casos de prueba.
5. Acceso a la intranet corporativa:
    a. Acceder a la interfaz de la intranet mediante un navegador con Plugin de Adobe Flash instalado. (http://localhost:8080/ControlHorario.html)
    b. Realizar el login con dos diferentes usuarios en función del roll que se pretenda probar:
        i. Roll Administrador: Login: - Password:
        ii. Roll Usuario: Login: - Password:

Test funcional para Roll Administrador

El usuario de prueba sin privilegios, Prueba0, ya se encuentra dado de alta en el sistema con información biométrica, marcajes, e incidencias. Parte del cometido de este test es la comprobación de la correcta visualización de esta información por parte de usuarios con roll administrador.

Jornada Laboral	**Descripción**	**Resultado**
Caso 1	Crear Jornada Laboral	VERIFICADO
Caso 2	Modificar Jornada Laboral	VERIFICADO
Caso 3	Eliminar Jornada Laboral	VERIFICADO
Caso 4	Eliminar Jornada Laboral asociada a un empleado	VERIFICADO

Empleado	**Descripción**	**Resultado**
Caso 5	Alta Empleado correcta para empleado Prueba1.	VERIFICADO
Caso 6	Alta Empleado correcta para empleado Prueba1.	VERIFICADO
Caso 7	Modificación empleado Prueba1	VERIFICADO
Caso 8	Modificación empleado Prueba1	VERIFICADO
Caso 9	Baja Empleado Prueba1	VERIFICADO
Caso 10	Alta empleado Prueba1 tras baja previa	VERIFICADO
Caso 11	Verificación empleado Prueba1	VERIFICADO
Caso 12	Subsanación empleado Prueba1 innecesaria	VERIFICADO
Caso 13	Verificar empleado Prueba1, eliminado del terminal	VERIFICADO
Caso 14	Subsanación empelado Prueba1	VERIFICADO

Marcajes	Descripción	Resultado
Caso 15	Marcaje de entrada para Prueba1	VERIFICADO
Caso 16	Marcaje de salida para Prueba1	VERIFICADO

Terminal	Descripción	Resultado
Caso 17	Comprobar el número de marcajes en el terminal	VERIFICADO
Caso 18	Borrado de marcajes en el terminal	VERIFICADO
Caso 19	Comprobación de conectividad con terminal	VERIFICADO
Caso 20	Comprobar conectividad con terminal desconectado.	VERIFICADO
Caso 21	Desbloqueo de terminal en estado no bloqueado.	VERIFICADO
Caso 22	Desbloqueo de terminal en estado bloqueado.	VERIFICADO
Caso 23	Modificación de configuración del terminal	VERIFICADO

Marcajes e Incidencias	Descripción	Resultado
Caso 24	Comprobación de los marcajes de la semana en curso para el usuario Prueba0.	VERIFICADO
Caso 25	Comprobación de los marcajes del mes en curso para el usuario Prueba0.	VERIFICADO
Caso 26	Comprobación de los marcajes del año en curso para el usuario Prueba0.	VERIFICADO
Caso 27	Comprobación de las incidencias de la semana en curso para el usuario Prueba1.	VERIFICADO
Caso 28	Comprobación de las incidencias del mes en curso para el usuario Prueba0.	VERIFICADO
Caso 29	Comprobación de las incidencias del año en curso para el usuario Prueba0	VERIFICADO
Caso 30	Comprobación de las incidencias por tipo.	VERIFICADO

Test funcional para Roll Usuario

El usuario de prueba sin privilegios, Prueba0, ya se encuentra dado de alta en el sistema con información biométrica, marcajes, e incidencias. El cometido de este test es la comprobación de la correcta visualización de esta información para Prueba0 de sus propios marcajes e incidencias.

Marcajes e Incidencias	**Descripción**	**Resultado**
Caso 1	Comprobación de los marcajes de la semana en curso.	VERIFICADO
Caso 2	Comprobación de los marcajes del mes en curso.	VERIFICADO
Caso 3	Comprobación de los marcajes del año en curso.	VERIFICADO
Caso 4	Comprobación de las incidencias de la semana en curso.	VERIFICADO
Caso 5	Comprobación de las incidencias del mes en curso.	VERIFICADO
Caso 6	Comprobación de las incidencias del año en curso.	VERIFICADO
Caso 7	Comprobación de las incidencias por tipo.	VERIFICADO

9.2 Pruebas modelo de negocio

9.2.1 Especificación

A continuación se especifican los casos de prueba a desarrollar. Cada uno de los tests especificados pertenecerá a una o varias de las suits de pruebas (agrupación de casos de pruebas) según la naturaleza de la misma. La pertenencia de cada uno de los casos a dichas suists viene indicado en la las diferentes tablas de este capítulo. Existen 3 suits diferentes:

Casos de prueba manual (1)

En esta suit se engloban los casos de prueba que se realizan utilizando la intranet conectada al terminal biométrico. Al ejecutar la suit será necesario interactuar con el

terminal para el desarrollo de la misma., para la captura de datos biométricos en las altas de empleados.

Casos de prueba automático con un solo terminal (2)

En esta suit se engloban los casos de prueba que se realizan utilizando la intranet conectada al simulador de terminal biométrico en configuración de un solo terminal.

Casos de prueba automático con múltiples terminales (3)

En esta suit se engloban los casos de prueba que se realizan utilizando la intranet conectada al simulador de terminal biométrico en configuración de múltiples terminales.

Alta Usuario	**Descripción**	**1**	**2**	**3**
Caso 00	Alta exitosa de empleado en terminal/terminales.	X	X	X
Caso 01	Alta fallida de empleado en terminal. Empleado ya creado.	X	X	
Caso 03	Alta fallida del empleado en terminal (error en la captura de huellas [sin captura]).	X	X	
Caso 04	Alta empleado en terminal después de alta fallida (error en la captura de huellas [sin captura]).	X	X	
Caso 05	Alta fallida en terminal (Jornada Laboral no existe).	X	X	
Caso 06	Alta fallida. Terminal bloqueado	X	X	

Baja Usuario	**Descripción**	**1**	**2**	**3**
Caso 00	Baja exitosa de empleado en terminal/terminales.	X	X	X
Caso 01	Baja fallida de empleado en terminal. Empleado no existente.	X	X	
Caso 02	Baja fallida de empleado en terminal. EmpleadoTK3 no dado de Alta.	X	X	
Caso 03	Baja fallida de empleado en terminal. Empleado de baja.	X	X	
Caso 04	Baja fallida. Terminal bloqueado.	X	X	

Verificación Usuario	**Descripción**	**1**	**2**	**3**
Caso 00	Verificación de un usuario en terminal/terminales. Verificación correcta. Empleado completamente integro.	X	X	X
Caso 01	Verificación masiva en terminal/terminales. Resultado de las verificaciones: -ID_EMPLEADO: Correcto -ID_EMPLEADO_52: De baja en persistencia pero existe en terminal. -ID_EMPLEADO_53: De alta en persistencia pero no existe en terminal. -ID_EMPLEADO_54: Existe tanto en persistencia como en terminal pero con minucias diferentes -ID_EMPLEADO_55: Existe tanto en persistencia como en terminal pero con valor del atributo 'pin activo' diferente. Sistema necesita subsanación.	X	X	X
Caso 02	Verificación de un usuario en terminal/terminales. Verificación con resultado erróneo. Empleado de baja en persistencia pero de alta en el terminal.	X	X	X
Caso 03	Verificación de un usuario en terminal/terminales. Verificación con resultado erróneo. Empleado de alta en persistencia y existe en terminal, pero minucias diferentes	X	X	X
Caso 04	Verificación de un usuario en terminal/terminales. Verificación con resultado erróneo. Empleado de alta en persistencia y existe en terminal, pero con distinto pin.	X	X	X

Modificación Usuario	**Descripción**	**1**	**2**	**3**
Caso 00	Modificación exitosa de empleado en terminal/terminales.	X	X	X
Caso 01	Modificación fallida de empleado en terminal. Empleado no existente.	X	X	

Caso 03	Modificación fallida de empleado en terminal. EmpleadoTK3 no dado de Alta.	X	X	
Caso 04	Modificación fallida. Terminal bloqueado.	X	X	
Caso 05	Modificación fallida de empleado (Jornada laboral no existe).	X	X	
Caso 06	Modificación exitosa de empleado en terminal.	X	X	
Alta Masiva	**Descripción**	**1**	**2**	**3**
Caso 00	Alta masiva en terminal.	X	X	X
Caso 01	Alta masiva fallida. Terminal bloqueado.	X	X	

Baja Masiva	**Descripción**	**1**	**2**	**3**
Caso 00	Baja masiva en terminal.	X	X	X
Caso 01	Baja masiva fallida. Terminal bloqueado.	X	X	

Verificación Masiva	**Descripción**	**1**	**2**	**3**
Caso 00	Verificación masiva en terminal/terminales. Todas las verificaciones correctas. Sistema completamente integro.	X	X	X
Caso 01	Verificación masiva en terminal/terminales. Resultado de las verificaciones: -ID_EMPLEADO: Correcto -ID_EMPLEADO_52: De baja en persistencia pero existe en terminal. -ID_EMPLEADO_53: De alta en persistencia pero no existe en terminal. -ID_EMPLEADO_54: Existe tanto en persistencia como en terminal pero con minucias diferentes -ID_EMPLEADO_55: Existe tanto en persistencia como en terminal pero con valor del atributo 'pin activo' diferente. Sistema necesita subsanación.	X	X	X

Op. Terminal	**Descripción**	**1**	**2**	**3**
Caso 00	Verificación de conectividad del terminal	X	X	
Caso 01	Desbloqueo terminal exitoso.	X	X	
Caso 02	Desbloqueo de terminal fallido. Terminal no bloqueado.	X	X	
Caso 03	Obtención de la configuración de todos los terminales.	X	X	
Caso 04	Obtención de la configuración del terminal.	X	X	

Subsanación Usuario	**Descripción**	**1**	**2**	**3**
Caso 00	Todas las verificaciones correctas. Sistema completamente integro, no necesita de subsanación		X	X
Caso 01	Tras el resultado de las verificaciones en terminales: -ID_EMPLEADO: Correcto -ID_EMPLEADO_52: De baja en persistencia pero existe en terminal. -ID_EMPLEADO_53: De alta en persistencia pero no existe en terminal. -ID_EMPLEADO_54: Existe tanto en persistencia como en terminal pero con minucias diferentes -ID_EMPLEADO_55: Existe tanto en persistencia como en terminal pero con valor del atributo 'pin activo' diferente. Sistema es subsanado correctamente.		X	X

Jornada Laboral	**Descripción**	**1**	**2**	**3**
Caso 00	Inserción Jornada Laboral exitosa.		X	
Caso 01	Inserción Jornada Laboral fallida (Jornada laboral ya existe).		X	
Caso 03	Eliminación Jornada Laboral exitosa.		X	
Caso 04	Eliminación Jornada Laboral fallida (Jornada laboral no existe).		X	

Caso 05	Eliminación Jornada Laboral fallida (Algún empleadoTK3 tiene asociado esta Jornada Laboral).		X	
Caso 06	Modificación Jornada Laboral exitosa.		X	
Caso 07	Modificación Jornada Laboral fallida (Jornada laboral no existe).		X	

Marcajes	**Descripción**	**1**	**2**	**3**
Caso 00	Borrado marcajes en terminal.		X	
Caso 01	Obtención de marcajes de terminal/terminales.		X	X
Caso 03	Obtención de número de marcajes almacenados en terminal.		X	
Caso 04	Sincronización de marcajes de terminal (0 marcajes sincronizados).		X	
Caso 05	Sincronización de marcajes de terminal (4 marcajes sincronizados).		X	

Simulador de terminales

Las pruebas automatizadas requieren de un simulador de terminales con los siguientes requisitos en función de las pruebas especificadas:

a) Deberá comportarse como uno o varios terminales biométricos conectados a la red, disponiendo de un puerto por cada terminal configurado por los que aceptará operaciones en el mismo formato que el terminal biométrico Kreta 3 y responderá de forma adecuada.
b) Las operaciones que deberá aceptar son todas aquellas necesarias para la ejecución de casos de prueba:
    - Captura de minucia biométrica
    - Alta de empleado
    - Baja empleado
c) Deberá enviar bajo demanda un conjunto de marcajes previamente configurados en el simulador. Cada conjunto de marcajes estará insertado en un fichero xml que definirán un comportamiento. Mediante un puerto

de control se enviarán los comandos necesarios para la ejecución de los diferentes comportamientos.

d) La configuración del simulador se realizará mediante un archivo xml al efecto donde. se incluirán los datos de configuración de todos los terminales configurados en el simulador, así como su comportamiento: SIMPLE: se comporta como un solo terminal conectado a la red solo teniendo en cuenta el primero de los terminales configurados, o MULTIPLE: se comportara como varios terminales biométricos accesibles por red, con tantos terminales tenga configurados. Este archivo también contendrá la configuración del puerto de control y la definición de una o varias minucias dactilares que se utilizarán como respuesta a las operaciones de captura de huellas. Así como los ficheros de simulación de marcajes disponibles asociados a un identificador, que será empleado para su ejecución mediante el puerto de control.

e) El simulador contendrá un modelo de datos para hacer frente a toda la funcionalidad necesaria. Este incluirá los terminales configurados en el sistema, los empleados dados de alta en cada uno de los terminales y los marcajes asociados a cada empleado producidos en cada terminal. El modelo se describirá en el siguiente apartado.

9.2.2 Diseño

Framework de pruebas

Como se ha comentado con anterioridad los casos de usos que requieren peticiones al terminal biométrico tienen un funcionamiento asíncrono. El método de la fachada retorna un valor una vez las ordenes al terminal se han cursado, pero el resultado final de la ejecución llega a través de JMS, mediante el empleo de topics. Debido a esta complejidad y para facilitar el desarrollo de los casos de prueba se ha desarrollado un framework que nos permitirá abstraernos en cierta medida de esta particularidad.

En el diagrama de la Figura 28 el diagrama de clases del framework. En este se observan que los tests creados heredan de la clase TestBase que es la que fianlmente

hereda de TestCase perteneciente a la librería JUnit. Como métodos principales a utilizar de la clase base diseñada se encuentran:

- inicializarComprobadorEventos: mediante este método se inserta un arrayList de objetos del tipo ParEventoEsperadoAccion compuesto por un objeto perteneciente a la clase base EventoOperacion y a la interfaz Accion. Con este array se define el conjunto de eventos esperados durante la ejecución del test. Además se definen las acciones a realizar (Altas, Bajas, etc) con la llegada de cada evento, lo que nos permitirá introducir secuencialidad entre ordenes en la ejecución de un test. La definición de acciones para cada evento esperado es opcional. Los eventos esperados se definen empleado el conjunto de clases que heredan todas de la clase base EventosOperación. Mientras que las acciones implementarán las interfaz Acción.
- comprobarAserciones: Generalmente llamado después de la invocación a una operación de servidor realiza una serie de a Asserts entre los datos de los eventos esperados y los que han llegado a través de topics de la Api JMS. El emparejamiento entre eventos esperados y eventos acaecidos se realiza por estricto orden de llegada.
- esperar: Los casos permiten declarar un conjunto de semáforos utilizados para operaciones de sincronización. Indicando el semáforo sobre el que se desea actuar esta operación detiene la ejecución del caso de prueba hasta que se produzca la invocación de una acción de continuar asociada a la llegada de un evento.

Para la realización de estas operaciones la case base emplea fundamentalmente las clases Assercion y CompruebaEventos.

Las pruebas se realizan instanciando directamente la fachada remota *`UsuarioFacadeRemote`* empleando para ello la utilización de la factoría `FachadasRemotasFactory`. Una vez instanciada se invoca a los diferentes métodos remotos para su verificación.

Paras las operaciones relacionadas con la operación de tearDown que tiene por propósito dejar la persistencia en el mismo estado que tenía antes de comenzar el test se empleará la clase `OperacionesBD` que define diferentes métodos para este propósito.

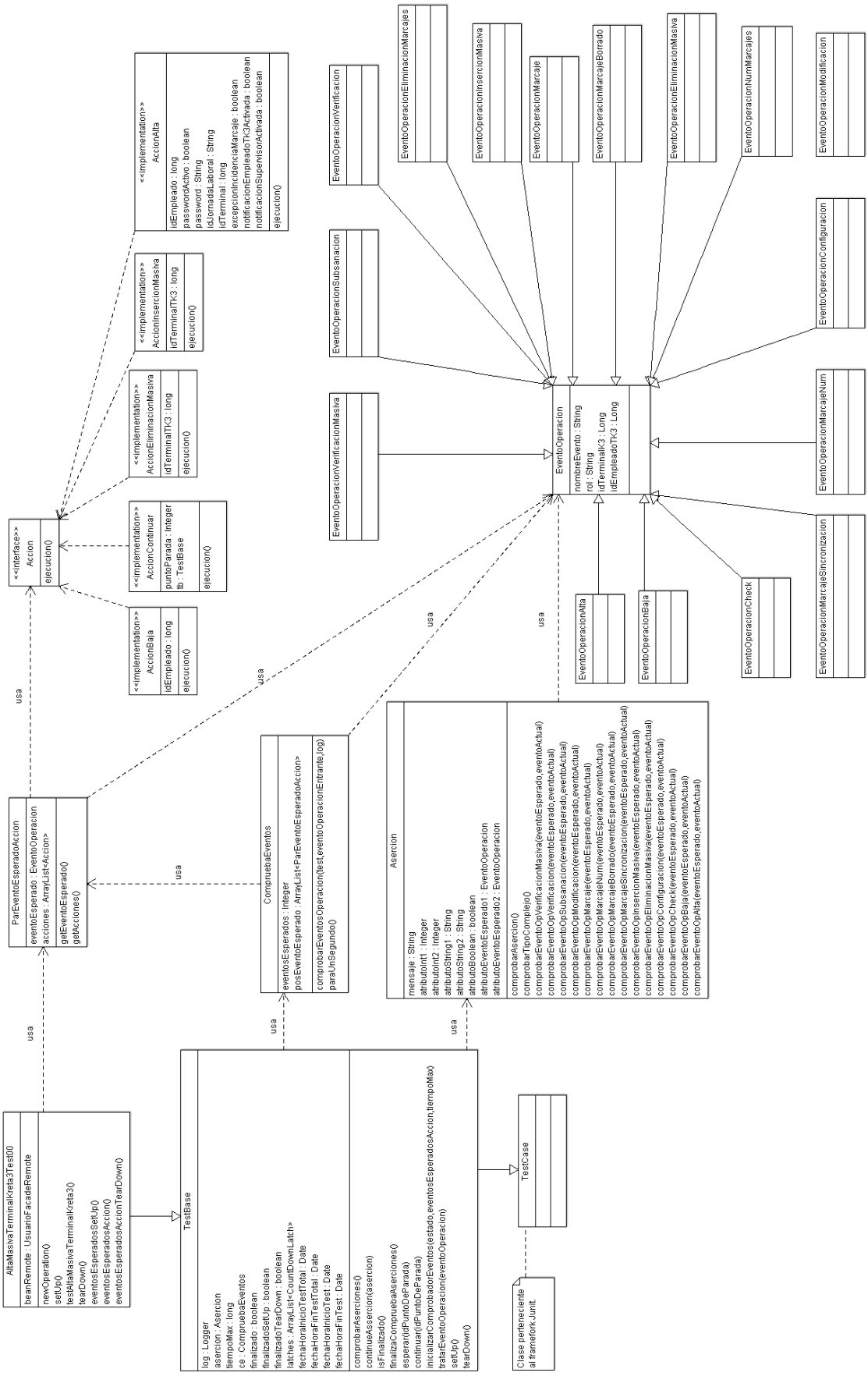

Figura 28. Diagrama de clases del framework de pruebas.

Simulador de terminales

Como elemento principal a modelar se encuentra el diagrama de clases de Figura 29, que conforma el modelo de datos del simulador encargado de contener todos los datos pertenecientes a Terminales, Empleados y Marcajes tal y como lo hace el Terminal Kreta 3, para una correcta simulación de su funcionamiento.

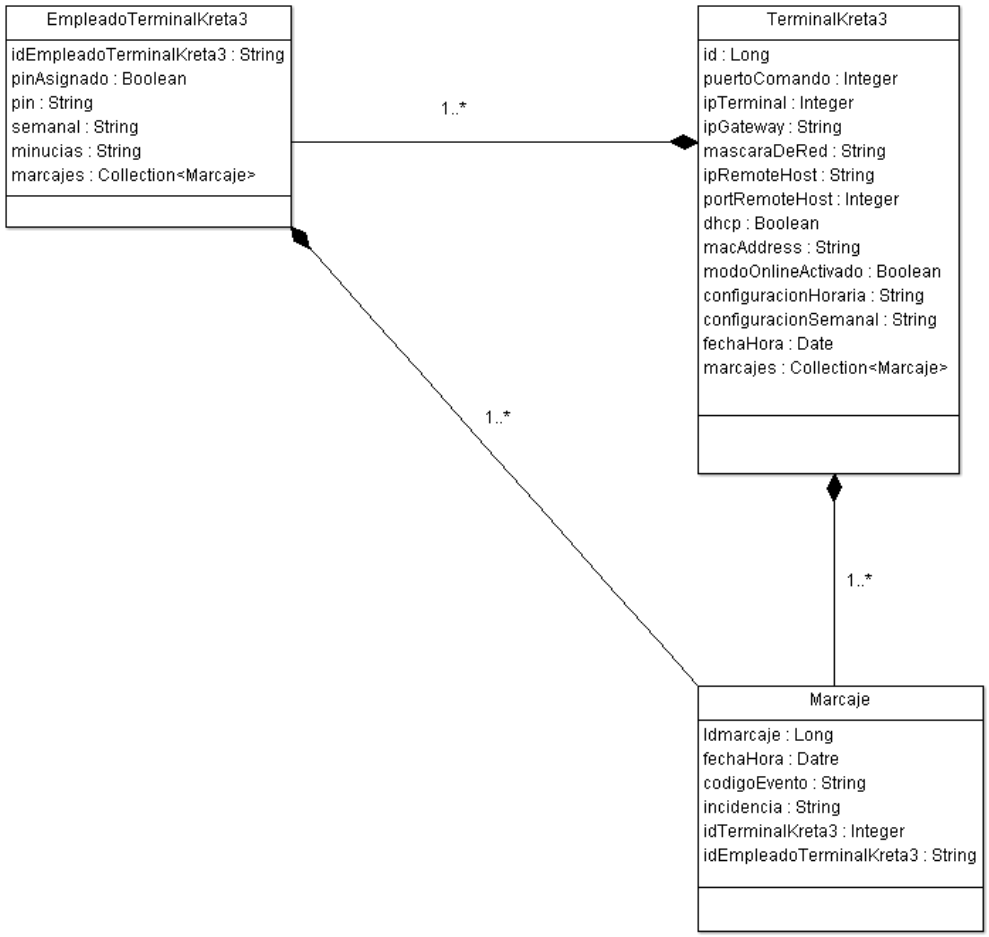

Figura 29. Diagrama de clases del modelo de datos mantenido por el simulador.

Se muestra en diagrama de la Figura 30 un diagrama de componentes formado por los diferentes elementos funcionales que componen el simulador, así como sus interfaces externas.

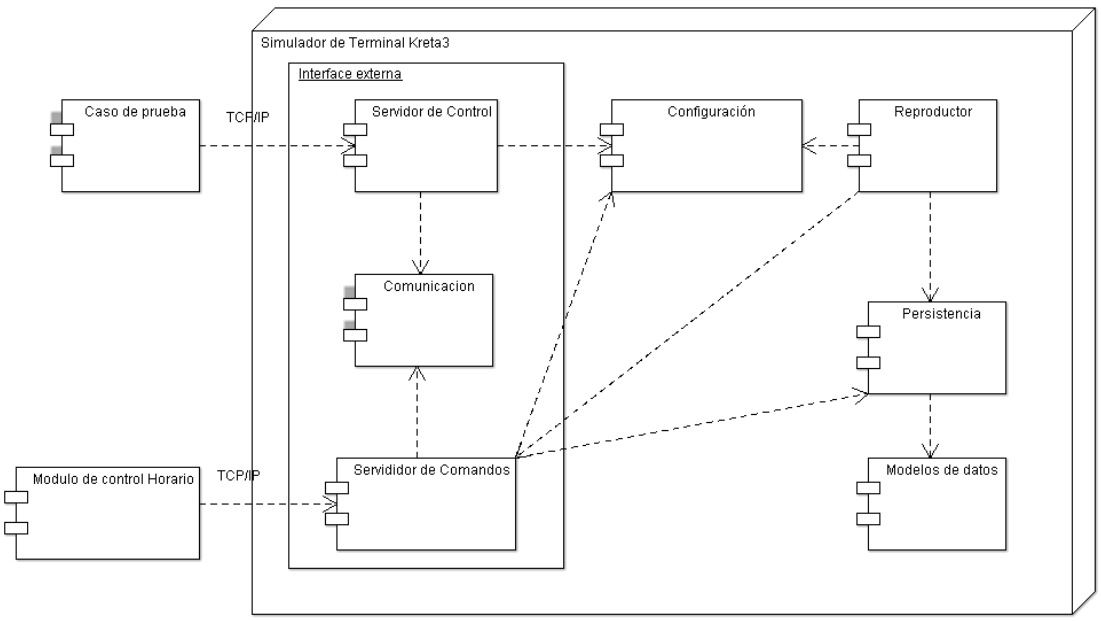

Figura 30. Daigrama de compoenntes del simulador de terminales biometricos Kreta 3

Por último se muestran una serie de diagramas de secuencia en donde se especifica el flujo de ejecución durante el arranque del simulador y tras la recepción de comandos, ya sean propios del conjunto de órdenes del Terminal Kreta3 o comandos de control para la ejecución de comportamientos.

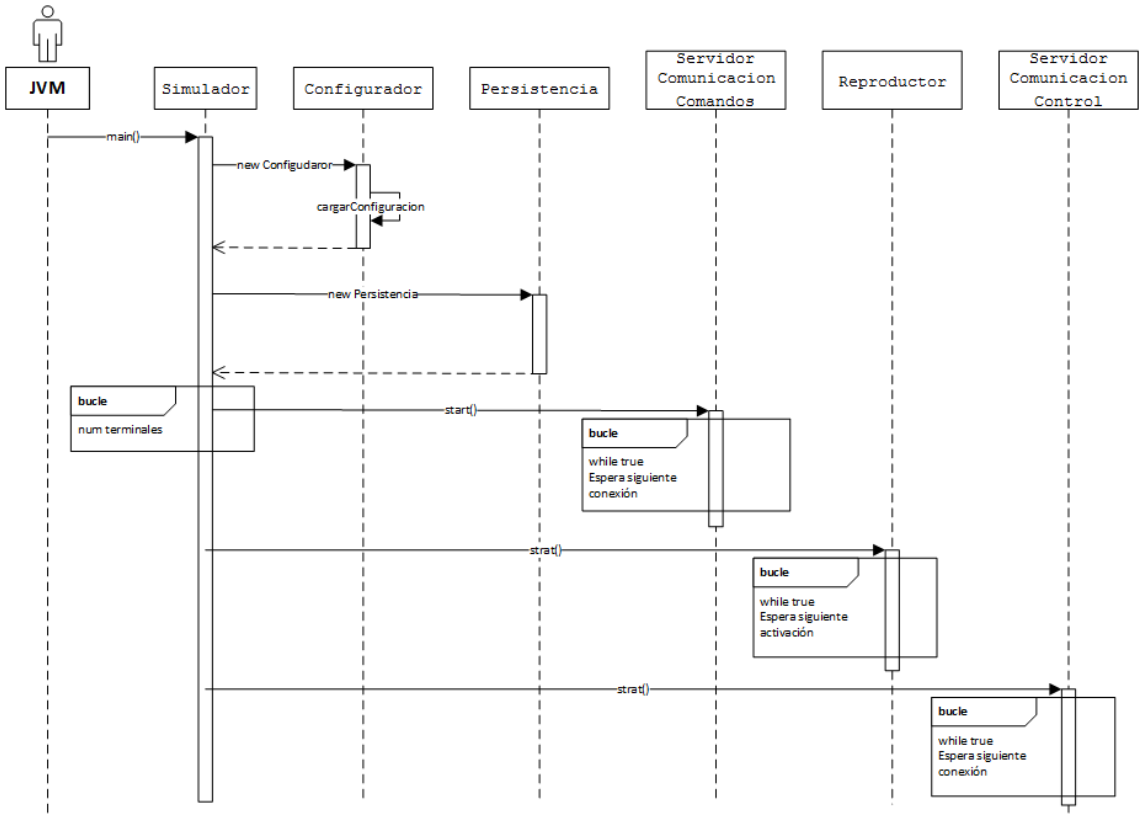

Figura 31. Daigrama de secuencia del flujo de recepción y ejecución de comandos en el simulador (1ª parte)

La Figura 31 muestra el diagrama de colaboración durante el arranque del simulador desde la función main hasta que los hilos principales de ejecución son iniciados y quedan a la espera de algún evento.

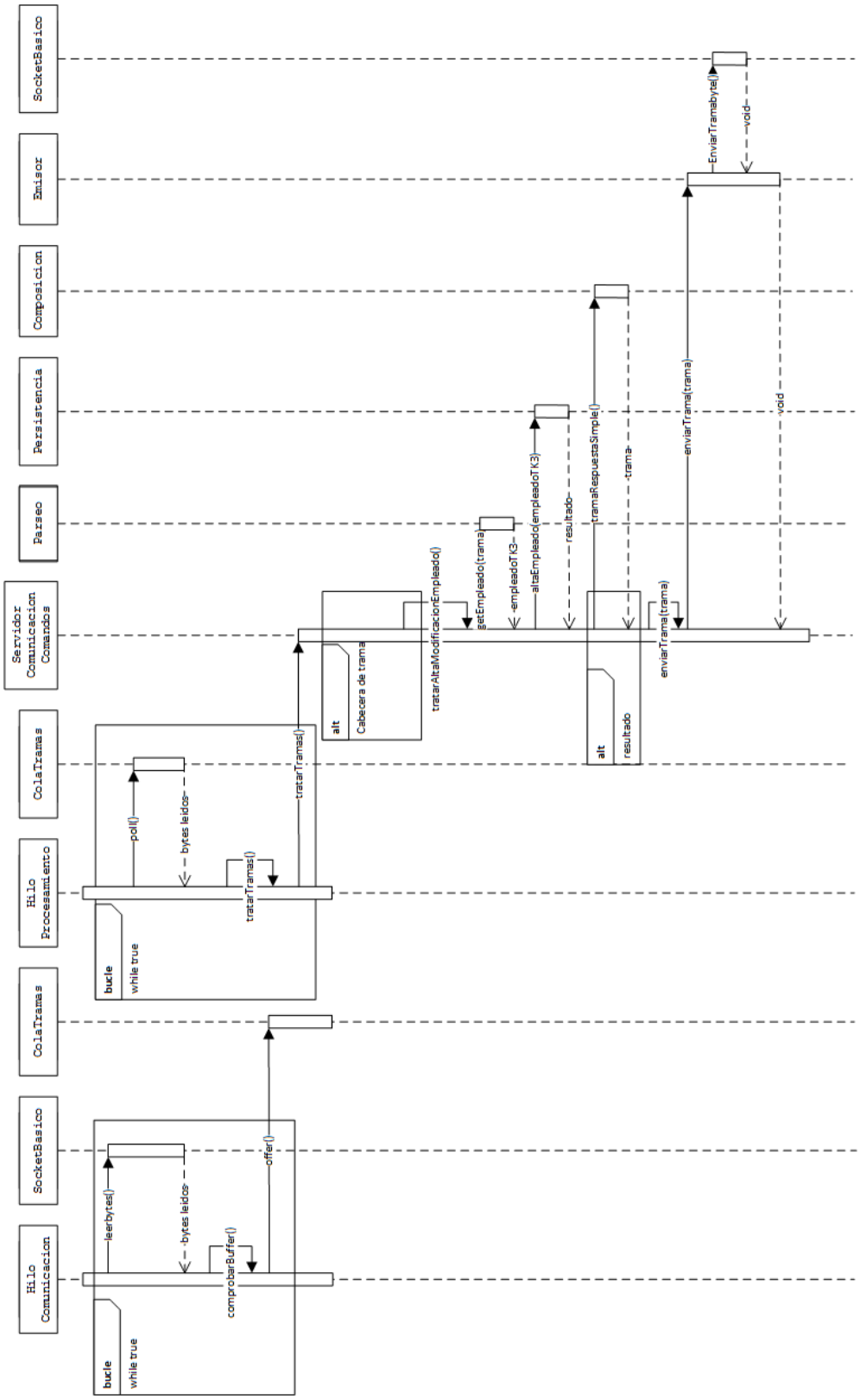

Figura 32. Daigrama de secuencia del flujo de recepción y ejecución de comandos en el simulador (1ª parte)

La Figura 32 muestra el diagrama de secuencia operación de alta diferida, desde que el comando es recibido, procesado y la respuesta al mismo es devuelta.

9.2.3 Implementación

El proyecto que constituye la implementación de las pruebas para la aplicación corresponde a un proyecto Java que puede ser editado y compilado con la herramienta Eclipse. Fundamentalmente el proyecto contiene un framework propio de pruebas que pretende hacer más fácil la creación de casos de prueba, los propios casos de prueba y un simulador de terminal que permite automatizar pruebas y realizar las mismas con configuración de múltiples terminales. El framework desarrollado emplea la librería JUnit. El proyecto de nombre `TestIntranetEF` contiene los siguientes elementos:

- `src/test`: Contiene el código fuente estructurado en paquetes.
- `clases`: Compilados de la aplicación.
- `build.xml`: Fichero de entrada para la ejecución de diferentes tareas mediante ant

Se muestra a continuación la estructuración del código fuente en el directorio src:

- `com.edosoftfactory.test.controlHorario`. Clases fundamentales del framework de pruebas.
- `com.edosoftfactory.test.controlHorario.accion`: Conjunto de clases de acciones que pueden ser asociadas a eventos esperados.
- `com.edosoftfactory.test.controlHorario.conectores`: Conectores de diversos tipos (JMS, RMI, SQL, TCP) que permiten las conexiones necesarias para el funcionamiento del entorno de pruebas. Permitiendo la recepción de mensajes, la interacción con la fachada de la lógica de negocio, modificación directa de la base de datos y control para tests del simulador de terminales.
- `com.edosoftfactory.test.controlHorario.configuracion`: Datos de configuración que utiliza el conector del simulador para establecer comunicación al puerto de control del mismo.
- `com.edosoftfactory.test.controlHorario.constantes`: Constantes empleadas por los tests. Datos que permiten definir, referenciar, empleados, jornadas laborales, terminales, etc.

- `com.edosoftfactory.test.controlHorario.modelo`: Conjunto de clases de eventos que sirven para definir los eventos esperados en un caso de prueba.
- `com.edosoftfactory.test.controlHorario.simuladorTK3.comunicacion`: Contiene todas las clases relacionadas con la comunicación del simulador: Sockets, colas e hilos de comunicación y procesamiento genérico de tramas de órdenes de terminal y control.
- `com.edosoftfactory.test.controlHorario.simuladorTK3.configuracion`: Contiene archivos relativos a la configuración del simulador y los comportamientos para el envío asíncrono de marcajes.
- `com.edosoftfactory.test.controlHorario.simuladorTK3.constantes`: Contiene la clase donde se define las contantes utilizadas el en código de simulador.
- `com.edosoftfactory.test.controlHorario.simuladorTK3.modelo`: Contiene las clases que conforman el modelo de datos necesario para el funcionamiento del simulador.
- `com.edosoftfactory.test.controlHorario.simuladorTK3.run`: Contiene las clases principales del simulador, incluida la clase principal Simulador.java
- `com.edosoftfactory.test.controlHorario.simuladorTK3.util`: Contiene la clase de apoyo para el parseo y la conformación de tramas.
- `com.edosoftfactory.test.controlHorario.suite.automatico.multiplesTerminales`: Conjunto de casos de pruebas que testean la intranet conectada al simulador de terminal en su configuración de múltiples terminales.
- `com.edosoftfactory.test.controlHorario.suite.automatico.terminalUnico`: Conjunto de casos de pruebas que testean la intranet conectada al simulador de terminal en su configuración de terminal único.
- `com.edosoftfactory.test.controlHorario.suite.manual`: Conjunto de casos de pruebas que testean la intranet conectada al terminal biométrico
- `com.edosoftfactory.test.controlHorario.suite.util`: Contiene la clase Utils que agrupa diferentes métodos estáticos que se utilizan de forma generalizada en los casos de prueba. También contiene la clase `LoggerUtil` para facilitar el logeo del framework de pruebas.

La compilación se realiza en la versión del JDK 1.6 utilizando tal y como se ha comentado la librería JUnit 4.

9.2.4 Configuración y ejecución

Tal y como se comenta en el apartado anterior existen 3 suits diferentes para las pruebas del Modelo de Negocio, cada una de las cuales requiere una serie de pasos de configuración previa y posterior ejecución. Este apartado parte de un correcto despliegue de la aplicación en un entorno de desarrollo o de preproducción. Para todas las Suits es necesario la creación previa de un usuario en la base de datos llamado "test" con password "test000", y con accesos permitidos a la base de datos `Icef` desde el host donde se ejecuten las pruebas. Con este usuario se realizaran operaciones directas sobre la base de datos con el fin de restaurar su estado al momento antes de la ejecución de cada caso de prueba. Además, como en el caso de las pruebas de la interfaz gráfica, será necesario la ejecución de un script almacenado en el proyecto de pruebas cuyo nombre y ruta es `/recursos/icef_prod_recreate_20110709.sql`, para la inserción de la base de datos de los datos iniciales necesarios para la ejecución de las suits. La ejecución solo será necesaria una vez.

Suit manual

En esta suit se engloban los casos de prueba que se realizan utilizando la intranet conectada al terminal biométrico. Al ejecutar la suit será necesario interactuar con el terminal para el desarrollo de la misma que deberá estar configurado y conectado a la red de manera adecuada. Para la instalación y configuración inicial del terminal se han de seguir las siguientes instrucciones:

- Alimente Kreta3, con el cable Ethernet conectado.
- Ejecute KSearch, para configurar los parámetros IP de Kreta3. KSearch es la utilidad software que permite detectar y configurar los dispositivos Kimaldi conectados a la red y ha sido proporcionado junto con el terminal.
    a) El puerto SLK se abre automáticamente
    b) Buscar dispositivos Kimaldi conectados a nuestra LAN (marque "Forzar respuesta en modo broadcast…" si nuestra LAN no corresponde a la dirección 192.168.123.xx)
    c) Kreta3 tendría que aparecer en la lista. Asegúrese de que la HW Address (MAC Addr.) se corresponde con el dispositivo a configurar.

d) Leer configuración, de modo que se pueda editar en la parte baja de la pantalla.

e) Especifique IP Address y Remote Host (la dirección IP del ordenador), o marque Obtener una IP automáticamente (DHCP)

f) Escribir configuración, para enviar los nuevos datos a Kreta3

g) Aplicar configuración. Kreta3 reiniciará.

h) Después de unos segundos, Buscar de nuevo y Kreta3 tendría que aparecer otra vez en la tabla, esta vez con la nueva IP Address.

i) Leer configuración otra vez, para estar seguros que Remote Host se ha actualizado correctamente. Remote Host es necesario para ejecutar DemoKreta.

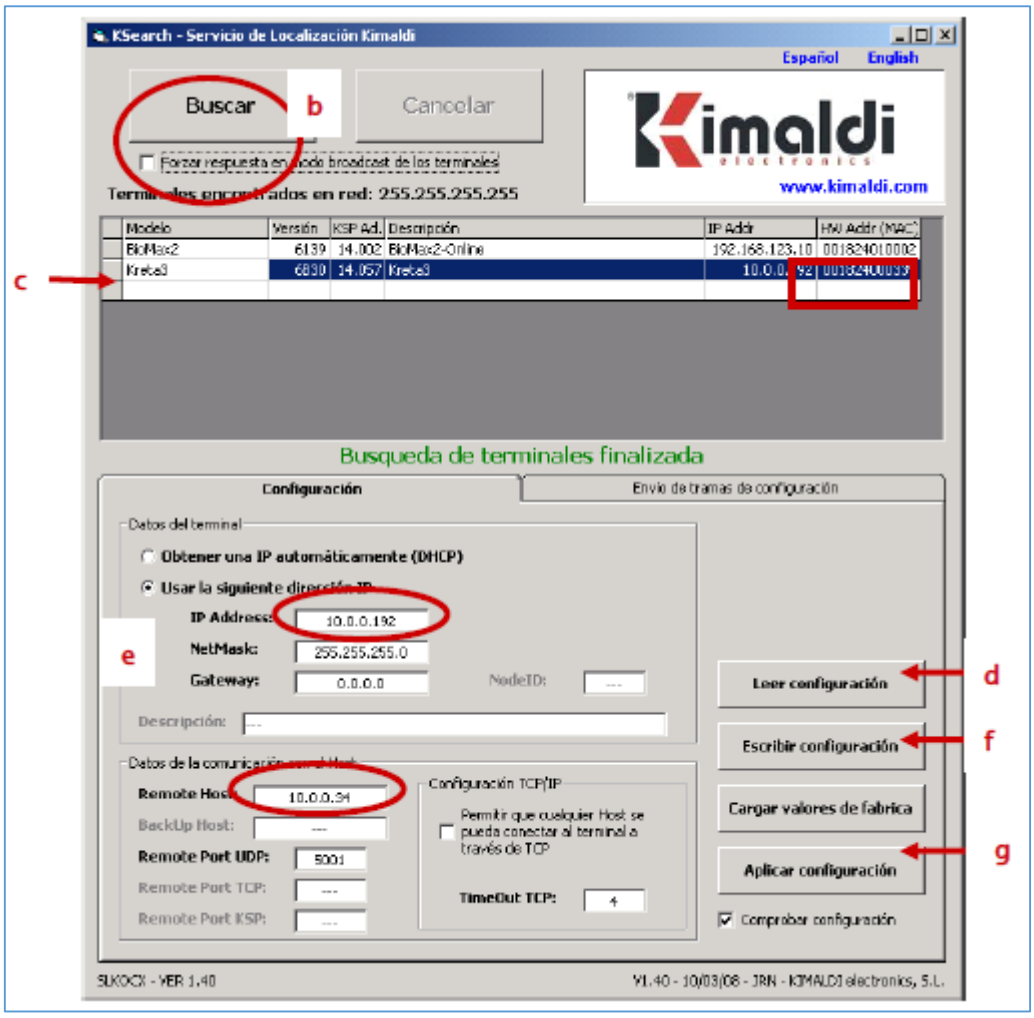

Figura 33. Pantalla principal del KSearch.

Los valores adecuados para las pruebas para los diferentes parámetros son los siguientes:

DIRECCION IP:192.168.1.10

PUERTO: 6000

GATEWAY: 192.168.1.1

MASCARA DE RED: 255.255.255.0

DHCP: False

HOST REMOTO: 192.168.1.35

PUERTO REMOTO: 6001

Nota: *La dirección del terminal biométrico, del Gateway y el host remoto podrán variar en función de la configuración de red donde se encuentre instalando el terminal y el host desde donde se ejecutan las pruebas.*

Posteriormente se configura el conector de la intranet para una adecuada comunicación con el terminal mediante el archivo Kreta3TerminalConnector\src\META-INF\ra.xml En este se debe comentar las configuraciones para el resto de pruebas y eliminar los comentarios del apartado que aplica para este caso:

```
<!-- Configuracion terminal unico real -->
     <config-property>
       <config-property-name>terminalKreta3Hosts</config-property-name>
       <config-property-type>java.lang.String</config-property-type>
       <config-property-value>PRINCIPAL@192.168.1.10</config-property-value>
     </config-property>
      <config-property>
       <config-property-name>terminalKreta3Ports</config-property-name>
       <config-property-type>java.lang.Integer</config-property-type>
       <config-property-value>1001</config-property-value>
     </config-property>
      <config-property>
       <config-property-name>terminalKreta3CaracterSeparador</config-property-name>
       <config-property-type>java.lang.String</config-property-type>
       <config-property-value>;</config-property-value>
     </config-property>
```

Una vez concluido el paso anterior se realiza un nuevo despliegue del conector con la configuración modificada mediante la ejecución de tag `kreta3Terminal-`

`connector-deploy` del build.xml del proyecto `Kreta3TerminalConnector` mediante la herramienta Ant de Apache desde el eclipse, y se relanza el servidor de aplicación.

En último lugar se ejecuta desde el eclipse la suit como una Suit de JUnit mediante la clase que la contiene:

`com.edosoftfactory.test.controlHorario.suite.manual.TestsTerminalUnico.java`

Suit automática con un único terminal

En esta suit se engloban los casos de prueba que se realizan utilizando la intranet conectada al simulador de terminal biométrico en configuración de un solo terminal. En este caso no es necesario disponer de ningún terminal conectado. Tan solo hay necesidad de ejecutar el Simulador de terminal con la configuración de un solo terminal y la configuración y despliegue del conector con los parámetros adecuados como en el caso anterior.

Para configurar el Simulador en modo de simulación de un solo terminal se modifica el fichero de propiedades

`com.edosoftfactory.test.controlHorario.simuladorTK3.modelo.Config.xml`

cambiando el valor de la propiedad CONFIGURACION a SIMPLE

```
<!-- MULTIPLE O SIMPLE -->
       <entry key="CONFIGURACION">SIMPLE</entry>
```

A continuación se ejecuta el simulador desde el eclipse mediante la clase:

`com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador`

obteniendo por consola los diferentes mensajes del arranque del simulador y quedando este a la espera de conexión por parte del servidor de la aplicación.

```
2014-04-15 11:08:12,414  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:21) - Cargo las propiedades iniciales
2014-04-15 11:08:12,466  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:24) - Se inicializa la persistencia de TK3s, empleados y marcajes
2014-04-15 11:08:12,467  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:29) - Se inicializa el hilo de comunicación de comandos CORE->: Terminal [1]
2014-04-15 11:08:12,469  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:35) - Se crea el hilo reproductor que empieza parado
2014-04-15 11:08:12,469  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:39) - Se inicializa el hilo de comunicación de control TEST->: espera por conexión
2014-04-15 11:08:12,472  INFO [Thread-2]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.ServidorComunicacionControl.
run(ServidorComunicacionControl.java:77) - Servidor comunicación control: esperando
por conexión
```

De igual manera se configura el conector de la intranet para una adecuada comunicación con el terminal mediante el archivo Kreta3TerminalConnector\src\META-INF\ra.xml En este se debe comentar las configuraciones para el resto de pruebas y borrar el comentario del apartado que aplica para este caso:

```
<!-- Configuracion terminal unico simulado: desarrollo -->
     <config-property>
       <config-property-name>terminalKreta3Hosts</config-property-name>
       <config-property-type>java.lang.String</config-property-type>
       <config-property-value>PRINCIPAL@127.0.0.1</config-property-value>
     </config-property>
      <config-property>
       <config-property-name>terminalKreta3Ports</config-property-name>
       <config-property-type>java.lang.Integer</config-property-type>
       <config-property-value>6000</config-property-value>
     </config-property>
      <config-property>
       <config-property-name>terminalKreta3CaracterSeparador</config-property-name>
       <config-property-type>java.lang.String</config-property-type>
       <config-property-value>;</config-property-value>
     </config-property>
```

Una vez concluido el paso anterior se realiza un nuevo despliegue del conector con la configuración modificada mediante la ejecución de tag “kreta3Terminal-

connector-deploy" del build.xml del proyecto Kreta3TerminalConnector mediante la herramienta Ant de Apache desde el eclipse y se relanza el servidor de aplicaciones.

En último lugar se comienza la ejecución desde el eclipse la suit como una Suit de JUnit mediante la clase que la contiene:

`com.edosoftfactory.test.controlHorario.suite.automatico.terminalUnico.TestsTerminalUnicoAutomatico.java`

Suit automática con múltiples terminales

En esta suit se engloban los casos de prueba que se realizan utilizando la intranet conectada al simulador de terminal biométrico en configuración de múltiples terminales. En este caso no es necesario disponer de varios terminales conectados a la red. Tan solo es preciso ejecutar el Simulador de terminal con la configuración de múltiples terminales y la configuración y despliegue del conector con los parámetros adecuados como en el caso anterior.

Para configurar el Simulador en modo de simulación de un solo terminal se modifica el fichero de propiedades

`com.edosoftfactory.test.controlHorario.simuladorTK3.modelo.Config.xml`

cambiando el valor de la propiedad CONFIGURACION a MULTIPLE

```
<!-- MULTIPLE O SIMPLE -->
       <entry key="CONFIGURACION">MULTIPLE</entry>
```

A continuación se ejecuta el simulador desde el eclipse mediante la clase:

`com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador`

obteniendo por consola los diferentes mensajes del arranque del simulador y quedando este a la espera de un total de 4 conexiones por parte del servidor de la aplicación.

```
2014-05-08 19:58:57,526  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:21) - Cargo las propiedades iniciales
2014-05-08 19:58:57,557  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:24) - Se inicializa la persistencia de TK3s, empleados y marcajes
2014-05-08 19:58:57,558  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:29) - Se inicializa el hilo de comunicacion de comandos CORE->: Terminal [1]
2014-05-08 19:58:57,561  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:29) - Se inicializa el hilo de comunicacion de comandos CORE->: Terminal [2]
2014-05-08 19:58:57,561  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:29) - Se inicializa el hilo de comunicacion de comandos CORE->: Terminal [3]
2014-05-08 19:58:57,561  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:29) - Se inicializa el hilo de comunicacion de comandos CORE->: Terminal [4]
2014-05-08 19:58:57,561  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:35) - Se crea el hilo reproductor que empieza parado
2014-05-08 19:58:57,562  INFO [main]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.Simulador.main(Simulador.jav
a:39) - Se inicializa el hilo de comunicacion de control TEST->: espera por conexion
2014-05-08 19:58:57,566  INFO [Thread-5]
com.edosoftfactory.test.controlHorario.simuladorTK3.run.ServidorComunicacionControl.
run(ServidorComunicacionControl.java:77) - Servidor comunicacion control: esperando
por conexion
```

De igual manera se configura el conector de la intranet para una adecuada comunicación con el terminal mediante el archivo Kreta3TerminalConnector\src\META-INF\ra.xml En este será preciso comentar las configuraciones para el resto de pruebas y descomentar el apartado que aplica para este caso:

```
<!-- Configuracion terminales multiples simulados: desarrollo y pruebas -->

     <config-property>

       <config-property-name>terminalKreta3Hosts</config-property-name>

       <config-property-type>java.lang.String</config-property-type>

       <config-property-
value>PRINCIPAL@127.0.0.1;SECUNDARIO1@127.0.0.1;SECUNDARIO2@127.0.0.1;SECUNDARIO3@12
7.0.0.1</config-property-value>

     </config-property>

      <config-property>

       <config-property-name>terminalKreta3Ports</config-property-name>

       <config-property-type>java.lang.String</config-property-type>

       <config-property-value>6000;7000;8000;9000</config-property-value>

     </config-property>

      <config-property>

       <config-property-name>terminalKreta3CaracterSeparador</config-property-name>

       <config-property-type>java.lang.String</config-property-type>

       <config-property-value>;</config-property-value>

     </config-property>
```

Una vez concluido el paso anterior se realiza un nuevo despliegue del conector con la configuración modificada mediante la ejecución de tag "kreta3Terminal-connector-deploy" del build.xml del proyecto Kreta3TerminalConnector mediante la herramienta Ant desde el eclipse y se relanza el servidor de aplicación.

Finalmente se lanza la ejecución desde el eclipse la suit, como una Suit de JUnit mediante la clase que la contiene:

`com.edosoftfactory.test.controlHorario.suite.automatico.multiplesTerminales.TestsMultiplesTerminalesAutomatico.java`

9.2.5 Resultados

Suit manual

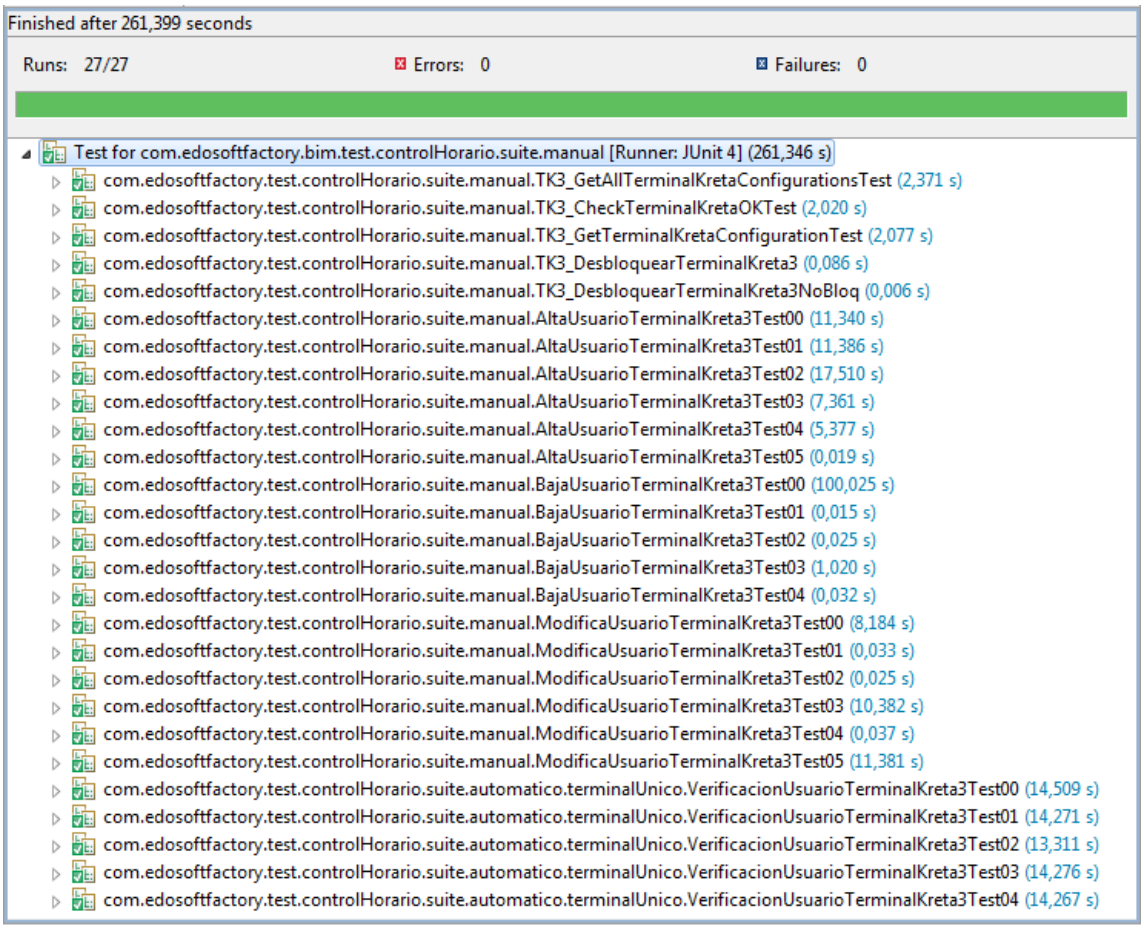

Figura 34. Resultado de la suit de pruebas manual con un único terminal real.

Suit automática con un único terminal

Figura 35. Resultado de la suit de pruebas automática con un único terminal simulado.

Suit automática con múltiples terminales

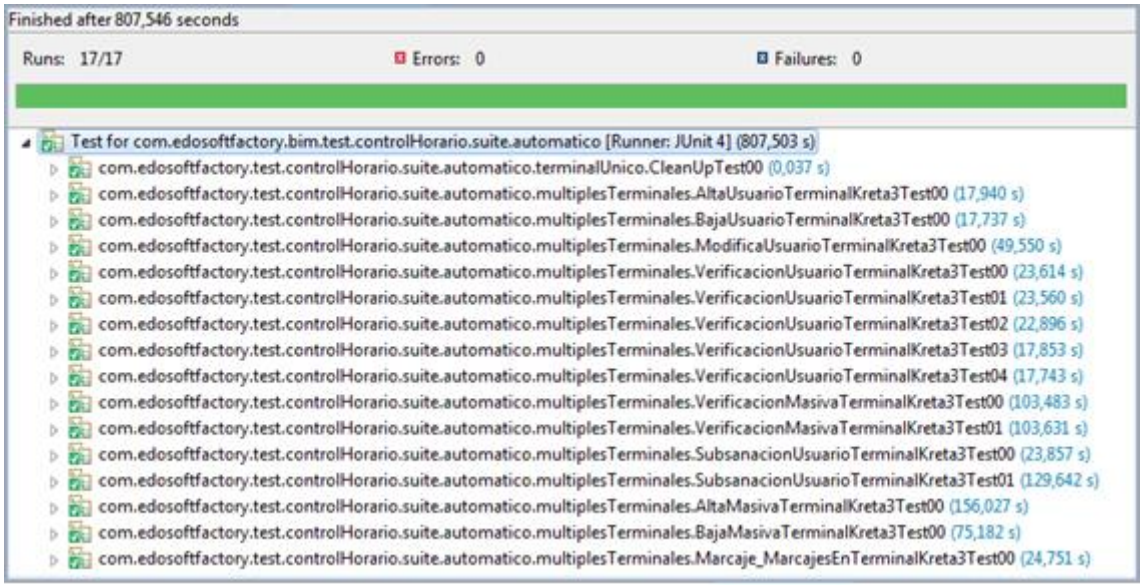

Figura 36. Resultado de la suit de pruebas automática con múltiples terminales simulados.

10. Desarrollos futuros

Durante las diferentes fases de este proyecto se han identificado diversas oportunidades de mejora y ampliación del trabajo realizado que se incluyen en este capítulo.

Migración del servidor a tecnologías más actuales.

Las tecnologías empleadas para el desarrollo de este proyecto de fin de carrera eran tecnologías de actualidad cuando se comenzó con el mismo. Debido al largo tiempo transcurrido, en la actualidad muchas han caído en desuso, han aparecido otras mejoras, o mejores versiones de las mismas.

En primer lugar sería necesario actualizar la versión del servidor de aplicaciones. La versión empleada de JBoss, 4.2.2.GA, ya no dispone de soporte, y aunque sus posibles vulnerabilidades son menos explotables, al ser una aplicación de intranet, solo disponible en la red de la oficina, urge la migración a una versión más reciente, por ejemplo la versión 7 del mismo, o la última de GlashFish de Sun.

Junto con la migración de servidor de aplicaciones con el fin de actualizar completamente la parte servidora sería conveniente migrar la Intranet de J2EE 1.4 a J2EE 6.

Ampliación de la funcionalidad de la GUI.

En esta primera versión, por falta de tiempo, el apartado de muestra de datos elaborados y significativos mediante gráficos (de barras, tarta, etc.), han sido poco explotados. Para futuras versiones se pueden incluir una diversidad de gráficos con datos referentes al absentismo, media por semana y mensuales de horas trabajadas por empleado y relacionadas con la media total de la empresa, gráficos sobre incidencias en el horario, etc.

Creación de pruebas de sistemas para la aplicación.

Las pruebas realizadas se han limitado a pruebas funcionales automatizadas para la prueba del backend y una prueba manual para la prueba del frontend. Sería necesario para evaluar las capacidades y la robustez de la parte servidora realizar la simulación de algunos escenarios reales donde se realicen altas, bajas, modificaciones, acceso a los datos de terminal y de usuarios(marcajes, incidencias y datos generales) mientras se reciben un gran número de marcajes a diferentes horas del día. Para esta prueba tan solo habrá que configurar una prueba que realice secuencial o paralelamente, depende del caso, las operaciones anteriormente mencionadas. Y para el envío de marcajes, aprovechando la función reproductora del Simulador, tan solo habrá que desarrollar un fichero con un gran número de marcajes aleatorios para los usuarios dados de alta en cada momento.

Configuración de relés para el control de apertura de las puertas de la oficina.

El horario flexible proporciona numerosas ventajas, pero también conlleva algunas desventajas. Entre estas se encuentra la complejidad de organización para que el primero empleado en llegar disponga de acceso al centro de trabajo mediante la llave en cuestión. El terminal dispone de relés que mediante su correcta configuración podría actuar sobre pestilleras eléctricas instaladas en las puertas de la oficina, tras un marcaje correcto de personal con permisos para esta acción. Este podría ser un interesante desarrollo futuro con numerosos retos, tales como el aseguramiento de la seguridad y la fiabilidad.

Transaccionalidad en operaciones de Altas, Bajas y Modificaciones: Ocultación y automatización de las operaciones de verificación y subsanación.

Tras el uso del nuevo módulo de control y gestión de horarios durante un tiempo considerable, con un único terminal, las operaciones de verificación y posterior subsanación no han sido apenas empleadas. Si con la compra de algún terminal más para los otros accesos a la oficina se continuara esta tendencia, se podría valorar la automatización de las operaciones de verificación y subsanación, quedando fuera del control de la interfaz. Para cada operación de alta, baja o modificación se evaluaría que ha finalizado correctamente. En caso contrario se lanzaría automáticamente un proceso de verificación y uno posterior de subsanación, retornando al sistema al estado anterior al

fallo producido. Esto convertiría a las operaciones de altas bajas y modificaciones sobre los diferentes terminales instalados, en transaccionales.

Adquisición e instalación de un lector de huella dactilar de sobremesa conexión a PC

Se ha demostrado que las altas en el sistema empleando el mismo terminal biométrico instalado en el punto de acceso a las instalaciones puede resultar algo incómodo. Al estar alejado el terminal biométrico del PC donde se gestiona el alta, señalar la correcta posición del dedo para la captura, o indicar la necesidad de realizar una nueva captura si es necesario, obliga en determinadas ocasiones al gestor a desplazarse desde su puesto de trabajo al terminal durante esta operación. Añadir al sistema un lector de huella dactilar de sobremesa conectado al PC para la captura de los datos biométricos que después emplear en el alta diferida mejoraría el procedimiento de altas. Esta opción se valoró en el proceso de requisitos de usuario, pero se suprimió para no incrementar el gasto del proyecto, y más aun teniendo en cuenta que las operaciones de Alta de empleado en sistema son de escasa frecuencia para una PYME como Edosoft. En cualquier caso se podría incluir en un desarrollo futuro como mejora o si se decidiera comercializar la intranet o el módulo de control y gestión de horarios por separado sería indispensable.

Funcionalidad de detección y configuración de terminales biométricos

En la actualidad antes de conectar el terminal biométrico Kreta 3 a la red para su utilización como parte del sistema es necesaria su configuración. Para ello como ya se ha comentado con anterioridad se emplea el programa Servicio de localización, mediante el cual se realiza su detección y configuración de red. Una vez accesible se configura la intranet para una adecuada conexión al mismo.

Incluido en el software del terminal Kimaldi aporta unas dlls bibliotecas de enlace dinámico, para realizar esta configuración de red inicial para cualquier terminal Kreta 3 conectado a la misma, como parte de la funcionalidad de un desarrollo propio.

Sería interesante añadir la posibilidad de configurar terminales desde la propia intranet, facilitando de este modo la operación de altas de terminales en el sistema.

Automatización de los despliegues

Completar la funcionalidad que se realiza mediante Ant para poder realizar despliegues en máquinas remotas de todos los componentes que constituyen la Intranet, tanto de los ya existentes en el momento de comienzo de este proyecto, core del servidor, y GUI, como los nuevos desarrollados, conector y planificador.

Automatización del cambio de Modo (Entrada y Salida) y empleo del antipassback

Implementación de un planificador que dos veces al día se ejecute (A media noche y al medio día) cambiando el modo en que está configurado el terminal (Entradas o salidas masivas), evitando que el primer usuario que ficha una salida o una entrada en una jornada laboral tenga que realizar el cambio manual de modo, interactuando con el teclado. Esta funcionalidad es necesaria para evitar en gran medida que algunos usuarios, por error, tenga seleccionado un modo inadecuado a la hora de fichar, produciéndose, por ejemplo, dos marcajes de entrada en un día, o un primer marcaje de salida al empezar el día. La segunda entrada, o un marcaje de salida sin uno previo de entrada, no son computadas según las especificaciones, pero el usuario no se da cuenta de esta eventualidad hasta que le llega una incidencia o hasta revisar sus marcajes realizados en la intranet.

Este cambios al resultar bastante sencillo de implementar y necesario para mejorar la usabilidad del sistema fueron implementados y puestos en producción al poco tiempo de la implantación del módulo de control y gestión horaria. Está modificación forma parte del código entregado y así consta en el capítulo 8.2.3 Implementación de los planificadores.

11. Resultados y conclusiones

Según los objetivos planteados inicialmente se ha llegado a unos resultados satisfactorios, obteniendo un sistema de control y gestión de horarios operativo, en el contexto de una aplicación existente, una intranet corporativa para la gestión de los procesos empresariales.

Actualmente el módulo de control y gestión de horarios se encuentra en producción, con un número ínfimo de bugs hallados durante un periodo considerable de tiempo, signo de un excelente proceso de desarrollo.

Aunque durante el tiempo de desarrollo y el periodo que la aplicación lleva en producción se han especificado nuevos requisitos, lo requisitos iniciales se han identificado correctamente e implementados en su totalidad, dando como resultado un módulo con las características justas y necesarias para el control y la gestión horario de los empleados de Edosoft.

Cabe destacar el concienzudo trabajo de ingeniería inversa para conocer el estado de la intranet en los inicios del proyecto y la validación de integración del nuevo módulo a desarrollar, tras la se pudo afirmar con rotundidad que la nueva funcionalidad podría ser desarrollada a partir de la ya existente sin que surgiera ningún tipo de problema de incompatibilidad a posteriori o dificultad no contemplada.

Otro de los éxitos del desarrollo ha sido la fase de pruebas, cuyas pruebas funcionales abarcan la totalidad de los casos de uso, con varias pruebas automatizadas para cada uno de estos, teniendo como punto de entrada los métodos de fachada del backend de la aplicación. Por otro lado la interfaz gráfica también dispone de un conjunto de pruebas manuales especificadas, que abarcan también el total de las operaciones que se realizan desde la misma, aunque debido a la complejidad de estas, gran parte de los pocos fallos encontrados estaban ubicados en el frontend de la aplicación.

La elección del terminal y el proveedor ha sido otro de los grandes aciertos. El terminal robusto, fiable, con una documentación impecable, y con una funcionalidad adecuada a las necesidades ha facilitado el trabajo de integración del mismo como elemento hardware en el nuevo módulo desarrollado. Quizás solo un pero, la funcionalidad que el terminal ofrecía era muy superior a lo que el proyecto requería, y

aunque algunas de estas características se podían haber empleado en el transcurso del desarrollo para realizar en el terminal cierto control, se ha optado por realizar el mismo como parte de la lógica de negocio, para independizar lo máximo posible el modulo del terminal empleado, pudiendo cambiar el mismo por otro, siempre y cuando cumpla con un número reducido de requisitos. Y qué decir del proveedor, el proceso de compra ágil con un asesoramiento adecuado, y un soporte durante el transcurso del proyecto inmejorable.

Como única sombra del desarrollo del proyecto se ha de destacar la infraestimación temporal de alguna de las tareas, que unido a la falta de continuidad en la ejecución del mismo ha provocado un retraso temporal de un 25% con respecto al total estimado. Es difícil en cualquier caso atribuir que parte del retraso corresponde a la infraestimación temporal de algunas tareas, como por ejemplo la de pruebas, y que porcentaje a la falta de continuidad y el trabajo en periodos cortos de dedicación diaria.

A pesar de esta eventualidad, el producto obtenido es de gran calidad y está dando unos resultados realmente buenos en su funcionamiento diario.

12. Herramientas de edición, compilación y control de la configuración

Implementación

-IDE Eclipse Ganimedes: Codificación y compilación de la aplicación servidora y el proyecto de pruebas.

-Cliente Git: Control de versiones.

-IDE Adobe Flex Builder 3: Codificación y compilación de la Interfaz Gráfica de Usuario.

Memoria

-Argo UML 0.32.1: Elaboración de diagramas de clases del servidor (Java) y diagramas de componentes en general.

-Microsoft Visio 2013: Diagramas de secuencia.

-Balsamiq Mockups For Desktop: Prototipo de pantallas de la GUI.

-UML4AS: Elaboración de diagramas de clases del cliente (ActionScript).

-Microsoft Word 2013: Redacción y edición de esta memoria.

-Microsoft Project 2013: Diagrama de Gannt.

13. Acrónimos

AFAS: Automatic Fingerprint Autentication System.

CORBA: Common Object Request Broker Architecture

DHCP: Dynamic Host Configuration Protocol

GUI: Graphic User Interface

JDBC: Java Database Connectivity

JMS: Java Message Service

J2EE: Java 2 Platform, Enterprise Edition

LAN: Local Area Network

MAC: Media Access Control

MXML: Macromedia Extensible Markup Language

MVC: Modelo Vista Controlador

RFID: Radio Frequency IDentification

RIA: Rich Internet applications

RMI: Remote Method Invocation

SMTP: Simple Mail Transfer Protocol

CVS: Concurrent Versions System

UOI: Unidad Organizativa Interna

VO: Value Object

XML: Extensible Markup Languag

14. Bibliografía

[1]. Biometría,
http://es.wikipedia.org/wiki/Biometria.

[2]. Sensor de huella dactilar,
http://es.wikipedia.org/wiki/Sensor_de_huella_digital.

[3]. D. Maltoni, D. Maio, A. K. Jain, S. Prabhakar "Handbook of Fingerprint Recognition" Springer, 2009.

[4]. J. Areitio Bertolín, M. T. Areitio Bertolín, "Análisis en torno a la tecnología biométrica para los sistemas electrónicos de identificación y autenticación" Revista española de electrónica, núm. 630, pp. 56-67, 2007.

[5]. Introducción a la biometría,
http://www.monografias.com/trabajos43/biometria/biometria.shtml.

[6]. A. K. Jain, S. Z. Li, "Encyclopedia of biometrics" Springer, 2009

[7] http://es.wikipedia.org/wiki/Servidor_de_aplicaciones

[8] Kimaldi "Kreta3 Manual de Instalación y Programación v. 1.01" 2009

[9] Kimaldi "Primeros pasos con Kreta3" 2009

Apéndice I. Manual de usuario

El acceso al módulo de control y gestión horaria se realiza a través del menú “Control horario” de la barra de menús. El menú está compuesto por dos opciones: “Listar" y "Revisar”.

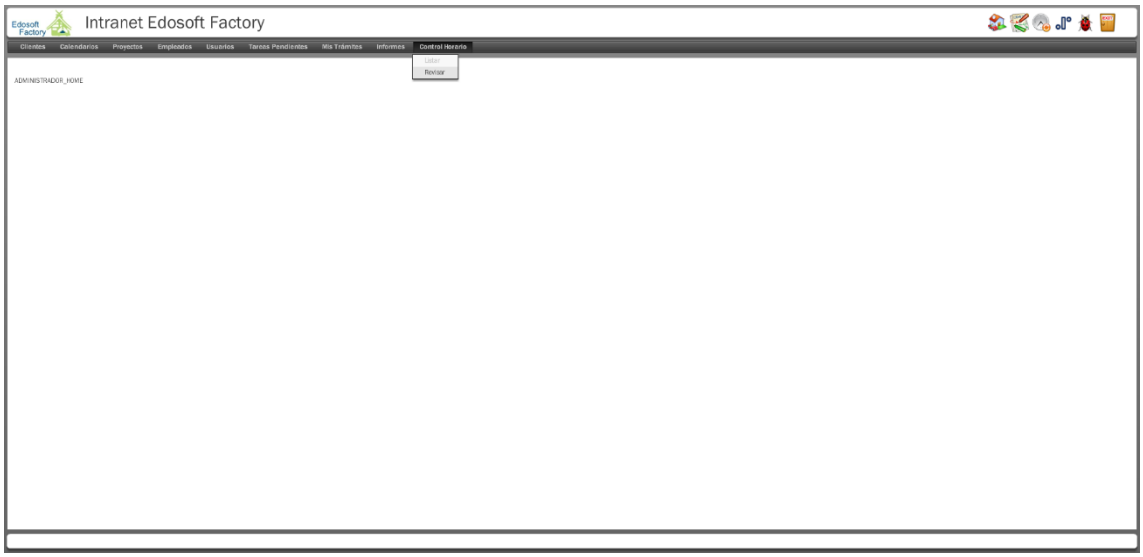

Figura 37. Menú Control horario.

Con la primera de estas opciones, habilitada para todo tipo de usuarios, se accede a la información de nuestros propios marcajes e incidencias asociadas. La única precondición para tener acceso a esta opción del menú es estar dado de alta en el sistema de control horario. Con la segunda opción, habilitada solo para usuarios con privilegios (Administradores y Jefes de UOI), se puede acceder a la información de marcajes e incidencias de todos los usuarios dados de alta, información de terminales Kreta3 conectados, gestión de jornadas laborales, y a toda la funcionalidad necesaria para realizar el resto de operaciones que permiten gestionar el sistema de Control horario.

1. Opción Listar

Tal y como se muestra en la Figura 38. Pantalla con información de marcajes y jornada laboral pertenecientes a la opción del menú Listar., el usuario pude obtener información de los marcajes realizados e incidencias asociadas, mediante tres pestañas a la derecha del área de visualización, bajo la barra de menús.

Figura 38. Pantalla con información de marcajes y jornada laboral pertenecientes a la opción del menú Listar.

En la primera pestaña se muestran todos los marcajes de entrada y salida realizados por el usuario. En la segunda pestaña se muestran los marcajes agrupados por días, junto con las horas totales y las horas realmente computadas (descontada la hora del almuerzo para jornadas de más de 7 horas) para cada una. Para cada una de estas pestañas en la parte inferior se muestra el número de horas totales y número de horas computadas. Una tercera pestaña muestra información de las incidencias en marcajes que se han producido para el usuario en cuestión. La siguiente tabla muestra los códigos de incidencias con su correspondiente significado:

Código de incidencia	Significado.
001	Excedida la hora de entrada máxima fijada
002	Incumplida la hora de salida mínima fijada
003	Incumplida la hora de entrada mínima fijada
004	Excedida hora de salida máxima fijada
005	Fichaje de entrada no debido (Fichaje en un día festivo, vacaciones o definido como no laborable)

006	Sin fichaje de salida (Solo presente el fichaje de entrada)
007	Sin fichaje (Falta fichaje de entrada y salida en un día laborable)
008	Incumplido el cómputo de horas semanales definidas.

Tabla 4

Las descripciones de las incidencias se pueden visualizar como tooltips, posicionando el cursor sobre cada una de ellas.

Para cada una de las tres pestañas existe la posibilidad de filtrar los datos mostrados mediante el selector al efecto. Por semana, mostrará la información de la semana en curso, por mes la del mes en curso, y por año la del año en curso.

En la parte izquierda del área de visualización es mostrada la información sobre la jornada laboral que el usuario tiene asignada. En esta se especifica una jornada ordinaria, activa durante unos determinados meses del año, un número total de horas semanales a realizar, y un horario de mínimo cumplimiento para cada día de la semana. Además opcionalmente puede presentarse una jornada intensiva para algunos meses del año, con otro número de horas y horario mínimo de exigido cumplimento. Esta jornada está pensando para los horarios de verano con jornada continua.

2. Opción Revisar (Disponible para Jefe UOI y Administrador)

Tal y como se muestra en la Figura 39. Opción revisar., los usuarios con privilegios pude gestionar el sistema de control horario mediante esta opción de menú.

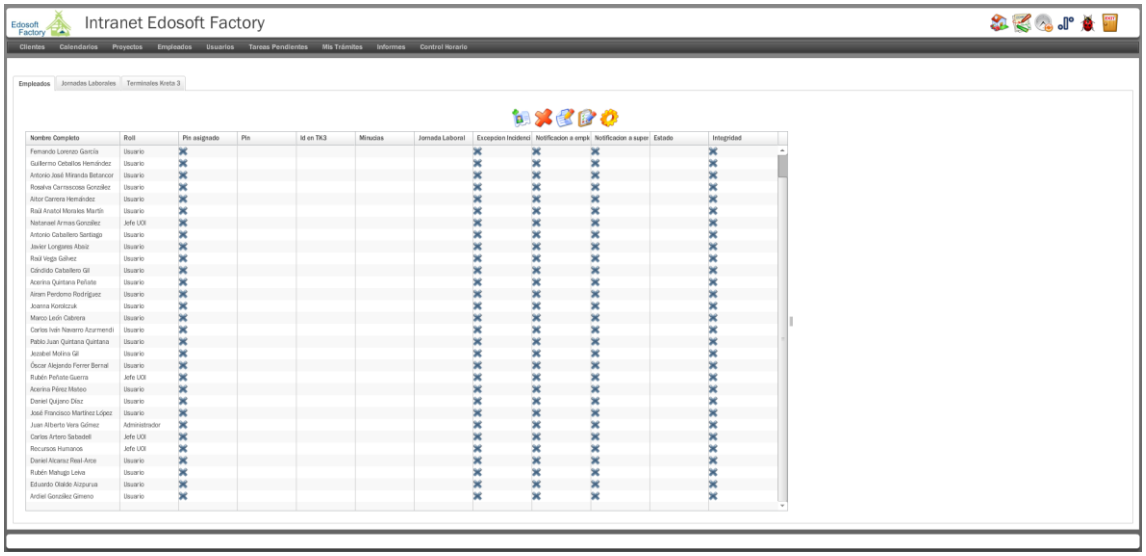

Figura 39. Opción revisar.

Las operaciones y muestra de datos relativos al control horario se divide en tres apartados diferentes, correspondientes a cada una de las pestañas mostradas a la izquierda del área de visualización: Empleados, Jornadas Laborales y Terminales Kreta 3.

2.1 Empleados

Permite visualizar los empleados dados de alta en la intranet, y entre estos los dados de alta en sistema de Control horario**,** en una tabla al efecto, con información del perfil configurado del usuario en el sistema de Control horario. De izquierda a derecha encontramos los siguientes atributos para cada usuario:

- Nombre completo: Nombre y apellidos con el que fue inscrito el usuario en la Intranet corporativa.
- Roll: Roll que tiene asignado el usuario, que determina los permisos para realizar las distintas operaciones de la Intranet corporativa. En el caso de no estar dado de alta en sistema de Control horario este será el último dato del usuario que constará.
- Pin asignado: Un aspa roja determina que el usuario no tiene un pin asignado que deberá introducir tras identificarse en el terminal mediante su huella dactilar. Con un check verde indica que si tiene un PIN de cuatro dígitos asociado que deberá introducir, el PIN se muestra en la siguiente columna de la tabla.
- PIN: Pin asignado al usuario, vacío en caso de no tener un PIN asignado.
- Id en TK3: identificador con el que el usuario ha sido dado de alta en cada uno de los terminales biométricos dados de alta en el sistema.
- Minucias: Datos biométricos almacenados para el usuario en cada terminar, con los que se realizará su autentificación.
- Jornada Laboral: Jornada Laboral asignada al usuario.
- Excepción incidencia: Indica si para el usuario se crearán incidencias de marcaje si incumple el horario establecido. Aspa roja en caso negativo check verde en caso afirmativo.
- Notificación a empleado: Indica si se enviará una notificación al empleado cuando se produzca una incidencia de marcaje. Aspa roja en caso negativo, check verde en caso afirmativo.

- Notificación a supervisor: Indica si se enviará una notificación al supervisor del empleado cuando se produzca una incidencia de marcaje. Aspa roja en caso negativo, check verde en caso afirmativo.
- Estado: Indica el estado del usuario en sistema. Una vez dado de alta tendrá el estado INSERTADO. Tras una baja mostrará el estado ELIMINADO.
- Integridad: Muestra el estado de integridad del usuario en el sistema tras una operación de verificación. Un aspa roja indica que hay algún fallo de integridad, y un check verde indica que el usuario está dato de alta en todos los terminales con los mismos datos en todos ellos, e iguales a su vez que los que constan en la base de datos. Pulsando sobre el aspa podremos obtener más información del problema existente.

Para cada uno de estos se muestra en la parte izquierda de la pantalla la información de marcajes e incidencias asociados, en función del empleado seleccionado y de la pestaña visible.

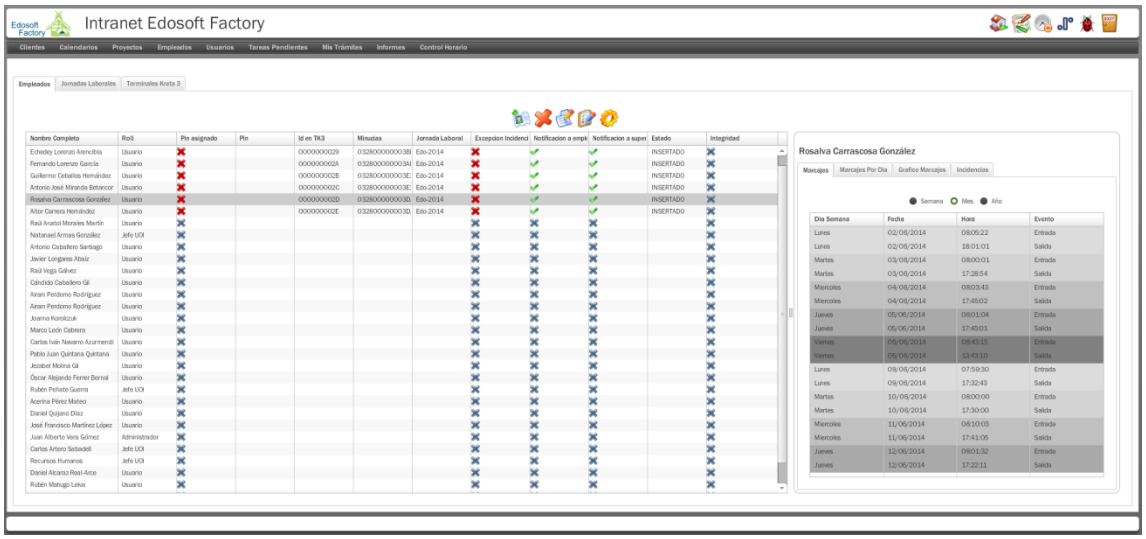

Figura 40. Opción revisar, pestaña empleados, visualización de pestaña Marcajes.

En la primera pestaña se muestran todos los marcajes de entrada y salida realizados por el usuario. En la segunda pestaña se muestran los marcajes agrupados por días, junto con las horas totales y las horas realmente computadas (descontada la hora del almuerzo para jornadas de más de 7 horas) para cada una. Para cada una de estas pestañas en la parte inferior se muestra el número de horas totales y número de horas computadas.

Una tercera pestaña donde se muestra un diagrama de barras con las horas totales computadas en un día en color verde, más la hora del almuerzo, si procede, que no computa como lectiva, en naranja.

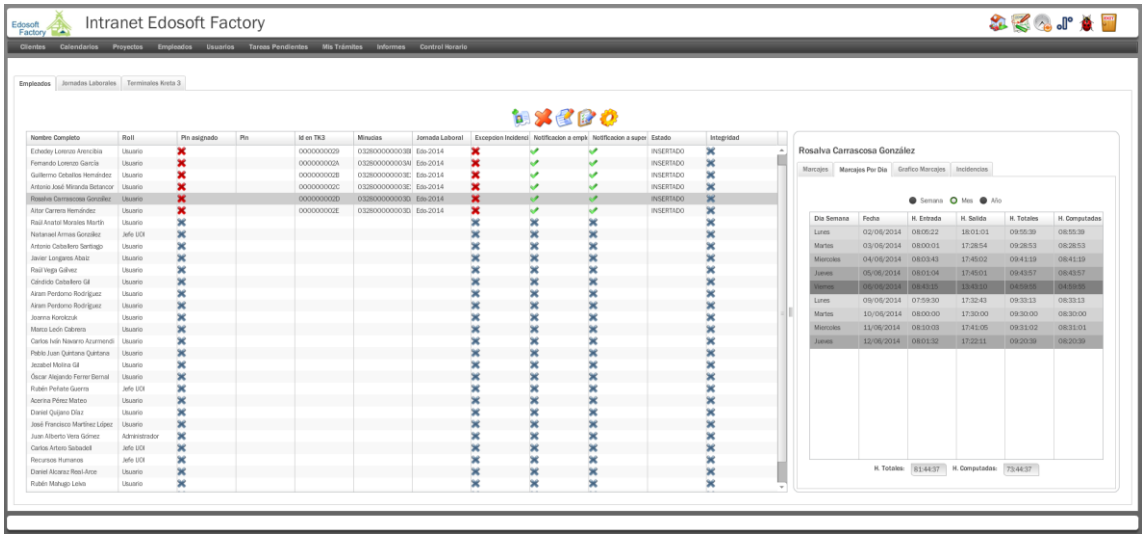

Figura 41. Opción revisar, pestaña empleados, visualización de pestaña Marcajes por Día.

Y una cuarta pestaña muestra información de las incidencias en marcajes que se han producido para el usuario en cuestión. La tabla 4 muestra los códigos de incidencias con su correspondiente significado. Las descripciones de las incidencias se pueden visualizar como tooltips, posicionando el cursor sobre cada una de ellas.

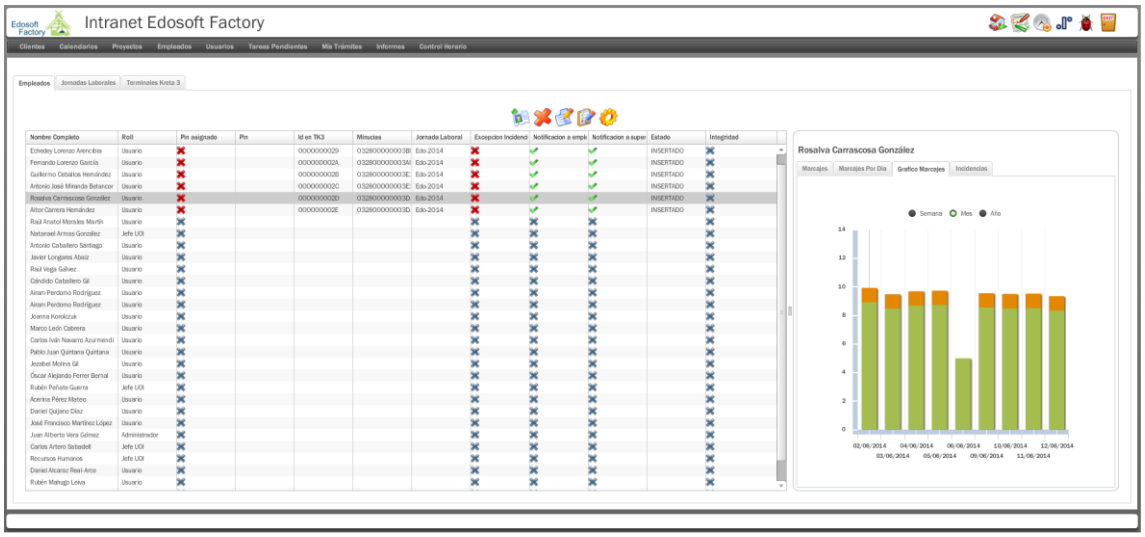

Figura 42. Opción revisar, pestaña empleados, visualización de pestaña Grafico de Marcajes.

Para cada una de las cuatro pestañas existe la posibilidad de filtrar los datos mostrados mediante el selector al efecto. Por semana, mostrará la información de la semana en curso, por mes, la del mes en curso, y por año, la del año en curso.

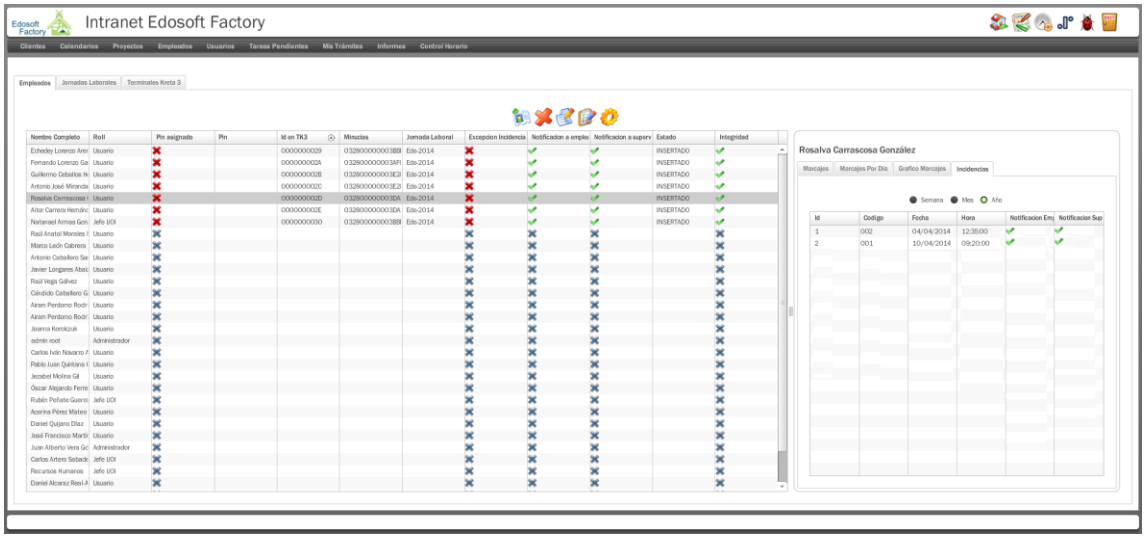

Figura 43. Opción revisar, pestaña empleados, visualización de pestaña Incidencias

Además se. Mediante la botonera situada sobre esta tabla se podrá realizar las diferentes operaciones: Altas, Modificaciones, Bajas, etc.

Alta de Empleado

Pulsando sobre el primer botón de la botonera contextual se abre una ventana mediante la que se permite establecer el alta en del empleado seleccionado en el sistema de Control horario.

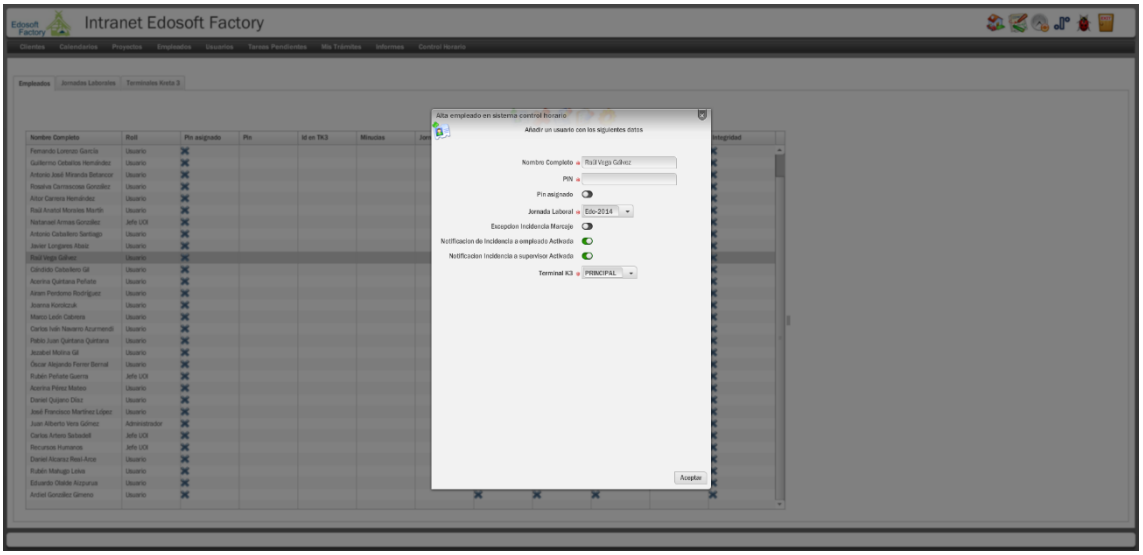

Figura 44. Alta de empleado en el sistema de Control Horario

En esta se especifica los siguientes elementos:

- PIN: Pin asignado al usuario, vacío en caso de no tener un PIN asignado.
- Pin asignado: Selector que indica si el usuario tiene asignado un pin asignado que deberá introducir tras identificarse en el terminal mediante su huella dactilar o no.
- Jornada Laboral: Jornada Laboral asignada al usuario.
- Excepción incidencia: Selector que indica si para el usuario se crearán incidencias de marcaje si incumple el horario establecido.
- Notificación al empleado: Selector que indica si se enviará una notificación al empleado cuando se produzca una incidencia de marcaje.
- Notificación al supervisor: Selector que indica si se enviará una notificación al supervisor del empleado cuando se produzca una incidencia de marcaje.

El empleado no debe estar dado de alta previamente en el sistema de Control horario, o haber sido dado de baja para que se permita esta operación. Tras pulsar el botón de aceptar comienza el proceso de Alta en el terminal mostrándose la ventana que irá informando del estado de la operación.

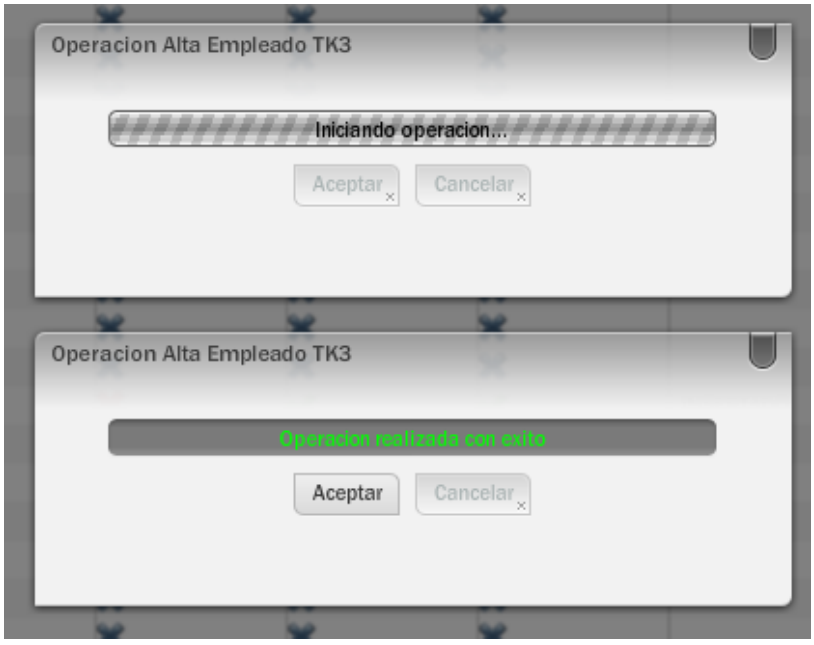

Figura 45. Mensajes de estado durante el proceso de Alta de empleado en el sistema de Control Horario.

El terminal avisará con un pitido para que coloquemos el dedo sobre el terminal, seguido de un segundo pitido para que lo coloquemos de nuevo, para la captura de una segunda huella. Se recomienda el dedo índice en la posición indicada con un círculo azul en el dibujo de la Figura 46.

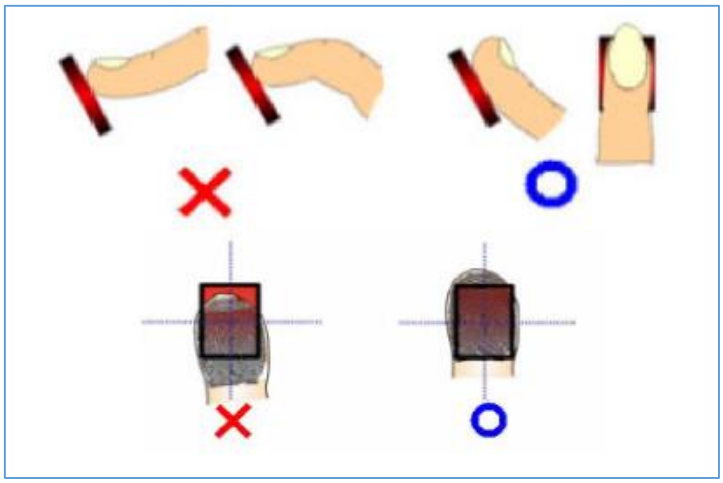

Figura 46. Posicionamiento recomendado del dedo en el sensor biométrico.

Tras la captura de huellas continua con el alta en cada terminal conectado a la intranet y finalizará mostrando con el mensaje "Operación realizada con éxito" en caso de la finalización exitosa de la operación de alta o un mensaje con el error que se ha producido en caso contrario. Se finalizará pulsando el botón aceptar. Si la operación no finaliza existe la posibilidad de cancelarla. Si esto se produce se deberá desbloquear el terminal y verificar el estado del empleado, pudiendo ser necesario una nueva alta o una subsanación.

Modificación de Empleado

Pulsando sobre el segundo botón de la botonera contextual se abre una ventana similar a la de Alta de empleado mediante la que se permite modificar los datos del empleado seleccionado. El empleado debe estar dado de alta en el sistema de Control horario para que se permita su modificación.

Baja de Empleado

Pulsando sobre el tercer botón de la botonera contextual se da de baja el empleado seleccionado en la tabla de empleados del sistema de Control horaria, quedando reflejado esta operación mediante el estado ELIMINADO.

Verificación de Empleado

Pulsando sobre el cuarto botón de la botonera contextual se verifica el empleado seleccionado en la tabla de empleados en el sistema de Control horario. Tras pulsar el botón de aceptar en la ventana ce confirmación comienza el proceso de Verificación del empleado en los diferentes terminales conectados, mostrándose la ventana que irá informando del estado de la operación.

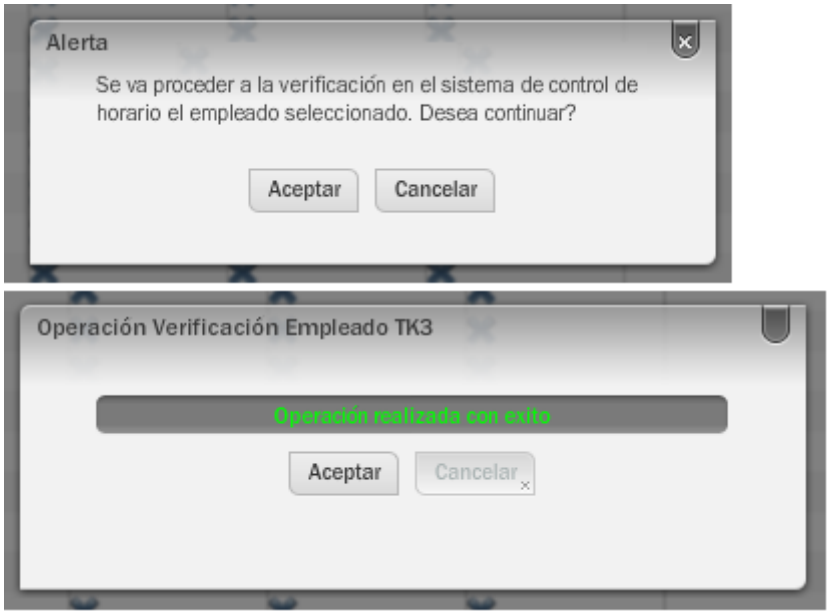

Figura 47. Mensajes de estado durante el proceso de Verificación de empleado en el sistema de Control Horario.

*L*a verificación finalizará mostrando el mensaje "Operación realizada con éxito" en caso de la finalización exitosa de la operación de verificación o un mensaje con el error que se ha producido en caso contrario. Se concluirá pulsando el botón aceptar. Si la operación no finalizara existe la posibilidad de cancelarla. Si esto se produce se deberá, si algún terminal ha quedado bloqueado, proceder a desbloquearlo accediendo a la pestaña de Terminles Kreta 3. Así mismo se deberá comprobar el estado del empleado verificado, pudiendo ser necesario una nueva verificación una vez solucionado el problema indicado mediante el correspondiente mensaje.

Finalmente en la columna Estado se indica el estado del empleado con respecto a su integridad en el sistema. Un aspa roja indica que hay algún fallo de integridad, y un check verde indica que el usuario está dato de alta en todos los terminales con los mismos datos en todos ellos, e iguales a su vez que los que constan en la base de datos. Pulsando sobre el aspa podremos obtener más información del problema existente.

Subsanación de Empleado

Pulsando sobre el quinto botón de la botonera contextual se subsana la perdida de integridad en el estado del empleado seleccionado en la tabla de empleados en el sistema de Control horario. Tras pulsar el botón de aceptar en la ventana ce confirmación comienza el proceso de Subsanación del estado del empleado en los diferentes terminales conectados, mostrándose la ventana que irá informando del estado de la operación.

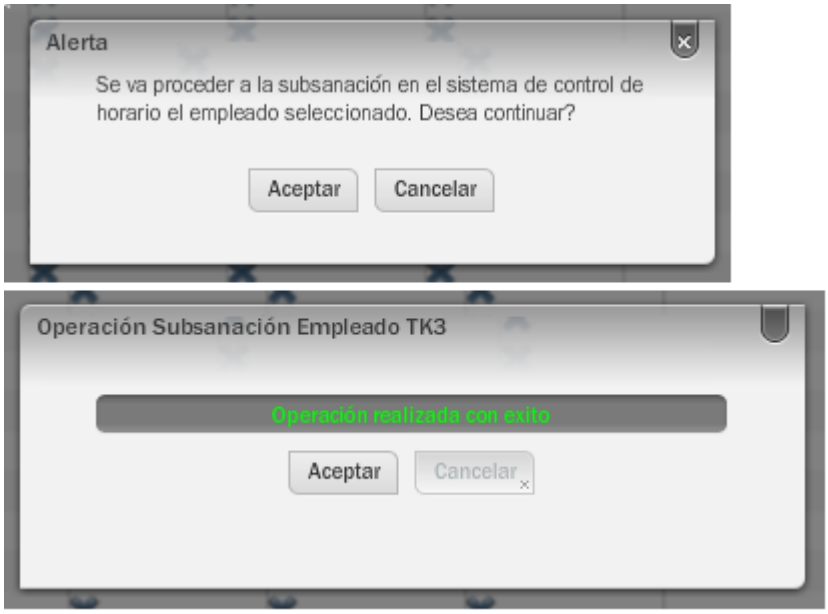

Figura 48. Mensajes de estado durante el proceso de Subsanación de empleado en el sistema de Control Horario.

La subsanación finalizará mostrando el mensaje "Operación realizada con éxito" en caso de la finalización exitosa de la operación de subsanación o un mensaje con el error que se ha producido en caso contrario. Se concluirá pulsando el botón aceptar. Si la operación no finalizara existe la posibilidad de cancelarla. Si esto se produce se deberá, si algún terminal ha quedado bloqueado, proceder a desbloquearlo accediendo a la pestaña de Terminles Kreta 3. Así mismo se deberá comprobar el estado del empleado subsanado, pudiendo ser necesario una nueva subsanación una vez solucionado el problema indicado mediante el correspondiente mensaje.

Finalmente en la columna Estado se indicará el estado del empleado con respecto a su integridad en el sistema. Un aspa roja indica que hay algún fallo de integridad, y un check verde indica que el usuario está dato de alta en todos los terminales con los mismos datos en todos ellos, e iguales a su vez que los que constan en la base de datos. Pulsando sobre el aspa podremos obtener más información del problema existente.

2.2 Jornada laboral

Permite visualizar las jornadas laborales dadas de altas en el sistema, en una tabla al efecto, con información relacionada con las mismas. De izquierda a derecha encontramos los siguientes atributos para cada jornada laboral:

- Nombre: Nombre identificativo de la Jornada laboral.
- Fecha de alta: Fecha de creación de la Jornada laboral. Una vez dada de alta podrá ser asignada a un usuario.
- Fecha de baja: Fecha de eliminación de la Jornada laboral, en blanco en caso de estar activa.

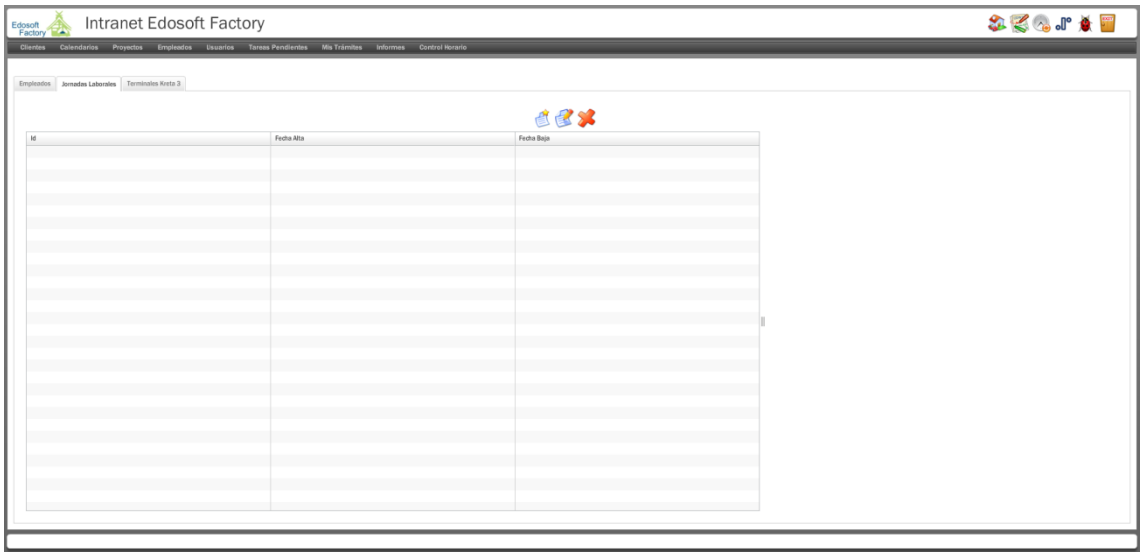

Figura 49. Opción revisar, pestaña Jornadas Laborales.

En la parte izquierda del área de visualización es mostrada la información sobre la jornada laboral seleccionada. En esta se especifica una jornada ordinaria, activa durante unos determinados meses del año, un número total de horas semanales a realizar, y un horario de mínimo cumplimiento para cada día de la semana. Además opcionalmente puede presentarse una jornada laboral intensiva para algunos meses del año no asignados a la jornada ordinaria, con otro número de horas y horario mínimo de exigido cumplimento. Esta jornada está pensada para los horarios de verano con jornada continua.

Mediante la botonera situada en la parte superior se podrá realizar las diferentes operaciones: Alta de Jornadas, Modificación de Jornadas y Bajas de Jornadas.

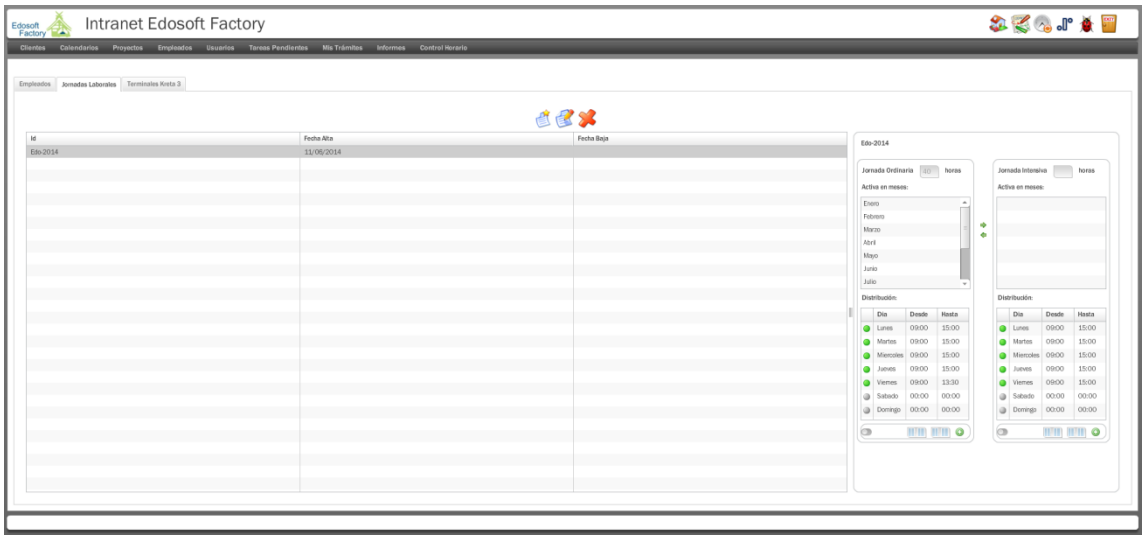

Figura 50. Opción revisar, pestaña Jornadas Laborales, visualización de los datos de una Jornada Laboral.

Alta de Jornada Laboral

Pulsando sobre el primer botón de la botonera contextual se abre una ventana mediante la cual se realiza la configuración de una nueva Jornada Laboral.

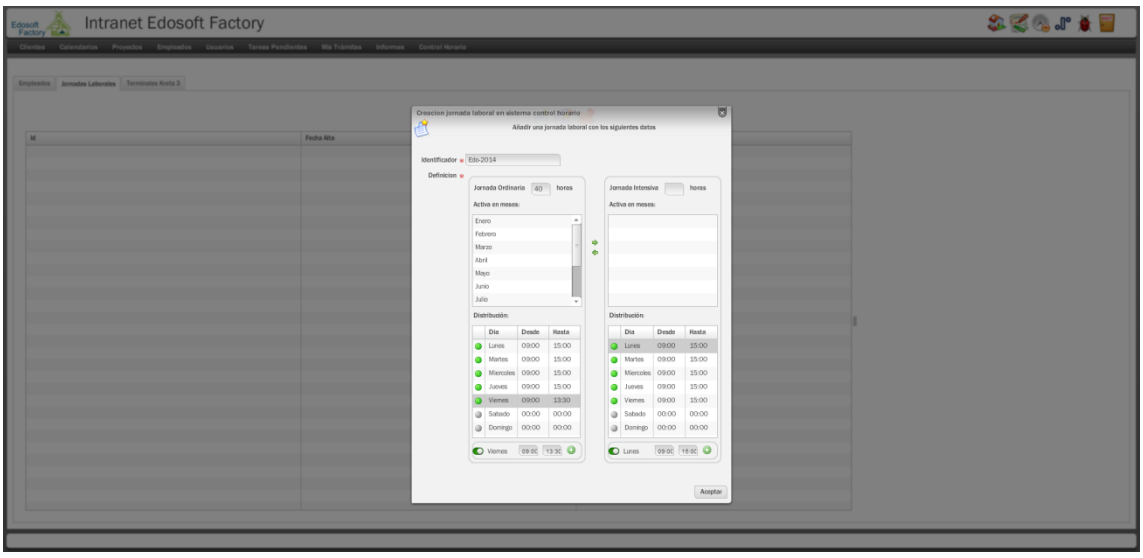

Figura 51. Creación de Jornada laboral.

En esta se especifica los siguientes elementos:

- Identificador: Identificador único de Jornada Laboral.
- Jornada ordinaria:
    - Se especifica el número de horas mínimo a realizar por semana.

- Se selecciona mediante las flechas verdes los meses del año en que la jornada estará activa. Los meses seleccionados pasan a la columna de jornada intensiva y deberá ser configurada.
- Se configura un horario de mínimo cumplimiento para cada día de la semana. Se selecciona el día deseado de la tabla y sus datos pueden ser modificados mediante los compontes bajo la tabla. El selector permite configurar los días de la semana lo lectivos. Desde es la hora de llegada más tarde permitida. Y hasta es la hora de salida más temprana permitida

- Jornada intensiva: Se configura de la misma manera que la ordinaria, es opcional y cuando está activa no se resta la hora correspondiente a la comida aunque se superen las 7 horas y 15 minutos.

Modificación de Jornada Laboral

Pulsando sobre el segundo botón de la botonera contextual se abre una ventana igual a la anterior mediante la que se permite modificar la Jornada Laboral seleccionada en la tabla de Jornadas Laborales.

Baja de Jornada Laboral

Pulsando sobre el tercer botón de la botonera contextual se elimina la Jornada Laboral seleccionada en la tabla de Jornadas Laborales. A la jornada laboral se la asigna una fecha de baja y ya no puede ser asignada a ningún usuario. Es necesario que la Jornada Laboral no esté asignada a ningún usuario para que sea permitida su eliminación.

2.3 Terminal Kreta 3.

Permite la visualización de los terminales biométricos instalados en el sistema en una tabla al efecto, con diferente información de configuración relacionada con los mismos. De izquierda a derecha encontramos los siguientes atributos para cada terminal configurado:

- Identificador: Identificador único del terminal

- Descripción: Descripción asociada al terminal. Ej: Terminal principal.
- Ip Terminal: Direccíon IP del terminal.
- Ip Gateway: Puerta de enlace configurada en el terminal.
- Mascara De Red: Mascara de red configurada en el terminal.
- Ip Host Remoto: Ip del host remoto desde el cual se permite conexiones al terminal configurado en este.
- Puerto Host Remoto: Puerto del host remoto desde el cual se permite conexiones al terminal configurado en este
- Dhcp: Indica si el dhcp está habilitado o no. Con un check verde en caso afirmativo y un aspa roja en caso contrario.
- MAC: Indica la dirección MAC del terminal.
- Modo Online: Indica si el terminal tiene el modo online habilitado, función que permitirá la recepción de marcajes provenientes de los terminales en tiempo real.
- Número de marcajes: Número de marcajes almacenados en un terminal.
- Fecha Alta: Fecha en que se conectó por primera vez el terminal a la intranet corporativa.
- Operaciones Marcajes: Esta columna no muestra información relativa al terminal, tan solo un conjunto de botones que permiten hacer en los mismos diversas operaciones relativas a marcajes.
- B (Bloqueado): Incida si el terminal está bloqueado mediante un testigo rojo o verde en caso contrario. El bloqueo de un terminal se puede producir por un error crítico en alguna operación que requiere el uso exclusivo del terminal, o por la cancelación de una operación por parte del usuario antes de su finalización.
- E (Estado): Indica el estado del terminal biométrico tras una actualización de su estado. Un testigo verde indica que está en línea, un testigo rojo que no tiene conectividad.

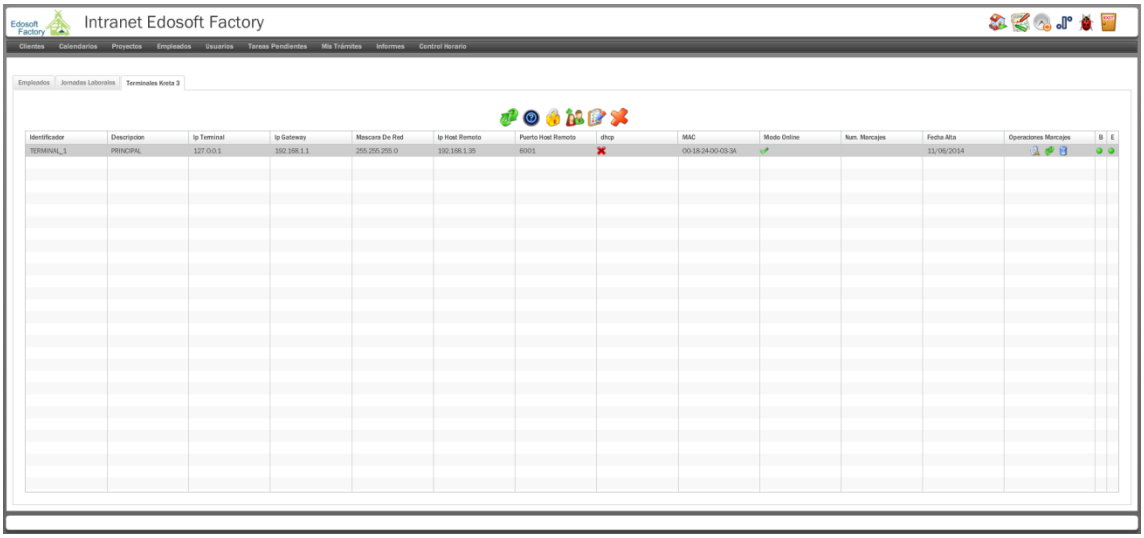

Figura 52. Opción revisar, pestaña Terminales Kreta 3.

Mediante la botonera situada en la parte superior se podrá realizar las diferentes operaciones relacionadas con los terminales o con gestión general del sistema tales como: Comprobar la conectividad de un terminal, refrescar la información de configuraciones de un terminal, dar de baja todos los marcajes e incidencias almacenados en el sistema, etc. Así mismo con los botones situados en la columna de operaciones de marcaje de la tabla de terminales, se pueden realizar diversas operaciones relacionadas con los marcajes almacenados en cada terminal: Sincronizar los marcajes almacenados en un determinado terminal biométrico con los almacenados en la base de datos de la intranet, eliminar los marcajes almacenados en un terminal biométrico o actualizar el número de marcajes almacenador en un terminal biométrico, cantidad que se muestra en la tabla de terminales.

Refrescar información de configuración del terminal

Pulsando sobre el primer botón de la botonera contextual se refresca la información de configuración del terminal seleccionado. La información del número de marcajes así como la del estado del terminal no se refresca con esta operación.

Comprobar conectividad del terminal

Pulsando sobre el segundo botón de la botonera contextual se comprueba la conectividad del terminal seleccionado, pasando el testigo de la columna E (Estado) a color gris en el trascurso de la operación, tal y como muestra la Figura 53.

El resultado de la operación será testigo verde que indica que está en línea, o un testigo rojo que no tiene conectividad.

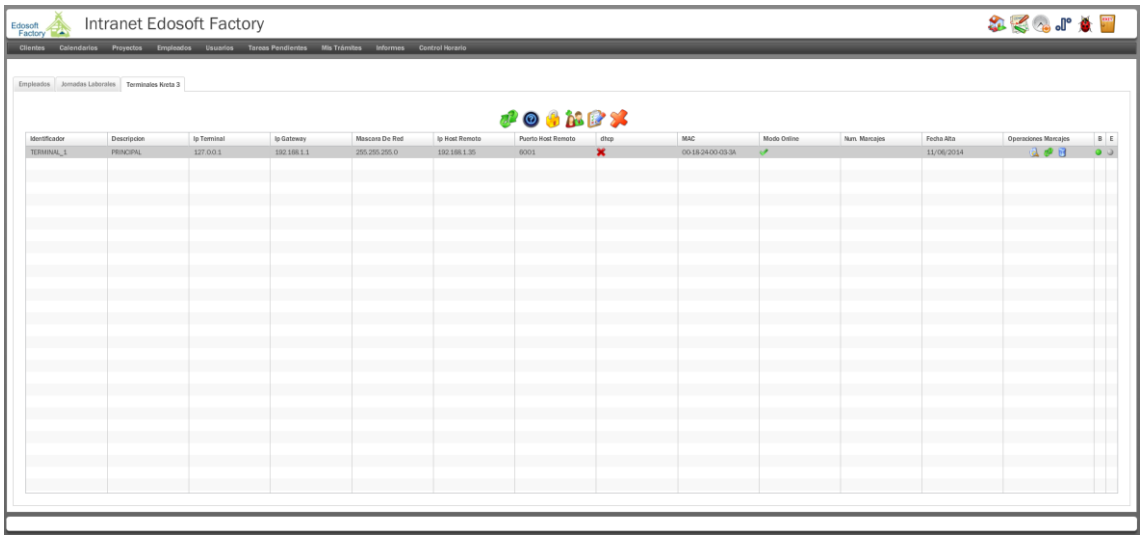

Figura 53. Proceso de comprobación de conectividad del terminal biométrico.

Desbloquear terminal

Pulsando sobre el tercer botón de la botonera contextual se desbloquea el terminal seleccionando, permitiendo realizar operaciones con el mismo, que de otra manera no se podrían realizar. Para poder lanzar esta operación el testigo en la columna B (Bloqueado) debe estar en color rojo. Una vez finalizado pasará a color verde.

Inserción masiva en terminal

Pulsando sobre el cuarto botón de la botonera contextual se inicializa la inserción masiva de empleados en el terminal seleccionando. Esta operación se realiza sobre un terminal vacío para cursar en éste todas altas de los empleados del Control horario almacenados en el sistema. Tras pulsar el botón de aceptar en la ventana de confirmación comienza el proceso de Inserción masiva de empleados en el terminal seleccionado, mostrándose la ventana que irá informando del estado de la operación.

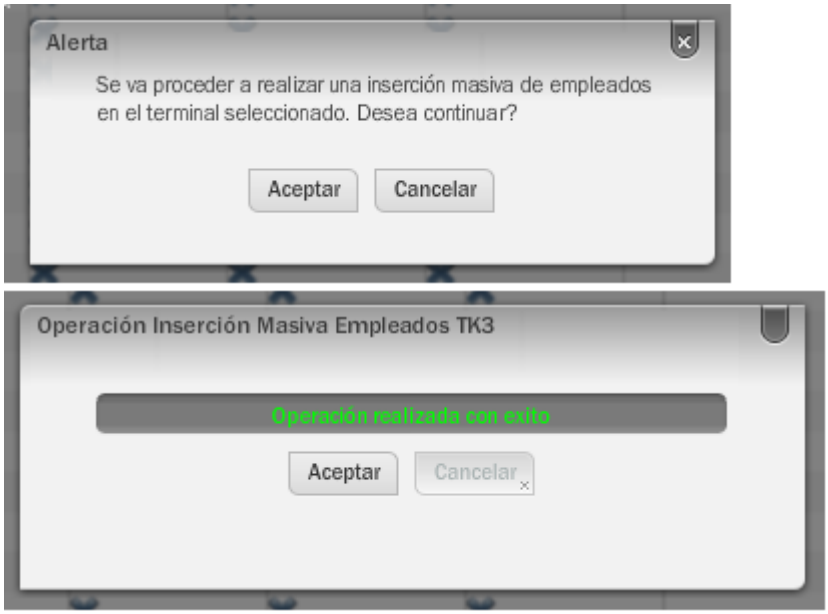

Figura 54. Mensajes de estado durante el proceso de Alta masiva de empleados en un determinado terminal biométrico.

La inserción masiva finalizará mostrando el mensaje "Operación realizada con éxito" en caso de la finalización exitosa de la operación de verificación o un mensaje con el error que se ha producido en caso contrario. Se concluirá pulsando el botón aceptar. Si la operación no finalizara existe la posibilidad de cancelarla. Si esto se produce se deberá proceder a desbloquear el terminal sobre el que operamos accediendo a la pestaña de Terminles Kreta 3. Así mismo se deberá comprobar el estado de todos los empleados mediante una verificación global.

Verificación masiva

Pulsando sobre el quinto botón de la botonera contextual se verifica la integridad de todos los empleados en el sistema de Control horario. Tras pulsar el botón de aceptar en la ventana ce confirmación comienza el proceso de Verificación masiva de todos los empleados en los diferentes terminales conectados, mostrándose la ventana que irá informando del estado de la operación.

Figura 55. Mensajes de estado durante el proceso de Verificación masiva de todos los empleados.

La Verificación masiva finalizará mostrando el mensaje "Operación realizada con éxito" en caso de la finalización exitosa de la operación de verificación o un mensaje con el error que se ha producido en caso contrario. Se concluirá pulsando el botón aceptar. Si la operación no finalizara existe la posibilidad de cancelarla. Si esto se produce se deberá, si algún terminal ha quedado bloqueado, proceder a desbloquearlo accediendo a la pestaña de Terminles Kreta 3. Así mismo se deberá comprobar el estado de todos los empleados mediante una verificación global de su estado.

Baja de marcajes e incidencias

Pulsando sobre el quinto botón de la botonera contextual se dan de baja todos los marcajes e incidencias almacenadas el sistema de Control horario, entrando a formar parte del histórico. Tras pulsar el botón de aceptar en la ventana de confirmación comienza el proceso de baja.

Obtener y eliminar los marcajes almacenados en un terminal

Pulsando sobre el primer botón de la columna de Operación marcajes se sincronizan todos los marcajes almacenados en el terminal seleccionado con la base de datos, es decir se copian a la base de datos todos aquellos marcajes que no han sido almacenados previamente. Así mismo son eliminados todos los marcajes del terminal

biométrico. Tras pulsar el botón de aceptar en la ventana de confirmación comienza el proceso de obtención y eliminación.

Obtener el número de marcajes almacenados en un terminal

Pulsando sobre el segundo botón de la columna de Operación marcajes se refresca la información mostrada en la columna Núm Marcajes del terminal seleccionado, donde se muestra el número de marcajes almacenados en el terminal biométrico y el total que permite almacenar. Tras pulsar el botón de aceptar en la ventana de confirmación comienza el proceso de refresco.

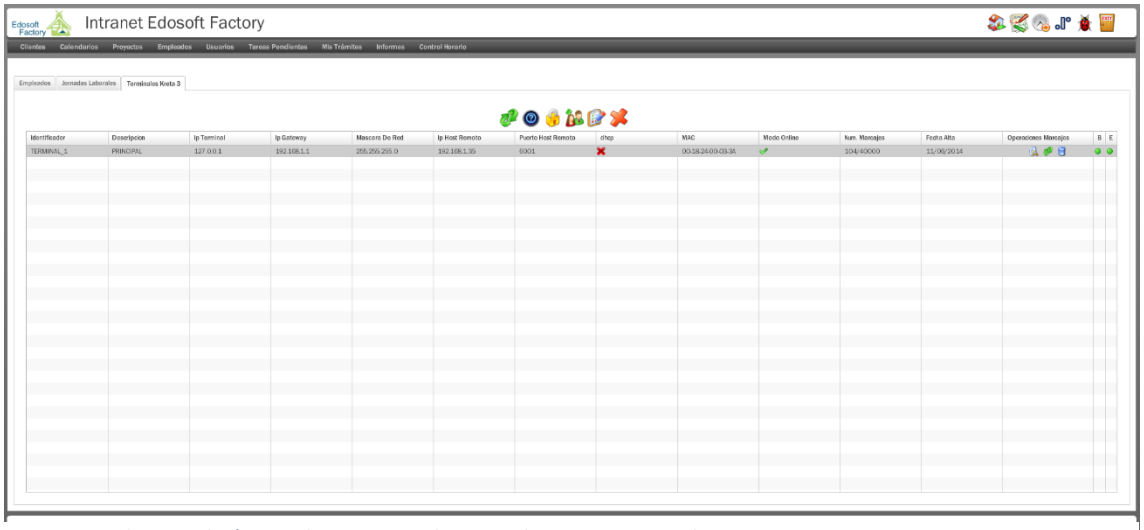

Figura 56. Obtener el número de marcajes almacenados en un terminal.

Eliminar los marcajes almacenado en un terminal

Pulsando sobre el tercer botón de la columna de Operación marcajes quedarán eliminados todos los marcajes almacenados en el terminal seleccionado sin una previa sincronización. Tras pulsar el botón de aceptar en la ventana de confirmación comienza el proceso de eliminación.

3. Instrucciones de marcaje en el terminal

Al comienzo y final de la jornada laboral los empleados dados de alta en el módulo de control y gestión horaria deberán realizar sendos marcajes de entrada y salida, autenticándose mediante el terminal biométrico al efecto, situado junto a la puerta de la Oficina 4.

Figura 57. Terminal biométrico instalado en Oficina 4.

3.1 Marcajes de entrada

Al iniciar la jornada laboral el empleado deberá realizar un marcaje de entrada. En primer lugar se deberá cerciorar que el terminal se encuentra en modo entrada, que vendrá indicado mediante la palabra Entrada mostrada en el display del terminal (Ver Figura 58). Si no fuera así, y el modo seleccionado fuera Salida, pulsar el botón de flecha verde situado en el teclado del mismo pasando de esta manera al modo deseado (Ver Figura 59). Finalmente situar el dedo en el sensor biométrico tal y como indica el diagrama de la Figura 46.

Figura 58. Display mostrando el modo entrada.

Si la identificación se ha producido correctamente se mostrará el mensaje por pantalla quedando registrado en marcaje de entrada. Si no ha habido una identificación positiva se mostrara el mensaje pertinente y un nuevo intento de marcaje de entrada deberá ser realizado.

Figura 59. Teclado del terminal

Si el empleado tuviera activado la opción de PIN, se le solicitará tras la captura biométrica y deberá insertar los cuatro dígitos del mismo finalizando con el botón "Enter", situado en la esquina inferior derecha del teclado (Ver Figura 59).

3.2 Marcajes de salida

Al finalizar la jornada laboral el empleado deberá realizar un marcaje de salida. En primer lugar se deberá cerciorar que el terminal se encuentra en modo salida, que vendrá indicado mediante la palabra Salida mostrada en el display del terminal (Ver Figura 58). Si no fuera así, y el modo seleccionado fuera Entrada, pulsar el botón de flecha roja situado en el teclado del mismo pasando de esta manera al modo deseado (Ver Figura 59). Finalmente situar el dedo en el sensor biométrico tal y como indica el diagrama de la Figura 46.

Figura 60. Display mostrando el modo salida.

Si la identificación se ha producido correctamente se mostrará el mensaje por pantalla quedando registrado en marcaje de entrada. Si no ha habido una identificación positiva se mostrara el mensaje pertinente y un nuevo intento de marcaje de en salida deberá ser realizado.

Apéndice II. JavaDoc

En este apéndice se muestra únicamente el JavaDoc de la interfaz remota de usuario, como fachada principal del modelo de negocio, y tan solo los métodos relativos al módulo de control y gestión de horarios. Es la fachada que emplea la Interfaz Gráfica y establece la interfaz de comunicaciones entre esta y el modelo de negocios. El resto de javaDocs entre la que destaca la fachada de ControlHorario se entrega como parte del código.

Interface UsuarioFacadeRemote

- All Known Implementing Classes:
  UsuarioFacadeBean

```
public interface UsuarioFacadeRemote
```

Interfaz remota que implementa el bean de sesión UsuarioFacadeBean que es instanciado para las nuevas sesiones que son creadas.

- **Method Detail**
  - **getMenuPorIdUsuario**

    ```
    java.lang.String getMenuPorIdUsuario(java.lang.String idEm
    pleado)
    ```

  - **checkTerminalKreta3OK**

    ```
    int checkTerminalKreta3OK(long idTerminal)

    Comprueba la conexión con Terminal de control de horarios.
    El resultado de esta operación asíncrona será enviado por
    topic. Operación de terminal.

    Parameters:
          idTerminal - Identificador del terminal.
    Returns:
          ControlHorarioFacade.OPERACION_CORRECTA si la
          petición se ha realizado con éxito
          o ControlHorarioFacade.NAMING_EXCEPTION, ControlHorar
          ioFacade.RESOURCE_EXCEPTION UsuarioFacade.TERMINAL_NO
          _ENCONTRADO_O_CONEXION_NO_ESTABLECIDA en caso de
          excepciones.
    ```

- **getTerminalKreta3Configuration**

```
int getTerminalKreta3Configuration(long idTerminal)

Realiza la petición serializada de parámetros de
configuración al Terminal de control de horarios. El
resultado de esta operación asíncrona será enviado por
topic. Operación de terminal.

Parameters:
      idTerminal - Identificador del terminal.
Returns:
      ControlHorarioFacade.OPERACION_CORRECTA si la
      petición se ha realizado con éxito
      o ControlHorarioFacade.NAMING_EXCEPTION, ControlHorar
      ioFacade.RESOURCE_EXCEPTION UsuarioFacade.TERMINAL_NO
      _ENCONTRADO_O_CONEXION_NO_ESTABLECIDA en caso de
      excepciones o error.
```

- **getAllTerminalKreta3Configurations**

```
int getAllTerminalKreta3Configurations()

Realiza la petición serializada de parámetros de todos los
Terminales de control de horarios configurados en el
sistema. El resultado de esta operación asíncrona será
enviado por topic. Operación de terminal.

Returns:
      ControlHorarioFacade.OPERACION_CORRECTA
```

- **altaEmpleadoTerminalKreta3**

```
int altaEmpleadoTerminalKreta3 (long idEmpleado,
boolean pinAsignado, java.lang.String pin,
java.lang.String idJornadaLaboral,
boolean excepcionIncidenciaMarcaje,
boolean notificacionEmpleadoTK3Activada,
boolean notificacionSupervisorActivada, long idTerminal)

Realiza el alta de un empleado en el sistema de control de
horarios. El resultado de esta operación asíncrona será
enviado por topic. Operación de empleadoTK3.

Parameters:
      idEmpleado - Identificador del empleado de la
      intranet corporativa que se desea dar de alta en el
      sistema de control de horario.
      pinAsignado - Valor que indica si el usuario tiene un
            pin activo a incluir cada vez que realiza un
            marcaje.
      pin - String de 4 caracteres obtenida del terminal
      que representa el pin asociado al empleadoTK3.
      idJornadaLaboral - Identificador de la jornada
      laboral asociada al empleado
      exencionIncidenciaMarcaje - Indica si el empleado
      está exento de ser objeto de incidencias asociadas al
      marcaje
```

notificacionEmpleadoTK3Activada - Indica si el empleado será notificado vía email cuando se produzca un incidente de marcaje (solo si exencionIncidenciaMarcaje está a false).
notificacionSupervisorActivada - Indica si el supervisor del empleado será notificado vía email cuando se produzca un incidente de marcaje (solo si exencionIncidenciaMarcaje está a false).
idTerminal - Identificador del terminal donde se captura la huella.

Returns:

Devuelve el resultado de la operación ControlHorarioFacade.OPERACION_CORRECTA cuando la petición de captura de huella biométrica se ha realizado correctamente ControlHorarioFacade.EMPLEADO_TK3_ALTA_PREVIA_BAJA empleado dado de alta sin necesidad de captura de huella. ControlHorarioFacade.NAMING_EXCEPTION oControlHorarioFacade.RESOURCE_EXCEPTION caso de excepciones en el proceso de captura. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO, UsuarioFacade.DATOS_INCORRECTOS o UsuarioFacade.TERMINAL_NO_ENCONTRADO_O_CONEXION_NO_ESTABLECIDA en caso de diferentes errores.

- **bajaEmpleadoTerminalKreta3**

```
int bajaEmpleadoTerminalKreta3(long idEmpleado)
```

Realiza la baja de un empleado en el sistema de control de horarios. El resultado de esta operación asíncrona será enviado por topic. Operación de empleadoTK3.

Parameters:

idEmpleado - Identificador del empleado de la intranet corporativa que se desea dar de alta en el sistema de control de horario.

Returns:

Devuelve el resultado de la operación: ControlHorarioFacade.OPERACION_CORRECTA cuando la petición de captura de huella biométrica se ha realizado correctamente ControlHorarioFacade.EMPLEADO_TK3_ESTADO_BAJA si existe empleadoTK3 de baja. ControlHorarioFacade.EMPLEADO_NO_ASOCIADO_A_EMPLEADO_TK3 si no existe empleadoTK3 asociado a empleado. ControlHorarioFacade.EMPLEADO_TK3_NO_EXISTENTE_EN_PERSISTENCIA fallo en la búsqueda de empleadoTK3 durante el proceso (fallo grave ya que siempre debiera estar según comprobaciones previas). ControlHorarioFacade.NAMING_EXCEPTION oControlHorarioFacade.RESOURCE_EXCEPTION en caso de excepciones en el proceso de baja en terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO en caso de error en la operación por bloqueo de terminal.

- **modificaEmpleadoTerminalKreta3**

```
int modificaEmpleadoTerminalKreta3(long idEmpleado,
boolean pinAsignado, java.lang.String pin,
java.lang.String idJornadaLaboral,
boolean excepcionIncidenciaMarcaje,
boolean notificacionEmpleadoTK3Activada,
boolean notificacionSupervisorActivada)
```

Realiza la modificación de un empleado en el sistema de control de horarios. El resultado de esta operación asíncrona será enviado por topic. Operación de empleadoTK3.

Parameters:

idEmpleado - Identificador del empleado de la intranet corporativa que se desea modificar en el sistema de control de horario.
pinAsignado - Nuevo valor para el parámetro pinAsignado (Indica si el usuario tiene un pin activo a incluir cada vez que realiza un marcaje).
pin - Nuevo valor para el parámetro pin (String de 4 caracteres obtenida del terminal que representa el pin asociado al empleadoTK3.
idJornadaLaboral - Nuevo valor para el parámetro idJornadaLaboral (Identificador de la jornada laboral asociada al empleadoTK3).
exencionIncidenciaMarcaje - Nuevo valor para el parámetro exencionIncidenciaMarcaje (Indica si el empleadoTK3 está exento de ser objeto de incidencias asociadas al marcaje).
notificacionEmpleadoTK3Activada - Nuevo valor para el parámetro notificacionEmpleadoTK3Activada (Indica si el empleadoTK3 será notificado vía email cuando se produzca un incidente de marcaje (solo si exencionIncidenciaMarcaje está a false)).
notificacionSupervisorActivada - Nuevo valor para el parámetro notificacionSupervisorActivada(Indica si el supervisor del empleadoTK3 será notificado via email cuando se produzca un incedente de marcaje (solo si exencionIncidenciaMarcaje está a false)).

Returns:

Devuelve el resultado de la operación: ControlHorarioFacade.OPERACION_CORRECTA cu ando la petición de captura de huella biométrica se ha realizado correctamente. ControlHorarioFacade.EMPLEADO_TK3_ESTADO_BAJA Si existe empleadoTK3 de baja. ControlHorarioFacade.EMPLEADO_NO_ASOCIADO_A_EMPLEADO_ TK3si no existe empleadoTK3 asociado a empleado. ControlHorarioFacade.EMPLEADO_TK3_NO_EXISTENTE_EN_PER SISTENCIA fallo en la búsqueda de empleadoTK3 durante el proceso (fallo grave ya que siempre debiera estar según comprobaciones previas). ControlHorarioFacade.NAMING_EXCEPTION oCont rolHorarioFacade.RESOURCE_EXCEPTION en caso de excepciones en el proceso de modificación en terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO, Us uarioFacade.DATOS_INCORRECTOS en caso de diferentes errores.

- **altaMasivaEmpleadosEnTerminalKreta3**

```
int altaMasivaEmpleadosEnTerminalKreta3(long idTerminal)

Realiza el inserción masiva de todos los empleados dados
de alta en el sistema de control de horario en el terminal
identificado por su id. El resultado de esta operación
asíncrona será enviado por topic. Operación de
empleadoTK3.

Parameters:
      idTerminal - Identificador del terminal donde se
      realiza la inserción masiva.
Returns:
      Devuelve el resultado de la
      operación: ControlHorarioFacade.OPERACION_CORRECTA cu
      ando la petición de captura de huella biométrica se
      ha realizado
      correctamente ControlHorarioFacade.EMPLEADO_TK3_NO_EX
      ISTENTE_EN_PERSISTENCIAfallo en la búsqueda de
      empleadoTK3 durante el proceso (fallo grave ya que
      siempre debiera estar según comprobaciones
      previas). ControlHorarioFacade.NAMING_EXCEPTION o Con
      trolHorarioFacade.RESOURCE_EXCEPTION en caso de
      excepciones en el proceso de inserción en
      terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO,
      UsuarioFacade.TERMINAL_NO_ENCONTRADO_O_CONEXION_NO_ES
      TABLECIDA o UsuarioFacade.TERMINAL_CON_ALGUN_ALTA_PRE
      VIA en caso de diferentes errores.
```

- **bajaMasivaEmpleadosEnTerminalKreta3**

```
int bajaMasivaEmpleadosEnTerminalKreta3(long idTerminal)

Realiza la eliminación masiva de todos los empleados dados
de alta en el sistema de control de horario en el terminal
identificado por su id. El resultado de esta operación
asíncrona será enviado por topic. Operación de
empleadoTK3.

Parameters:
      idTerminal - Identificador del terminal donde se
      realiza la eliminación masiva.
Returns:
      Devuelve el resultado de la
      operación: ControlHorarioFacade.OPERACION_CORRECTA cu
      ando la petición de captura de huella biométrica se
      ha realizado
      correctamente ControlHorarioFacade.EMPLEADO_TK3_NO_EX
      ISTENTE_EN_PERSISTENCIA fallo en la búsqueda de
      empleadoTK3 durante el proceso (fallo grave ya que
      siempre debiera estar según comprobaciones
      previas). ControlHorarioFacade.NAMING_EXCEPTION o Con
      trolHorarioFacade.RESOURCE_EXCEPTION en caso de
      excepciones en el proceso de inserción en
      terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO oUs
      uarioFacade.TERMINAL_NO_ENCONTRADO_O_CONEXION_NO_ESTA
      BLECIDA en caso de diferentes errores.
```

- **verificacionEmpleadoEnTerminalesKreta3**

```
int verificacionEmpleadoEnTerminalesKreta3(long idEmpleado
)

Realiza la verificación de integridad del empleado
identificado por idEmpleado en todos los terminales
configurados en el sistema de control de horario. El
resultado de esta operación asíncrona será enviado por
topic. Operación de verificación empleadoTK3.

Parameters:
      idEmpleado - Identificador del empleado de la
      intranet corporativa del que se desea verificar su
      integridad en el sistema de control de horario.
Returns:
      Devuelve el resultado de la
      operación: ControlHorarioFacade.OPERACION_CORRECTA si
      las peticiones de obtención de empleado a los
      terminales han sido
      correctas, ControlHorarioFacade.EMPLEADO_TK3_NO_EXIST
      ENTE_EN_PERSISTENCIA fallo en la búsqueda de
      empleadoTK3 durante el proceso (fallo grave ya que
      siempre debiera estar según comprobaciones
      previas). ControlHorarioFacade.NAMING_EXCEPTION o Con
      trolHorarioFacade.RESOURCE_EXCEPTION en caso de
      excepciones en el proceso de obtención de empleadoTK3
      en
      terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO en
      caso de error.
```

- **verificacionMasivaEmpleadosEnTerminalesKreta3**

```
int verificacionMasivaEmpleadosEnTerminalesKreta3()

Realiza la verificación masiva de todos los empleados en
el sistema de control de horario (incluso los de baja) en
todos los terminales, comprobando que los empleados de
alta en persistencia están presentes en todos los
terminales configurados con sus datos correctos. Así mismo
comprueba que los empleados de baja no están presentes en
ningún terminal. El resultado de esta operación asíncrona
será enviado por topic. Operación de terminal.

Returns:
      Devuelve el resultado de la
      operación: ControlHorarioFacade.OPERACION_CORRECTA si
      la petición inicial se ha realidado con
      exito. ControlHorarioFacade.EMPLEADO_TK3_NO_EXISTENTE
      _EN_PERSISTENCIA fallo en la búsqueda de empleadoTK3
      durante el proceso (fallo grave ya que siempre
      debiera estar según comprobaciones
      previas). ControlHorarioFacade.NAMING_EXCEPTION o Con
      trolHorarioFacade.RESOURCE_EXCEPTION en caso de
      excepciones en el proceso de subsanación de
      empleadoTK3 en
      terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO en
      caso de error.
```

- **subsanacionEmpleadoEnTerminalesKreta3**

```
int subsanacionEmpleadoEnTerminalesKreta3(long idEmpleado)

Realiza la subsanación de integridad del empleado
identificado por idEmpleado en todos los terminales
configurados en el sistema de control de horario que sea
necesario. El resultado de esta operación asíncrona será
enviado por topic. Operación de verificación empleadoTK3.

Parameters:
      idEmpleado - Identificador del empleado de la
      intranet corporativa del que se desea subsanar su
      integridad en el sistema de control de horario.
Returns:
      Devuelve el resultado de la
      operación: ControlHorarioFacade.OPERACION_CORRECTA si
      las peticiones de corrección de empleado a los
      terminales han sido
      correctas, ControlHorarioFacade.EMPLEADO_TK3_NO_EXIST
      ENTE_EN_PERSISTENCIA fallo en la búsqueda de
      empleadoTK3 durante el proceso (fallo grave ya que
      siempre debiera estar según comprobaciones
      previas). ControlHorarioFacade.NAMING_EXCEPTION o Con
      trolHorarioFacade.RESOURCE_EXCEPTION en caso de
      excepciones en el proceso de obtención de empleadoTK3
      en
      terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO en
      caso de error
```

- **getNumeroMarcajesEnTerminalKreta3**

```
int getNumeroMarcajesEnTerminalKreta3(long idTerminal)

Realiza la petición del número de marcajes almacenados al
terminal identificado por su id. El resultado de esta
operación asíncrona será enviado por topic. Operación de
marcaje.

Parameters:
      idTerminal - Identificador del terminal donde se
      realiza la operación.
Returns:
      Devuelve el resultado de la
      operación: ControlHorarioFacade.OPERACION_CORRECTA si
      la petición de numero de marcajes en terminal ha sido
      correcta, ControlHorarioFacade.NAMING_EXCEPTION o Con
      trolHorarioFacade.RESOURCE_EXCEPTION en caso de
      excepciones en el proceso de obtención de empleadoTK3
      en
      terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO o U
      suarioFacade.TERMINAL_NO_ENCONTRADO_O_CONEXION_NO_EST
      ABLECIDA en caso de diferentes errores.
```

- **eliminarMarcajesEnTerminalKreta3**

```
int eliminarMarcajesEnTerminalKreta3(long idTerminal)
```

Realiza la petición de eliminar marcajes almacenados en el terminal identificado por su id. El resultado de esta operación asíncrona será enviado por topic. Operación de marcaje.

Parameters:
idTerminal - Identificador del terminal donde se realiza la operación.
Returns:
Devuelve el resultado de la operacion: ControlHorarioFacade.OPERACION_CORRECTA si las peticiones de corrección de empleado a los terminales han sido correctas, ControlHorarioFacade.NAMING_EXCEPTION o ControlHorarioFacade.RESOURCE_EXCEPTION en caso de excepciones en el proceso de obtención de empleadoTK3 en terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO o UsuarioFacade.TERMINAL_NO_ENCONTRADO_O_CONEXION_NO_ESTABLECIDA en caso de diferentes errores.

- **sincronizarMarcajesConTerminalKreta3**

int sincronizarMarcajesConTerminalKreta3(long idTerminal)

Realiza la petición de sincronizar marcajes almacenados en el terminal identificado por su id. La sincronización provocara el borrado de los marcajes del terminal. El resultado de esta operación asíncrona será enviado por topic. Operación de marcaje.

Parameters:
idTerminal - Identificador del terminal donde se realiza la operación.
Returns:
Devuelve el resultado de la operación: ControlHorarioFacade.OPERACION_CORRECTA si las peticiones de corrección de empleado a los terminales han sido correctas, ControlHorarioFacade.NAMING_EXCEPTION o ControlHorarioFacade.RESOURCE_EXCEPTION en caso de excepciones en el proceso de obtención de empleadoTK3 en terminal. UsuarioFacade.EXISTE_TERMINAL_BLOQUEADO o UsuarioFacade.TERMINAL_NO_ENCONTRADO_O_CONEXION_NO_ESTABLECIDA en caso de diferentes errores.

- **getEmpleados**

java.util.Collection<PojoEmpleado> getEmpleados()

Obtiene y devuelve información relativa al empleado identificado por su nombre de usuario Operación de empleado. Comunicación solo CORE: Síncrono

Parameters:
userName - Nombre de usuario del empleado que se pretende obtener.
Returns:

PojoEmpleado Pojo con información del empleado buscado.

- **getEmpleado**

  PojoEmpleado getEmpleado(java.lang.String userName)

  Obtiene y devuelve información relativa al empleado identificado por su nombre de usuario Operación de empleado. Comunicación solo CORE: Síncrono

  Parameters:
  userName - Nombre de usuario del empleado que se pretende obtener.
  Returns:
  PojoEmpleado Pojo con información del empleado buscado.

- **getAllTerminalesKreta3**

  java.util.Collection<PojoTerminalKreta3> getAllTerminalesKreta3()

  Obtiene y devuelve información relativa a los terminales almacenados en el sistema. Operación de terminal. Comunicación solo CORE: Síncrono

  Returns:
  Collection Colección de pojos de terminales.

- **getTerminalKreta3**

  PojoTerminalKreta3 getTerminalKreta3(long idTerminal)

  Obtiene y devuelve información relativa al terminal que corresponde al id idTerminal. Operación de terminal. Comunicación solo CORE: Síncrono

  Parameters:
  idTerminal - Identificador del terminal que se pretende obtener
  Returns:
  PojoTerminalKreta3 Pojo con información del terminal buscado.

- **desbloquearTerminalKreta3**

  int desbloquearTerminalKreta3()

  Desbloquea el terminal que en el momento de ejecutar el método este bloqueado modificando el atributo terminalBloqueado. Operación usualmente utilizada para desbloquear el terminal tras una operación bloqueante que no ha finalizado correctamente. Operación de terminal. Comunicación solo CORE: Síncrono

  Returns:

ControlHorarioFacade.OPERACION_CORRECTA si la
petición se ha realizado con éxito
o ControlHorarioFacade.NINGUN_TERMINAL_BLOQUEADO en
caso de no haber hallado ningún terminal bloqueado.

- **getEmpleadoTerminalKreta3**

PojoEmpleadoTerminalKreta3 getEmpleadoTerminalKreta3(long idEmpleado)

Obtiene y devuelve información relativa al empleadoTK3 relacionado con el empleado cuyo id es idEmpleado sin incluir marcajes del mismo. Operación de empleado. Comunicación solo CORE: Síncrono

Parameters:
idEmpleado - Identificador del empleado relacionado con el empleadoTK3 que se pretende obtener
Returns:
PojoEmpleadoTerminalKreta3 Pojo con información del empleadoTK3 buscado.

- **getEmpleadoTerminalKreta3yMarcajes**

PojoEmpleadoTerminalKreta3 getEmpleadoTerminalKreta3yMarcajes(long idEmpleado)

Obtiene y devuelve información relativa al empleadoTK3 relacionado con el empleado cuyo id es idEmpleado incluido marcajes del mismo.

Parameters:
idEmpleado - Identificador del empleado relacionado con el empleadoTK3 que se pretende obtener
Returns:
PojoEmpleadoTerminalKreta3 Pojo con información del empleado buscado incluido marcajes.

- **getAllEmpleadosTerminalKreta3**

java.util.Collection<PojoEmpleadoTerminalKreta3> getAllEmpleadosTerminalKreta3()

Obtiene y devuelve información relativa a los empleadoTK3 (altas y bajas) almacenados en el sistema.

Returns:
Collection Colección de pojos de empleadoTK3.

- **getAllEmpleadoTerminalKreta3yMarcajes**

java.util.Collection<PojoEmpleadoTerminalKreta3> getAllEmpleadoTerminalKreta3yMarcajes()

Obtiene y devuelve información relativa a los empleadoTK3 (altas y bajas) almacenados en el sistema, incluido marcajes.

Returns:
Collection Colección de pojos de empleadoTK3, cada uno de los cuales contiene información de los marcajes realizados por el empleadoTK3.

- **getEstadisticasEmpleadoTerminalKreta3**

PojoEstadisticasEmpleadoTK3 getEstadisticasEmpleadoTerminalKreta3(long idEmpleado)

Compone y devuelve información estadística relativa al empleadoTK3 relacionado con el empleado cuyo id es idEmpleado. (horas computadas por semana y por mes, número de incidencias por semana y por mes y numero de incidencia por tipo).

Parameters:
idEmpleado - Identificador del empleadoTK3 del que se pretende obtener información estadística de marcajes e incidencias.
Returns:
PojoEstadisticasEmpleadoTK3 Pojo con diferente información estadística de marcajes e incidencias del empleado.

- **getMarcajes**

java.util.Collection<PojoMarcaje> getMarcajes (int codigoPeticion, long idEmpleado)

Obtiene y devuelve información relativa a los marcajes del empleadoTK3 relacionado con el empleado cuyo id es idEmpleado. Solo aquellos que están en vigor, es decir los que no están dados de baja.

Parameters:
codigoPeticion - Especifica un rango temporal para el filtrado de los marcajes a obtener. 0: Retorna los marcajes pertenecientes al dia en curso. 1: Retorna los marcajes pertenecientes a la semana en curso. 2: Retorna los marcajes pertenecientes al mes en curso. 3: Retorna los marcajes pertenecientes al año en curso.
idEmpleado - Identificador del empleado relacionado con el empleadoTK3 del que se pretende obtener marcajes.
Returns:
Collection Colección de pojos de marcajes realizados por el empleadoTK3 cuyo id corresponde a idEmpleadoTK3 que cumplen con la regla de filtrado especificada mediante codigoPeticion.

- **getMarcajesEnIntervaloFechas**

java.util.Collection<PojoMarcaje> getMarcajesEnIntervaloFechas(long idEmpleado, java.util.Date inicioIntervalo, java.util.Date finIntervalo)

Obtiene y devuelve información relativa a los marcajes del empleadoTK3 relacionado con el empleado cuyo id es idEmpleado. Solo aquellos que están en vigor, es decir los que no están dados de baja.

Parameters:
idEmpleado - Identificador del empleado relacionado con el empleadoTK3 del que se pretende obtener marcajes.
inicioIntervalo - Especifica el inicio del intervalo al que deben pertenecer los marcajes para ser retornados.
finIntervalo - Especifica el fin del intervalo al que deben pertenecer los marcajes para ser retornados.
Returns:
Collection Colección de pojos de marcajes realizados por el empleadoTK3 cuyo id corresponde a idEmpleadoTK3 que pertenecen al intervalo especificado.

- **bajaMarcajesEnIntervaloFechas**

java.util.Collection<java.lang.String> bajaMarcajesEnIntervaloFechas(java.util.Date inicioIntervalo, java.util.Date finIntervalo)

Elimina todos aquellos marcajes comprendidos en el intervalo temporal especificado por los parámetros.

Parameters:
inicioIntervalo - Especifica el inicio del intervalo al que deben pertenecer los marcaje para ser dados de baja.
finIntervalo - Especifica el fin del intervalo al que deben pertenecer los marcaje para ser dados de baja.
Returns:
Retorna todos los ids de marcajes dados de baja en la operación.

- **getHistoricosMarcajes**

java.util.Collection<PojoMarcaje> getHistoricosMarcajes(long idEmpleado)

Obtiene y devuelve información relativa a los marcajes de baja del empleadoTK3 relacionado con el empleado cuyo id es idEmpleado.

Parameters:
idEmpleado - Identificador del empleado relacionado con el empleadoTK3 del que se pretende obtener marcajes.
Returns:
Collection Colección de pojos de marcajes realizados por el empleadoTK3 cuyo id corresponde a idEmpleadoTK3 que han sido dado de baja previamente.

- **getIncidencias**

  java.util.Collection<PojoIncidencia> getIncidencias(int codigoPeticion, long idEmpleado)

  Obtiene y devuelve información relativa a las incidencias del empleadoTK3 relacionado con el empleado cuyo id es idEmpleado. Solo aquellas que están en vigor, es decir las que no están dadas de baja.

  Parameters:
  codigoPeticion - Especifica un rango temporal para el filtrado de las incidencias a obtener. 0: Retorna las incidencias pertenecientes al día en curso 1: Retorna las incidencias pertenecientes a la semana en curso 2: Retorna las incidencias pertenecientes al mes en curso 3: Retorna las incidencias pertenecientes al año en curso
  idEmpleado - Identificador del empleado relacionado con el empleadoTK3 del que se pretende obtener incidencias.
  Returns:
  Collection Colección de pojos de incidencias del empleadoTK3 cuyo id corresponde a idEmpleadoTK3 y que cumplen con la regla de filtrado especificada mediante codigoPeticion.

- **getIncidenciasEnIntervaloFechas**

  java.util.Collection<PojoIncidencia> getIncidenciasEnIntervaloFechas(long idEmpleado,java.util.Date inicioIntervalo, java.util.Date finIntervalo)

  Obtiene y devuelve información relativa a las incidencias del empleadoTK3 relacionado con el empleado cuyo id es idEmpleado. Solo aquellas que están en vigor, es decir las que no están dadas de baja.

  Parameters:
  idEmpleadoTK3 - Identificador del empleado relacionado con el empleadoTK3 del que se pretende obtener las incidencias.
  inicioIntervalo - Especifica el inicio del intervalo al que deben pertenecer las incidencias para ser retornadas.
  finIntervalo - Especifica el fin del intervalo al que deben pertencer las incidencias para ser retornados.
  Returns:
  Collection Colección de pojos de incidencias del empleadoTK3 cuyo id corresponde a idEmpleadoTK3 que pertenecen al intervalo especificado.

- **bajaIncidenciasEnIntervaloFechas**

  java.util.Collection<java.lang.String> bajaIncidenciasEnIntervaloFechas(java.util.Date inicioIntervalo, java.util.Date finIntervalo)

Elimina todas aquellas incidencias comprendidas en el intervalo temporal especificado por los parámetros.

Parameters:
inicioIntervalo - Especifica el inicio del intervalo al que deben pertenecer las incidencias para ser dadas de baja.
finIntervalo - Especifica el fin del intervalo al que deben pertenecer las incidencias para ser dadas de baja.
Returns:
Retorna todos los ids de incidencias dadas de baja en la operación.

- **getHistoricosIncidencias**

java.util.Collection<PojoIncidencia> getHistoricosIncidencias(long idEmpleado)

Obtiene y devuelve información relativa a las incidencias de baja del empleadoTK3 que corresponde al id idEmpleado.

Parameters:
idEmpleado - Identificador del empleado relacionado con el empleadoTK3 del que se pretende obtener las incidencias.
Returns:
Collection Colección de pojos de incidencias pertenecientes al empleadoTK3 cuyo id corresponde a idEmpleadoTK3 que han sido dado de baja previamente.

- **getJornadasLaborales**

java.util.Collection<PojoJornadaLaboralXML> getJornadasLaborales()

Obtiene y devuelve información relativa a las jornadas laborales dadas de alta en el sistema.

Returns:
Collection Colección de pojos de jornadas laborales. Cada uno de estos contiene la especificación de la jornada laboral en formato XML. Ejemplo de definición de jornada laboral en
XLM: Enero;Febrero;Marzo;Abril;Mayo;Junio;Septiembre;Octubre;Noviembre;Diciembre 0900;1500 0900;1500 0900;15000900;1500 0900;1500 41 Julio;Agosto 0900;1400 0900;1400 0900;1400 0900;1400 0900;1330 35

- **getJornadaLaboral**

PojoJornadaLaboralXML getJornadaLaboral(java.lang.String idJornadaLaboral)

Obtiene y devuelve información relativa a la jornada laboral identificada por idJornadaLaboral.

Returns:
PojoJornadaLaboralXML Pojo de jornada laboral, que contiene la especificación del al jornada laboral en formato xml.

- **setJornadaLaboral**

PojoJornadaLaboralXML setJornadaLaboral(java.lang.String idJornadaLaboral, java.lang.String jornadaLaboralXML)

Añade una nueva jornada laboral al sistema de control de horarios.

Parameters:
idJornadaLaboral - Identificador de la nueva jornada laboral.
jornadaLaboralXML - Especificación de la nueva jornada laboral.
Returns:
PojoJornadaLaboralXML Pojo de la nueva jornada laboral en caso de que la inserción se haya realizado correctamente, null en caso contrario.

- **modificacionJornadaLaboral**

int modificacionJornadaLaboral(java.lang.String idJornadaLaboral, java.lang.String jornadaLaboralXML)

Modifica una jornada laboral existente en sistema de control de horarios.

Parameters:
idJornadaLaboral - Identificador de la jornada laboral a modificar.
jornadaLaboralXML - Nueva especificación para la jornada laboral.
Returns:
ControlHorarioFacade.OPERACION_CORRECTA en caso de que la modificación haya sido exitosa, ControlHorarioFacade.JORNADA_LABORAL_NO_EXISTE en caso de fallo.

- **eliminacionJornadaLaboral**

int eliminacionJornadaLaboral(java.lang.String idJornadaLaboral)

Elimina una jornada laboral existente en sistema de control de horarios.

Parameters:
idJornadaLaboral - Identificador de la jornada laboral a eliminar.
Returns:
ControlHorarioFacade.OPERACION_CORRECTA en caso de que la eliminación haya sido exitosa, ControlHorarioFacade.JORNADA_LABORAL_NO_EXISTE en caso de fallo.

Apéndice III. Especificaciones Terminal Biométrico Kreta3

Dimensiones:	Terminal en caja Plástico: 160 mm x 220 mm x 65 mm; Peso 700g.
Alimentación:	Tensión: 5 VDC ± 10%
	Corriente (sólo electrónica): 600mA máx.
	Corriente (conjunto en caja): 1A típico
Contactos relé:	4 o 6 contactos relé, normalmente abiertos, 24V / 1ª
Display:	Display LCD, 20x2 caracteres
Teclado:	Teclado matricial de 4 filas x 4 columnas
Entradas digitales:	4 entradas digitales tipo relé. En circuito abierto (contacto abierto) su valor lógico será 0. En contacto a masa (contacto cerrado), valor lógico 1.
Buzzer:	1 Buzzer Interno (actúa junto con lector principal y teclado)
Entradas Clock & Data:	
	Cada módulo UI permite la conexión de un dispositivo que transmita en Clock&Data con codificación ABA, Track 2.
Interfaz RS232:	Hasta 3 puertos serie configurables: comunicación con el Host o con lectores de tarjeta. Baud Rate: 9600, 19200 ó 38400. Otros parámetros: n,8,1.
Salida para la alimentación del lector externo:	
	Kreta3 proporciona 5Vdc para la alimentación del lector auxiliar con una capacidad de corriente de 200mA.
Conexión Ethernet:	Protocolos: TCP/IP, UDP, ARP, DHCP, ICMP (ping)
	Velocidad: 10/100 BaseT
	Conector: RJ-45
Reloj en Tiempo Real	
Microcontrolador:	Doble (DB + UI), arquitectura RISC de 32 bits
Frecuencia de reloj:	60 MHz en electrónica DB, 48 MHz en UI
Memoria:	1024 kB, SRAM, no volátil (con pila de litio, CR2032)

Apéndice IV. Contenido CD-Rom

A continuación mostramos el contenido del CD-Rom que se adjunta junto con esta memoria. Para cada uno de las carpetas y subcarpetas se incluye una breve descripción:

- Proyectos: Incluye los diferentes proyectos software para diferentes entornos de programación que conforman la intranet corporativa de Edosoft: los cuales contienen el código fuentes y JavaDocs de las partes más significativas del código los proyectos Java. Compuesto por los siguientes proyectos:
    - ControlHorario: Conforma la implementación de la GUI y corresponde a un proyecto Flex que puede ser editado y compilado con la herramienta Adobe Flex Builder 3. El contenido del mismo está explicado en el capítulo 8.1 Implementación del cliente (GUI).
    - IntranetEF: Conforma la implementación de la parte servidora de la aplicación que mantiene la lógica de negocio y corresponde a un proyecto Java Empresarial que puede ser editado y compilado con la herramienta Eclipse. El contenido del mismo está explicado en el capítulo 8.2.1 Implementación de la lógica de negocio.
    - Kreta3TerminalConnector: Conforma la implementación del conector y corresponde a un proyecto Java que puede ser editado y compilado con la herramienta Eclipse. El contenido del mismo está explicado en el capítulo 8.2.2 Implementación del conector.
    - Scheduler: Conforma la implementación de los planificadores y corresponde a un proyecto Java que puede ser editado y compilado con la herramienta Eclipse. El contenido del mismo está explicado en el capítulo 8.2.3 Implementación de los planificadores.
    - TestIntranetEF: Constituye la implementación de las pruebas para la aplicación y corresponde a un proyecto Java que puede ser editado y compilado con la herramienta Eclipse. El contenido del mismo está explicado en el capítulo 9.2.3 Implementación.

Tal y como se ha comentado con anterioridad el código entregado contiene tan solo el código fuente desarrollado para este proyecto de fin de carrera, y el propio de la ICEF que es estrictamente necesarios para la compilación y su correcta ejecución, modificado incluso para la eliminación de la referencias a

código suprimido. Se han eliminado por tanto todos los módulos de la ICEF desarrollados por Edosoft, permaneciendo únicamente el módulo de control y gestión de horarios.

- Memoria de proyecto: Incluye esta memoria y las diferentes capturas, figuras y diagramas utilizadas en la misma.
- Miscelánea: Otros elementos de interés:
    - icef_data_demo_memoria.sql: Script para la carga de datos iniciales de ejemplo en la base de datos. Como es lógico los datos de carácter personal de los usuarios han sido sustituidos por otros ficticios.
    - datos_varios.txt: Contiene información sobre algunos usuarios pertenecientes a los datos iniciales de ejemplo para que sea posible hacer login en la intranet con estos. Así mismo contiene otros datos de interés tales como el nombre y valor de la variable de sistema necesaria para la encriptación y desencriptación de los datos almacenados en la base de datos. Además de la dirección IP, puerto, usuario y contraseña empleada por el backend de la aplicación para el acceso a la base de datos mysql en la versión de desarrollo (servidor de mysql en la misma máquina que el JBoss).
    - jboss-4.2.2.GA-icef: Servidor de aplicaciones JBoss con las librerías necesarias para la compilación y ejecución de todos los proyectos que corren en éste incluidas. Así mismo contiene un despliegue de desarrollo en "`\jboss-4.2.2.GA-icef\server\default\deploy`" de todos elementos ear, jar, rar, war, etc. necesarios para la ejecución de la intranet.

Como es lógico sin contar con terminal biométrico Kreta 3 de Kimaldi el funcionamiento del software entregado es limitado. No podrá ejecutarse la ICEF del modo para el cual ha sido ideada, conectada al terminal biométrico, aunque mediante la configuración y el arranque del simulador se podrán ejecutar las Suits automáticas tal y como indica el capítulo 9.2.4 Configuración y ejecución. Incluso mediante el empleo de este simulador se podrá acceder al módulo de control y gestión horaria en la ICEF donde visualizar los terminales conectados, simulados en este caso, y realizar alguna operación básica soportada por el simulador, por ejemplo, altas en sistema de control y gestión horaria de alguno de los usuarios incluidos en los datos iniciales de ejemplo.