UNIVERSIDAD DE ZARAGOZA

Centro Politécnico Superior

Ingeniería Informática

ANEXOS

Proyecto Fin de Carrera

DESARROLLO DE UNA APLICACIÓN PARA LOS OPERADORES DE TELEASISTENCIA INTEGRADA EN TELEFONÍA IP

AUTORA: LAURA LACARRA ARCOS

DIRECTOR: ROBERTO CASAS MILLÁN

PONENTE: JESÚS ALASTRUEY BENEDÉ

Departamento Informática e Ingeniería de Sistemas

Zaragoza, Julio 2010

A mi familia, amigos y
compañeros de trabajo

TABLA DE CONTENIDOS

Lista de figuras

Lista de Tablas

UNIVERSIDAD DE ZARAGOZA

Centro Politécnico Superior

Ingeniería Informática

ANEXO A: PLATAFORMA IP

[1/5]

PFC: DESARROLLO DE UNA APLICACIÓN PARA LOS OPERADORES DE TELEASISTENCIA INTEGRADA EN TELEFONÍA IP

AUTORA: LAURA LACARRA ARCOS

DIRECTOR: ROBERTO CASAS MILLÁN

PONENTE: JESÚS ALASTRUEY BENEDÉ

Departamento Informática e Ingeniería de Sistemas

Zaragoza, Julio 2010

ANEXO A PLATAFORMA IP

A.1 Introducción

La plataforma IP diseñada por el equipo de sistemas de la empresa Diaple tiene dos fases:

- Primera: basada en la instalación de la infraestructura de VozIP [21] junto con un servidor local en el centro de teleasistencia.
- Segunda: basada en un pequeño servidor local en el centro de teleasistencia y dar servicio Voz IP a través de la línea de Internet.

La inversión económica para la primera fase es superior a la segunda ya que se necesita más tecnología.

La segunda fase está basada en la definición de SaaS. Se propone dar servicio IP como parte del producto. La instalación que tendrá la empresa sería de magnitud más pequeña. No obstante esta fase presenta necesita tener infraestructura o servicio para:

- Garantizar el servicio en caso de caída de Internet: la voz llega por Internet, es imprescindible tener el servicio.
- Garantizar que una llamada de emergencia, es realizada en el mismo municipio que el centro de teleasistencia: al ofrecer voz IP como servicio, si el servidor está instalado en otro municipio, las llamadas de emergencia '112' son direccionadas al municipio del servidor.

A continuación se explican ambas fases y algunas de las alternativas. La plataforma dependerá en gran medida:

- Al número de operadores estimados por el centro.
- Al número de líneas para operadores que opte como backup.

A.2 Primera Fase

A.2.1 Primera solución

En la primera fase se ha instalado una plataforma IP junto con un servidor. La salida que aporta la plataforma es una línea de datos a cada puesto de trabajo de los operadores y su función es integrar las llamadas telefónicas, aviso de alertas y conexiones con la base de datos.

La entrada de la plataforma es la línea RTB (red de telefonía básica). Ésta va conectada a un dispositivo, pasarela, encargado de interconectar las dos arquitecturas de red y por lo tanto traducirlo al protocolo apropiado para VozIP. De ahí pasa a la centralita IP que se comunicará con la base de datos y una cola de eventos para mandar alarmas a los operadores.

La arquitectura se muestra en la Figura A-1. Se observa dos instalaciones de clientes, con diferente protocolo que están conectados a la empresa de teleasistencia, por medio de la RTB. La plataforma IP dispone del servidor configurado para el funcionamiento del sistema AsisT: la Centralita IP, la cola de mensajes y la base de datos. El protocolo IP con el que se señalizan los dispositivos de la red local es SIP.

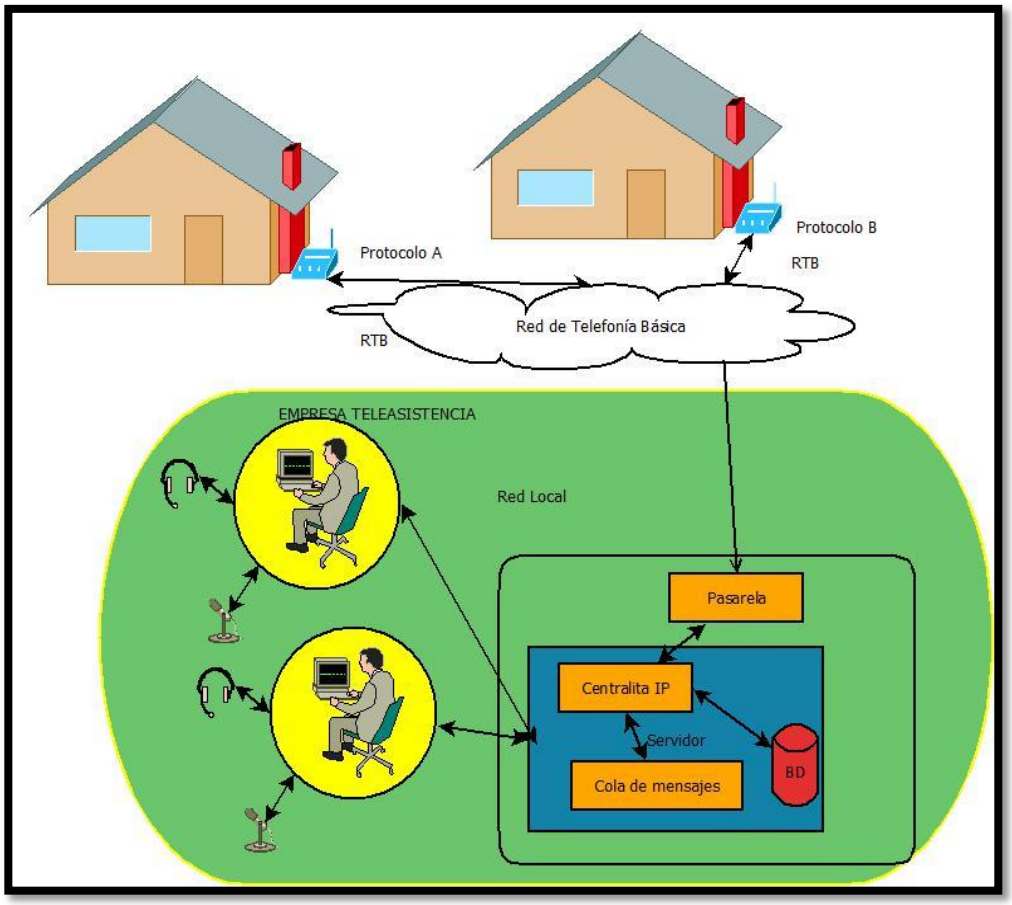


Figura A-1 Primera fase

El precio de la pasarela o *gateway* varía con el número de conexiones que permita. Por ejemplo, si la empresa dispone de 23 operadores, una opción es un *gateway* primario que permita 30 estaciones de teléfono IP.

Una vez conectado la RTB con SIP pasa a la centralita IP. Su funcionalidad es:

- Decodificar las llamadas entrantes identificando el terminal, el cliente o instalación y el tipo de alarma.
- Registrar en la base de datos los datos que reciba de la alarma y/o llamada.
- Publicar un mensaje en la cola de mensajería de los datos de la alarma nueva.
- Establecer contacto con la aplicación para gestionar la entrada y salida de llamadas.

Las ventajas que tiene esta plataforma son:

- Multiprotocolo: la nueva plataforma ahorra los costes de hardware necesarios para traducir protocolos de distintos fabricantes.
- Centralita más barata: la centralita IP es mucho más barata que una centralita de teleasistencia.
- Integración: la propia centralita está interactuando con la base de datos y la cola de mensajes.
- Escalabilidad:
    - Crecer a la empresa en cuanto al número de operadores ya que depende del servidor, no del hardware de una centralita.
    - Crecer en el número de clientes que tenga dados de alta: al ser multiprotocolo por software, no hay limitación por hardware en número de traducciones simultáneas entrantes a la centralita.

- Sin problemas de caída de Internet: la voz IP es obtenida en la propia instalación del centro, por lo tanto, todas las llamadas tanto entrantes como salientes se obtienen directamente por la RTB y no por Internet.
- Sin problemas respecto a las llamadas de emergencia: al estar la centralita IP en el centro, evita un mal direccionamiento de las llamadas de emergencia.
- Evolución con las tecnologías: la tecnología IP está en auge y expansión.

A.3 Segunda Fase

A.3.1 Subcontratar el servicio IP

Los costes de VozIP pueden subcontratarse a una empresa especializada en ello de forma que ahorre los costes que supondría una pasarela o *gateway* de gran envergadura. La central de teleasistencia, contaría con el servicio IP sin necesidad de la infraestructura de la primera fase.

Esta instalación también contaría con una pasarela de magnitud y coste más pequeño. Su funcionalidad es de seguridad ante fallos de conexión a Internet, además de dar salida a las llamadas de emergencia.

Como puede verse en la Figura A-2 , la voz IP va conectada a la centralita IP, que se encargará, de la misma manera, de la gestión de llamadas.

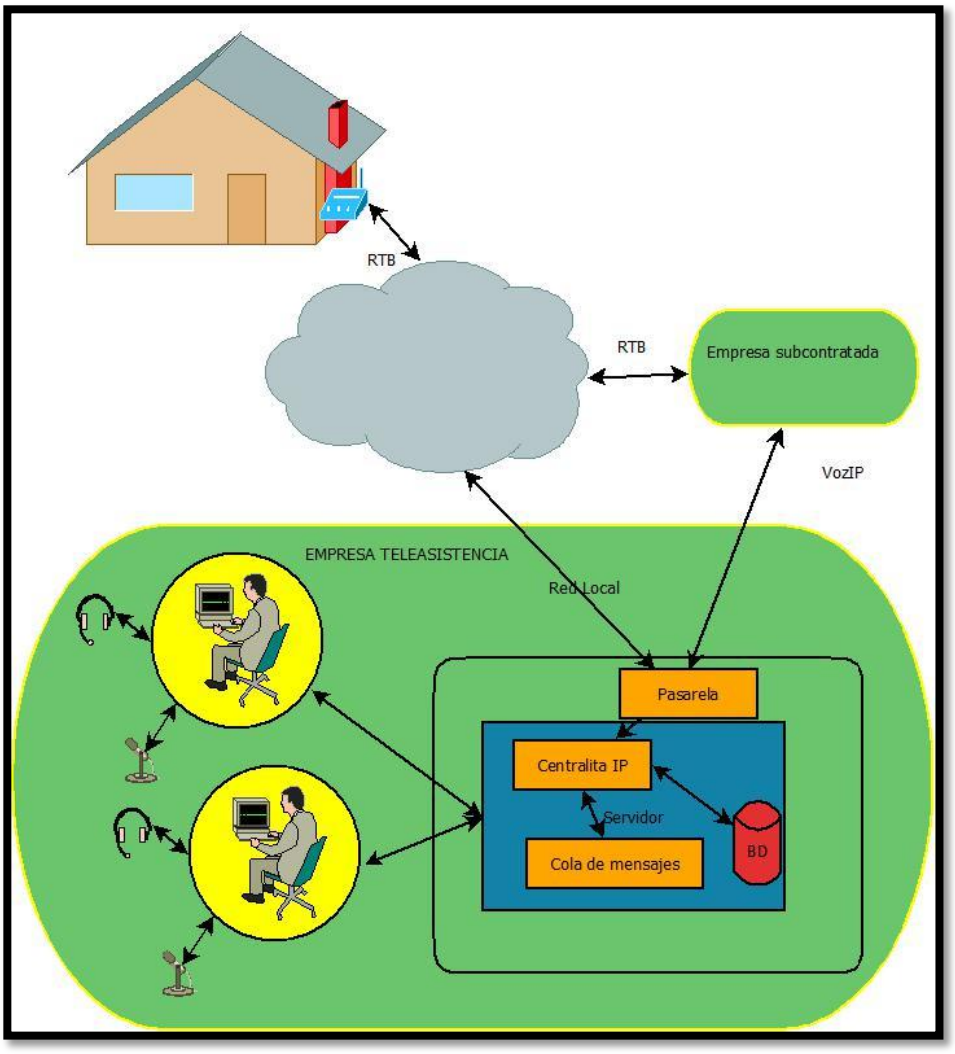


Figura A-2 Subcontratar servicio IP

Las ventajas que se obtienen con esta plataforma:

- Ventajas de VozIP: todas las ventajas ya comentadas respecto a tener una centralita IP (multiprotocolo, económico, mejor escalabilidad, etc).
- Económico: es más económico que la primera fase. Los costes de inversión, también son más baratos, ya que no es necesario tener una gran infraestructura inicial.

Los problemas que supone la VozIP, comentados en la introducción, deben contratarse con la empresa que ofrece el servicio para garantizar el servicio continuo a través de Internet.

A.3.2 Software As A Service (SAAS)

Esta solución encaja con la idea de "SAAS". En este caso, el servicio que se ofrece a la empresa contratante es de:

- Servicio de telefonía IP a la empresa.
- Servidor pequeño instalado en la empresa de teleasistencia para las llamadas de emergencia y en caso de caída de Internet.
- Servidor externo a la empresa donde funcione toda la lógica de teleasistencia.
- Mantenimiento y servicio de la plataforma completa.

La Figura A-3 representa las características de la arquitectura.

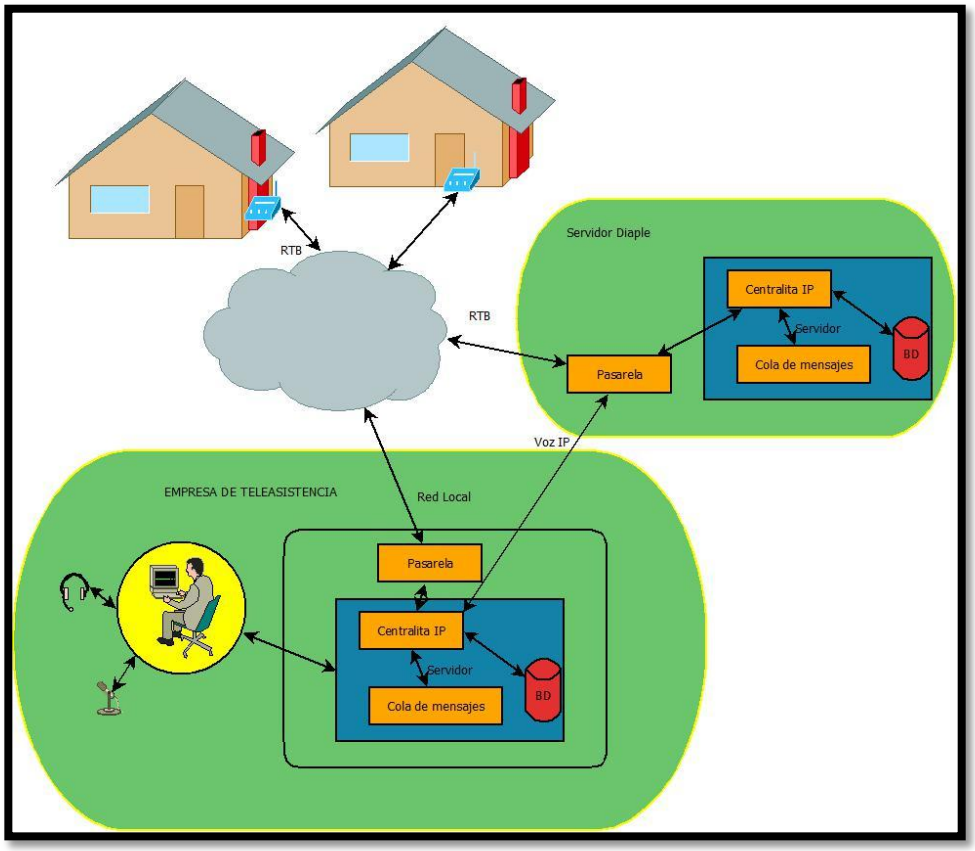


Figura A-3 Solucion Software as a service

Esta plataforma puede modificarse en la medida de las necesidades de las empresas. Cada empresa, tendrá diferente política económica para la instalación. Se recomienda:

- Tener dos líneas de Internet: se configuran en el sistema en modo pasivo-activo garantizando la conexión a Internet.
-

Además de las ventajas mencionadas en las anteriores fases, esta solución destaca por:

- Ahorro en costes instalación: el servidor, se encuentra ajeno a la empresa. Éste pertenece a la empresa Diaple Networking, por lo tanto, no es necesario subcontratar el servicio con otra empresa o la compra de un *gateway* como en la primera fase.
- Gran escalabilidad: debido a que la gestión de llamadas es competencia del sistema que da el servicio, en caso de aumentar el número de clientes de teleasistencia o el número de operadores por centro, pueden crecer a petición de la empresa de teleasistencia.
- Mantenimiento: garantizar el servicio prestando asistencia 24h.

UNIVERSIDAD DE ZARAGOZA

Centro Politécnico Superior

Ingeniería Informática

ANEXO B: ANÁLISIS DEL SISTEMA DE INFORMACIÓN

[2/5]

PFC: DESARROLLO DE UNA APLICACIÓN PARA LOS OPERADORES DE TELEASISTENCIA INTEGRADA EN TELEFONÍA IP

AUTORA: LAURA LACARRA ARCOS

DIRECTOR: ROBERTO CASAS MILLÁN

PONENTE: JESÚS ALASTRUEY BENEDÉ

Departamento Informática e Ingeniería de Sistemas

Zaragoza, Julio 2010

ANEXO B ANÁLISIS DEL SISTEMA DE INFORMACIÓN

B.1 Introducción

Todo sistema requiere de un análisis. Para la realización del análisis del sistema AsisT se ha realizado una interpretación libre basada en las especificaciones de Métrica 3 [30]. Los objetivos que se persiguen son:

- Obtener una especificación detallada de lo que realmente se quiere hacer.
- Dividir el sistema en subsistemas.
- Estructurar la información que se dispone tanto en la base de datos como en el diagrama de clases.
- Obtener el comportamiento de los objetos ante eventos.
- Localizar patrones de diseño.
- Realizar un prototipo de la aplicación.

Dada la envergadura del sistema, se ha simplificado en algunos casos los diagramas para mejorar la legibilidad.

Las actividades han sido elegidas con el criterio de aportar nueva información al sistema. En ocasiones, se ha permutado el orden de las actividades o se han fusionado varias.

El orden de las actividades, es el estricto en el que se ha desarrollado. A continuación se detalla cada actividad.

B.2 Actividad ASI1: definición del subsistema

B.2.1 Información disponible

La fuente de los requisitos proviene del convenio-marco ofrecido por las organizaciones IMSERSO [3] y FEMP [4].

Dicho documento [5] contiene las especificaciones del sistema completo: hardware y software, por lo tanto, el catálogo de requisitos, ofrece las necesidades básicas que el sistema del software necesita, teniendo en cuenta la integración hardware especificada.

Los requisitos también contemplan deficiencias del software actual. La empresa Diaple NetWorking S.L, ha tenido acceso a pruebas de un software de operadores, las cuales se han tenido en cuenta en los requisitos.

B.2.2 Definición y alcance del sistema

Para completar esta información, se recomienda leer el dic**c**ionario de datos, véase GLOSARIO

Se desea desarrollar una aplicación para los operadores de una empresa de teleasistencia. La empresa, dispone de varios centros, ubicados en diferentes emplazamientos, que dan servicio a distintas zonas. Según la organización interna elegida por la empresa pueden ser: distritos, barrios, códigos postales o municipios.

La aplicación que se instale en cada puesto de trabajo de los operadores, necesita gestionar:

- Información de usuarios de la propia aplicación.

- Información de clientes: información de asistencias sanitarias, hospitales, médicos, características personales, personas de su entorno a recurrir, instalación, historiales de alarmas y llamadas.

El sistema debe ser capaz de avisar de la llegada de una alarma en el centro de asistencia. Solo un operador podrá capturarla y gestionarla indicando una o varias actuaciones. Las actuaciones, como por ejemplo: "llamar a usuario" o "llamar a vecino", quedarán registradas tanto las llamadas, como los datos de las mismas (fechas, operador que la atiende, etc).

Los operadores podrán ver todos los clientes y acceder a un expediente con su información: vivienda, instalación, personas a recurrir en caso de emergencia, características, estados dentro del servicio (ausencia, baja, activo), asistencia sanitaria, según las especificaciones del IMSERSO [3].

El sistema debe estar integrado en un entorno por VozIP. Las llamadas se realizarán desde la propia aplicación. Con solo pulsar un botón se iniciará una llamada, así como aceptar llamadas externas.

B.3 Actividad ASI 2: catálogo de requisitos

El catálogo de requisitos en un documento que se ha revisado tras cada actividad realizada en el análisis. Lo que se ha hecho es analizar el convenio-marco [5] y establecer las bases para una primera versión del programa, de forma que sea completa y a la vez que cubra la funcionalidad esencial para la atención personalizada de alarmas.

Para estructurarlos, se han distinguido entre funcionales (explican qué hace la aplicación), seguridad y no funcionales.

Los requisitos funcionales, se han subdivido en distintos bloques debido a la información en común.

B.3.1 Requisitos funcionales

- Gestión de los usuarios de la aplicación: la aplicación debe poder dar de alta/baja/modificación de los usuarios de la aplicación.
- Gestionar el expediente cliente de un cliente
    - Información personal
        - Crear/ modificar /eliminar / mostrar los datos de un cliente.
        - CLIENTE={nombre, apellidos, D.N.I, sexo, fecha de nacimiento, domicilio, teléfono, número de Expediente, código de usuario (CIU), tipo de usuario, tipología de usuario}
        - TIPO_USUARIO = {usuario principal, titular del servicio, usuario con UCR propia, usuario sin UCR}
        - TIPOLOGÍA= {Personas mayores de 65 años, discapacitados físicos, discapacitados psíquicos, discapacitados sensoriales}
    - Viviendas
        - Crear / modificar /eliminar / mostrar viviendas de un cliente.
        - Crear una instalación para un único cliente a una única vivienda.
        - Modificar la vivienda en la que está la instalación.
        - VIVIENDA={dirección, tipo de vivienda, modo de acceso, empresas que aportan servicio a la vivienda, teléfonos}
        - TIPO_VIVIENDA={edificio de vecinos, unifamiliar urbana , unifamiliar aislada}
    - EMPRESA_SERVICIOS={nombre, teléfonos}

        - Mostrar / asociar / desasociar empresas de servicios del hogar de los domicilios de los clientes.
    - Características
        - Crear / modificar /eliminar / mostrar características de una persona según el tipo
        - TIPO_CARACTERISTICA = {riesgo, físicas, psicológicas, sensoriales}
    - Personas a recurrir en caso de emergencia:
        - Crear / modificar /eliminar / mostrar los datos de una persona de convivencia de un cliente.
        - PERSONA_CONVIVENCIA={nombre, apellidos, fecha de nacimiento, parentesco o relación, observaciones}
        - Crear / modificar / eliminar / mostrar los datos de una persona a recurrir en caso de emergencia de un cliente
        - PERSONA_A_RECURRIR={nombre, apellidos, relación, direcciones, teléfonos, disponibilidad, tiene o no llaves}
    - Asistencia sanitaria
        - Mostrar / asociar / desasociar datos de una asistencia sanitaria que tiene asociado un cliente
    - Médicos
        - Mostrar/ asociar / desasociar los datos de un médico que tiene asociado un cliente para una asistencia sanitaria
        - MEDICO = {Nombre, apellidos, teléfonos, disponibilidad}
    - Hospitales
        - Mostrar asociar / desasociar los datos de un hospital que tiene asociado un cliente para una asistencia sanitaria
        - HOSPITAL = {nombre, dirección, teléfonos}
    - Instalación / Equipos
        - Mostrar / asociar / desasociar equipos a una instalación de un cliente
    - Llamadas
        - Registrar / mostrar llamadas entrantes y salientes beneficiarias del cliente
    - Estados
        - Registrar / mostrar la situación de un usuario
        - Registrar / mostrar el motivo de la situación de un cliente
- Atender alarmas
    - En tiempo real, recibir un aviso de alarma y poder aceptarla:
        - En caso de que otro usuario la acepte, la llamada debe quedar inhabilitada.
    - Mientras que atiende una alarma, no puede recibir más avisos, sólo puede atender una alarma.
    - Al atender una alarma, automáticamente el operador tendrá a su disposición de forma simultánea:
        - Tipo de alarma.
        - Terminal del que procede.
        - Nombre y apellidos del cliente.
        - Acceso al expediente completo.
        - Hora del inicio del alarma.
    - El usuario u operador para llevar a cabo la actuación debe:
        - Completar los datos del tipo de alarma que se ha generado.
        - Elegir una actuación de las disponibles.
        - Realizar las llamadas para el beneficio del cliente a consecuencia de la actuación elegida.

        - Establecer la llamada a dicho numero.
- Llamadas:
    - El operador debe poder llamar sin tener que marcar el número de teléfono.
    - El operador debe poder cortar y liberar la línea telefónica.
    - El sistema no permite que el operador entre en contacto con el domicilio de manera distinta a una llamada convencional, excepto en el caso de alarmas no técnicas.
    - El sistema tendrá capacidad para registrar información sobre llamadas que emita o reciba el centro receptor , con identificación de:
        - Fecha y hora.
        - Línea de entrada.
        - Tiempo de ocupación real de la línea durante la llamada.
        - En el caso de llamadas de alarma entrantes, la identificación del terminal del que procede, el tipo de alarma o llamada, y la actuación seguida por el centro.
        - Tras la gestión de una alarma, el operador del sistema del centro receptor será quién envíe las señales de corte y liberación de la línea.
- Historial de llamadas
    - El operador debe poder ver las llamadas realizas por él mismo con la información relevante.
- Historial de alarmas
    - El operador debe poder ver las alarmas atendidas por él.

B.3.2 Requisitos de seguridad

- Debe cumplir la Ley de protección de datos [12].

B.3.3 Requisitos no funcionales

- Aplicación rápida e intuitiva.
- Rápido acceso en base de datos.
- Multiplataforma.
- Multilenguaje: que soporte varios lenguajes.

B.4 ACTIVIDAD ASI 4: casos de uso

Para el estudio de los subsistemas que se pueden localizar en AsisT se han elaborado unos diagramas que aportan claramente información acerca de la interacción de los usuarios con el sistema.

B.4.1 Casos de uso

Actores

Se ha observado que cada empresa puede tener diferentes perfiles o permisos que deben ser configurables según las propias necesidades. Por ejemplo, una empresa puede permitir que un operador atienda alarmas y manipule información de los clientes mientras otra no.

Se hace mención de usuario a cualquier persona que está en el puesto de trabajo independientemente del rol.

En esta primera versión de AsisT se identifican tres actores.

- Súper-administrador: tiene acceso a toda la funcionalidad de la aplicación.

- Administrador: tiene acceso a la dar de alta / baja o modificar operadores y clientes.
- Operador: ver y administrar la información de los clientes y atender las alarmas.

Validación de un usuario en el sistema

El usuario, cuando inicia la aplicación, la primera acción que llevará a cabo es acceder al sistema con un nombre y contraseña válidos. El sistema determinará a que funcionalidades e información tiene acceso.

El sistema AsisT identifica los perfiles y los permisos y da acceso a dicha funcionalidad. La Figura B-1 Validación de usuario muestra el caso de uso de autentificación de los tres actores.

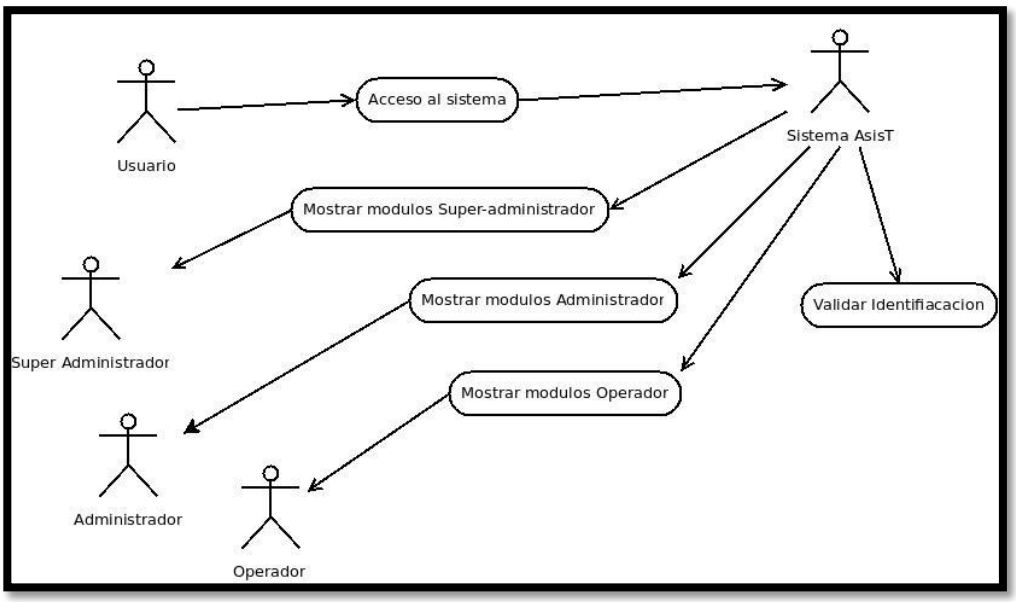


Figura B-1 Validación de usuario

Gestión usuarios y clientes

En la Figura B-2 se observa la funcionalidad del súper-administrador. Tendrá la lógica para crear, modificar y eliminar un operador y la misma funcionalidad para los clientes.

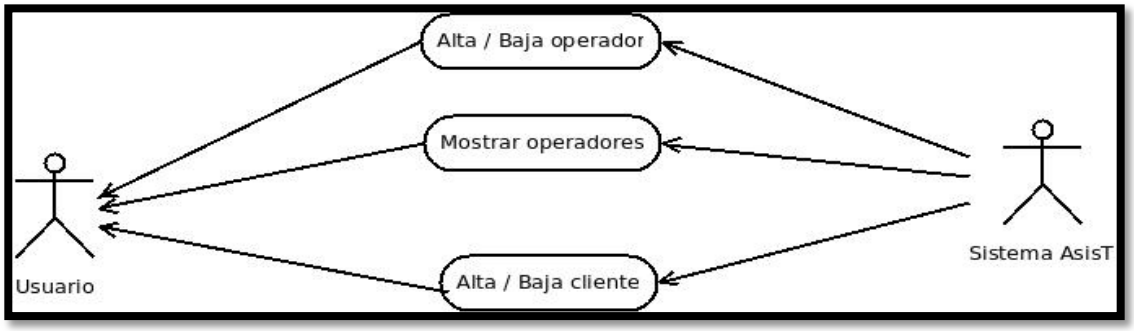


Figura B-2 Gestión usuarios y clientes

Expediente cliente

En la Figura B-3 se muestra la interacción del usuario con la información de los clientes según la documentación [5] del Convenio-marco. Se han distinguido distintos bloques en el expediente.

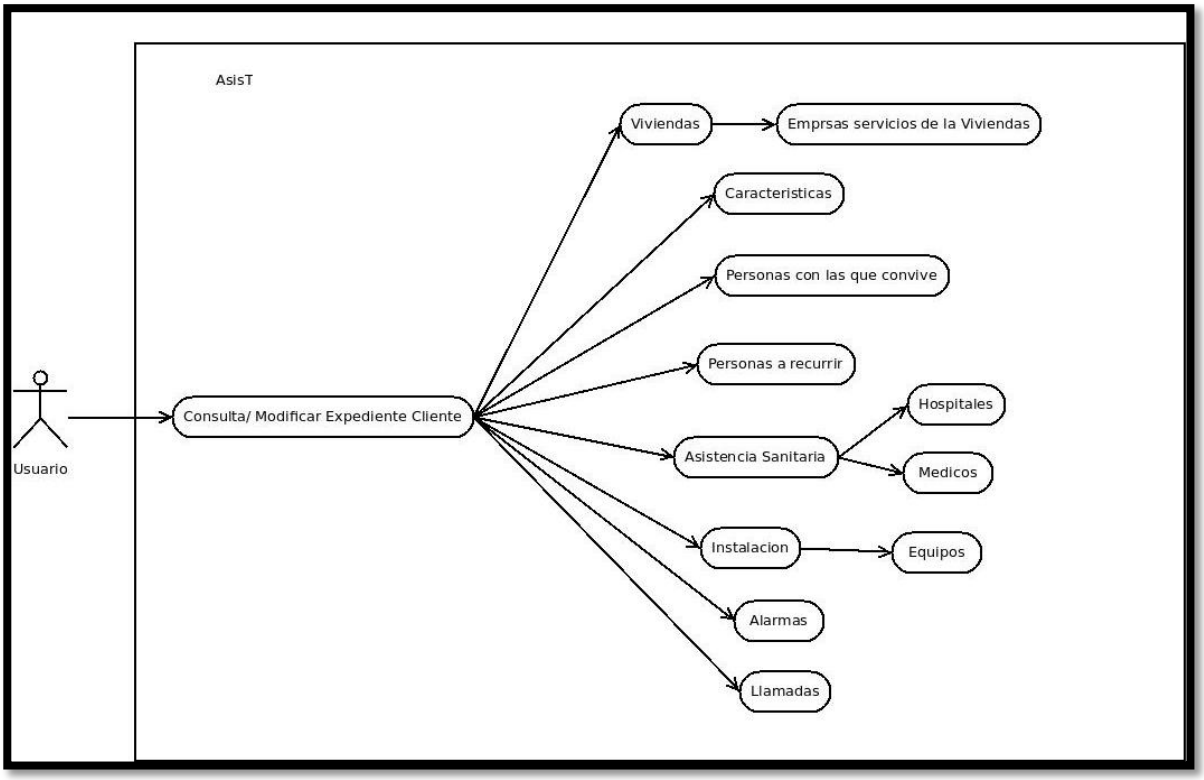


Figura B-3 Gestión del expediente de un cliente

El administrador será capaz de crear y modificar los datos de cada bloque.

Gestión de alarmas

En la siguiente figura se observa cómo el sistema debe avisar al usuario de la llegada de una alarma. Una vez recibida, el usuario inicia la atención y realiza las llamadas pertinentes a través del sistema.

En todo momento se muestra la información disponible y manipulada.

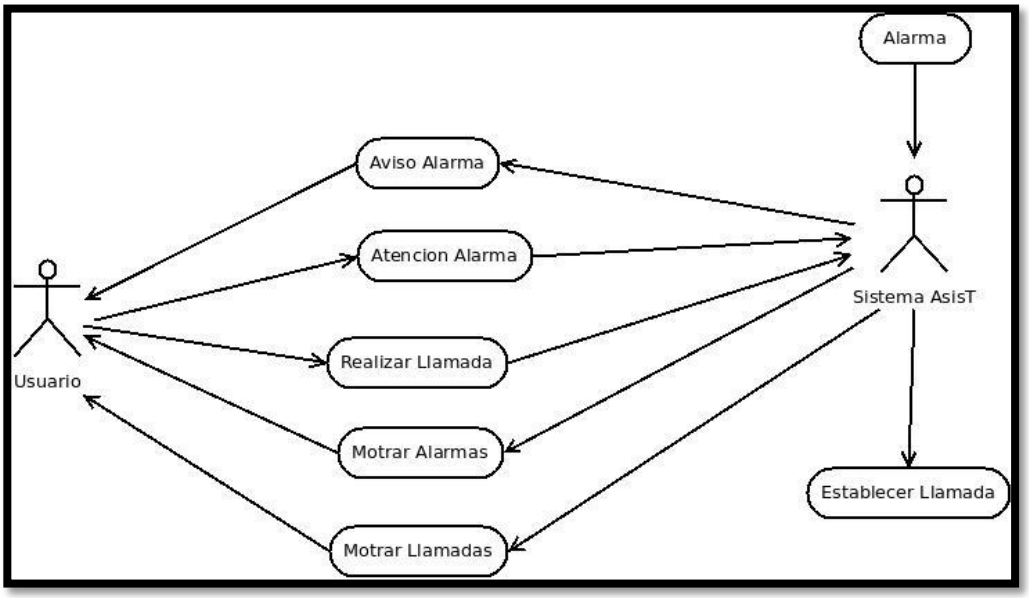


Figura B-4 Gestión de las alarmas

B.5 Actividad ASI 3: identificación de subsistemas

Dada la información obtenida en los requisitos y en los casos de uso, se han identificado tres subsistemas de forma que se aíslen aquellas partes del programa con la misma funcionalidad, mejore el desarrollo y el mantenimiento del sistema.

Los subsistemas localizados en AsisT:

- Subsistemas de bases de datos: parte del sistema que crea e inicializa la información de la base de datos en un gestor apropiado y configurado para el propósito.
- Subsistema de la información: contiene toda la interfaz y gestión de la información.
- Subsistemas de aviso de alarmas: gestiona la llegada de alarmas externas al puesto de los operadores.
- Subsistema teléfono: gestiona recibir y realizar llamadas por medio de un teléfono integrado en la aplicación

El la Figura B-5 se plasma la distribución de los subsistemas y su conexión:

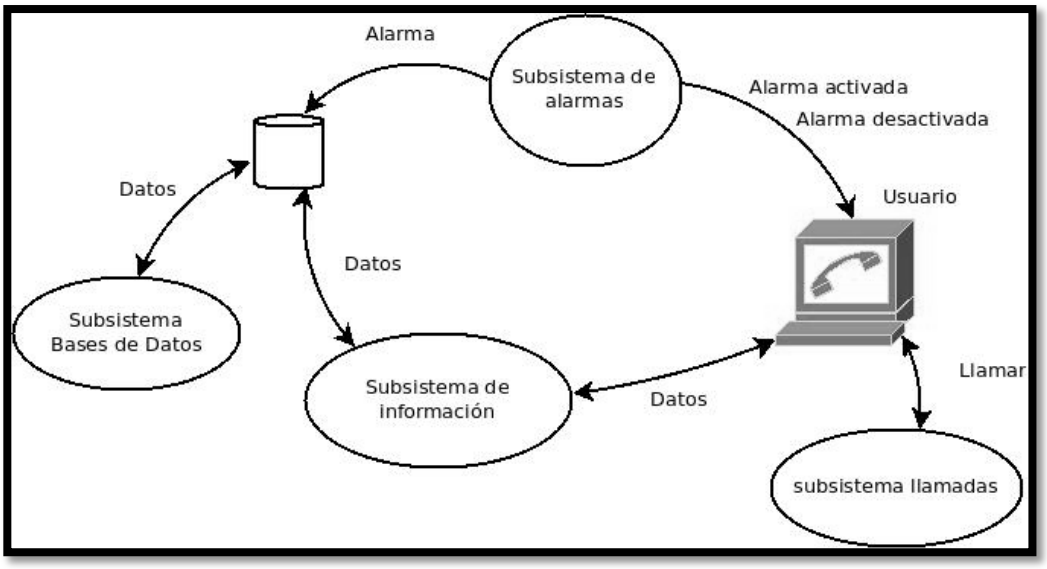


Figura B-5 Subsistemas de AsisT

El subsistema de la información, abarca diferentes módulos. Se ha realizado un Diagrama de flujo de datos hasta nivel 1 para distinguirlos:

La Figura B-6 muestra las entradas y salidas del subsistema de información en el nivel 0.

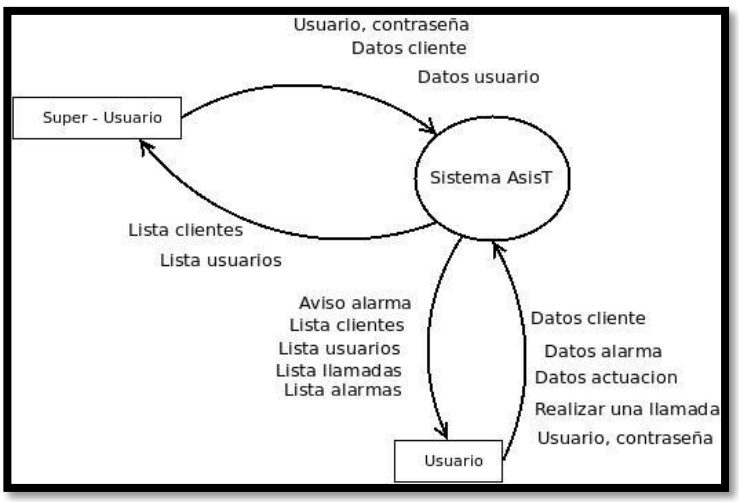


Figura B-6 DFD Nivel 0

La Figura B-7 corresponde al nivel 1 del DFD. El diagrama revela los diferentes módulos contenidos en subsistema de información.

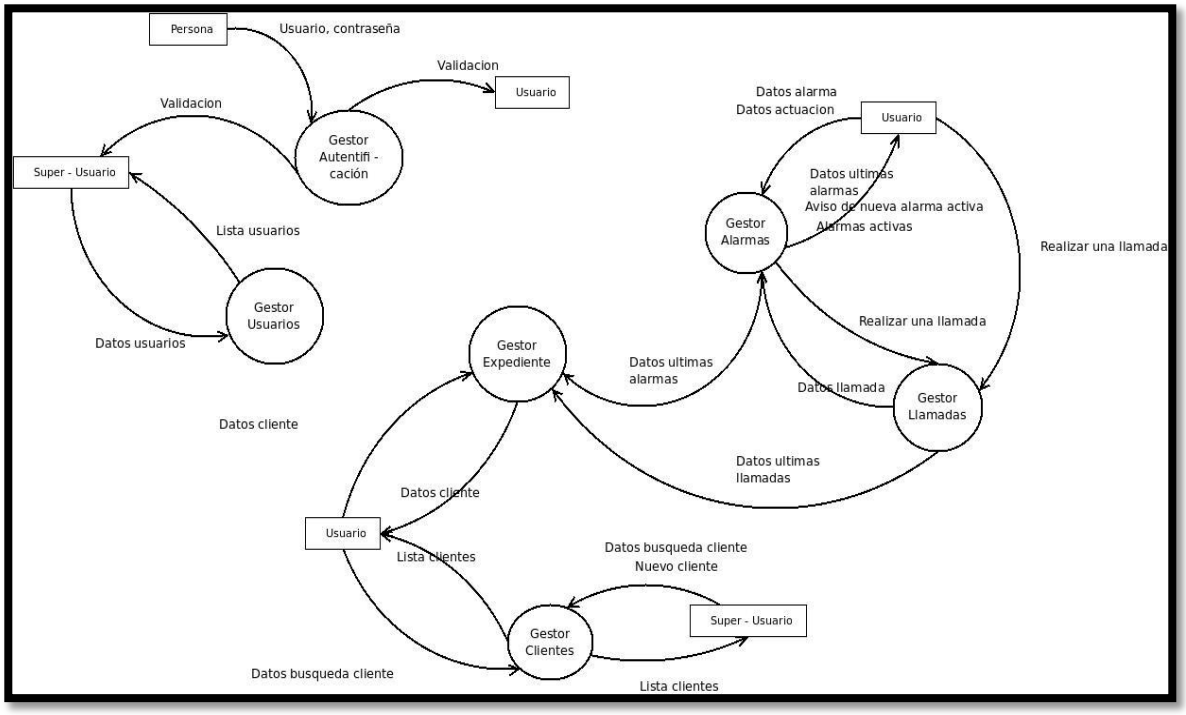


Figura B-7 DFD Nivel 1

La notación empleada es *Yourdon and De Marco* [29]. Los módulos encargados de modificación y alteración de la información son:

- Módulo de acceso a la aplicación: encargado de validar un usuario y cargar los módulos que tiene acceso.
- Módulo de usuarios: dar de alta o modificar los datos de los usuarios de la aplicación
- Módulo de clientes: dar de alta o modificar los datos de los clientes
- Módulo expediente: contiene toda la información de los clientes. Crear / modificar los datos de los clientes
- Módulo gestión alarmas: encargado de la atención y gestión de las alarmas. Integrado con el subsistema de aviso de alarmas y con el subsistema teléfono.
- Módulo historial alarmas: encargado de mostrar las alarmas atendidas o activas por el usuario.
- Módulo historial llamadas: encargado de mostrar las llamadas realizadas o recibidas por el usuario.

B.6 Actividad ASI 6: diseño entidad – relación

Antes del diagrama de clases se ha realizado el diseño de la base de datos. El motivo es la abundante información que contiene la especificación del sistema. El análisis de clases será auto-contenido de la base de datos, y no al revés.

Para mejorar la legibilidad del análisis entidad-relación, se ha dividido en partes. Los diagramas que se adjuntan son la última versión actualizada.

La base de datos está analizada para que sea escalable y abarque posibles alternativas que en posteriores versiones de la aplicación puedan aparecer. Por ejemplo, la especificación de

IMSERSO [5] establece "Alarmas tipo" ya definidas. La base de datos soporta que puedan incorporarse nuevos tipos de datos a petición de la empresa que contrate el servicio.

B.6.1 Empresa

En la Figura B-8, se identifican distintas empresas: empresa de teleasistencia, centro de teleasistencia, servicios hogar y equipos.

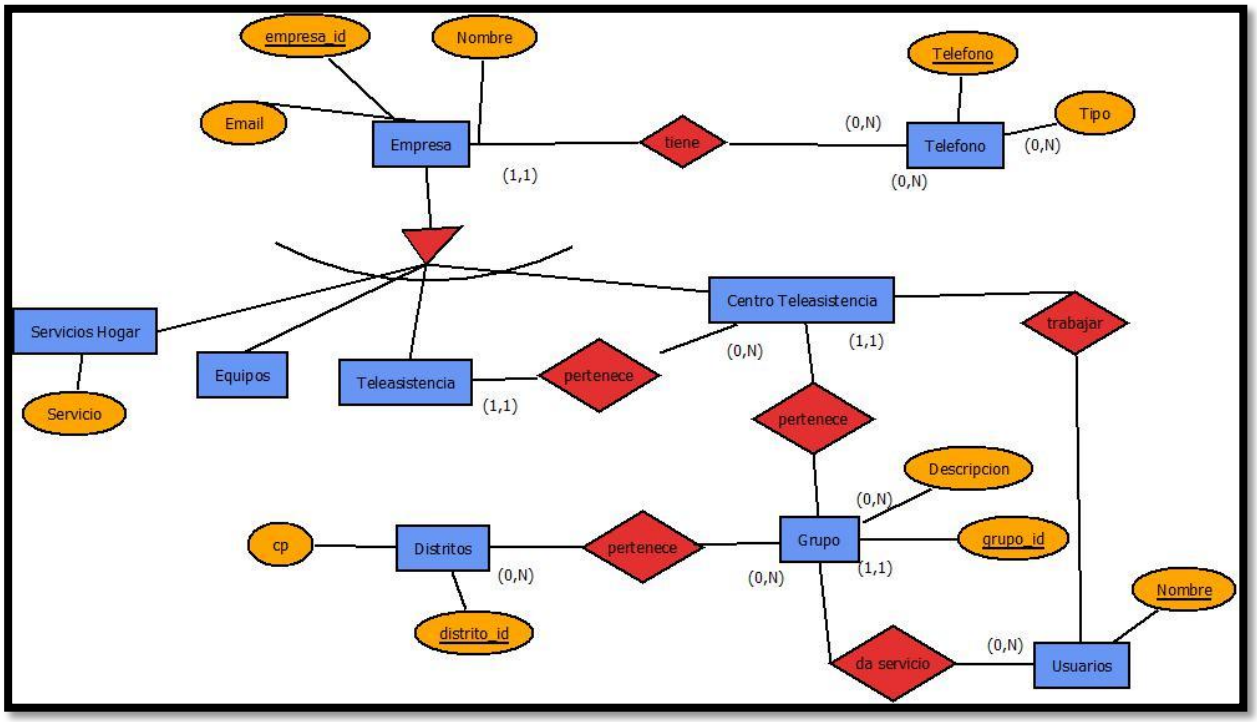


Figura B-8 ER - Empresas

Una de las decisiones que se ha tomado a la hora de plasmar el esquema es que cada empresa tiene una tabla distinta. El motivo ha sido para mantener la consistencia en las claves foráneas y que sean escalables. Posteriormente, en el apartado B.6.4 se plasman la empresa de servicios hogar y en el apartado B.6.5 la empresa de fabricantes de equipos.

Un grupo es una organización dentro de la empresa. Un conjunto de operadores, pertenezca a un grupo, y darán servicio a los códigos postales que tenga dicho grupo. Esta idea se ha pensado para mejorar la organización de una empresa.

empresa_teleasisistencia(empresa_teleasistencia_id: entero;
nombre: cadena(150);
email: cadena(120))

centro(centro_id: entero;
teleasisistencia_id: entero;
nombre: cadena(120);
email: cadena(120))
empresa_teleasisistencia_id clave ajena
empresa_teleasisistencia(empresa_teleasisistencia_id) *no nulo*

grupo (grupo_id: entero;
descripción: cadena(120))

grupo_distrito(grupo_distrito_id: entero;
grupo_id: entero;

distrito_id: entero
grupo_id clave ajena de grupo
distrito_id: calve ajena de distrito);

distrito (distrito_id: entero
cp: cadena(5));

En la Figura B-9, se observa la relación de Usuario de la aplicación para soportar perfiles y permisos a la aplicación.

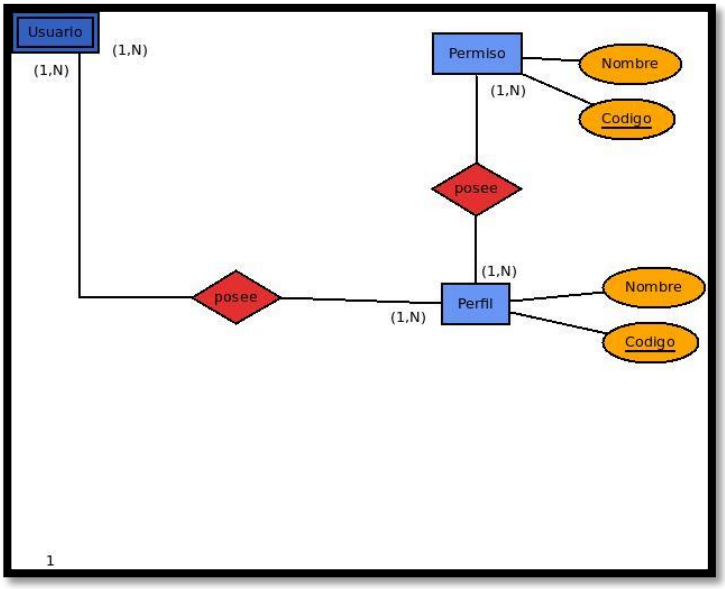


Figura B-9 ER - Usuarios

usuario (<u>usuario_id</u>: entero;
nombre: cadena(50);
apellido1: cadena(50);
apellido2: cadena(50);
perfil_id: entero;
centro_id: entero;
email: cadena(150);)
centro_id clave ajena centro(centro_id) *no nulo*
perfil _id clave ajena perfil (perfil _id) *no nulo*

En la Figura B-10 se plasma el esquema de los organismos y recursos. “Recursos entidad” son servicios de urgencia propios de una entidad, mientras que “Organismos” son servicios de urgencia como ambulancias, bomberos, etc.

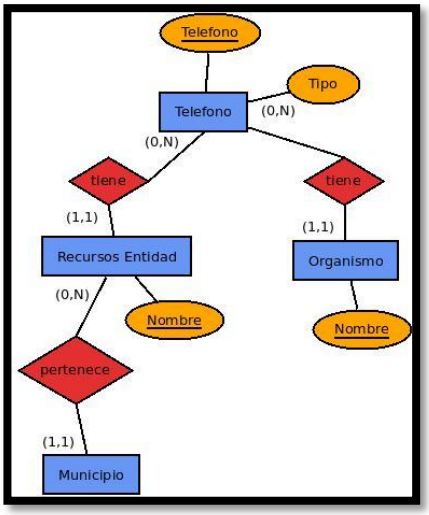


Figura B-10 ER – Organismos y recursos

recursos_entidad(recursos_entidad_id: entero;
municipio_id: entero;
nombre: cadena(100) *no nulo*)
municipio_id clave ajena de municipio(municipio_id) *no nulo*
organismo(organismo_id: entero;
nombre: cadena(100) *no nulo*);

Los teléfonos es uno de los aspectos más importantes de la aplicación. La forma de reflejarlos en el esquema de relación es:

//Teléfono
telefono_tipo(telefono_tipo_id: entero;
descripcion: cadena(100) *no nulo*)

telefono_numero(telefono_numero_id: entero;
telefono_tipo: entero *no nulo*
numero: cadena(100) *no nulo*)

telefono_destino(telefono_destino_id: entero,
descripcion: cadena(100) *no nulo*)

telefono(telefono_id: entero;
telefono_destino_id: entero;
telefono_numero_id: entero;
clave: entero;
telefono_destino_id clave ajena telefono_destino(telefono_destino_id) *no nulo*
telefono_numero_id clave ajena telefono_numero(telefono_numero_id) *no nulo*

La idea es crear un listín de teléfonos, en el que todos los tipos de destino estén juntos, en vez de tablas separadas. El motivo es que al ser tan importantes, si el cliente necesita crear una nueva entidad que tenga teléfonos, no haga falta crear una nueva tabla, sino un nuevo tipo de en la tabla telefono_destino.

La tabla teléfono tiene las siguientes restricciones: Tabla teléfono según telefono_destino_id, la clave debe existir en:

- "telefono_destino_id"= 1 "clave" clave ajena cliente(cliente_id)

- "telefono_destino_id"= 2 "clave" clave ajena persona_convivencia(persona_id)
- "telefono_destino_id"= 3 "clave" clave ajena persona_avisar(persona_id)
- "telefono_destino_id"= 4 "clave" clave ajena organismo(organismo_id)
- "telefono_destino_id"= 5 "clave" clave ajena asistencia_sanitaria(as_id)
- "telefono_destino_id"= 6 "clave" clave ajena hospital(hospital_id)
- "telefono_destino_id"= 7 "clave" clave ajena medico(medico_id)
- "telefono_destino_id"= 8 "clave" clave ajena empresa_servicios(empresa_servicios_id)
- "telefono_destino_id"= 9 "clave" clave ajena empresa_equipos(empresa_equipos_id)
- "telefono_destino_id"= 10 "clave" clave ajena recursos_entidad(recursos_entidad_id)
- "telefono_destino_id"= 11 "clave" clave ajena empresa_teleasistencia(empresa_teleasistencia_id)
- "telefono_destino_id"=12 "clave" clave ajena (centro_id)
- "telefono_destino_id"=13 "clave" clave ajena vivienda(vivienda_id)
- "telefono_destino_id"=12 "clave" clave ajena direccion(direccion_id)

B.6.2 Personas

En la base de datos van a registrarse los diferentes tipos de personas:

- Usuarios de la aplicación.
- Clientes.
- Personas que conviven con el cliente.
- Personas a avisar en caso de emergencia.
- Médicos.

Hay una serie de restricciones que impiden que estén todos en una tabla:

- En médicos, el DNI puede ser desconocido.
- Los usuarios de la aplicación no pueden ser clientes

Por lo tanto el diagrama diseñado según la Figura B-11.

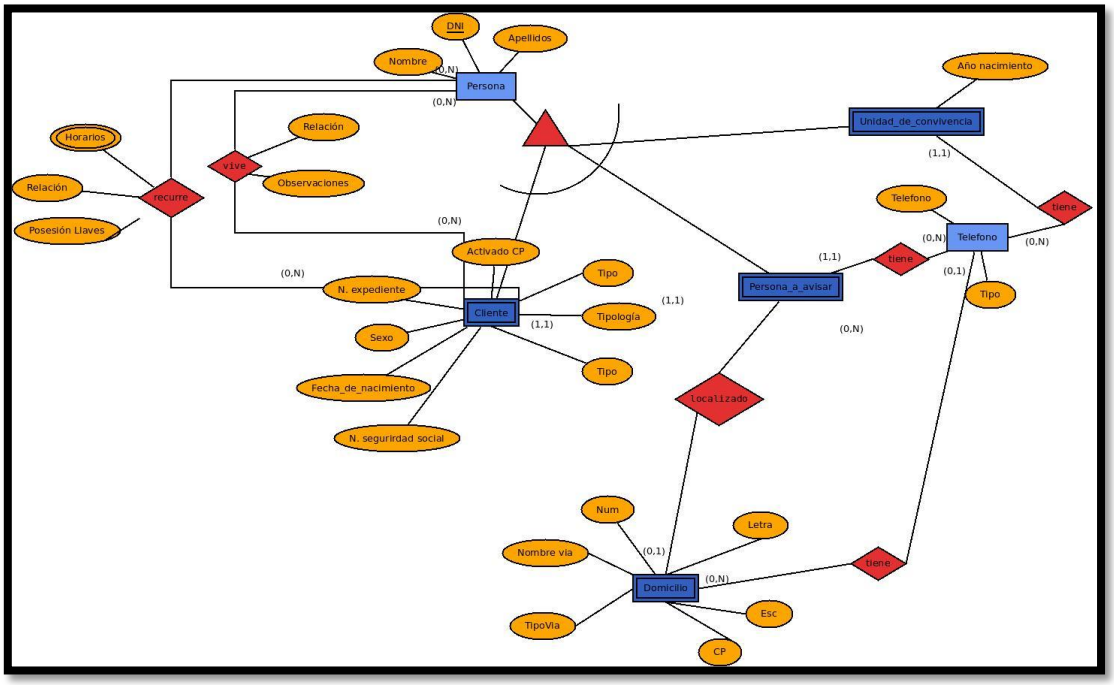


Figura B-11 ER – Personas

Comentarios

- Una persona es especialización de cliente, persona_convivencia y persona_avisar.
- Un cliente puede vivir con cualquier tipo de persona
- Un cliente puede recurrir en caso de emergencia a cualquier tipo de persona
- Varios clientes pueden tener en común varias personas de convivencia / persona a recurrir.
- Tipo de cliente es la codificación según:
  1. Titular del servicio con UCR
  2. Titular del servicio sin UCR
  3. No titular del servicio con UCR
- Tipología de cliente es la codificación según:
  1. Mayor de 65
  2. Discapacitado físico
  3. Discapacitado psíquico
  4. Discapacitado sensorial

Activado CP de cliente indica si está activado el servicio de control de pasividad (véase GLOSARIO

Esquema de relación

//Cliente – Información personal

persona = (persona_id; integer *no nulo*
dni: cadena *no nulo unico*;
nombre: tpnombre *no nulo;*
apellido1: tpnombre *no nulo,*
apellido2: tpnombre *no nulo,*
);

tipo_cliente = (tipo_cliente_id: integer *no nulo*
descripcion: cadena *no nulo*
);

tipologia =(tipologia_id: integer *no nulo*
descripcion: cadena *no nulo*
);

persona_cliente = (cliente_id; integer *no nulo*
dni: cadena *no nulo*;
n_expediente: cadena *no nulo*;
ciu: cadena *no nulo;*
sexo: tpsexo *no nulo*;
activado_cp: booleano *no nulo*;
estado_id: entero *no nulo*;
tipo_cliente_id: tptipocliente *no nulo*;
tipologia_id: entero *no nulo*;
fecha_nacimiento: tpfecha *no nulo*;
n_seguridad_social: cadena *no nulo*)
cliente_id clave ajena de persona(persona_id)
tipo_cliente_id clave ajena de tipo_cliente(tipo_cliente_id)
tipologia_id clave ajena de tipologia(tipologia_id)

//Personas de convivencia y a recurrir en caso de emergencia

relacion = (relacion_id: entero *no nulo;*
descripcion: cadena *no nulo)*

persona_convivencia = (persona_id; entero *no nulo*
fecha_nacimiento: tpfecha *no nulo*)
persona_id clave ajena de persona(persona_id)

persona_avisar = (persona_id; entero *no nulo*
disponibilidad: cadena *no nulo*)
dni clave ajena de persona(dni)
persona_id clave ajena de persona(persona_id)

persona_avisar_domicilio = (persona_id; entero *no nulo*
direccion_id: entero *no nulo*)
persona_id clave ajena de persona(persona_id) *no nulo*
direccion_id clave ajena de direccion(direccion_id) *no nulo*

cliente_vive_ persona = (cliente_id: entero *no nulo;*
persona_id: entero *no nulo*
relación: entero *no nulo*;
observacion: cadena(100))
persona_id clave ajena de persona(persona_id) *no nulo*
cliente_id clave ajena de cliente(cliente_id) *no nulo*

cliente_recurre_persona = (cliente_id: entero *no nulo;*
persona_id; entero *no nulo*
relación: entero *no nulo*;
observacion: cadena(100);
posesion_llaves: booleano *no nulo*)
persona_id clave ajena de persona(persona_id) *no nulo*
cliente_id clave ajena de cliente(cliente_id) no nulo

B.6.3 Características y estados

El siguiente diagrama, Figura B-12, explica la relación de ambas entidades con cliente.

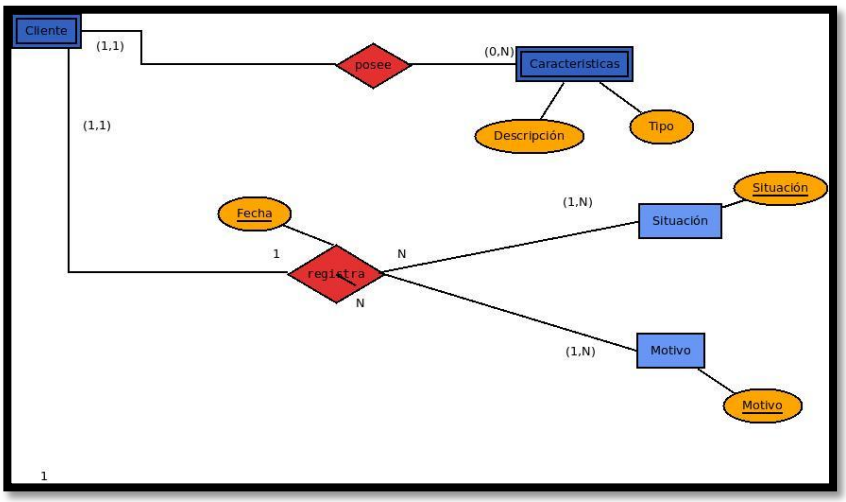


Figura B-12 ER – Características y situación

Comentarios

- situacion_id de Cliente indica si:
  1. En activo: recibiendo el servicio.
  2. Ausencia domiciliaria inferior a 24 horas: estado en activo, sale del domicilio y lo comunica.
  3. Suspensión temporal: habiendo estado en activo, deja provisionalmente de recibir el servicio por periodo superior a 24 horas..
  4. Baja definitiva: habiendo estado en activo, causa baja definitivamente en el servicio.
  5. Sin iniciar actividades: servicio solicitado y asignado, aún no ha causado alta.
  6. Baja definitiva de solicitantes que nunca estuvieron en activo.
- Un Cliente puede registrar varios tipos de características.
- TpTipoCaracterística es la codificación según:
  1. Riesgos.
  2. Físicas
  3. Psicológicas
  4. Sensoriales
- La característica es única por cada Cliente, ya que la descripción es personalizada.

Esquema entidad – relación

//Cliente – Estados

situacion= (situacion_id: entero *no nulo*;
descripción: cadena(20) *no nulo*;)

motivo = (motivo_id: entero *no nulo*;
descripción: cadena(20) *no nulo*;)

estados = (estado_id: entero;
cliente_id: tpdni *no nulo*;
situacion_id: tpsituacion *no nulo*;
motivo_id: tpmotivo *no nulo*;
fecha_inicio: tpfecha *no nulo;*
fecha_fin: tpfecha *no nulo*)
cliente_id clave ajena de cliente(cliente_id) *no nulo;*
situacion_id clave ajena de situacion(situacion_id) *no nulo;*
motivo_id clave ajena de motivo(motivo_id) *no nulo;*

caracteristica_tipo = (caracteristica_tipo_id: entero *no nulo;*
descripcion: cadena *no nulo);*

caracteristica = (caracteristica_id: entero;
cliente_id: entero *no nulo;*
caracteristica_tipo_id: entero *no nulo*;
descripción: cadena(100))
cliente_id clave ajena, cliente(cliente_id), *no nulo*
caracteristica_tipo_id clave ajena caracteristica_tipo(caracteristica_tipo_id) *no nulo*

Restricciones

- Dado una situación sólo es posible la siguiente combinación con motivo:

Situación de cliente / En el proyecto / Motivo de la situación	En activo	Ausencia	Suspensión temporal	Baja definitiva	Sin iniciar actividades	Baja solicitantes
En espera de evaluación	-	-	A	A	A	A
Petición del interesado	-	-	A	A	A	A
Obtención del recurso por otros medios	-	-	-	A	-	A
Traslado domicilio propio	-	-	A	A	A	A
Traslado con familiares o amigos	-	-	A	A	A	A
Ingreso en centro residencial	-	-	A	A	A	A
Derivado a otra empresa/entidad	-	-	-	A	-	A
Vacaciones	-	-	A	-	A	-
Enfermedad	-	-	A	-	A	-
Ingreso en centro hospitalario	-	-	A	-	A	-
No aceptar las condiciones del servicio	-	-	A	A	A	A
Solución del problema	-	-	-	A	-	A
Fallecimiento	-	-	-	A	-	A
Traslado expediente a otro centro de atención de teleasistencia de la empresa/entidad	-	-	A	A	-	A
Carencia de recursos humanos	-	-	A	-	A	-
Carencia de recursos materiales	-	-	A	-	A	-
No tener cubiertas las necesidades básicas	-	-	A	-	A	-
Recibiendo el servicio actualmente	A	-	-	-	-	-
Ausencia domiciliaria inferior a 24 horas	-	A	-	-	-	-
Otros	-	-	A	A	A	A
Nota:A: relación aceptada -: relación no aceptada						

Tabla B-1 Estado: situación-motivo.

B.6.4 Vivienda

El siguiente diagrama, Figura B-13, se refleja la distribución de dirección con municipio – provincia. Dirección tiene especial importancia ya que está relacionado con las entidades: vivienda y persona_avisar.

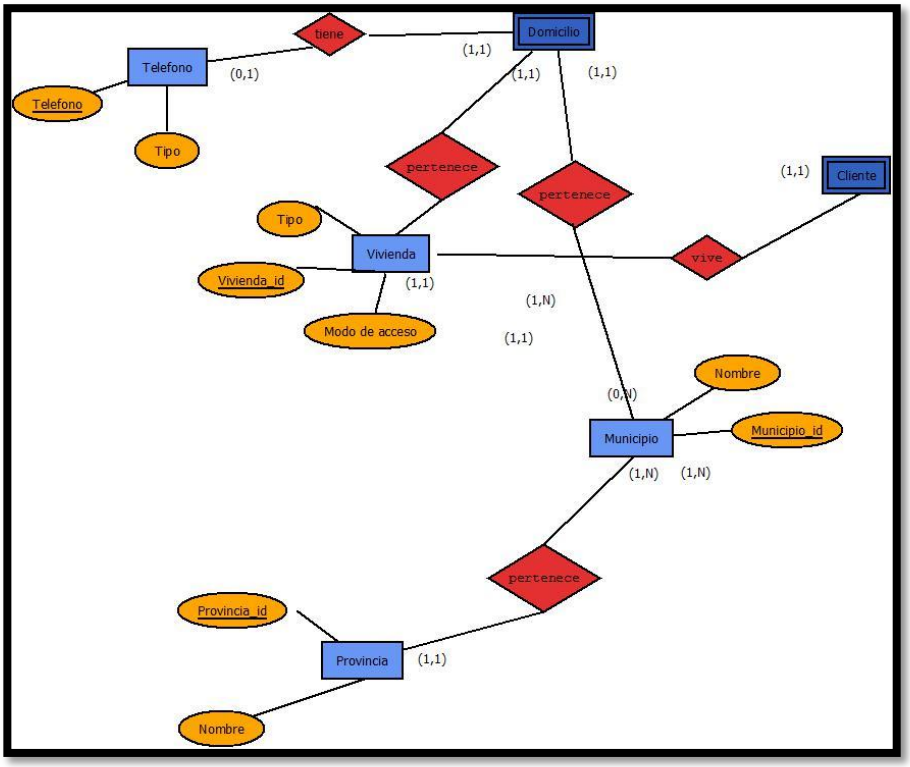


Figura B-13 ER – Vivienda

Comentarios

Se observa que aunque en los requisitos de esta versión no se permita que varios clientes vivan en la misma vivienda, se ha permitido en la base de datos para que sea escalable.

- Varios clientes puedan compartir la misma vivienda.
- La vivienda pertenece a un único municipio
- El municipio pertenece a una única provincia
- Tipo de Vivienda es la codificación de:
  1. Edificio de vecinos
  2. Unifamiliar urbana
  3. Unifamiliar aislada
- Una vivienda puede tener varias empresas contratadas (gas, electricidad, etc) lo mismo que estas empresas pueden estar en alguna o muchas viviendas.

Esquema entidad-relación

//Direcciones

municipio = (municipio_id: entero;
nombre: tpcadena(100) *no nulo*;
codigo: cadena(3) *no nulo*
provincia_id: entero*)*
provincia_id clave ajena de provincia(provincia_id) *no nulo*;

provincia = (provincia_id: entero;
nombre: tpcadena(100) *no nulo*;
codigo: cadena(3) *no nulo)*
identidad clave ajena de entidad_local(identidad) *no nulo*;

tipo_via(tipo_via_id: entero;

descripcion: cadena(100) *no nulo*)

direccion (direccion_id: entero;
tipo_via_id: entero *no nulo*;
nombre_via: cadena(150) *no nulo*;
numero: cadena(4);
piso: cadena(6);
letra: cadena(6);
esc: cadena(6);
cp: cadena(5) *no nulo*;
municipio_id: entero;
telefono_id: enetero;
telefono_id clave ajena telefono(telefono_id) *no nulo*
tipo_via_id clave ajena tipo_via(tipo_via_id) *no nulo*
municipio_id clave ajena municipio(municipio_id) *no nulo*

//Cliente – Vivienda

vivienda_tipo = (vivienda_tipo_id: entero;
descripción: cadena(20) *no nulo)*

vivienda = (vivienda_id: entero;
direccion_id: entero *no nulo*;
vivienda_tipo_id: entero *no nulo*;
modo_acceso: tpcadena (100);
direccion_id clave ajena direccion(direccion_id) *no nulo;*

servicio_tipo = (servicio_tipo_id: entero *no nulo*;
descripción: cadena(20) *no nulo*)

empresa_servicios = (empresa_servicios_id: entero;
nombre; tpcadena(50);
servicio_tipo_id: entero *no nulo)*
servicio_tipo_id clave ajena de servicio_tipo(servicio_tipo_id) *no nulo;*

vivienda_empresa = (vivienda_id: entero ;
empresa_servicios_id: entero)
vivienda_id clave ajena de vivienda(vivienda_id) *no nulo;*
empresa_servicios_id clave ajena de empresa_servicios(empresa_servicios_id) no nulo;

cliente_vivienda = (cliente_id: entero *no nulo;*
vivienda_id: entero *no nulo)*

B.6.5 Instalación

Comentarios

- Un cliente tiene contratado una instalación.
- Varios clientes pueden beneficiarse de la misma instalación.
- En caso de que varios clientes tengan la misma instalación, solo uno puede ser titular.

- Una instalación está formada por al menos uno (que sería el terminal) o muchos equipos.
- Existe una base de datos de los equipos registrados con los detalles técnicos.
- Dichos equipos son fabricados por una única Empresa.

Pensado en ser escalable, la base de datos permite que varios usuarios compartan una instalación. En el siguiente diagrama, Figura B-14 se observa el diseño:

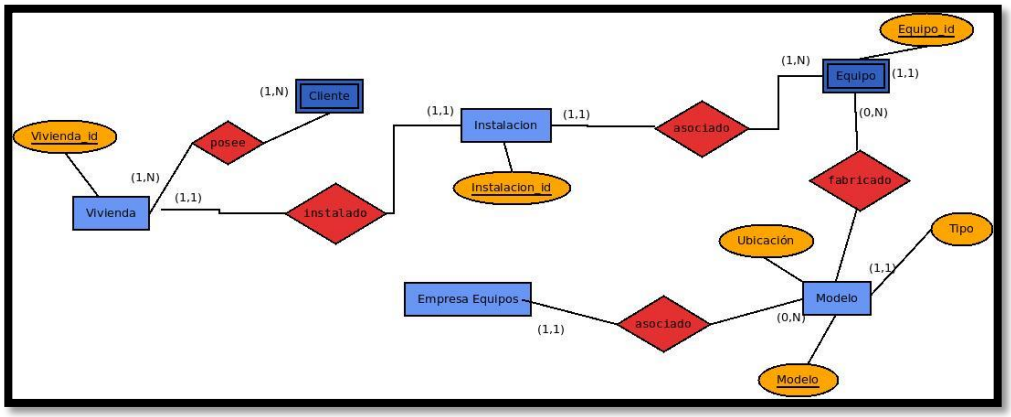


Figura B-14 ER – Instalación - Equipos

Esquema entidad relación

//Cliente – Instalacion

instalacion (instalacion_id: entero;
vivienda_id: entero);
vivienda_id clave ajena de vivienda(vivienda_id) no nulo

vivienda_instalacion = (vivienda_id: entero;
instalacion_id: entero)
vivienda_id clave ajena vivienda(vivienda_id) *no nulo;*
instalacion_id clave ajena de instalacion(instalacion_id) *no nulo;*

//Datos equipos

modelo_tipo = (modelo_tipo_id: entero;
descripcion: cadena(100) *no nulo*)

modelo (modelo_id: entero;
nombre: cadena(100) *no nulo*;
empresa_equipos_id: entero ;
modelo_tipo_id: entero;
descripcion: cadena(200))
modelo_tipo_id clave ajena de modelo_tipo(modelo_tipo_id) *no nulo*
empresa_equipos_id clave ajena de empresa_equipos(empresa_equipos_id) *no nulo*

equipo (equipo_id: entero;
nombre: cadena(50) *no nulo);*

estado_equipo (estado_equipo_id: entero;
descripcion: cadena(50) *no nulo*);

equipo_detalles (equipo_id: entero;
modelo: entero *no nulo*;
ubicación: tpcadena(100) *no nulo*;
instalacion_id: entero);
equipo_id clave ajena de equipo(equipo_id) *no nulo*
modelo clave ajena de tabla_equipos *no nulo*

equipo_chequeo(equipo_chequeo_id: entero;
equipo_id: entero;
usuario_id: entero;
fecha: tpFecha *no nulo*;
estado_equipo_id: entero)
usuario_id clave ajena de usuario(usuario_id) *no nulo*
equipo_id clave ajena de equipo(equipo_id) *no nulo*
estado_id clave ajena de estado(estado_id) *no nulo*

empresa_equipos(empresa_equipos_id: entero;
nombre: tpnombre
email: cadena(100));

B.6.6 Datos sanitarios

Comentarios

- Un cliente tiene asignado ningún o alguna asistencia sanitaria
- Cada dirección del cliente, tiene diferente distancia con la de la asistencia sanitaria. Una asistencia sanitaria cubre cero o varios hospitales.
- En una asistencia sanitaria trabajan cero o varios médicos.
- Un médico puede estar en varias asistencias sanitarias.
- Un hospital puede estar en varias asistencias sanitarias.
- El cliente tiene asignado ninguno o varios médicos que trabajan en una asistencia sanitaria concreta.
- El cliente tiene asignado ninguno o varios hospitales que pertenecen a una asistencia sanitaria concreta.

El siguiente diagrama, Figura B-15 muestra las relaciones observadas:

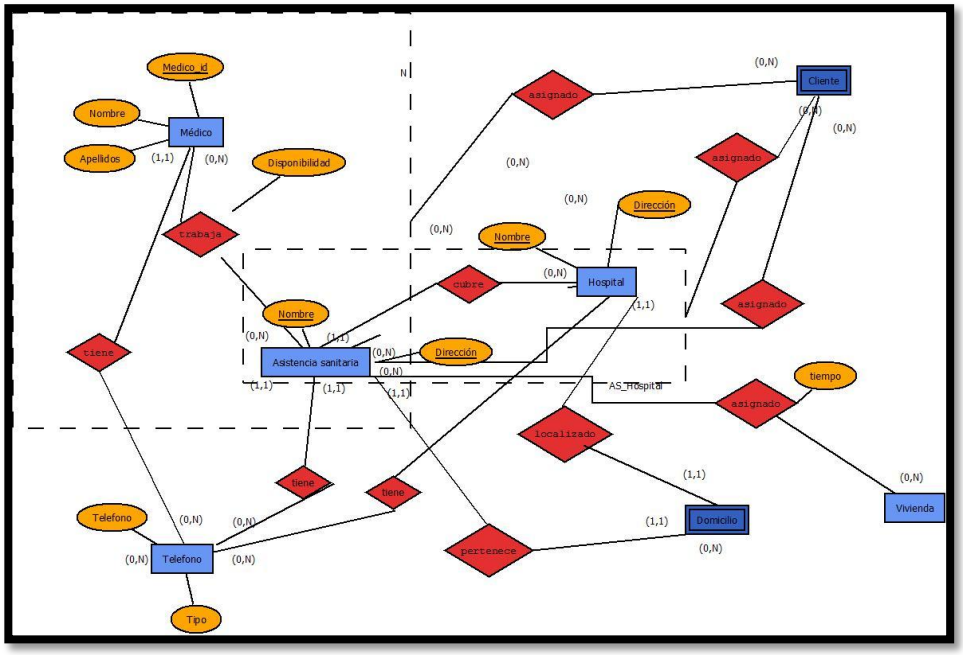


Figura B-15 ER – Asistencia sanitaria

Esquema entidad-relación

//Cliente – Datos sanitarios
asistencia_sanitaria(as_id: entero;
nombre: tpnombre *no nulo*;
direccion_id: entero *no nulo*;)

medico(medico_id: entero;
nombre: tpnombre *no nulo*;
apellido1: tpnombre *no nulo*,
apellido2: tpnombre *no nulo*)

as_medico(as_medico_id: entero;
as_id: entero;
medico_id: entero;
disponibilidad: text *no nulo*;)
as_id clave ajena de asistencia_sanitaria(as_id) *no nulo;*
medico_id clave ajena de medico(medico_id) *no nulo;*

as_medico_cliente(as_medico_cliente_id: entero;
as_medico_id: entero;
cliente_id: entero;)
as_medico_id clave ajena de as_medico_id (s_medico) *no nulo;*
cliente_id clave ajena de cliente(cliente_id) *no nulo;*

as_cliente (as_cliente_id:entero;
cliente_id: entero;
as_id: entero;)
as_id clave ajena de asistencia_sanitaria(as_id) *no nulo;*
cliente_id clave ajena de cliente(cliente_id) *no nulo;*

as_vivienda (as_vivienda_id:entero;
vivienda_id: entero;

as_id: entero;
tiempo: entero)
as_id clave ajena de asistencia_sanitaria(as_id) *no nulo;*
vivienda_id clave ajena de vivienda(vivienda_id) *no nulo;*

hospital (hospital_id: entero;
nombre: tpnombre *no nulo*;
direccion_id: entero *no nulo)*
direccion_id clave ajena de direccion(direccion_id) *no nulo;*

as_hospital (as_hospital_id: entero;
as_id: entero;
hospital_id: entero)
as_id clave ajena de asistencia_sanitaria(as_id) *no nulo*;
hospital_id ajena hospital (hospital_id) *no nulo;*

as_hospital_cliente(as_hospital_cliente_id: entero;
as_hospital_id: entero;
cliente_id: entero;
as_hospital_id clave ajena de as_hospital(as_hospital_id)
cliente_id clave ajena de cliente(cliente_id)

B.6.7 Alarmas - Llamadas

Por último se detalla el diagrama entidad relación y esquema de alarmas y llamadas en la Figura B-16.

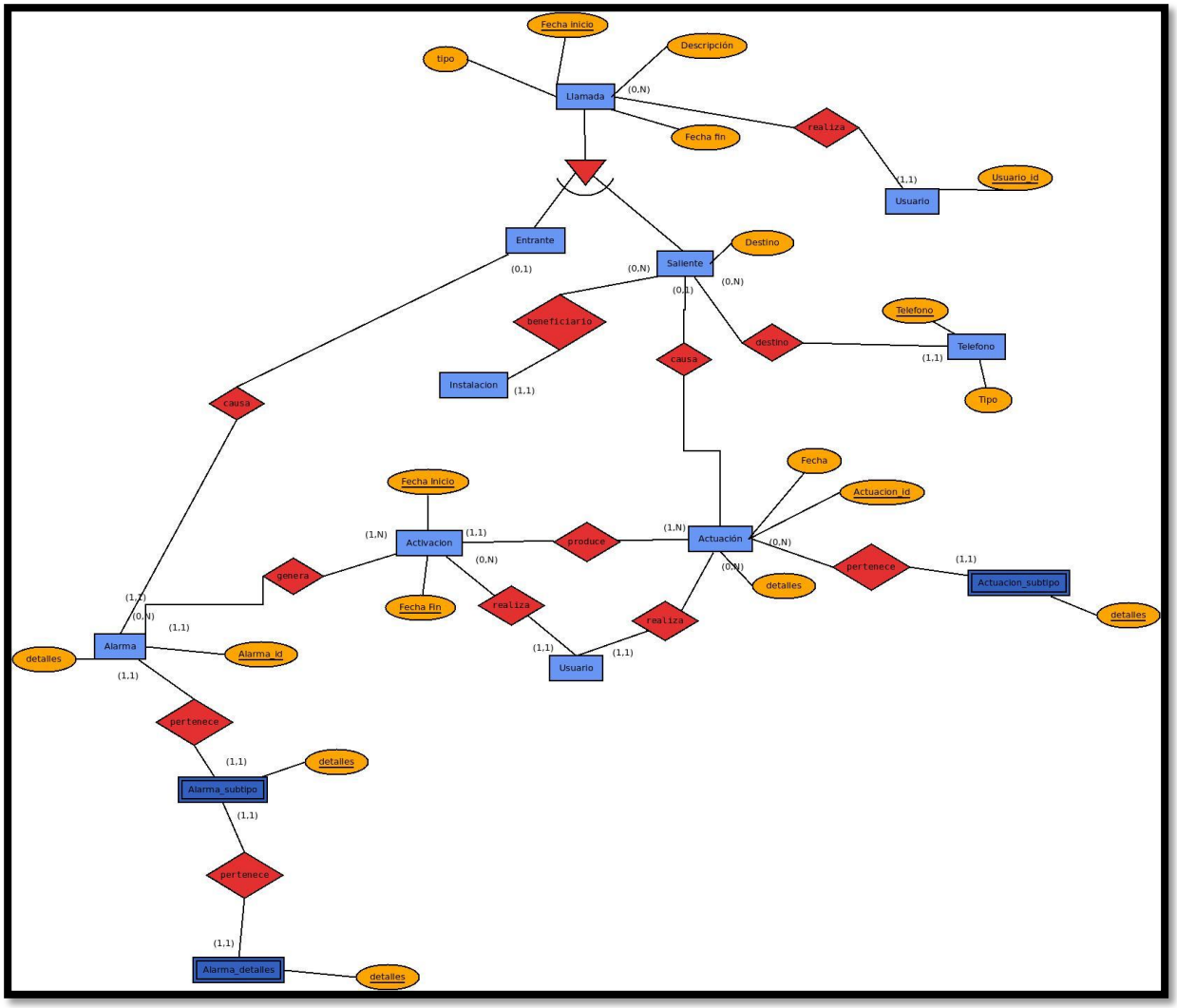


Figura B-16 ER – Alarmas - llamadas

Comentarios:

- Un cliente puede activar por activa o pasiva cero o varias alarmas en diferentes momentos del tiempo a partir de un equipo.
- La alarma activa es una llamada y la alarma pasiva es una alarma técnica
    - Técnica: originadas por un terminal (ya sea por un sensor de movimiento, humos, el terminal no responde, avisos de caída del sistema, etc)
    - Llamadas: originadas por el cliente son las alarmas por pulsación de uno de los dos dispositivos:
        - Terminal de cliente.
        - Unidad de control remota (UCR).
- Los códigos de las alarmas están almacenados en “alarma_tipo” y “alarma_subtipo”.
- Una llamada saliente es generada por un operador y para un cliente.
- Una alarma cuando se acepta queda activada.
- Una alarma puede tener al menos una o varias activaciones.
- Por cada activación tiene una o varias actuaciones.
- Activación y capturar una alarma tiene el mismo significado.

Esquema entidad-relación

//Alarmas
alarma_tipo (alarma_tipo_id: entero;
descripción: cadena(150));

alarma_subtipo (alarma_subtipo_id: entero;
alarma_tipo_id: entero;
descripción: cadena(150));
alarma_tipo_id clave ajena de alarma_subtipo(alarma_subtipo_id) *no nulo*

alarma_detalle (alarma_detalle_id: entero;
alarma_subtipo_id: entero;
descripción: cadena(150));
alarma_subtipo_id clave ajena de alarma_subtipo(alarma_subtipo_id) *no nulo*

alarma(alarma_id: entero;
alarma_tipo_id: entero;
alarma_subtipo_id: entero;
alarma_detalle_id: entero;
ciu: cadena(50);
cliente_id: entero;
llamada_entrante_id: entero)
alarma_tipo_id clave ajena de equipo(equipo_id) *no nulo*
alarma_subtipo_id clave ajena de alarma_subtipo(alarma_subtipo_id) *no nulo*
alarma_detalle_id clave ajena de alarma_detalle(alarma_detalle_id)
cliente_id clave ajena de cliente(cliente_id)
llamada_entrante_id clave ajena de llamada_entrante(llamada_entrante_id) *no nulo*

activacion (activacion_id: entero;
alarma_id: entero;
fecha_inicio: tpFecha *no nulo*;
fecha_fin: tpFecha;
usuario_id: entero;
usuario_id clave ajena de usuario(usuario_id) *no nulo*
alarma_id clave ajena de alarma(alarma_id) *no nulo*

actuacion_tipo (actuacion_tipo_id: entero;
descripción: cadena(150));

actuacion_subtipo (actuacion_subtipo_id: entero;
actuacion_tipo_id: entero;
descripción: cadena(150));

activacion_actuacion (activacion_actuacion_id: entero;
actuacion_tipo_id: entero;
actuacion_subtipo_id: entero;
usuario_id: entero;
comentario: cadena(300);
llamada_saliente_id: entero)
usuario_id clave ajena de usuario(usuario_id) *no nulo*
actuacion_tipo_id clave ajena de actuacion_tipo(actuacion_tipo_id) *no nulo*
actuacion_subtipo_id clave ajena de actuacion_subtipo(actuacion_subtipo_id) *no nulo*

llamada_saliente_tipo(llamada_saliente_tipo_id: entero;
descripcion: cadena(100))

llamada_saliente(llamada_saliente_id: entero;
usuario_id: entero;
llamada_saliente_tipo_id: entero;
telefono_id: entero;
instalacion_id: entero;
comentario: cadena(100);
fecha_inicio: tpfecha *no nulo*;
fecha_fin: tpfecha;
numero: cadena(15))
usuario_id clave ajena de usuario(usuario_id) *no nulo*
operador_id clave ajena de operador(operador_id) *no nulo*
llamada_saliente_tipo_id clave ajena de
llamada_saliente_tipo(llamada_saliente_tipo_id) *no nulo*
telefono_id clave ajena de telefono(telefono_id) *no nulo*
instalacion_id clave ajena de instalacion(instalacion_id) *no nulo*

llamada_entrante(llamada_entrante_id: entero;
cliente_id: entero;
usuario_id: entero;
telefono_id: entero;
comentario: cadena(100);
fecha_inicio: tpfecha *no nulo*;
fecha_fin: tpfecha)
usuario_id clave ajena de usuario(usuario_id) *no nulo*
operador_id clave ajena de operador(operador_id) *no nulo*
telefono_id clave ajena de telefono(telefono_id) *no nulo*
instalacion_id clave ajena de instalacion(instalacion_id) *no nulo*

B.7 Actividad ASI 5: diagrama de clases

En el apartado de subsistemas se han identificado unos módulos. En esta actividad se ha desarrollado la primera versión del diagrama de clases. Este diagrama proviene de las conclusiones desarrolladas en los casos de uso y la base de datos.

Las clases identificadas son familiares ya que los nombres y definiciones han sido arrastrados en todo el análisis. El la siguiente figura, Figura B-17, se muestra el diagrama.

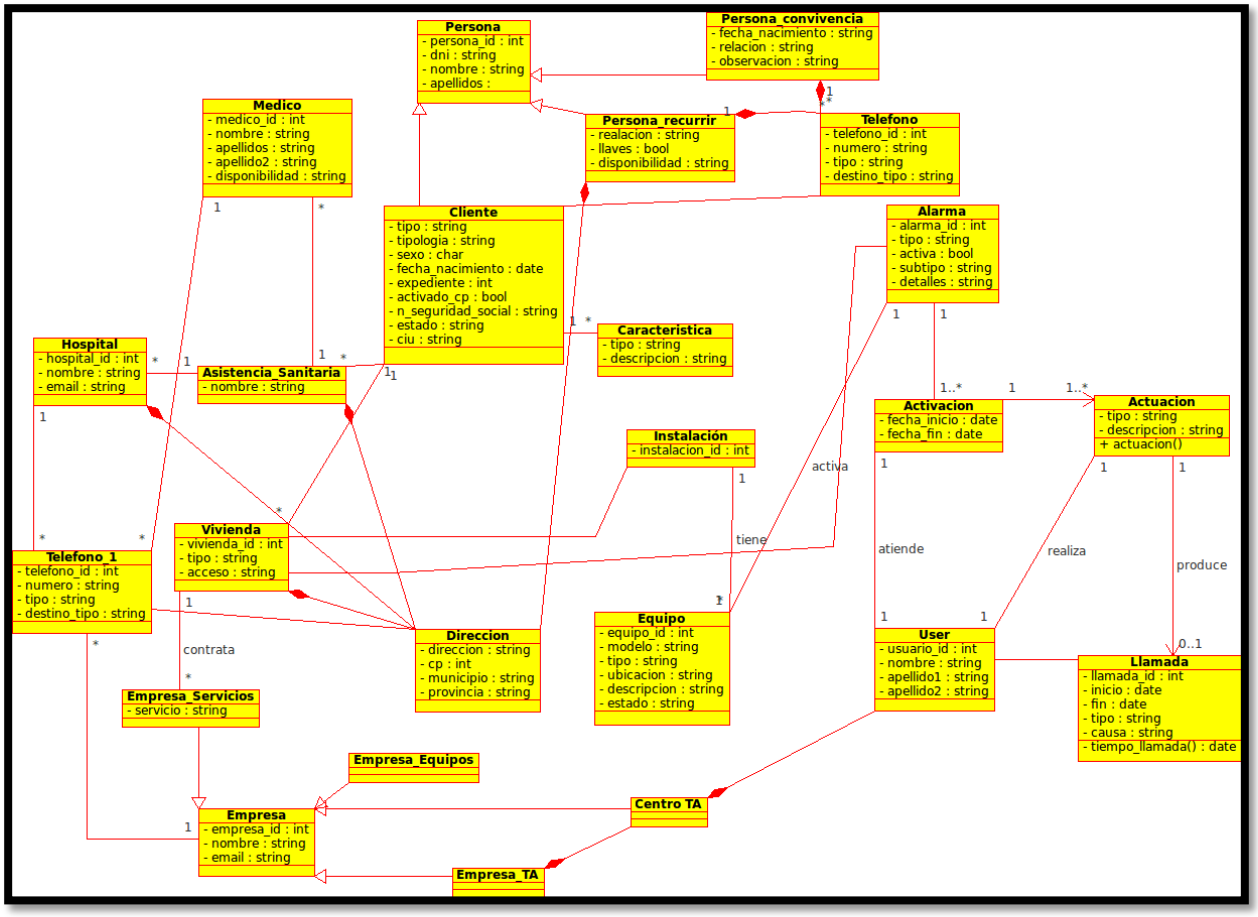


Figura B-17 Diagrama de clases

Observaciones:

- Herencia:
  - Cliente, Persona_recurrir y Persona_convivencia heredan de persona
  - Empresa_servicios, Empresa_Teleasistencia, Empresa_Equipos y Centro herendan de Empresa

Este diagrama se revisará en el diseño y en la implementación.

B.8 Actividad ASI 8: principios de la interfaz

En esta actividad se pretende establecer unas normas y patrones para el diseño de la interfaz y realizar el prototipo de la aplicación.

El objetivo de las normas es mantener la similitud y conexión en todas las ventanas, de forma que el usuario aprenda a usarla de forma fácil y cómoda.

B.8.1 Interfaz principal

- Menú a la izquierda y una zona principal o centro de operaciones.
  - Siempre visible.
- Parte superior nombre del operador.
- Cuando se pulse una opción del menú, se abrirá una pestaña en la zona de operaciones:
  - No serán ventanas emergentes.
  - Se podrán cerrar.

B.8.2 Tablas

- La información se mostrará en tablas, por ejemplo: personas de convivencia.
- La tabla siempre contenida en un panel.
- El panel en el que está la tabla, debe tener una línea exterior, con una etiqueta, donde se nombre lo que se está mostrando en dicha tabla.
- Habrá una tabla por pestaña, a no ser que tenga información derivada de esa tabla (tablas con dependencias).
- En tablas dependientes, cada vez que se seleccione una línea de la tabla principal, cambiará la parte de la tabla dependiente.
- En caso de varias tablas (por ejemplo: personas de convivencia y teléfonos), se situarán por prioridad (zona superior: personas), cada panel con el bordeado correspondiente y la etiqueta con la descripción.
- Sólo puede haber hasta tres tablas en una pestaña. En caso de necesitar más, dividir la información en más pestañas.
- Mostrar en la tabla principal la primera fila, siempre como seleccionada.
- Si la tabla no tiene dependencias, mostrar por defecto sin seleccionar; si tiene dependencias, debe aparecer la información en la tabla de dependencias de la primera fila de la tabla principal.
- La edición del contenido de la tabla se realizará siempre por medio de una ficha, que bloquee el acceso a la ventana principal hasta que no se haya terminado.
- Botones de edición de tablas siempre en la parte superior: agregar, modificar, asociar y/o desasociar.
- Opción de asociar, que muestre una lista desplegable en una ficha con las opciones que no tiene el cliente.
- Opción eliminar solamente si antes ha visualizado la información que va a eliminar.
- Opciones de ordenación por cada columna de la tabla.
- Opciones de editar el ancho de cada columna de la tabla.
- Al pulsar doble “click” sobre una fila de una tabla con información, se abrirá la ficha de edición en caso de tener dicha funcionalidad.

B.8.3 Pestaña

- Cada pestaña debe estar acompañada con un icono intuitivo a la izquierda.
- El botón de cerrar aparece a la derecha.

B.8.4 Botones

- Añadir – modificar – asociar – desasociar – eliminar – buscar: deben contener icono intuitivo.
- Preferiblemente, ubicarlos en una barra de herramientas.

B.8.5 Fichas

- Parte superior con el nombre de la ficha.
- Parte central con los campos a rellenar.
- Parte inferior con dos botones: borrar y guardar.
- Guardar en la parte inferior de la derecha.
- Opción de cerrar ventana, equivalente a no hacer cambios.

B.8.6 Aviso alarma

- Ventana emergente de aviso de alarma.
- Pequeño tamaño con información principal con etiquetas.

- Enlace de aceptar la llamada.

B.8.7 Errores

- Cuando haya un error, se mostrará una ventana emergente cuyo control no se devolverá a la pantalla principal hasta que no se haya dado a la información principal informando de dicho error.
- Antes de eliminar un elemento siempre preguntar con una ventana que informe de lo que se pretende eliminar y tenga dos botones “si” o “no”.

B.9 Prototipo de la interfaz de impresión

Como consecuencia de todas las actividades desarrolladas en el análisis se ha procedido a realizar el prototipo de la aplicación AsisT. Los diagramas se han realizado con la herramienta Pencil [31].

La siguiente Figura B-18 muestra el prototipo de la ventana principal de la aplicación. A la izquierda se dispondría de un menú de acceso rápido a los módulos, diferenciado un módulo de gestión y otro para atender alarmas. A la derecha se observa el centro de operaciones, donde contendrá las pestañas de información.

Se ha analizado una ventana principal, de forma que aporte de primera mano información imprescindible para la asistencia de un operador. El cuadro de mandos o ventana principal tendría una visualización como en la Figura B-18.

- Mostrar alarmas activas gestionadas por el usuario que está atendiendo
- Mostrar últimas alarmas finalizadas (atendidas) por un usuario
- Mostrar alarmas activas en tiempo real del centro de teleasistencia.

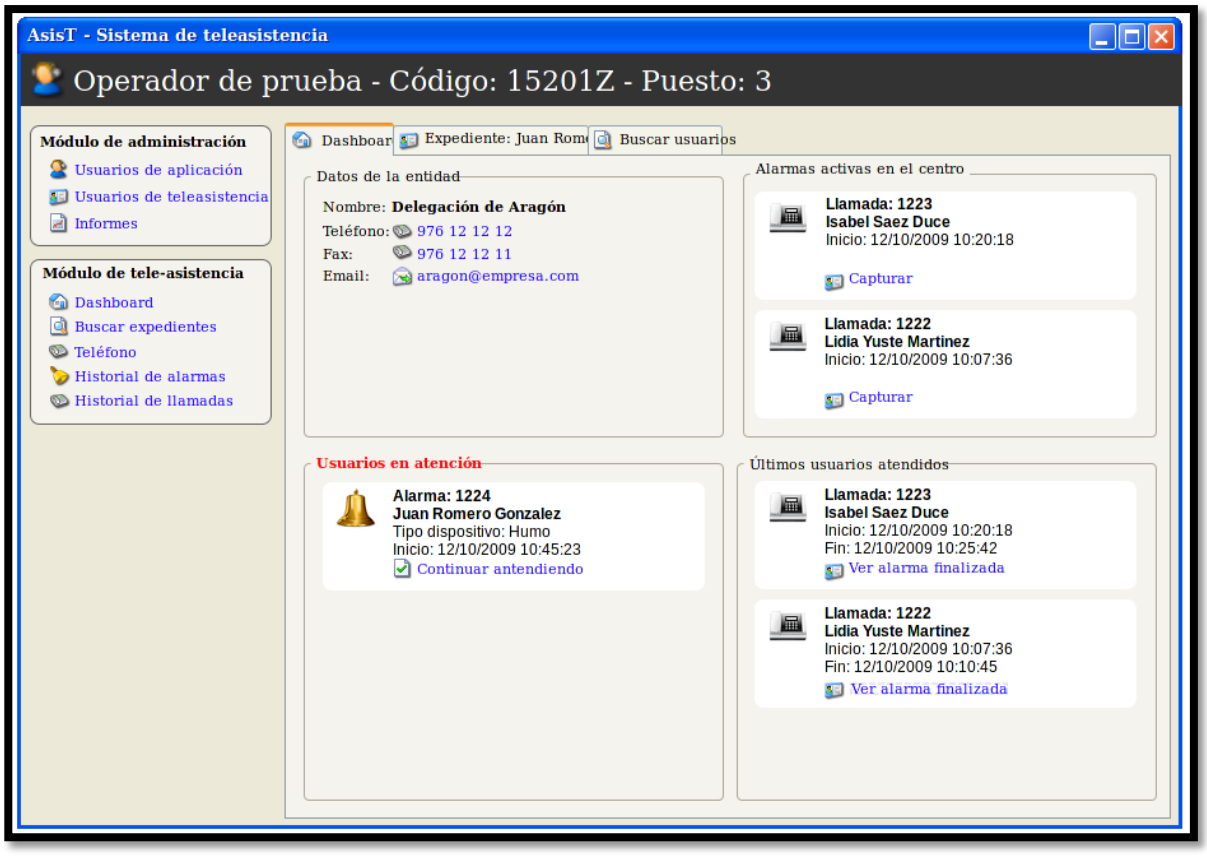


Figura B-18 Prototipo ventana principal

La ventana emergente se visualizará de la siguiente forma, Figura B-19.

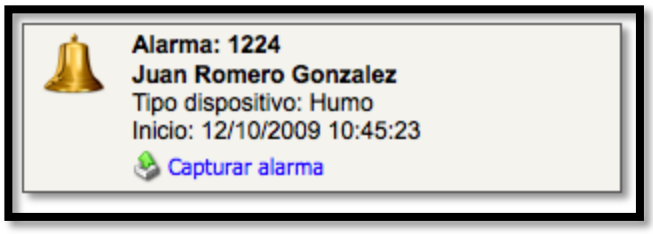


Figura B-19 Prototipo aviso de alarma

El expediente de un cliente deberá tener la siguiente interfaz, Figura B-20. En la parte superior y siempre visible, los datos personales del cliente. En la parte inferior, pestañas que no se podrán cerrar, con la información identificada como módulos, dentro del expediente.

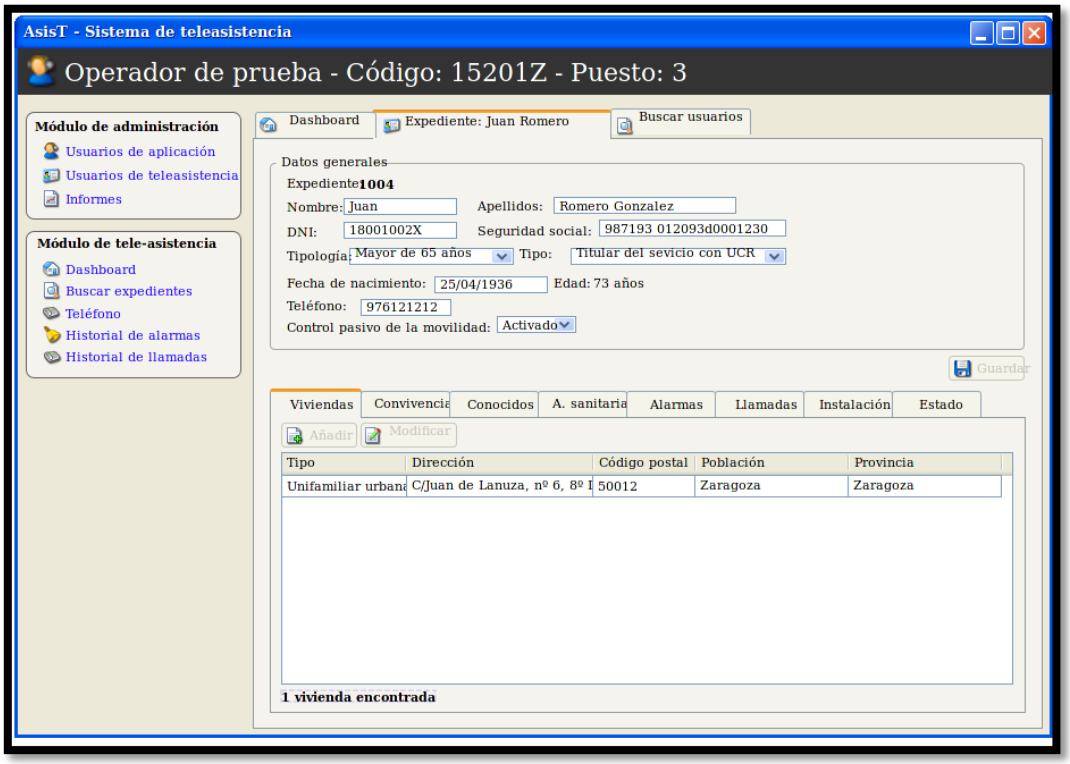


Figura B-20 Prototipo expediente

B.10 Conclusiones del análisis

En esta fase del sistema se han experimentado riesgos:

- El primer riesgo que se experimentó fue entender y comprender qué se quería hacer. La información aportada por el convenio-marco, en ocasiones, no era clara y tuvo que revisarse con cuidado debido a ambigüedades.
- Otro riesgo fue determinar la magnitud del sistema. Dado el tiempo disponible para la realización del proyecto, era importante establecer unos límites ya que una aplicación puede crecer en funcionalidad exponencialmente.

El documento ha estado abierto hasta el final del proyecto ante aspectos que no se tuvieron en cuenta. Sin embargo se ha evitado hacer modificaciones ya que un cambio en el análisis supone un cambio en cascada en todas las fases.

Como observación, la pantalla de Inicio, no está contenido en requisitos. No obstante, ha sido tomado como decisión de implementación por la gran utilidad que aporta a los operadores

Se recomienda continuar por el Anexo de Diseño, ANEXO C, para proseguir con el desarrollo del sistema.

UNIVERSIDAD DE ZARAGOZA

Centro Politécnico Superior

Ingeniería Informática

ANEXO C: DISEÑO DEL SISTEMA DE INFORMACIÓN

[3/5]

PFC: DESARROLLO DE UNA APLICACIÓN PARA LOS OPERADORES DE TELEASISTENCIA INTEGRADA EN TELEFONÍA IP

AUTORA: LAURA LACARRA ARCOS

DIRECTOR: ROBERTO CASAS MILLÁN

PONENTE: JESÚS ALASTRUEY BENEDÉ

Departamento Informática e Ingeniería de Sistemas

Zaragoza, Julio 2010

ANEXO C DISEÑO DEL SISTEMA DE INFORMACIÓN

C.1 Introducción

En este anexo se detalla el diseño del sistema AsisT. Este documento ha sufrido modificaciones desde su construcción hasta el final de la aplicación. El comienzo del mismo fue tras una primera versión del Anexo de Análisis, ANEXO B.

Al igual que el análisis de la aplicación, las actividades desarrolladas se basan en la metodología Métrica 3 [30].Los objetivos que se persiguen son diseñar:

- La arquitectura del sistema: definir los distintos niveles de la arquitectura.
- La tecnología del sistema: identificar los elementos o nodos que representan el sistema.
- Seguridad y control de acceso: especificar cómo implementar la seguridad de los datos.
- Normas y estándares para la construcción: identificar los patrones de diseño de forma que el sistema sea reutilizable por otro programador y que pueda ampliarse por medio de complementos a elección del comprador.
- Excepciones: especificar cómo controlar excepciones.
- Pruebas: elaborar el plan de pruebas.

Se han implementado las actividades de Métrica 3, acorde con las necesidades. Los siguientes apartados que se detallan siguen el orden de elaboración, indistintamente del orden que establece la metodología.

C.2 Actividad DSI 1: definición de niveles de arquitectura.

C.2.1 Identificación de los niveles de arquitectura

Para facilitar la compresión del sistema se han identificado los nodos elementales que sustentan la infraestructura del sistema. En esta parte del diseño se ha analizado la mejor alternativa y estudiado la mejor solución.

Los niveles de arquitectura que se van a estudiar son el puesto de los operadores, gestor de datos y servidor. El siguiente diagrama, Figura C- muestra la distribución y su conexión, partiendo de las conclusiones del análisis.

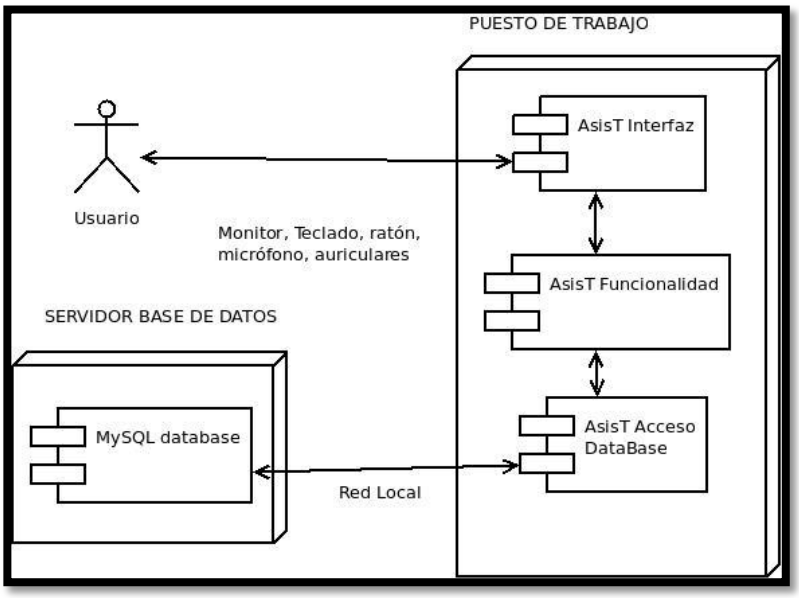


Figura C-1 Diagrama de componentes

C.2.2 Gestores de datos

En este apartado se va a elegir un gestor de base de datos por defecto. No obstante, la aplicación va a ser diseñada de forma que pueda migrarse a otro gestor de base de datos.

AsisT necesita un gestor de base de datos apropiado cuyas características sean:

- Compatible con Java por medio de un API ya elaborado.
- Soporte de claves foráneas.
- Soporte de transacciones.
- Rápido, seguro y robusto
- Software Libre.

Se han escogido para el estudio los gestores de MySQL y PostgreSQL. Prácticamente son muy similares: aceptan distintos tipos de datos, creación de funciones, transacciones, claves foráneas, disponen de API de Java, etc. No obstante PostgreSQL consume gran cantidad de recursos comparado con MySQL.

Debido a la importancia de que la aplicación sea rápida, el gestor por defecto elegido para AsisT ha sido MySQL.

El gestor de almacenamiento en el que se ha construido es InnoDB [27] ya que es condición indispensable la existencia de claves foráneas. Además soporta transacciones.

C.2.3 Puesto del usuario

El puesto del usuario de la aplicación debe estar conectado a la red local de la empresa de teleasistencia.

Las características de la computadora son:

- Instalado un sistema operativo (Linux, Windows, Mac OS X)
- Máquina de Java versión 1.6
- 512MB ram
- 1Gb de memoria
- Tarjeta de red
- Tarjeta de sonido

C.2.4 Especificación de excepciones

Las excepciones se mostrarán en una ventana emergente que contenga el mensaje de error. Hasta que no pulse "Aceptar", no saldrá de dicha ventana.

En cuanto a los errores que el usuario debe observar son:

- Fechas: día, mes y año válidos. En fecha de nacimiento no puede ser superior al día actual.
- Teléfonos: números válidos no de emergencia serán todos aquellos que empiecen por 6, 8 y 9 y contengan otros 8 dígitos válidos.
- Campos obligatorios: comprobar que se ha rellenado los campos obligatorios.
- Las cadenas son del tamaño adecuado (preferible a error en la base de datos por introducir mal la cadena).
- CP: código postal de 5 dígitos.

Comentario: finalmente no se ha realizado control en la introducción de DNI ya que podría ocasionar problemas al introducir otro tipo de documentos válidos para la identificación: la tarjeta de residencia, o DNI de extranjeros.

C.2.5 Nomenclatura

Base de datos

En la realización de la base de datos se ha utilizado una regla de nombrado que sigue los siguientes patrones:

- Tablas: <nombre de la tabla>.
- Identificador de la tabla: <nombre de la tabla>_id.
- Las claves foráneas deben nombrarse igual que la clave principal a la que hacen referencia.
- Tanto el nombre de tablas como el nombre de los campos, se representa en minúsculas.
- Las palabras de las tablas van separados por "_".

Código fuente

Se han tenido en cuenta las siguientes normas:

- Encabezados en todos los ficheros con nombre de autor.
- Constantes en mayúsculas.
- Nombre de clases: estilo UpperCamelCase. Ejemplo: ClienteDAO.java.
- Nombre de procedimientos: estilo lowerCamelCase.. Ejemplo: loadTable().

C.3 Actividad libre: identificación de subsistemas de diseño

C.3.1 Diseño de módulos de apoyo

Partimos de los subsistemas localizados en el anexo del análisis, ANEXO B. Desde este punto, se ha localizados los módulos que ayudan a la aplicación a proporcionar mejoras, independientes a la funcionalidad:

- Módulo de recursos: para que la aplicación sea intuitiva, se necesitarán iconos que aporten dinamismo.
- Módulo de traducciones: aporte diferentes traducciones de la aplicación.

- Módulo de pruebas: contenga las pruebas de la aplicación de forma que estén organizadas y agrupadas en un mismo entorno.

C.4 Actividad libre: identificación de mecanismos genéricos de diseño

En esta actividad se han identificado los patrones que facilitan el mantenimiento y reduzcan la complejidad del sistema.

C.4.1 Aislamiento de funcionalidad

Design Pattern VO (Value Object) – DAO (Data Access Object)

El objetivo de este patrón, es aislar la aplicación en capas de forma que el sistema sea escalable y fácil de mantener [20].

Para ello la mejor solución es aislar la lógica en clases diferentes:

- Interfaz: clase con la que el usuario interacciona
- InterfazAction: tiene la lógica de negocio del interfaz.
- ClaseVO: clase extraída del diagrama de clases. Sigue el patrón de diseño VO o también conocido como DTO: data transfer object [20]. No tiene funcionalidad, sólo almacenar y devolver su propia información.
- ClaseDAO: patrón DAO [20]. Abstrae las tareas propias de la base de datos para cada clase VO: insertar, modificar, eliminar y búsquedas.

El siguiente diagrama, Figura C-2, muestra la distribución de estas capas.

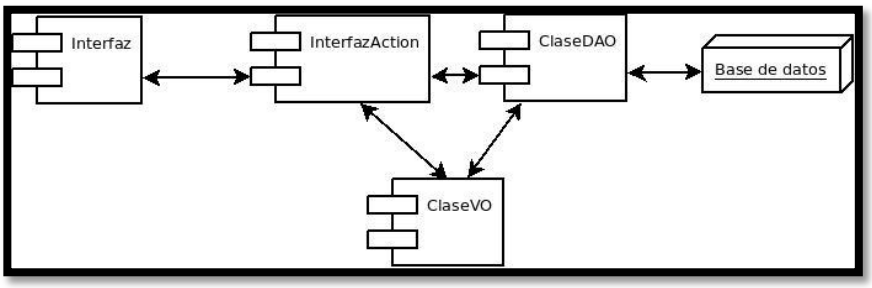


Figura C-2 VO-DAO

Design Patter *Singleton*

Este patrón debe está diseñado para no crear objetos pertenecientes de una clase. El objetivo es que la clase solo tenga una instancia y se use de forma global [20]. De esta forma, se pueden acceder a los métodos de la clase, sin crear el objeto, ni invocar al constructor.

Algunas de las clases en las que se ha usado este patrón es DAOFactory, ApplicationContext y JMSQueue.

Design Pattern DAOFactory

DAOFactory es la consecuencia de la utilización de dos patrones: Abstract Factory [20] y Factory Method [20].

A continuación se muestra un esquema, Figura C-3, de la estructura que sigue el patrón, aplicada a un ejemplo de las clases reales de la aplicación.

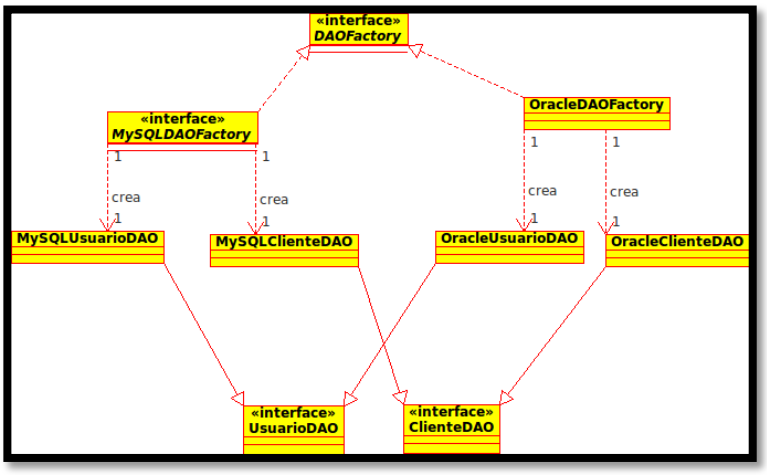


Figura C-3 DAOFactory

La síntesis de este patrón es:

- DAOFactory: interface de todas las factorías.
  - Utiliza el patrón *Singleton* para no instanciar dicha clase.
  - Contiene el método FactoryDefault() para poner por defecto el gestor elegido.
- MySQLDAOFactory & OracleDAOFactory: contiene toda la colección de los DAO del sistema. Implementa la interfaz de DAOFactory.
- MySQLUsuarioDAO & OracleUsuarioDAO: contiene las funciones insertar, modificar, eliminar y búsquedas propias de acceso a la base de datos.
- ClienteDAO & UsuarioDAO: interface cuya implementación está en MySQLClienteDAO & OracleClienteDAO. Son las funciones finales utilizadas en la aplicación.

De esta forma, al llamar mediante las funciones de <NombreClase>DAO, la migración de la factoría queda por cuenta de DAOFactory y no de la interacción entre la interfaz- lógica de la interfaz.

C.5 Actividad DSI 3: diseño de caso de uso reales

Se han diseñado unos escenarios o "casos de usos" de forma que aporte más información a la interacción de objetos. También se han planteado casos genéricos que faciliten su posterior construcción. Para el modelado se ha basado en las herramientas de UML [28].

C.5.1 Escenario: autentificación

1. La aplicación pide nombre y contraseña
2. El operador introduce nombre de usuario y contraseña
3. La base de datos comprueba los datos
4. La BD valida los datos
5. El usuario accede al cuadro de mandos

El siguiente diagrama, Figura C-4, se muestra la interacción entre clases.

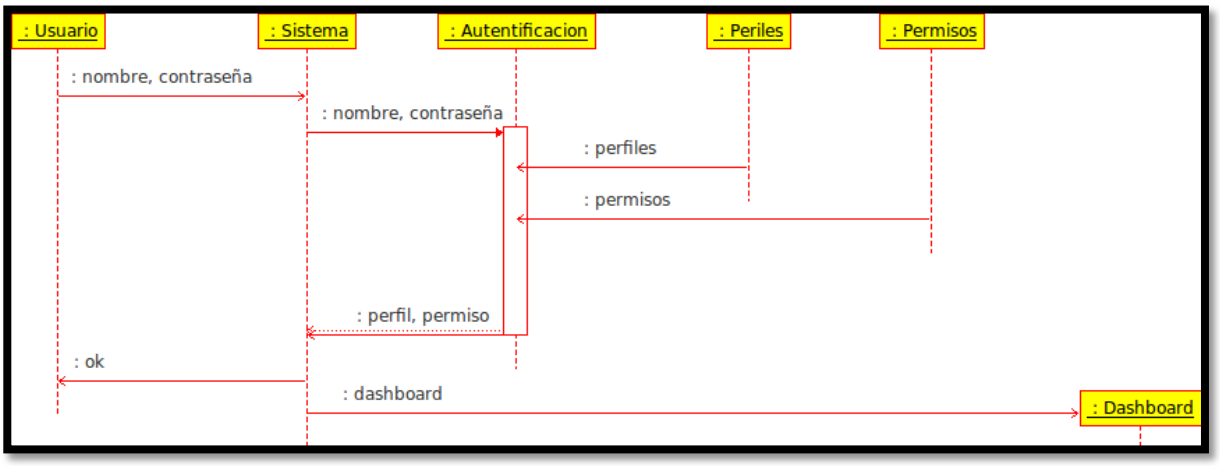


Figura C-4 Escenario autentificación

C.5.2 Escenario: cuadro de mandos

El cuadro de mandos se inicializa:

1. El cuadro de mandos muestra
   - Las alarmas activas
   - Las alarmas atendidas
2. Pulsa sobre una de las alarmas atendidas y ve la información
3. Desde la información pulsa un botón y ve más información del cliente.

El siguiente diagrama, Figura C-5, muestra la interacción entre clases:

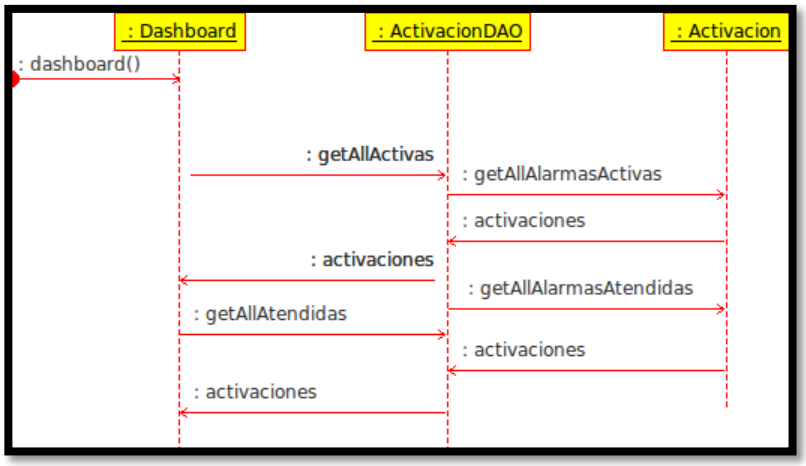


Figura C-5 Diagrama de secuencia del cuadro de mandos

C.5.3 Escenario: atender una alarma

1. El operador observa una alarma entrante generada por un equipo (llamada técnica).
2. El operador acepta la alarma.
3. La alarma pasa a estar activa.
4. La aplicación muestra datos del usuario que pertenece la alarma.
5. El operador observa el tipo de alarma “Por humos”.
6. El operador indica el tipo de actuación “Intervención solamente desde el centro”.
7. Se almacena en la BD el tipo de alarma y actuación.
8. La actuación genera una llamada al cliente.

9. Finaliza la llamada con el cliente, al no obtener respuesta.
10. El operador indica otro tipo de actuación, “Llamada a los bomberos”
11. La actuación genera una llamada a los bomberos.
12. El operador finaliza la llamada.
13. La alarma pasa a estar inactiva.

El siguiente diagrama, Figura C-6, muestra la interacción entre clases:

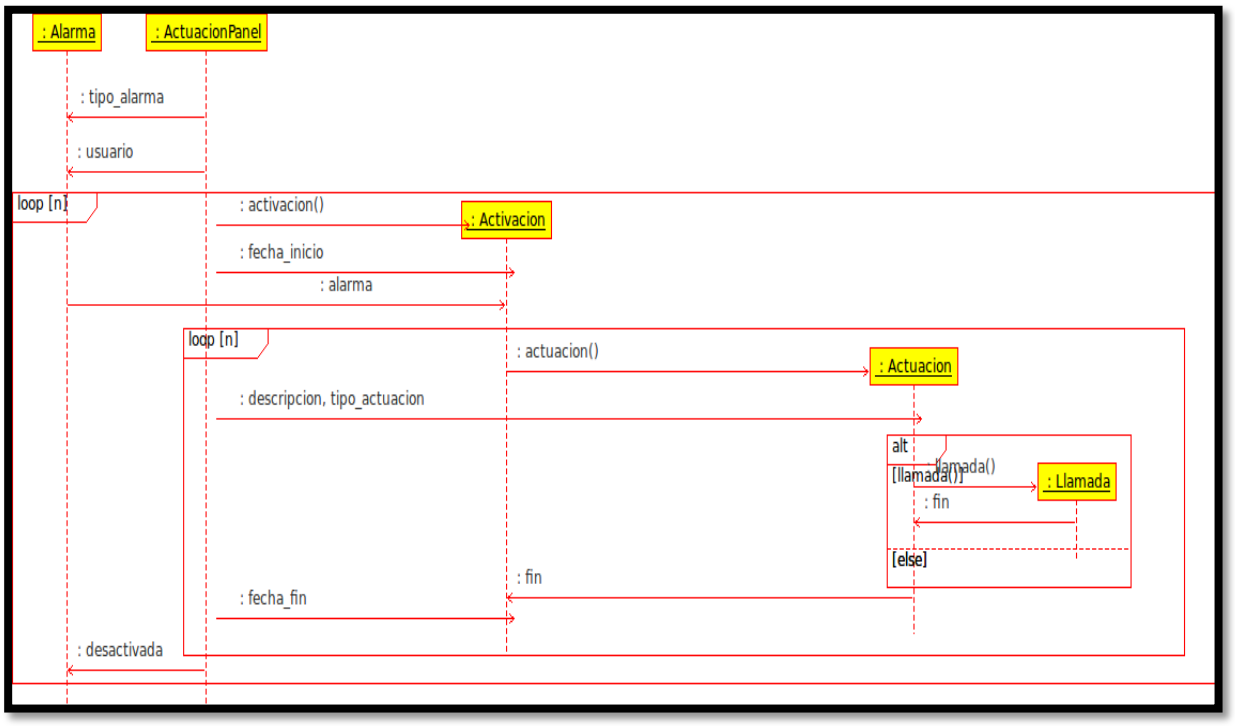


Figura C-6 Diagrama de secuencia atender una alarma

El siguiente diagrama de actividad, Figura C-7, completa la información de clases de atender una alarma.

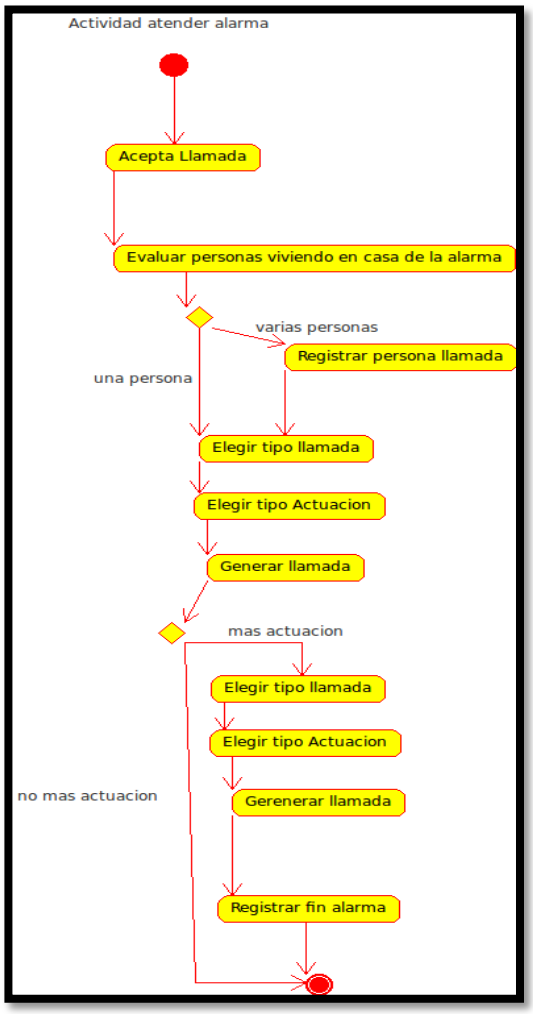


Figura C-7 Diagrama de actividad atender una alarma

C.5.4 Escenario genérico: eliminar elemento de un listado

1. Mostrar la tabla de datos:
    a. El usuario pulsa el botón agregar.
        i. Se inicia una ficha vacía.
    b. El usuario pulsa una fila de la tabla y el botón modificar.
        i. Se inicia una ficha con los datos cargados del elemento a eliminar.
2. Se actualiza la tabla.

El siguiente diagrama, Figura C-8, muestra el modelado de la lista a mostrar:

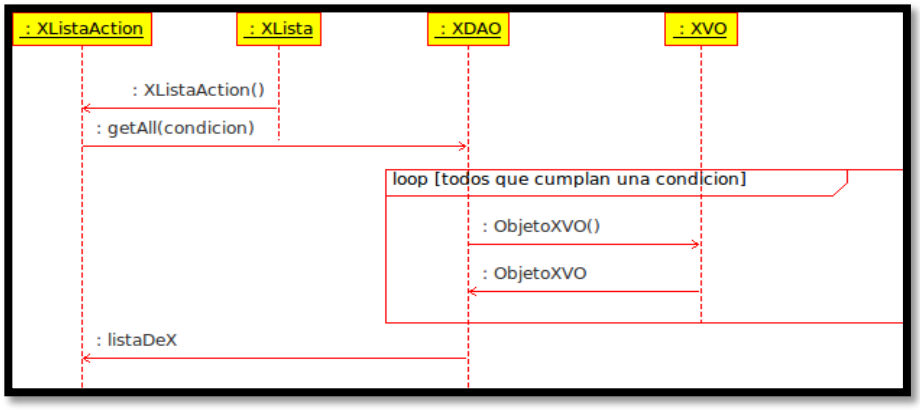


Figura C-8 Diagrama de secuencia mostrar lista

C.5.5 Escenario genérico: agregar/modificar un elemento de un listado

En una ficha pueden ocurrir los siguientes eventos:

1. Sea una ficha vacía o rellena:
    a. El usuario pulsa guardar:
        i. Se guardan los datos en la base de datos.
    b. El usuario pulsa borrar:
        i. Se eliminan los datos de la base de datos.
2. Desaparece la ficha.

Dentro de la ficha, siguiendo el escenario, el usuario pulsará guardar o borrar. En caso de que pulse guardar, si ha venido por una acción previa, la ficha debe estar vacía sino, carga el objeto en la ficha, observar la Figura C-9.

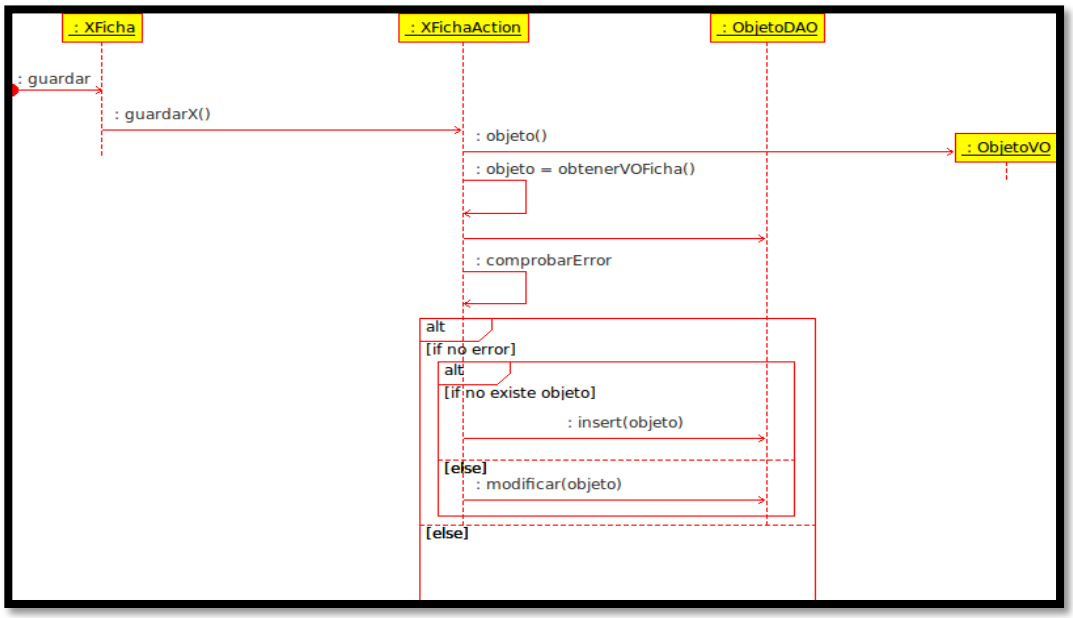


Figura C-9 Diagrama de secuencia guardar ficha

Por último se adjunta el diagrama de la inserción de un nuevo usuario, Figura C-10. Como se puede observar, combina los escenarios genéricos descritos.

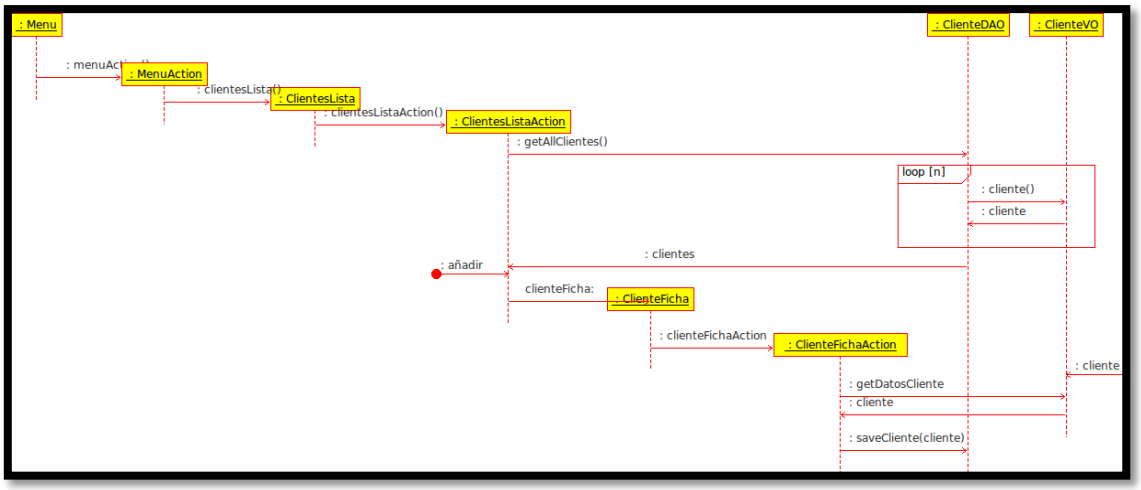


Figura C-10 Diagrama de secuencia nuevo usuario

C.5.6 Escenario: aviso alarma

Por un lado, la aplicación tendrá que avisar a los usuarios de dos eventos:

- Alarma nueva
- Alarma capturada por un usuario.

La Figura C-11 muestra la interacción del sistema cuando recibe dicho aviso de alarma.

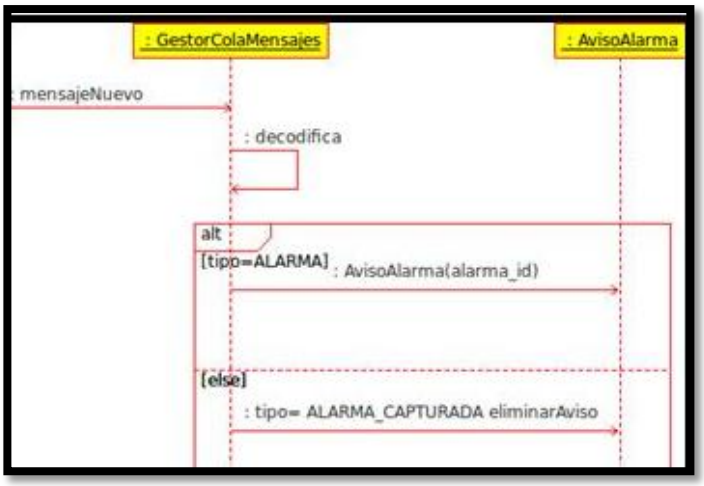


Figura C-11 Diagrama de secuencia aviso alarma

Una clase será la encargada de gestionar el tipo de mensaje recibido y realizar la operativa de:

- Mostrar aviso de alarma
- Quitar aviso de alarma

C.5.7 Escenario: realizar llamada

El siguiente diagrama, Figura C-12, muestra la interacción para la realización de una llamada.

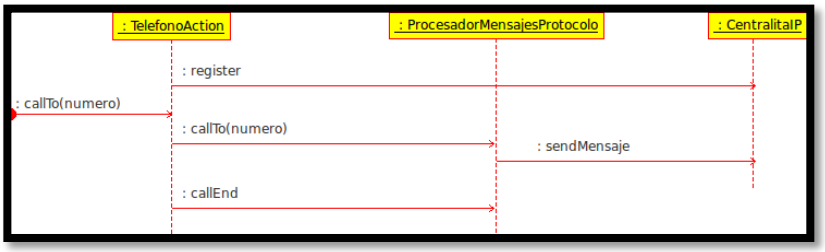


Figura C-12 Diagrama de secuencia realizar una llamada

En el siguiente caso se distingue una clase que tendrá que ser la receptora del número de teléfono al que hay que llamar (Por ejemplo: TelefonoAction.java) y una clase que se encargue de establecer la conexión, es decir, mantener una comunicación con la centralita IP.

La clase intermedia con la centralita IP, se encargará de traducir la operativa de la aplicación con el protocolo de señalización de la centralita.

El escenario de aceptar una llamada se muestra en la Figura C-13. Aceptar la llamada consistirá en mandar una llamada al número del que proviene.

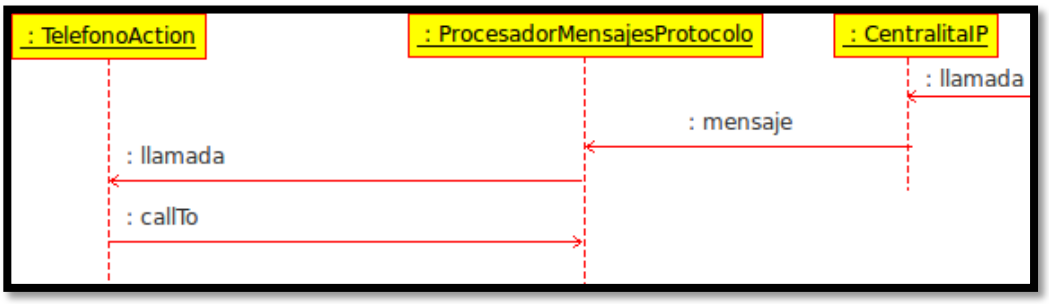


Figura C-13 Diagrama de secuencia aceptar una llamada

En este caso, la clase intermedia se encargará de señalizar el paso de mensajes para establecer la comunicación, siguiendo el protocolo de la centralita,

C.5.8 Conclusión escenarios

Se ha observado unos escenarios genéricos: listar, modificar, agregar y borrar clientes, usuarios, personas de convivencia, viviendas, etc, asociar, desasociar empresas sanitarias, médicos, hospitales, etc.

Se recomienda realizar unas plantillas para agilizar la construcción del sistema. En los patrones de diseño se observó un diagrama de componentes acerca de la relación que tendrán las clases. No obstante, en el siguiente diagrama se puede observar el paso de mensaje que experimentará con la interacción de fichas. Una ficha es una interfaz más, no obstante para cada manipulación de cada objeto con la interfaz se creará una ficha.

En la Figura C-14 se representa la interacción de clases para el patrón localizado.

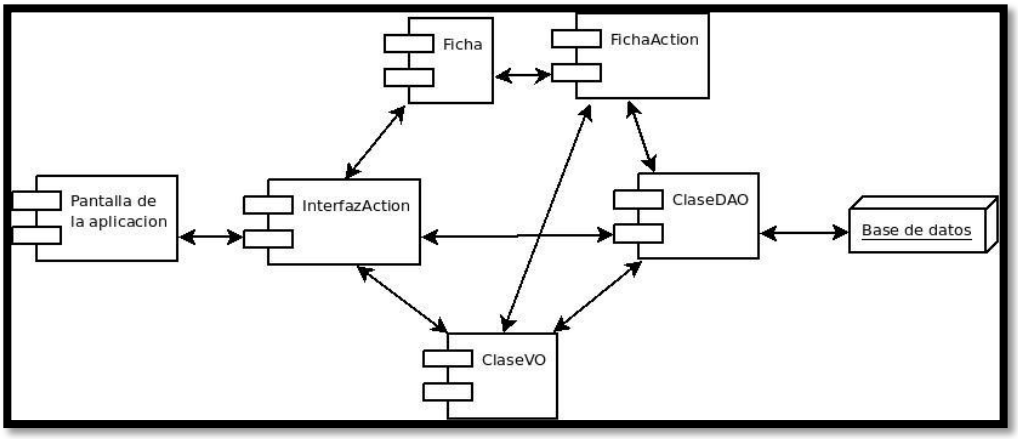


Figura C-14 Interfaz - InterfazAction

C.6 Actividad DSI 4: diagrama de clases

En el análisis del sistema se analizó y elaboró un diagrama de clases. Llegado a este paso previo a la implementación, es necesario tener un modelo consistente para realizar los menos cambios posibles.

Para ello, el diagrama de clases realizado en el análisis B.7, se modifica pensando en la base de datos y en la ampliación de funcionalidad.

En la aplicación, se aporta información al usuario de la que tendrá que elegir. Por ejemplo “tipología del cliente”, “tipo de alarma”, etc. En dichos casos y tal como aparece en los prototipos de las ventanas, aparecerán desplegables.

Un desplegable es una lista de un identificador y una cadena. Para abordar la solución de mostrar información estática es necesario crear más clases para ellos.

Por ejemplo, un cliente tiene un atributo que es "cliente_tipo" anteriormente definido como cadena. Si tenemos en cuenta que para insertar un cliente, será necesario que el usuario seleccione de una lista desplegable, un tipo y el tipo, proviene de una lectura de la base de datos, entonces conviene que el tipo cadena, pase a ser un objeto, que esté contenido en el cliente.

Teniendo en cuenta los patrones de diseño definidos en este mismo anexo, estos definen las clases VO, y por lo tanto hacen de contenedores. Será necesario implementar sus homólogos DAO que interaccionen con la base de datos.

La Figura C-15, contiene el diagrama de clases. Para simplificar el diagrama, la clase teléfono sólo aparece una vez. Falta su enlace con hospital, empresa, médico,

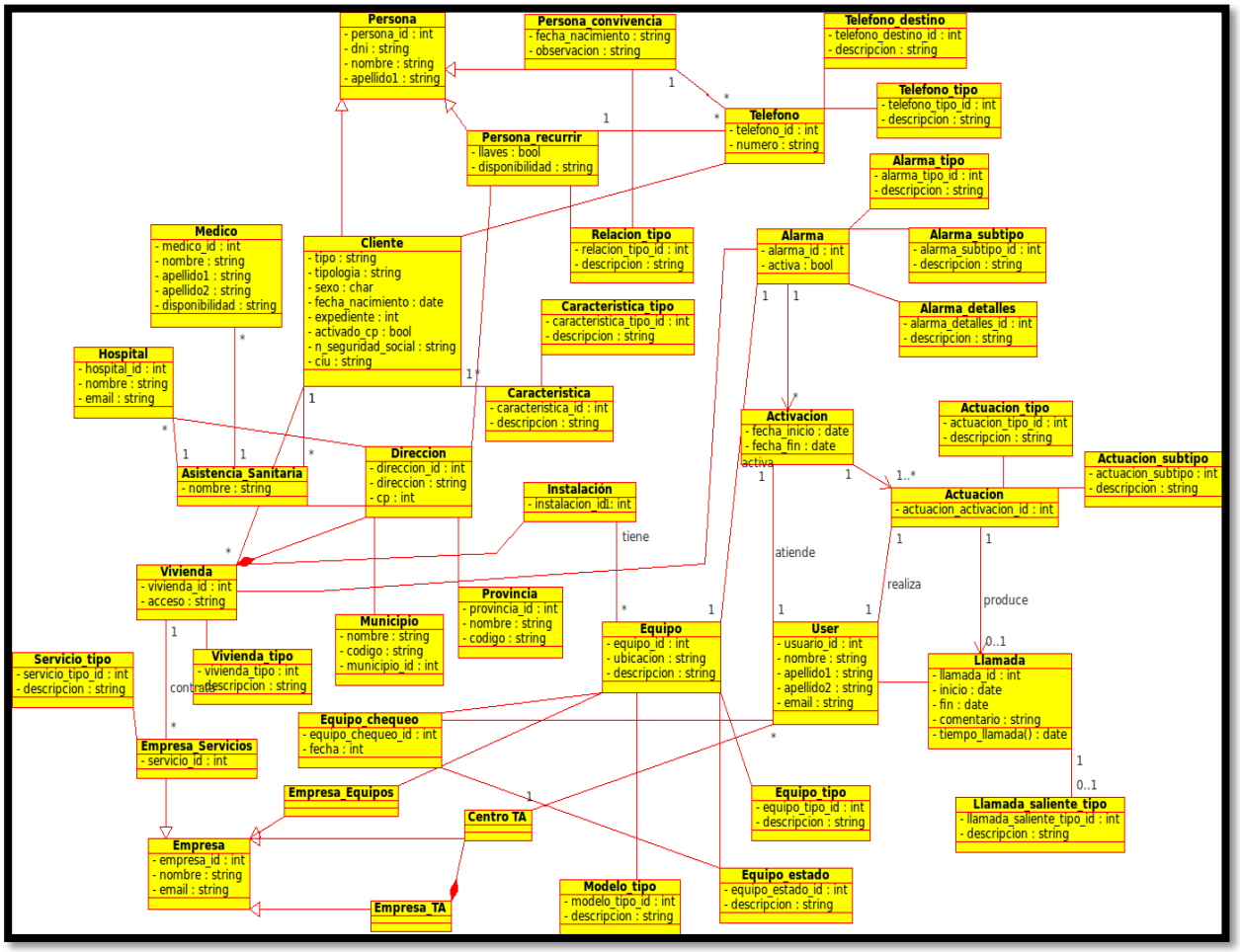


Figura C-15 Diagrama de clases tras diseño

C.7 Actividad DSI 10: pruebas

En este apartado se va a definir un estándar de pruebas para ponerlo en marcha en las tareas de implementación.

Las pruebas se realizarán después de:

- La construcción de un módulo dentro de un subsistema.
- El cambio de algo que ya se había testeado.
- La integración entre módulos.

- La integración entre subsistemas.

La herramienta utilizada para la elaboración es JUnit [19]. Gracias a la integración de NetBeans con JUnit, facilita la tarea de pruebas por clase y permite mantener consistencia de la aplicación, después de realizar cambios.

En fichas:

Test de introducción de datos:

- Comprobar las restricciones de los campos. Se han definido en la declaración de excepciones, en el apartado C.2.4.
- Lea y cargue bien un VO para diferentes valores.

En DAO:

- Realizar las opciones de insertado, modificación y eliminación por varios VO.
- Pruebas para cada búsqueda aplicando distintos valores.

En Listas:

- Probar para diferentes listas se muestra bien la información.

En aviso alarmas:

- Probar que recibe distintas alarmas.

En teléfono:

- Probar para distintos números de teléfonos que se producen llamadas salientes.
- Probar que recibe llamadas entrantes.

C.8 Conclusiones

En esta parte del desarrollo del sistema se han tomado decisiones respecto a lo analizado y sintetizado en el anexo anterior. Los objetivos que inicialmente se han propuesto, han sido desarrollados y argumentados en este documento.

En el diseño se ha definido:

- Patrones de diseño: la existencia de patrones que son de utilidad para las necesidades de la aplicación.
- Funcionalidades comunes entre clases: patrones propios localizados en el sistema.

Gracias a la organización planteada, permitirá facilitar el mantenimiento, ayudar en la construcción y ante todo: ahorrar tiempo.

UNIVERSIDAD DE ZARAGOZA

Centro Politécnico Superior

Ingeniería Informática

ANEXO D: CONSTRUCCIÓN DEL SISTEMA DE INFORMACIÓN

[4/5]

PFC: DESARROLLO DE UNA APLICACIÓN PARA LOS OPERADORES DE TELEASISTENCIA INTEGRADA EN TELEFONÍA IP

AUTORA: LAURA LACARRA ARCOS

DIRECTOR: ROBERTO CASAS MILLÁN

PONENTE: JESÚS ALASTRUEY BENEDÉ

Departamento Informática e Ingeniería de Sistemas

Zaragoza, Julio 2010

ANEXO D CONSTRUCCIÓN DEL SISTEMA DE INFORMACIÓN

D.1 Introducción

Este anexo es continuación del 0 y extensión de la explicación de la memoria.

Se explica el qué se ha implementado para que todas las conclusiones del análisis y el diseño hayan sido albergadas en el sistema. La metodología está basada en Métrica 3 [30].

La forma de abordar la explicación ha sido por orden de implementación.

El objetivo que se persiste en este documento es explicar:

- La tecnología utilizada en la implementación.
- Las soluciones implementadas de los subsistemas.
- El uso de los patrones de diseño.
- Los APIs en los que se ha sustentado la aplicación.
- La implementación de los módulos.

D.2 Preparación del entorno de generación

Puesto de trabajo personal:

- Sistema Operativo Ubuntu 9.4
- Herramienta de programación para java: NetBeans 6.8 para Java [13].
- Herramienta de programación para PHP: Aptana [14].[14]
- Herramienta de bases de datos: MySQL Query Browser.[44].
- Máquina virtual Java 1.6 [6]

El puesto de trabajo está conectado a la red local de la empresa Diaple NetWorking. En el servidor se ha instalado y configurado:

- MySQL [9]: gestor de bases de datos
- Servidor Apache: programar script PHP y comprobarlo.
- Broker ActiveMQ : agente de mensajería

D.3 Bases de datos

La primera implementación corresponde a la base de datos. Se han creado los archivos creaBD.sql y datosBD.sql basados del esquema entidad relación explicado en el análisis, sección B.6.

Para interactuar con MySQL se ha utilizado la herramienta MySQL Query Browser v1.2.12. La Figura D-1 muestra el programa.

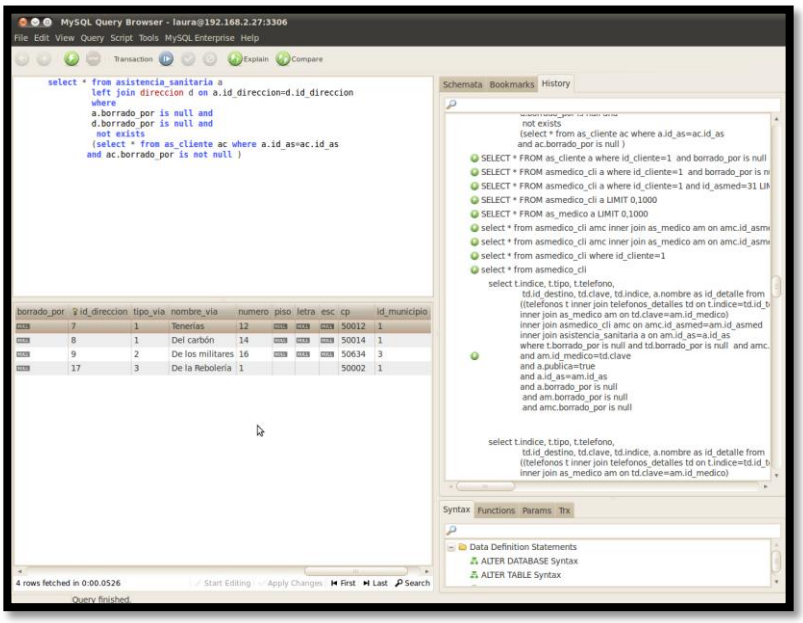


Figura D-1 Herramienta MySQLQuery Browser

D.4 Generación del código de los componentes y procedimientos

El sistema está desarrollado con la herramienta de NetBeans 6.8 [13] y la máquina virtual de java 1.6. En el inicio se hicieron las siguientes actividades:

- Adjuntar las librerías necesarias.
- Organizar el proyecto en subsistemas
- Implementar *main*: creación de la estructura principal del sistema.

Partiendo de las conclusiones del análisis y el diseño, se ha organizado AsisT en subsistemas. El orden en que se han implementado es: subsistema de bases de datos, subsistema de información, subsistema de aviso de alarmas y subsistema teléfono.

Una vez realizado uno, se ha integrado con los anteriores y realizado las pruebas pertinentes. Los pasos en la implementación de un subsistema han sido:

- Por cada módulo contenido en el subsistema:
  - Implementación de un módulo.
  - Generación y ejecución de las pruebas con JUnit [19].
  - Integración con el resto de módulos contenidos.
  - Generación y ejecución de las pruebas.
- Integración del subsistema con la clase principal (*main.java)*, módulos o subsistemas implementados.
  - Generación y ejecución de pruebas.

D.5 Organización del sistema en paquetes

El proyecto AsisT está formado por un conjunto de paquetes:

- Paquetes fuente: contienen los paquetes con la implementación del sistema.
- Librerías: conjunto de librerías ya disponibles.
- Librerías de test: contiene todas las clases para la realización de pruebas.

D.5.1 Librerías utilizadas

En este apartado se explica las librerías necesarias para la aplicación.

jcalendar-1.3.2.jar [56]

- Ofrece: interfaz para los datos tipo *Calendar*.
- Utilizado:
  - *JDateChooser*: para elegir una fecha.
  - *JspinField*: para elegir horas y minutos de distancia desde el centro en el que reside un cliente hasta un centro de asistencia.

mysql-connector-java-5.1.7-bin.jar [15]

- Ofrece: JDBC Driver para MySQL.
- Utilizado: establecer la conexión con la base de datos.

substance-lite.jar [16]

- Ofrece: efectos visuales en la interfaz.
- Utilizado: al iniciar la aplicación, para cambiar la apariencia de la aplicación.

diaple-desktop-fw-0.0.18.jar

- Ofrece: librería de interfaz para java.

Esta librería ha sido cedida por la empresa Diaple NetWorking. Aporta funcionalidades fundamentales que se repiten constantemente para aplicaciones distintas:

- Componentes de *Swing* : Extiende la funcionalidad con los métodos:
  - Método *getResourceBounde(String cadena):* carga la traducción de un archivo *properties*[23] dada una clave.
  - Método *getBindName(String cadena):* carga el valor de un atributo de un objeto, a un texto de las interfaces mencionadas.
  - Crear una tabla a partir de una lista de objetos.
  - Cargar un objeto en una ficha.
  - Obtener los datos de una ficha y meterlos en un objeto.
  - Cargar una lista de objetos en una lista desplegable.
  - Dada una clave o identificador de una lista desplegable, se cargue con el valor asociado. Esa clave puede estar en un objeto *(getViviendaVO().getTipo_Vivienda_id()*) o un objeto dentro de otro objeto (getViviendaVO() .getTipo_vivienda() .getTipo_vivienda_id()).
- *ApplicactionContext*: clase encargada de guardar objetos en la sesión, de forma que puedan ser accedidos en cualquier clase. Utiliza el patrón *singleton* [20].
  - Guardar objetos en la sesión, de forma que puedan ser utilizados en cualquier clase, en concreto:
    - El usuario que ha iniciado sesión.
    - Guardar *ParentFrame* de la aplicación, así evitar las ventanas huérfanas cuando se abre una ventana emergente.
    - El tipo de letra por defecto de la aplicación.
    - El acceso al archivo *properties* que contiene las traducciones e iconos

La librería se ha actualizado de forma que soporte más funcionalidad. Alguna de las soluciones implementadas:

- En los componentes de *swing*:

  - Crear una tabla a partir de una lista de objetos VO [20] y una lista de objetos conjunto de varios VO.
  - Cargar en una ficha, un objeto y los objetos que contiene. Ejemplo: objeto DireccionVO contenido en AsistenciaSanitariaVO.
  - Cerrar pestañas.
  - Editar el tamaño de las columnas.
  - Por cada fila de la tabla, guardar un identificador.
  - Ocultar una columna de una tabla.
  - Al hacer "click" en una fila obtener la clave principal de esa fila (*getSelectedItem*()) para poder operar con dicho identificador.
  - Propiedad *maxLength()* en TextField: limitar la longitud de cadena de forma que no permita escribir una palabra en el campo mayor al límite establecido.
- *ModuleManager*: Clase que gestiona un cargador de módulos en caliente. Al arrancar la aplicación, se inicia el fichero *ModuleManager.xml* el cual especifica los módulos a cargar dinámicamente. Los módulos que se añaden, son un conjunto de clases que en vez de tener un *main*, se añaden al núcleo principal. Las ventajas de esta clase son:
  - Añadir más funcionalidad a la aplicación.
  - Mantenimiento del sistema más cómodo.
  - Personalizar la configuración por módulos, según las necesidades de la empresa.

La Figura D-2 muestra el IDE NetBeans en la edición de pantallas. La imagen corresponde a la pantalla expedientes. A la derecha se puede observar la paleta implementada en Diaple_desktop, que extiende a s*wing*.

En la parte inferior, se observa las propiedades de una etiqueta seleccionada. *ResourceBundle* es la función que dado "expediente_app.labels.expediente" obtendrá su traducción "Expediente", presente en el fichero de traducciones.

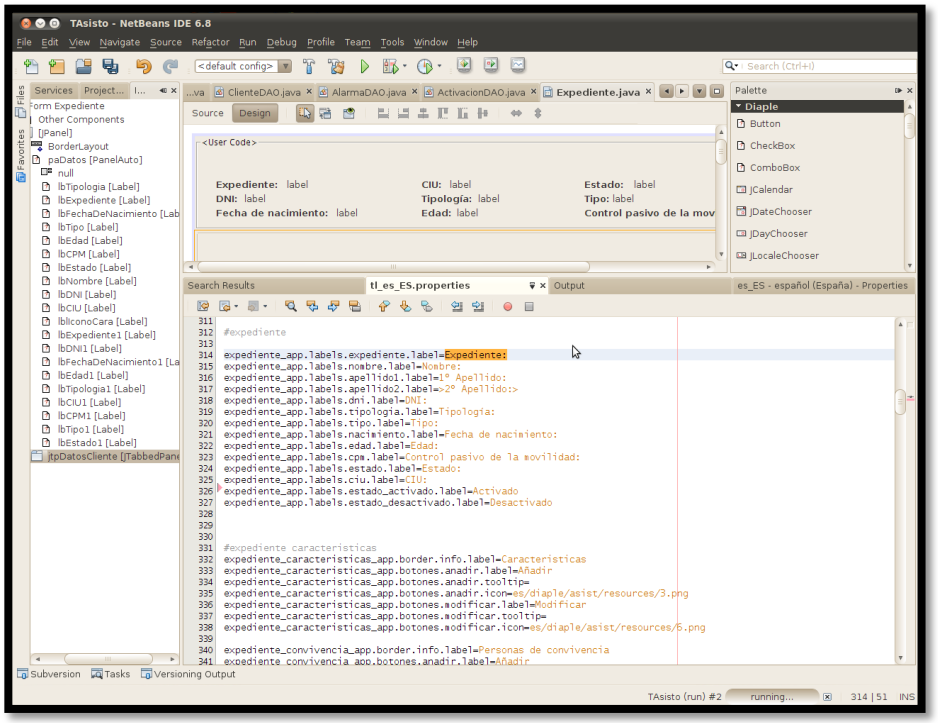


Figura D-2 Uso de diaple-desktop

jms.jar y javax.jms.jar [18]

- Ofrece: librería para interactuar con la cola de mensajería JMS.

- Utilizado: en la gestión de avisos de alarmas.

junit-4.8.1.jar [19]

- Ofrece: librería útil para realizar pruebas de la aplicación.
- Utilizada: en fichas, DAO y paneles.

Librería Jain [1]

- Ofrece: las clases para crear todos los elementos necesarios para una pila SIP [2].
- Utilizada: gestionar el teléfono integrado en la aplicación. Se entra en detalle en el apartado D.8, página 87.

D.5.2 Paquetes del sistema

Los paquetes se han creado siguiendo los subsistemas identificados en el análisis:

- Subsistema de información:
  - es.diaple.asist: *main*, menu y el cuadro de mandos (inicio).
  - es.diaple.asist.dao: todas las clases DAO.
  - es.diaple.asist.vo: todas las clases VO.
  - es.diaple.asist.view: la interfaz y su interacción con el usuario. Instancia a los DAO para que operen con la base de datos.
- Subsistema aviso de alarmas:
  - es.diaple.asist.jms: gestión de la cola de mensajería.
- Subsistema teléfono:
  - es.diaple.asist.sip: gestión del teléfono de la aplicación.
- Soporte:
  - es.diaple.asist.exceptions: captura de errores.
  - es.diaple.asist.resources: iconos de la aplicación.
  - es.diaple.asist.translations: ficheros de traducciones.

En el siguiente apartado se explica la implementación de cada subsistema.

D.6 Subsistema de información

Son todas las clases que intervienen la interfaz y la interacción con la base de datos. A continuación se detalla la distribución e implementación realizada. Cada título corresponde a un paquete implementado. Dentro del paquete, se detalla los módulos y clases desarrollados.

D.6.1 Paquetes

es.diaple.asist

Main.java: Es la clase raíz de toda la aplicación. Inicia:

- El fichero de traducciones (por defecto: en español).
- La letra.
- Inicializa la conexión con la base de datos.
- La subscripción al agente de mensajería.
- *ModuleManager*: carga todos los módulos configurados.
- La autenticación:
  - Si ha sido correcta, inicio de la sesión, carga el menú y el cuadro de mandos.
  - Si no, muestra un mensaje de error y la misma ventana de autentificación.

Menu.java – MenuAction.java – MenuActionGroup: Inicia el menú según los permisos de la persona que ha inicializado sesión.

DashBoard.java – DashBoardAction.java:

- Esta clase contiene el cuadro de mandos o inicio. Por defecto está siempre visible y está implementado de forma que no se pueda cerrar.
- Esta clase instancia los siguientes módulos:
    - es.diaple.asist.view.dashboard.atendiendo: muestra las alarmas que el operador ha capturado y están sin finalizar.
    - es.diaple.asist.view.dashboard.activas: muestra las alarmas activas por el centro y que ningún operador ha capturado.
    - es.diaple.asist.view.dashboard.atendidas: muestra las últimas alarmas atendidas por el operador.
    - es.diaple.asist.view.dashboard.centro muestra la información del centro.

es.diaple.asist.dao

Este paquete contiene todos los DAO de la aplicación. Ejemplos de DAO son: ClienteDAO.java, UsuarioDAO.java, MunicipioDAO.java, TipoCaracteristicaDAO.java, etc.

Se ha seguido el patrón DAOFactory explicado en el apartado C.4 del 0. Estas clases, como ClienteDAO.java, son la interface, mientras que la implementación queda en MySQL<NombreClase>DAO.java, es decir, en MySQLClienteDAO.java.

Se ha implementado en todas las clases DAO las siguientes funciones:

- List<NombreClaseVO> getAll(String sqlFilter,List parameterValues, String sqlOrder): devuelve una lista completa de los VO que no están borrados.
- <NombreClaseVO> get (int id): devuelve el VO de getAll cuya clave es id.

Todos los VO que son utilizados en las listas desplegables como TipoCaracteristicaVO, o TipoViviendaVO, sólo contienen las funciones que se acaban de describir. No se descarta que en posteriores versiones de la aplicación se permita editar alguno de estos tipos de datos.

En todos aquellos VO que puedan ser editables: insertar, modificar o eliminar datos, se han implementado las siguientes funciones:

- <NombreClaseVO> insert (<NombreClaseVO> objeto): inserta el objeto <NombreClaseVO> en la base de datos.
    - Se inserta atributos “creado_por” y “fecha_creación”, para mantener la ley de protección de datos. [12]
    - En caso de que sean varias tablas, como es el caso de un cliente (tabla persona y cliente) se insertan en ambas teniendo en cuenta que el *commit* (cometer la inserción) no se realiza hasta el final.
    - En caso de que <NombreClaseVO> tenga un objeto autoconenido, se ha insertado sólo el índice del objeto, El objeto autocontenido, se inserta posteriormente, teniendo en cuenta *commit*. Por ejemplo un objeto ViviendaVO tiene el objeto DirecciónVO y este objeto, el objeto TelefonoVO. Para insertar una vivienda nueva para un cliente, ViviendaDAO está implementado de forma que:
        1º. Inserta el teléfono (contenido en Dirección) → no realizar *commit*
        2º. Inserta la dirección (contenido en ViviendaVO) → no realizar *commit*
        3º. Insertar la vivienda → realizar *commit*
- void update (<NombreClaseVO> objeto): actualiza todos los atributos de objeto y marca modificado_por y fecha_modificación por los correspondientes valores.

    - Hay casos que un update modifica la asociación existente ente dos tablas: por ejemplo la asociación de un cliente con un médico "update (medico_id, cliente_id). Está función está contenida en el DAO más relevante, evitando la creación de nuevos DAO que entorpecen la legibilidad.
    - También se ha tenido en cuenta la correcta utilización de *commit.*
- Void delete (<NombreClaseVO> objeto): la función borrar (*delete)* establece valor a los campos "borrado_por" y "fecha_borrado". La información nunca se borra de la base de datos.

Implementación de búsquedas:

- List<NombreClaseVO> getAllFrom<Condicion>(Integer id<Condicion>): normalmente, ligado a la función *getAll*. El objetivo es realizar la búsqueda de toda la tabla que cumplan un patrón común. Un ejemplo:
    - List <Persona_conovivienciaDAO> *getAllFromCliente* (Integer cliente_id): obtiene todas las personas de convivencia de un cliente. Ejemplo de búsquedas:
        - Obtener los equipos instalados de un cliente, las características de un cliente, las viviendas de un cliente, los teléfonos de una persona de convivencia, etc.
        - Obtener los nombres de los médicos/hospitales de una asistencia sanitaria, que no tiene asignados el cliente.
        - Obtener los clientes de la aplicación con criterios de búsqueda.
        - Obtener los municipios que pertenecen a una provincia
        - Obtener los modelos de equipos de un tipo y una empresa seleccionados previamente.
- <NombreClaseVO> getFrom<Condicion>(Integer id<Condicion>): cuando se quiere obtener sólo un objeto.
    - Ejemplo: InstalaciónDAO
        - Instalacion getFromCliente(Integer cliente_id): devuelve la instalación de dicho cliente

También contiene las funciones para mostrar la información de una tabla. Estas funciones realizan búsquedas entre varias tablas y construyen la estructura compatible con la función implementada en la librería diaple-desktop-fw-0.0.18.jar, de construir una tabla, a partir de una lista de objetos VO.

- List<List> getTabla<Patron>(Integer id<Condicion>)
    - Ejemplo: ActivacionDAO, getTablaAlarmasFromCliente
        - Devuelve una lista de filas de la tabla de las alarmas. Para obtener la información, se ha realizado una búsqueda de las tablas: alarma (detalles de la alarma) y usuario (nombre y apellidos de la persona que la ha atendido).

es.diaple.asist.exceptions

ErrorManager.java – AsisTException.java: Clases utilizadas para la gestión de errores. En cualquier clase que se produzca un error se ha llamado al constructor AsistTexception(<mensajeError>) que lanza una excepción.

La Figura D-3 Ventana de errormuestra la ventana de error. Se ha utilizado JOptionPane [39].

El mensaje error, sigue las pautas establecidas en la codificación de excepciones.

Las ventajas de gestionar las excepciones es que pueden ser utilizadas para otros fines, como por ejemplo, ser enviadas a un centro de mantenimiento del software AsisT.

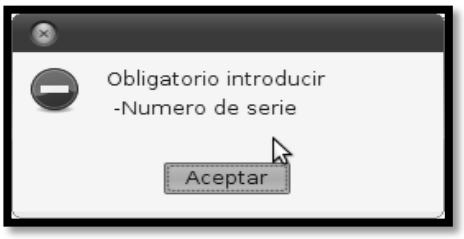


Figura D-3 Ventana de error

es.diaple.asist.view

Este paquete contiene toda la interfaz de la aplicación. Su subdivisión se ha hecho según las conclusiones del análisis. A continuación se explica algunas de las funciones similares que comparten.

En todos los paneles que se muestra un listado de tablas se ha implementado la clase *<nombre>Action.java* con las funciones:

- loadTable<nombre>(): construye y completa la tabla.
    - Obtiene la lista de objetos de la clase <nombre>DAO.java
    - Crea el modelo la tabla.
    - Muestra por pantalla el modelo.
    - Si es la primera vez que se muestra, la primera línea aparece seleccionada.
    - Si ha añadido una nueva fila, está será la última seleccionada.

En caso de que tenga una tabla dependiente:

- loadTable<nombre2>(integer id):
    - Misma funcionalidad que la anterior, pero dependiente del *id* de la fila seleccionada en la primera tabla.
    - Siempre que se actualiza la tabla principal, se actualiza la tabla dependiente.

Si la lista se puede editar: añadir / modificar:

- add<nombre>: consecuencia de pulsar el botón de la interfaz "Añadir".
    - Invoca la ficha de edición específica, cargándola vacía.
- modify<nombre>: consecuencia de pulsar el botón de la interfaz "Modificar".
    - Invoca la ficha cargando los datos del objeto.

Para borrar los datos, el usuario debe acceder a modificar y posteriormente en la ficha, pulsar borrar. De esta forma, el usuario observa los datos, antes de proceder a borrarlos.

Si la lista se puede editar: asociar (*link*)/desasociar (*separate*):

- link<nombre>: invoca la ficha de edición específica, cargando en una o varias listas desplegables las opciones disponibles para asociar de forma que:
    - El cliente no las tenga ya asociadas.
    - Si han sido borradas, las opciones vuelven a estar disponibles para asociar.
- separate<nombre>: pregunta confirmación de borrado y procede a desasociar la fila seleccionada.
- modify<nombre>: en este caso, carga una nueva ficha, con la información a editar. Los datos se actualizan de la asociación existente.

En todas las fichas, fichero <nombre>FichaAction se ha implementado:

- *checkError():* localiza los errores de entrada tales como campos en blanco obligatorios o formatos no válidos.
- <nombre_objeto>new(): instancia el objeto en el panel de la ficha. Necesario para introducir un nuevo dato. Posteriormente es necesario hacer *save()* si se desea guardar.
- < nombre_objeto>load(): carga el objeto en el panel de la ficha. Consecuencia de pulsar modificar en el panel del que proviene la ficha.
- < nombre_objeto>save(): guarda o modifica el objeto, detectando:
    - Si se ha cometido un error de entrada
        - Muestra el error por pantalla.
    - Si no hay errores:
        - El objeto es nuevo: realiza *insert*.
        - El objeto ya existe: realiza *update*.
        - En ambos casos, si hay objetos dentro del objeto, los objetos interiores se insertan o actualizan invocando a la función con commit=false. De esta forma, los datos no quedan modificados hasta que el objeto principal no se ha guardado con éxito.
- <nombre_objeto>remove(): elimina el objeto que se ha cargado en la ficha.

Otros detalles:

- Cuando se pulsa el botón "Añadir" en una lista, el botón "Borrar" de una ficha, queda oculto.
- Si se pulsa el botón "Asociar" en una lista, el botón "Desasociar" de una ficha, queda oculto.
- Si se han agotado las opciones para asociar, aparece el mensaje "no hay más opciones disponibles" en la lista desplegable y el botón "Guardar" queda oculto.
- Cuando se guarda un nuevo dato, la tabla o tablas, se actualizan.
- Si la tabla principal está vacía, las tablas dependientes o paneles dependientes, quedan inhabilitadas para operar.
- Cuando se añada o asocie un nuevo dato, la tabla dependiente queda lista para introducir nuevos datos que dependen directamente de la principal.

A continuación se aborda cada módulo implementado en es.diaple.asist.view, especificando ejemplos y funcionalidad de los mismos.

es.diaple.asist.view.autentificiacion

Autenticiación.java – AutentifiacionAction.java: Encargado de autentificar el usuario de la aplicación.

Se ha utilizado el algoritmo de encriptación MD5 para almacenar las contraseñas. La codificación se realiza en la aplicación, no en la base de datos. Esto permite que los datos viajen seguros en la red.

es.diaple.asist.view.clientes

ClientesFicha.java – ClientesFichaAction.java: añadir / modificar / eliminar un cliente.

ClientesLista.java – ClientesListaAction.java: muestra los clientes que han contratado al menos una vez, el servicio de teleasistencia, en función de unos criterios de búsqueda: expediente, CIU, nombre y apellidos, sexo, estado, disponibilidad de CPM, municipio, y provincia.

- Al pulsar "Buscar":
    - Identificación de los campos de búsqueda que han sido rellenados en los campos de texto o lista desplegable.

- Mediante un *hashMap* se ha introducido la información que ha sido escrita o seleccionada.
- ClienteDAO decodifica la información del haspMap en una cadena de búsqueda.
- La búsqueda se anida a la búsqueda corriente de *getAll* para obtener la lista filtrada.

es.diaple.asist.view.usuarios

UsuarioFicha.java – UsuarioFichaAction.java: dar de alta, de baja o modificar a un usuario de la aplicación.

UsuarioLista.java – UsuarioListaAction.java: muestra los usuarios pertenecientes a una empresa, que pertenecen a distintos centros.

PerfilFicha.java – PerfilFichaAction.java: muestra los perfiles asignados al usuario. Incorpora botón "asociar" que invoca a ficha PerfilAsociarFicha.java

PerfilAsociarFicha.java – PerfilAsociarlFichaAction.java: muestra una lista desplegable de perfiles que no tiene asociado el usuario. Agrega el seleccionado si pulsa "guardar".

es.diaple.asist.view.llamadas

LlamadasPanel.java – LlamadasPanelAction.java: muestra una lista de llamadas salientes y recibidas del usuario que ha iniciado sesión

es.diaple.asist.view.aviso

AvisoAlarma.java – AvisoAlarmaAction.java: ventana emergente que se instancia en es.diaple.asist.jms, cuando recibe una alarma.

También se utiliza en el view.dashboard.activas como panel de una alarma activa.

La ventana se inicializa con la información del cliente y la alarma. Al hacer "click" en capturar alarma se muestra el panel de actuaciones ActuacionPanel.java, que contiene la funcionalidad de gestionar la alarma.

es.diaple.asist.view.atentida

AtendidaAlarma.java – AtendidaAlarmaAction.java: formato similar a la ventana emergente. Este panel es utilizado en view.dashboard.atendidas.

Al pulsar sobre el enlace se muestra el panel de alarma finalizada: AlarmaPanel.java

es.diaple.asist.view.atendiendo

AtendiendoAlarma.java – AtendiendoAlarmaAction.java: formato similar a la ventana emergente. Este panel es utilizado en view.dashboard.atendidas.

Al pulsar sobre el enlace se muestra el panel de alarma terminada, perteneciente a view.actuacion cargando toda la información.

es.diaple.asist.view.dashboard.atendidas

AtendidasPanel.java - AtendidasPanelAction.java: panel de alarmas atendidas. Crea una pila de alarmas atendidas por un cliente, la primera es la última atendida. Cada alarma atendida es la instancia de la clase AtendidaAlarma.java

El número de alarmas que se carga es 10.

es.diaple.asist.view.dashboard.activas

ActivasPanel.java - ActivasPanelAction.java: crea el panel de alarmas que no han sido capturada por ningún usuario.

Al hacer "click" la alarma pasa a ser capturada y se muestra el panel de gestión de alarma.

Este panel se refresca cada 5 segundos si el usuario está viéndolo. El refresco de activas se ha implementado utilizando la clase *TimerTask* [41] de la siguiente manera:

- Se ha añadido a la sesión el panel de alarmas Activas para modificarlo desde otra clase.
- Se ha creado una clase *RefreshAlarmasActivas*, que extiende *TimerTask*.
- La clase sobreescribe la funcion "run" de *TimerTask*:
    - En el "run" se ha implementado que muestre el listado de alarmas activas.
- Para programar el refresco, en el panel de alarmas activas, se ha hecho uso de la función "Schedule" de java.util.Timer, indicando la fecha de inicio, el periodo de tiempo y el objeto new *RefreshAlarmasActivas()*.

El refresco es necesario, ya que las alarmas activas no dependen de un solo operador, sino todos los que están operativos en el centro.

También se refresca cuando el operador captura una alarma y pulsa sobre la pestaña de "Inicio".

es.diaple.asist.view.dashboard.atendiendo

AtendiendoPanel.java - AtendiendoPanelAction.java: crea pila de alarmas que el usuario ha capturado y aún no ha finalizado.

Al hacer "click" en el enlace se muestra el panel de gestión de alarma.

es.diaple.asist.view.dashboard.centro

Contiene la información del centro. Los números de teléfono, son enlaces cuya activación, inician una llamada por el *softphone*.

es.diaple.asist.view.alarmas

AlarmasPanel.java – AlarmasPanelAction.java: muestra una tabla de alarmas activas y finalizadas por el usuario que ha iniciado sesión. Al hacer doble "click" sobre la fila, si la alarma está activada se abre el panel de gestión de alarmas, sino, se muestran los detalles de la alarma finalizada.

es.diaple.asist.view.actuacion

ActuacionPanel.java – ActuacionPanelAction.java: son las clases encargadas de la gestión de una alarma.

El panel está dividido en tres partes:

- Información del cliente al que pertenece la alarma: el nombre contiene un enlace al expediente
- Información de la alarma: tipo alarma / subtipo alarma / detalles alarma.
- Información de las actuaciones: tipo actuación y subtipo de actuación.

Si la alarma es técnica difiere de si es una llamada:

- Técnica: al ser alarma técnica, se conocen y se muestran los dos niveles de información de la alarma: Tipo y subtipo de alarma. Ejemplo:
    - Tipo Alarma: Humo
    - Subtipo Alarma: Fuego
- Llamada: al ser una alarma con llamada, sólo se conoce el tipo de alarma, y aparecen dos listas desplegables a rellenar del tipo de alarma. Las listas, son dependientes según la opción elegida.
    - Se ha implementado Listentener – ActionListener en el control de las listas desplegables.

Según la actuación elegida, el programa busca los teléfonos relacionados, procedentes de los datos del cliente juntos los teléfonos almacenados por defecto (bomberos, empresa, etc).

Cuando se pulsa "ejecutar actuación", se muestra un cuadro de texto, para que el operador introduzca sus conclusiones de la asistencia. En este momento, las listas desplegables de tipos de alarmas y actuaciones quedan inhabilitadas.

Una vez que el usuario pulse "finalizar actuación", puede finalizar la alarma o puede realizar más actuaciones (los campos de actuaciones, vuelven a estar habilitados).

Aunque una alarma no tenga actuación, la pantalla no deja avanzar hasta que se haya marcado tal propósito y den por finalizada la alarma.

AlarmaPanel.java – AlarmaPanelAction.java: muestra la alarma finalizada. Desde esta pantalla se puede acceder al expediente del cliente y reactivar la alarma.

Este panel está dividido en tres partes:

- Información del cliente
- Tabla de información de cada captura: fecha de inicio / Fin y usuario que la ha capturado.
- Tabla de actuaciones por cada captura: tipo de actuación, teléfono al que se ha llamado, etc.

En el caso de que un usuario cierre la pestaña o la aplicación podrá acceder nuevamente a la alarma mediante las pantallas:

- Cuadro de mandos: panel de alarmas que está atendiendo.
- Historial alarmas: disponible desde el menú de la aplicación.
- Expediente del cliente: en la pestaña Alarmas.

Cuando se reactiva la alarma, ActuacionPanel.java se carga con los datos de la alarma. Se inhabilitan los campos de tipo de alarma, ya que una vez seleccionados, las actuaciones sólo pertenecen a ese tipo de alarma.

es.diaple.asist.view.expedienteCliente

Contiene la implementación completa del expediente de un cliente. Este paquete tiene 59 ficheros distribuidos:

- Al menos una clase por pestaña: ExpedienteViviendas.java – ExpedienteViviendasAction.java
- Ficha correspondiente para editar: ViviendaFicha.java – ViviendaFichaAction.java
- Además se han utilizado las siguientes clases para instanciarlas en más de un lugar del expediente:
    - TelefonoPanel.java – TelefonoPanelAction.java
    - DireccionPanel.java – DireccionPanelAction.java

Vamos a generalizar explicando la implementación en varias clases:

Expediente.java – ExpedienteAction.java

- Crea la instancia de todas las pestañas.
- Carga los datos de la primera pestaña.
- Para no sobrecargar la base de datos, sólo se inicializa la primera pestaña. Cada pestaña a la que se accede, su índice dentro del conjunto de pestañas, es introducido en un *ArrayList()* de forma que controle que no se vuelva a actualizar innecesariamente.

ExpedienteConviviencia.java – ExpedienteConvivenciaAction.java

- Muestra la lista de personas que conviven con el cliente.
- Se puede editar: añadir, modificar y borrar.
- Si se selecciona una fila de una tabla de la margen superior, es decir, las tablas que tienen tablas dependientes, la información de la tabla inferior se modifica, en este caso, al seleccionar distinta persona se cargan los teléfonos en la parte inferior.

ExpedienteAS.java – ExpedienteASAction.java

- Muestra la lista de asistencia sanitaria. En la parte inferior y según la asistencia sanitaria pulsada se muestran los datos de tiempo de las viviendas al centro de asistencia, teléfonos de la asistencia y médicos y hospitales asociados a la entidad sanitaria que dan servicio al cliente.
- Dispone de cuatro pestañas en la parte inferior que corresponden a cuatro instancias de diferentes clases: ViviendasAsPanel.java, TelefonosPanel.java, MedicosPanel.java, HospitalesPanel.java. Todas se inicializan al pulsar en la pestaña de asistencia sanitaria.
- Cuando se desasocia un Centro de asistencia, se eliminan también todas las asociaciones que tiene el cliente con el médico y los hospitales. La implementación de esto se ha realizado en ASFichaAction.java.
- Las asociaciones del tiempo entre las viviendas del cliente y la asistencia sanitaria, no se eliminan. En caso de eliminar la asistencia sanitaria por error, el tiempo permanece.

D.7 Subsistema aviso alarmas

D.7.1 Solución al aviso de alarmas

La solución que se ha adoptado para el aviso de alarmas es la utilización de un cliente de mensajería en forma publicador y suscriptor.

Se ha creado un tema en el agente llamado “AsisT”, en el que todos los operadores, al inicializar la aplicación, quedan subscritos de forma que los mensajes que se escriban en dicho tema, serán leídos por todos.

El agente utilizado es ActiveMQ [24]. Se ha utilizado para avisar de que ha llegado una alarma y comunicar al resto de operadores que se ha capturado.

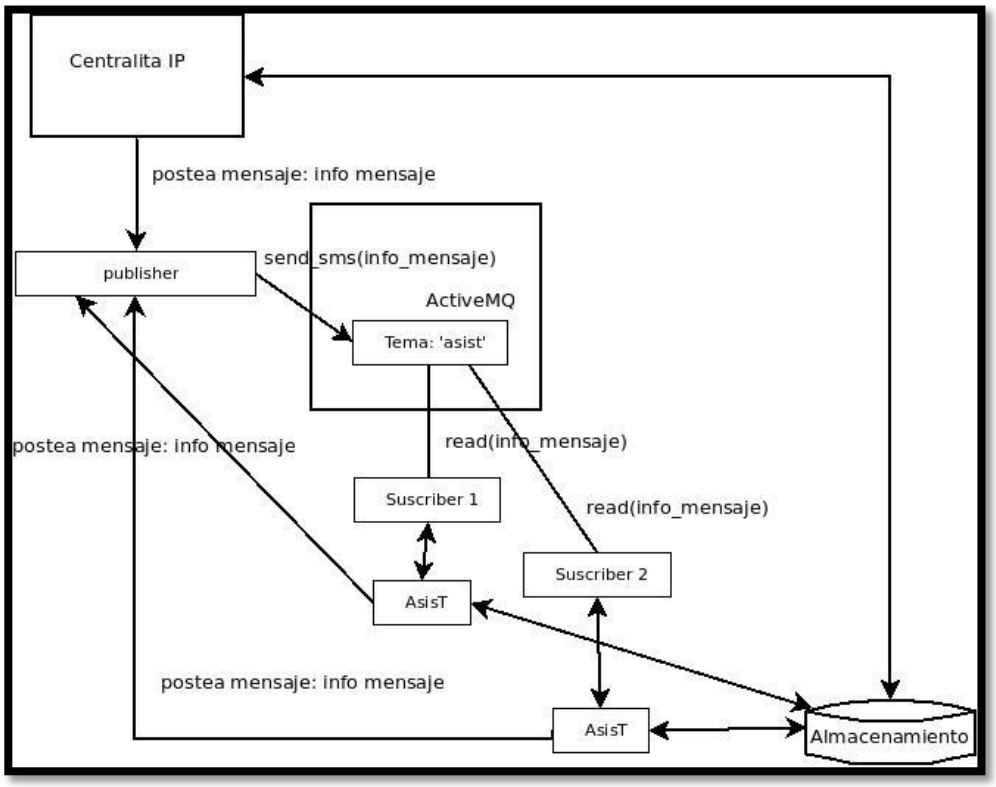


Figura D-4 Funcionamiento publicador y suscriptor

En Figura D-4 Funcionamiento publicador y suscriptor, se observa como AsisT o la centralita IP invoca el publicador para escribir una notificación en el tema de ActiveMQ. Las aplicaciones AsisT que están en funcionamiento, han instanciado a un suscriptor que lee los mensajes de notificación.

D.7.2 Paquetes

es.diaple.asist.jms

JMSQueue.java: esta clase se ha implementado según el patrón de diseño *Singleton*. Al instanciarla se suscribe ActiveMQ con el tema AsisT. Se encarga de gestionar, las funciones que se describen a continuación.

D.7.3 Aviso de nueva alarma

La alarma llega a la centralita IP. Se ha realizado un script en PHP que ejecuta la centralita, cuya funcionalidad es:

- Introducir la alarma en la base de datos
- Escribir un mensaje la cola de mensajería

La Figura D-5 muestra la herramienta Aptana [14] utilizada para la implementación:

- A la derecha aparece en fichero de prueba, el cual inicializa los datos de la invocación de *send_alarma*.
- A la izquierda está el fichero que implementa la función *send_alarma*:
    - Inserta la alarma en la base de datos
    - Notifica el mensaje en el tema: AsisT.alarmas.1.1

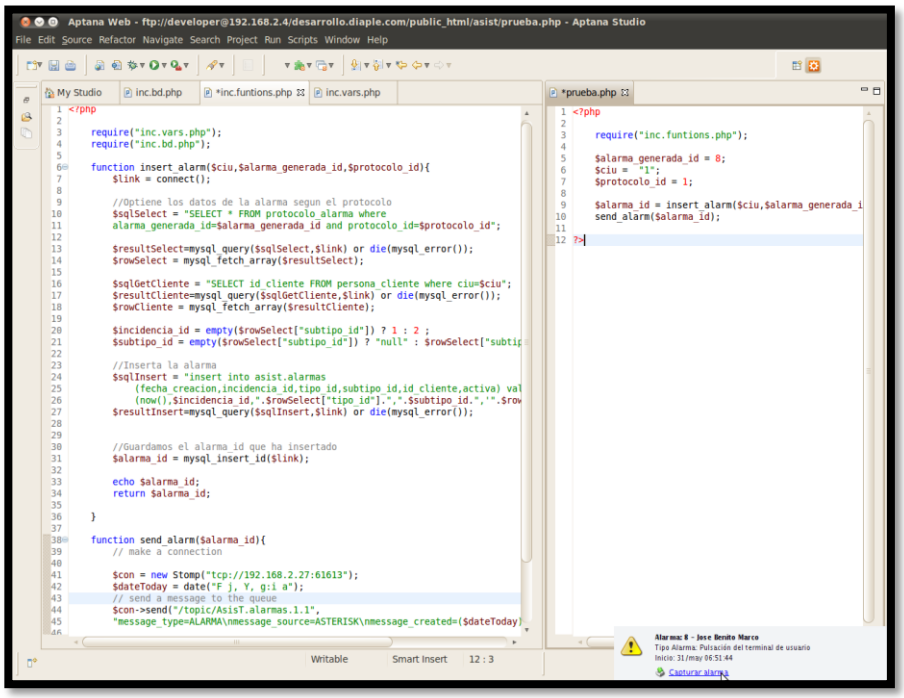


Figura D-5 Herramienta Aptana

El mensaje de aviso de alarma tiene la siguiente codificación:

message_type=ALARMA
message_source=ASTERISK
message_created=(FECHA_HORA_ACTUAL)
alarma_id=1201

La Figura D-6 explica cómo la aplicación es notificada de mostrar un aviso de alarma:

1. La centralita recibe la alarma y la almacena en la base de datos.
2. Invoca al publicador que postea el mensaje de tipo ALARMA en el tema: “AsisT”.
3. Las aplicaciones AsisT, leen por medio del suscriptor el mensaje y muestran el mensaje de aviso.

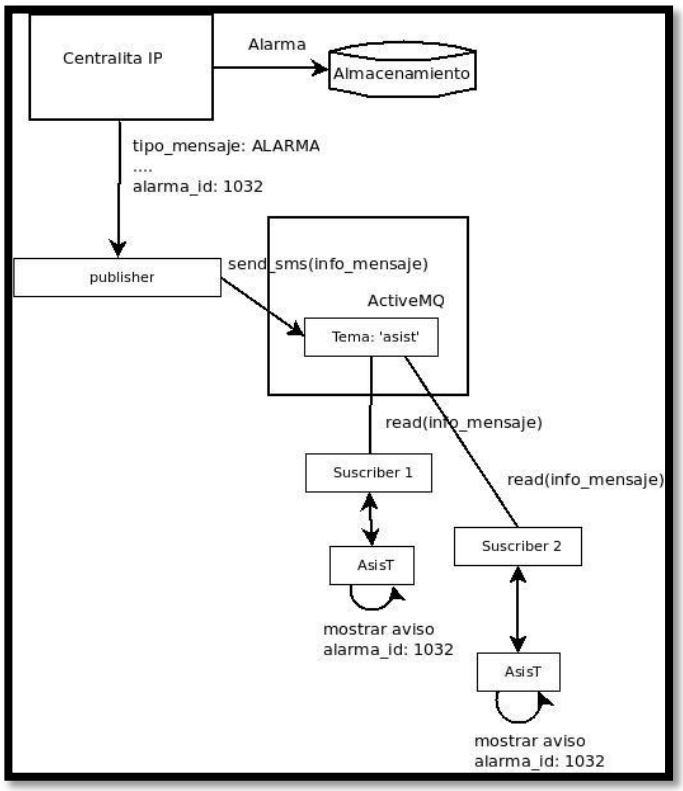


Figura D-6 Aviso de alarma por mensajería

Cuando el operador recibe el mensaje, al ser de tipo ALARMA, la aplicación muestra la ventana emergente de aviso de alarma.

Para acceder a la información de cada variable, se ha hecho uso de la clase *properties* [23] de Java. Los suscriptores acceden a la alarma según la Figura D-7.

1. Leen el mensaje los suscriptores de aviso de alarma.
2. Pulsan capturar. Si la "alarma_id" está
    a. Desactivada: Se inicia a gestión de alarmas y se actualiza el estado como Activa y se muestra el panel de gestión.
    b. Activa: Significa que otro usuario la ha obtenido y por tanto muestra un mensaje de que ya ha sido capturada

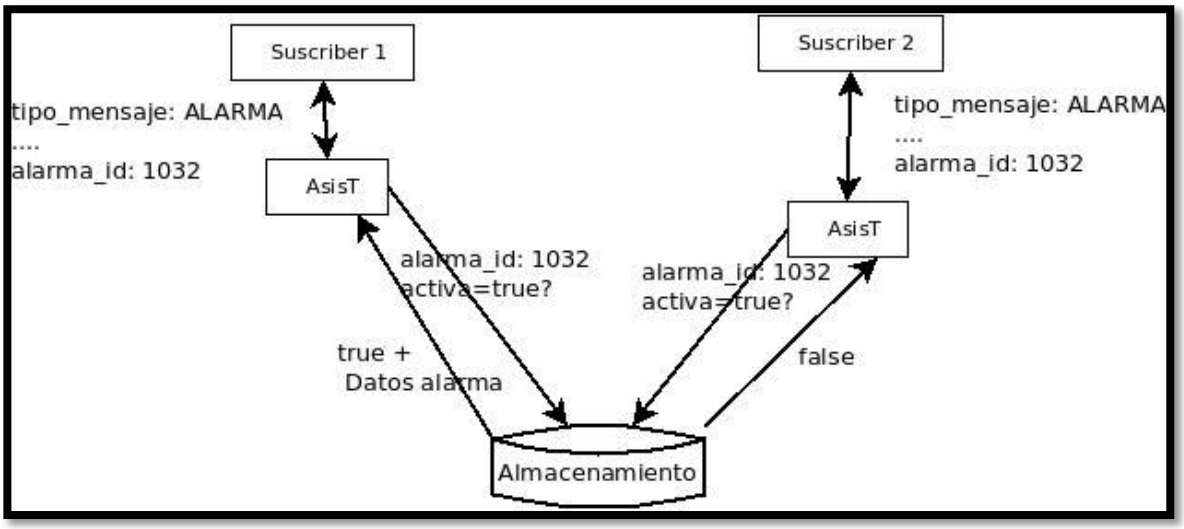


Figura D-7 Lectura de la alarma

Si el operador pulsa el enlace de capturar, la aplicación lee de la base de datos, la alarma cuyo id aparece en el mensaje. La búsqueda se ha hecho utilizando "select… for update" de forma que garantice el acceso en exclusiva a la zona de memoria y no permita que dos operadores gestionen una alarma a la vez.

La Figura D-8 explica el peligro de no controlar la transacción. Los dos usuarios hacen la consulta en la base de datos. Como el primero en llegar aún no ha realizado un commit (ha modificado la alarma como que ya no está activo y los cambios se han guardado en el gestor de base de datos), el segundo lee que está libre, accediendo a la misma información y gestionándola a la vez.

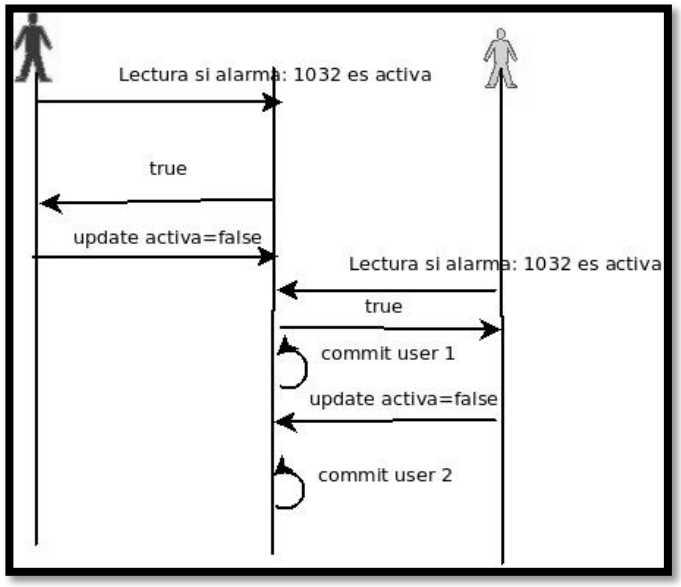


Figura D-8 Error de acceso a la alarma

D.7.4 Aviso de alarma capturada

La alarma ha sido capturada por un operador. En la aplicación Java, se publica un mensaje con la siguiente codificación:

message_type=ALARMA_CAPTURADA
message_source=1223
message_created=(FECHA_HORA_ACTUAL)
alarma_id=1201

En este caso "message_source" es el identificador del operador que ha recibido la alarma. La aplicación lee el mensaje publicado y elimina el panel de aviso de alarma.

La Figura D-9 representa la secuencia de hechos que se producen:

1. La aplicación AsisT que acepta la alarma invoca al publicador y notifica en el tema con el mensaje tipo ALARMA_CAPTURADA
2. Todas las aplicaciones leen la notificación y eliminan el aviso de alarma en la pantalla.

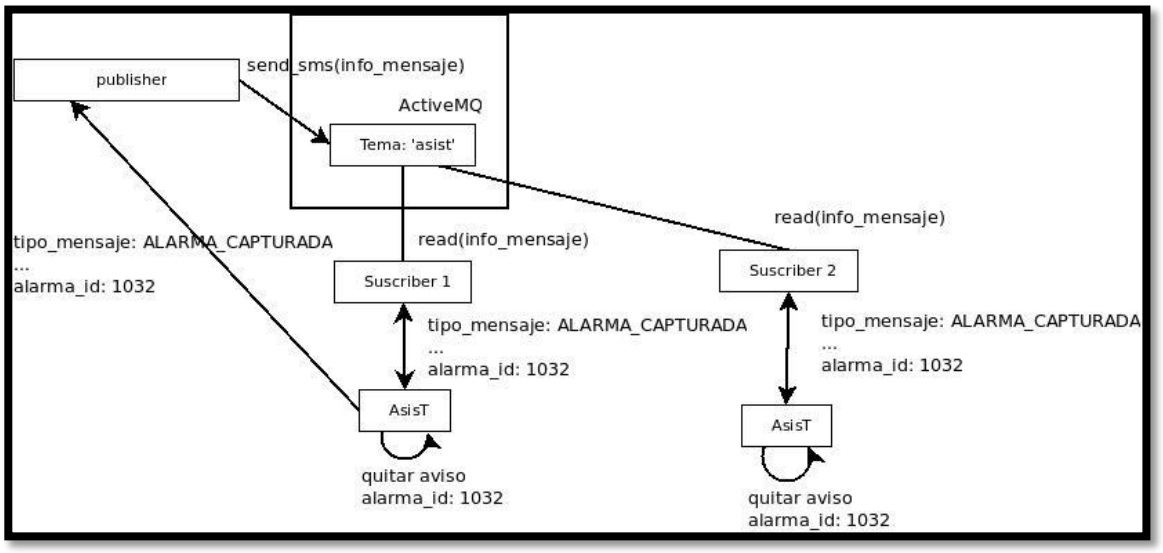


Figura D-9 Aviso de alarma capturada

D.7.5 Pruebas con la herramienta ActiveMQ

Para la comprobación del aviso de alarmas, se ha hecho uso de la herramienta web que se muestra en la Figura D-10.

El servidor web proporcionado por ActiveMQ [43], permite acceder a los temas y postear mensajes de la misma forma que lo hace el publicador. La aplicación está instalada en el servidor local. Mediante el navegador Mozilla FireFox [42] se ha accedido a la misma.

En el cuadro, "Message Body", se introducen los mensajes que los suscriptores leen. Se puede observar en la esquina inferior derecha, que al enviar el texto de "Message body", se ha enviado y la alarma y por consiguiente, aparece el aviso.

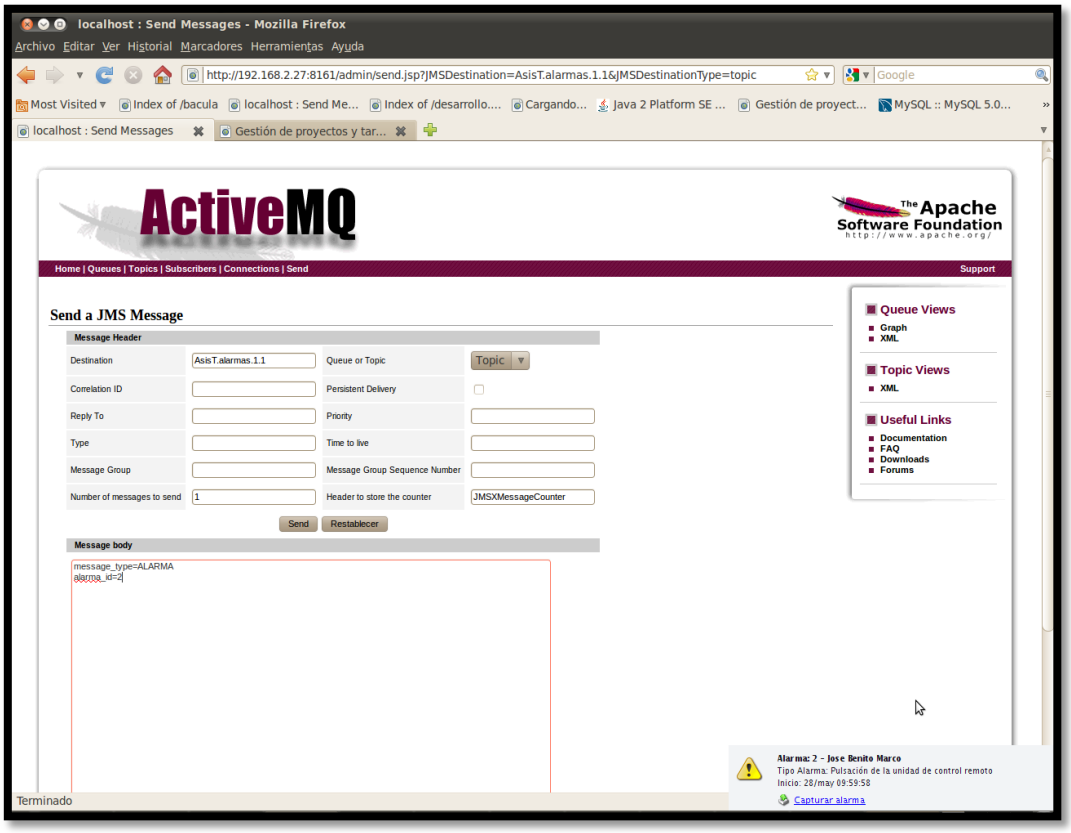


Figura D-10 Pruebas con ActiveMQ

D.8 Subsistema teléfono

D.8.1 Solución adoptada

Para abordar la solución del *softphone* se recurre a la arquitectura del sistema. La centralita IP es la configuración a medida para la gestión de llamadas, basado en el software libre Asterisk [22]. El software está instalado en el servidor local, junto a la base de datos y el software de mensajería.

La centralita o Asterisk, necesita comunicarse con los componentes de la red local. Para ello, necesita valerse de un protocolo de señalización. El utilizado para la plataforma de teleasistencia es SIP [2].

Para implementar SIP en la aplicación AsisT, se ha utilizado la librería Jain [1] de java. Esta librería abstrae la comunicación de bajo nivel en funciones de java.

Para implementar el *softphone* se ha aislado la funcionalidad en tres niveles de forma que cada capa de nivel superior se comunica con la inmediata inferior. De superior a inferior:

- 1er Nivel: funciones que utiliza el subsistema de información para establecer conexión, realizar una llamada y terminarla.
- 2º Nivel: funciones para construir paquetes de SIP acordes con la funcionalidad y configuración deseada.
- 3º Nivel: librería Jain: aporta funciones para la construcción de la pila de SIP.

Al iniciar la aplicación AsisT, establece la conexión con la centralita. A partir de este momento, el sistema queda a la escucha de mensajes que reciba o producir mensajes para la centralita.

D.8.2 Contexto de SIP

SIP es el protocolo de señalización utilizado entre la aplicación AsisT y la centralita IP. La información se manda en paquetes con cabecera y cuerpo. Cuando se inicia la comunicación, se inicia una transacción.

Uno de los atributos más importante del paquete es el tipo de petición: *invite*, *ack*, *option*, *bye*. Según el tipo, tanto el origen como el destino, actuarán según las secuencias establecidas del protocolo.

- Ocurren dos acciones importantes en el *softphone* cuyas secuencias son:
  - Se envía una petición "*request*" de conexión
  - Asterisk nos envía el *token* para identificarnos
  - Se envía los datos de conexión y el *token* para identificarnos
    .
- Aceptar / recibir llamada:
  - Envía *invite* a Asterisk para conectarse.
  - Obtiene la información de destino por medio de *ack*
  - Abre el canal de audio con la información intercambiada.

En el caso de AsisT, recibir una llamada es lo mismo que realizarla. El sistema, por medio de la base de datos, obtiene la alarma y su información. Por lo tanto, cuando reciba una alarma con llamada, será lo mismo que llamar al número que originó la alarma.

D.8.3 Librerías

En este apartado se explica las librerías necesarias para el *softphone*. Estás provienen del API de Jain:

- Librerías de sonido: trident.jar, mp3spi1.9.4.jar, jl10.jar y trionus_share.jar
- Librería de audio y video: jmf.jar
- Librerías de señalización : nist-sdp.jar, sip-sdp.jar, jain-sip-api-1.2.jar, jain-sip-ri-1.2.118.jar, jain-sip-tck-1.2.jar

D.8.4 Paquetes

es.diaple.asist.sip

SIPConfiguration.java: es la clase contenedor de la tabla de la base de datos "usuarios_config". Contiene todos los datos para gestionar el teléfono con la centralita.

Una vez que un usuario inicia la aplicación, necesita conectarse a la centralita. Para ello necesita los parámetros:

- stackName: nombre de la pila.
- stackIPAddress: IP de la pila.
- contactIPAddres: IP de contacto.
- outboundProxy: IP de Asterisk.
- outboundProxyPort: puerto de Asterisk.
- listeningPort: puerto de escucha del puesto del operador.
- signalingTransport: protocolo de transporte para la señalización (*tcp* o *udp*).
- mediaTransport: protocolo de transporte para el audio (*tcp* o *udp*).
- retransmissionFilter: extensión del teléfono dentro de la centralita.
- userName: usuario dentro de la centralita.
- userPassword: contraseña dentro de la centralita.

Los datos de SIPConfiguration.java, se obtienen en la clase UsuarioDAO, a partir de un "usuario_id". Estos datos son esenciales para construir todos los paquetes SIP.

PhoneListener.java

- Contiene los diferentes mensajes de la interfaz del *softphone* según el estado.
- Contiene la clase interface del método para pintar por pantalla: *onStateChange*. Contiene el interfaz del *softphone*
- La implementación del método *onStateChange* dependiente del estado del teléfono.

Phone.java: Contiene las funciones a nivel java del *softphone*:

- register(): registrar el teléfono a la centralita.
- callTo(numero): inicia la llamada telefónica con el número de entrada.
- cancelCall(): finaliza la llamada.

Estas funciones no son más que la capa superior a la implementación de la siguiente clase: Register invoca a sendRegister, callTo a sendInvite y canceCall a sendBye. A continuación se explican estas funciones, ya que están contenidas en la siguiente clase.

MessageProcesor.java: tiene todas las funciones de creación de paquetes y envío.

- Creación de la pila de SIP.
- Funciones genéricas: crear una petición y una respuesta ante petición.
- create<nombreTipoPetición>: crear una petición específica: *register*, *invite*, *ok*, *option*.
- send<nombreTipoPetición>: envía la petición.
- send<nombreTipoPetición><nombreTipoPetición>: envía la petición como respuesta a otra petición.
- Procces<nombreTipoPetición>: procesa la llegada de un paquete de ese tipo de petición, realizando las secuencias del protocolo SIP.

Las funciones principales implementadas:

- createSipStack: Devuelve la pila SIP. La simulación de la pila SIP se ha hecho mediante un fichero *properties* [23].
    - La información de la pila: entre los valores que se inician están la IP origen y destino (Asterisk).
- processInviteOk: procesa el caso de recibir un *ok* ante un *invite*. Envía un *ack* incrementando el identificador de secuencia.
- processRegistOK: es la función que recibe *ok* ante un *register*.
- sendRegister:
    - Crea una petición de tipo *register*.
    - Inicia la transacción.
    - Envía la petición *register*.
- createRequest: crea una petición genérica, dado el tipo y las URI (Uniform Resource Identifier, es decir, identificador del dispositivo en SIP) origen (*callee*) y destino (*caller*).
- sendRegisterWithAuth: envía una petición de tipo "*register*" incluyendo en la cabecera autentificación.
- sendOptionOK: crea una respuesta de tipo *ok* ante una petición de *option*.
- createResponse: crea una respuesta ante una petición.
- sendInvite:
    - Crea una sesión SDP (Session Description Protocol), con formato de audio negociado.
    - Obtiene la URI del que llama y el llamado y crea la petición de tipo *invite*.
    - Establece la sesión SDP.
    - Inicia la transacción y envía la petición.
- sendInviteWithAuth: establece las cabeceras para soportar la autentificación.
- sendBye: crea una petición de tipo *bye* y la envía.
- processInvite: Estado: llamada entrante. Ha recibido un *invite*, entonces:
    - Crea una respuesta *ringing*.
    - Configura dispositivo de audio.
    - Envía *ok* con información SDP.
    - Crea SDP con formato de audio negociado.
- processCancel: estado: llamada terminada.
- processBye: estado: llamada terminada.
- processRinging: estado: sonando.
- processForbiden: estado: prohibido. Finaliza la llamada.
- processDecline: estado: denegado. Finaliza la llamada.

MessageListener.java: Esta clase tiene la misma funcionalidad que un *Event Listener* de java [40]. En este caso, implementa los procedimientos de la clase que responden a eventos de petición o respuesta, generando los pasos del protocolo SIP. Utiliza las funciones implementadas en MessageListener para llevarlas a cabo.

Las funciones implementadas son:

- processRequest: Se invoca con el evento RequestEvent.
    - Obtiene la transacción de dicha petición.
    - Invoca a la función implementada en MessageProcessor.java send<nombreTipoPetición>.
- processResponse: Se invoca con el evento, ResponseEvent.
    - Según el estado, invoca las funciones de process de MenssageProcessor.java:
        - OK – invoca processInviteOK o processRegisterOK.
        - RINGING – invoca *processRinging*.

        - FORBIDEN – invoca *processForbiden*.
        - DECLINE – invoca *processDecline*.
        - PROXY_AUTHENTICATION_REQUIRED.
        - UNAUTHORIZED – Termina la petición y reenvía una nueva.
- processTimeout: Se invoca con el evento *TimeoutEvent*.
    - Informa de error de timeOut.
- processIOException: Se invoca con el evento *IOException*.
    - Informa de error de excepcion.
- processTransactionTerminated:
    - En caso de ser la transacción de *register*, se envía petición de registro con autentificación.
    - En caso de ser la transacción de *invite*, se envía petición de invite con autentificación.
- processDialogTerminated:
    - Informa que el diálogo ha terminado.

es.diaple.asist.sip.media

MediaManager.java: Esta clase gestiona el canal de audio y voz de la aplicación.

- preparedMediaSession: Establece los parámetros de audio: codecs, puertos, direcciones, etc.
- startMediaSession: Recibe y transmite los datos de la sesión.
- stopMediaSession: Cierra los canales de recibir y transmitir.

D.9 Paquetes extra

es.diaple.asist.resources

Contiene los iconos de la aplicación. El origen de los mismos es de libre distribución

es.diaple.asist.translations

tl_es_ES.properties [23]: este archivo contiene todas las traducciones. Todo texto de la aplicación ha sido introducido en este fichero manteniendo unas normas de nombrado.

Aunque existen botones, tablas o etiquetas cuyo icono o nombre ha sido idéntico (Ejemplo borrar), se ha replicado tantas veces como se ha utilizado. De esta forma, se asegura la consistencia en caso de modificaciones o eliminaciones.

En la siguiente Figura D-11 se observa el IDE NetBeans: el centro, la clase FichaDireccion, a la derecha, la paleta Diaple implementada en la librería Diaple-desktop y abajo, el fichero properties tl_es_Es.properties con los datos de traducciones de las labels de la ficha.

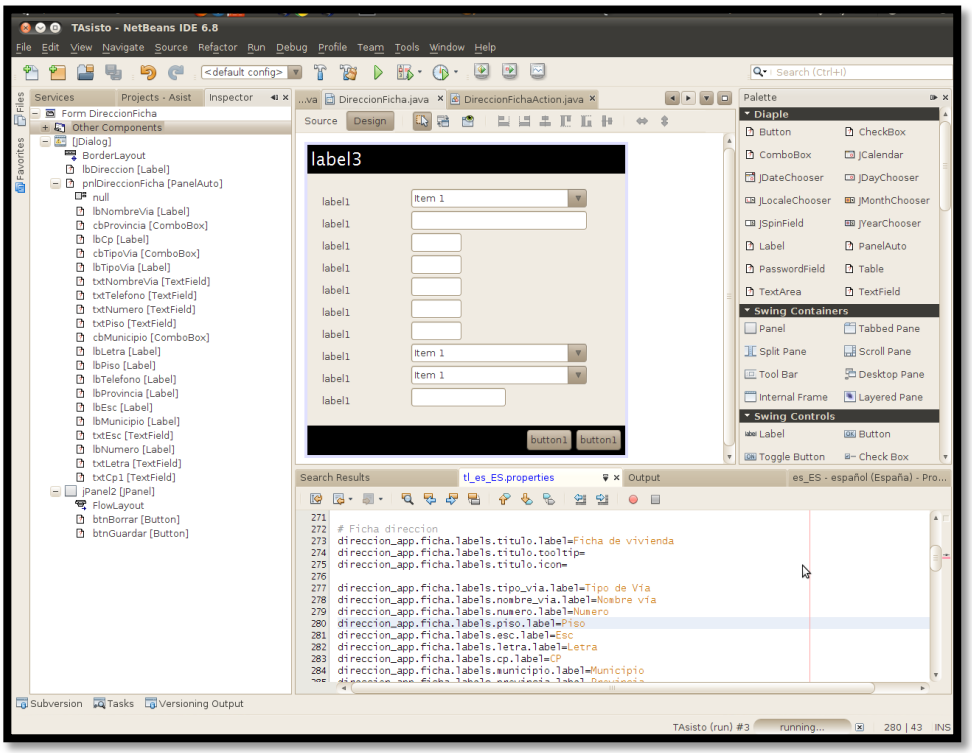


Figura D-11 Fichero tl_es_Es.properties

D.9.1 Ejecución de las pruebas

Siguiendo los planes del diseño, se han realizado las pruebas de testeo definidas.

Por cada clase se ha creado su correspondiente clase Test. Como ejemplo ClientesFicha.java → ClientesFichaTest.java.

Se han realizado las pruebas atendiendo a diferentes valores de entrada. Los casos especiales que se ha tenido especial importancia son con valores nulos tales como listas, objetos o campos.

D.9.2 Conclusiones de la implementación

Se ha empezado implementando la base de datos, luego el subsistema de información y por último se ha integrado con los subsistemas externos. Los objetivos que se plantearon en el inicio, han sido cubiertos exitosamente.

La implementación ha sufrido también riesgos:

- Necesidad de estudiar la materia:
  - Interfaz:
    - API *Swing*
  - Paso de mensajes con JMS
    - API jms.jar y javax.jms.jar
  - Protocolo SIP y Asterisk
    - Protocolo SIP.
    - API Jain.
  - Pruebas
    - API JUnit.
- Consultas tediosas: MySQL no ofrece detalles específicos en caso de error, en ocasiones era difícil encontrar el fallo.

- Una vez que se han implementado los DAO y aplicado, ha habido algún cambio en la base de datos que ha provocado cambios en todos los niveles.

En este punto es donde se aprecia la utilidad de haber realizado antes un análisis y diseño. Debido a la identificación de patrones y similitudes en las clases, se ha podido avanzar con unas plantillas para cada DAO, VO, “Action”, “Ficha”, etc.

UNIVERSIDAD DE ZARAGOZA

Centro Politécnico Superior

Ingeniería Informática

ANEXO E: MANUAL DE USUARIO

[5/5]

PFC: DESARROLLO DE UNA APLICACIÓN PARA LOS OPERADORES DE TELEASISTENCIA INTEGRADA EN TELEFONÍA IP

AUTORA: LAURA LACARRA ARCOS

DIRECTOR: ROBERTO CASAS MILLÁN

PONENTE: JESÚS ALASTRUEY BENEDÉ

Departamento Informática e Ingeniería de Sistemas

Zaragoza, Julio 2010

MANUAL DE USUARIO AsisT V1.0

Introducción

Bienvenidos al manual de la aplicación de AsisT. El objetivo de este manual es prestar ayuda en la aplicación AsisT.
Se distinguen en el manual los siguientes apartados:

- Autentificación
- Administración de la información.
  - Operadores de la aplicación
  - Clientes de tele-asistencia.
  - Gestión de un expediente.
- Gestión de alarmas
  - Inicio.
  - Teléfono.
  - Gestión de una alarma.
- Historiales
  - Historial de alarmas.
  - Historial de llamadas.

Los usuarios de la aplicación son las personas dentro de una empresa de teleasistencia que utilizan la aplicación AsisT para distintos fines.
Los usuarios de teleasistencia, son los clientes que han contratado el propio servicio. De aquí en adelante, vamos a definirlos clientes.

Requisitos mínimos

Se considera que la aplicación ha sido instalada por el equipo cualificado y que el puesto del operador está conectado a la red local de la empresa de teleasistencia. El equipo debe disponer del siguiente equipamiento:

- Cascos u altavoces.
- Micrófono

Los requisitos técnicos mínimos son:

- Sistema operativo: Windows, Linux, MAC OS X, etc.
- Tarjeta de red.
- Tarjeta de sonido con entrada de audio.
- 512 MB de RAM
- Máquina virtual de Java 1,6.

Iconos de manual

Los iconos del manual significan:


Información: Explican detalles de las pantallas mostradas


Consejos: Explican funcionalidad para mejorar la navegación y son comunes al resto de pantallas.


Advertencia: Detalles a tener en especial cuidado.

Autentificación

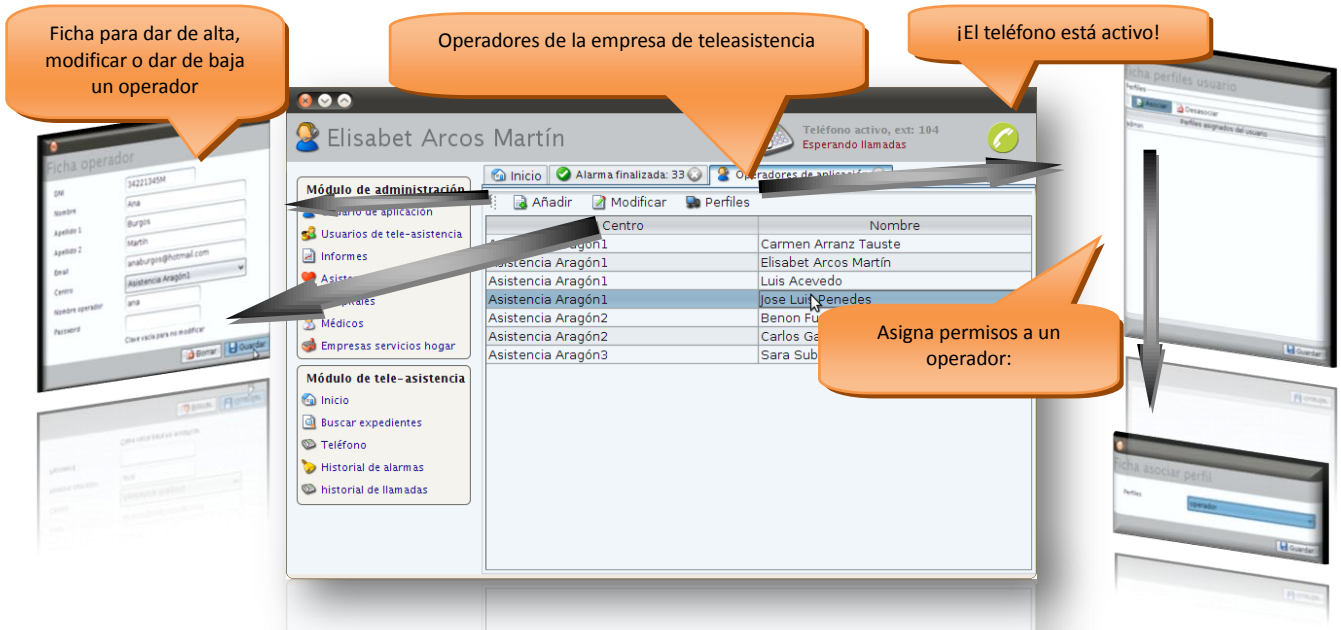
Introduzca nombre de usuario y contraseña para acceder a la aplicación.

Administración de la información

Accediendo a cada uno de los botones, la pantalla central mostrará la información en pestañas.
La parte superior del menú, corresponde a la administración de la información. Sólo acceden los operadores con perfil de administrador.
La parte inferior del menú, corresponde a la gestión de teleasistencia. Sólo acceden los operadores con perfil de operador.

- ✓ Ponga el ratón encima de una ventana y se mostrará un mensaje informativo.
- ✓ Si una ficha ya está abierta, no se vuelve a abrir.

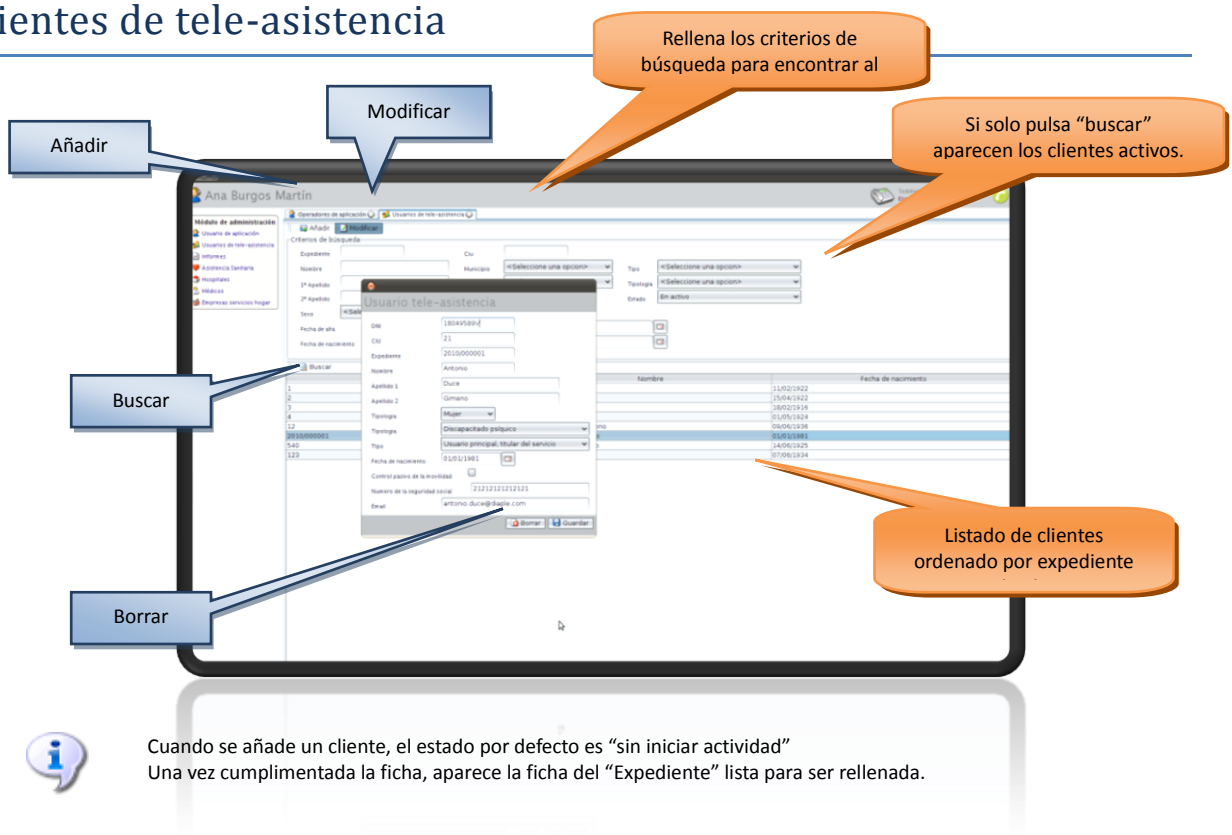
Operadores de la aplicación



Los operadores aparecen ordenados por centro.
El nuevo operador debe tener un nombre para acceder a la aplicación, único.
Los perfiles que puede tener un operador son:
Perfil de administrador: Tiene acceso único al módulo de administración.
Perfil de operador: Tiene acceso único al módulo de teleasistencia.
Sólo aparecen los perfiles que no tiene asignado el usuario.

- ✓ Pulse "doble click" sobre cualquier fila de una tabla para acceder directamente a los datos de la ficha y poder modificarlos.
- ✓ Siempre que quiera borrar un dato, será necesario que una segunda confirmación. ¡Así se evitarán disgustos!
- ✓ Cuando se asocia información, estamos aportando datos que ya existen, como es un perfil, a un operador. Si lo desasocias, volverá a aparecer en una lista desplegable por si desea volverlo a asociar.
- ✓ Cuando se añade información, estamos creando nueva información partiendo de cero, como es el caso de un nuevo operador.

Clientes de tele-asistencia

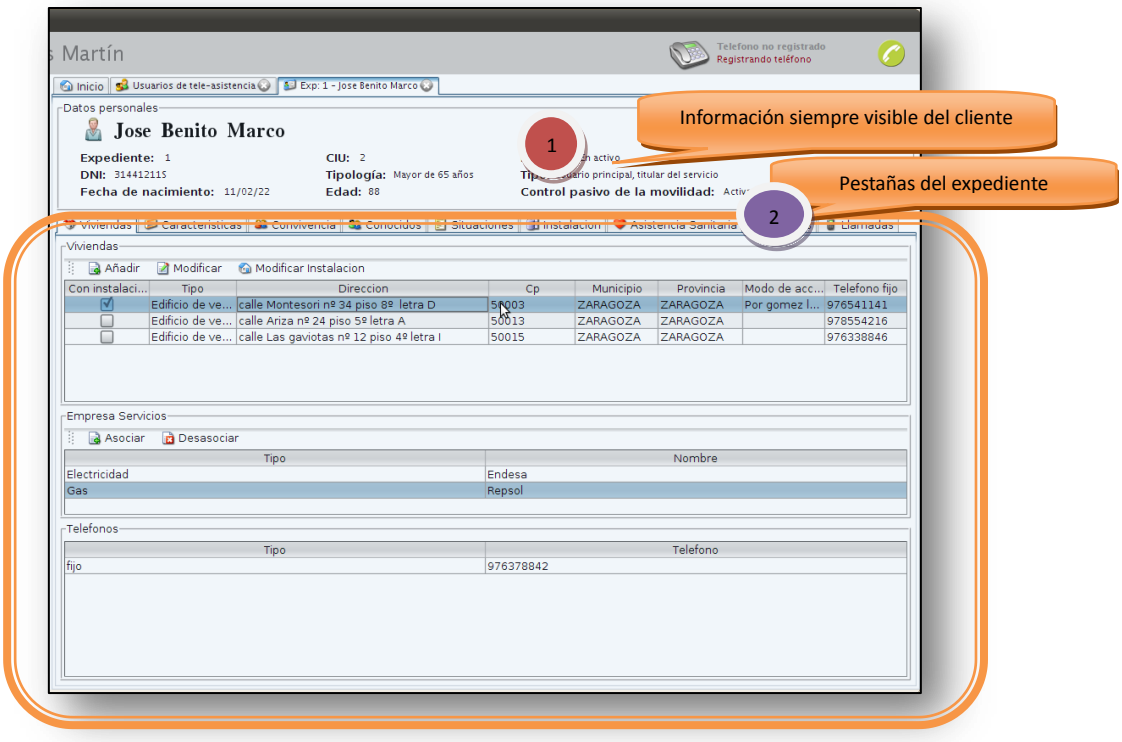


Cuando se añade un cliente, el estado por defecto es "sin iniciar actividad"
Una vez cumplimentada la ficha, aparece la ficha del "Expediente" lista para ser rellenada.

- ✓ Pulse "doble click" sobre cualquier fila de los clientes y accederá al expediente de la persona.
- ✓ En caso de introducir mal un dato en una ficha se mostrará un mensaje para informar del error.

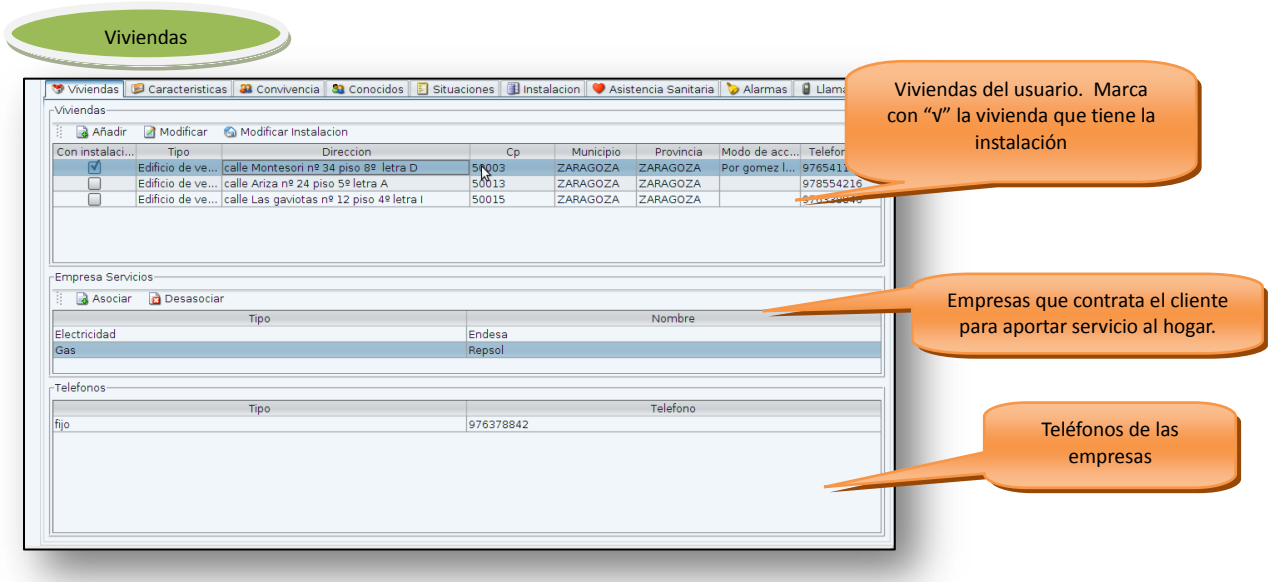
Gestión de un expediente

Se accede a un expediente desde las pestañas "Buscar expedientes", "Usuarios de tele-asistencia", "Atender alarma" y "Alarma finalizada"



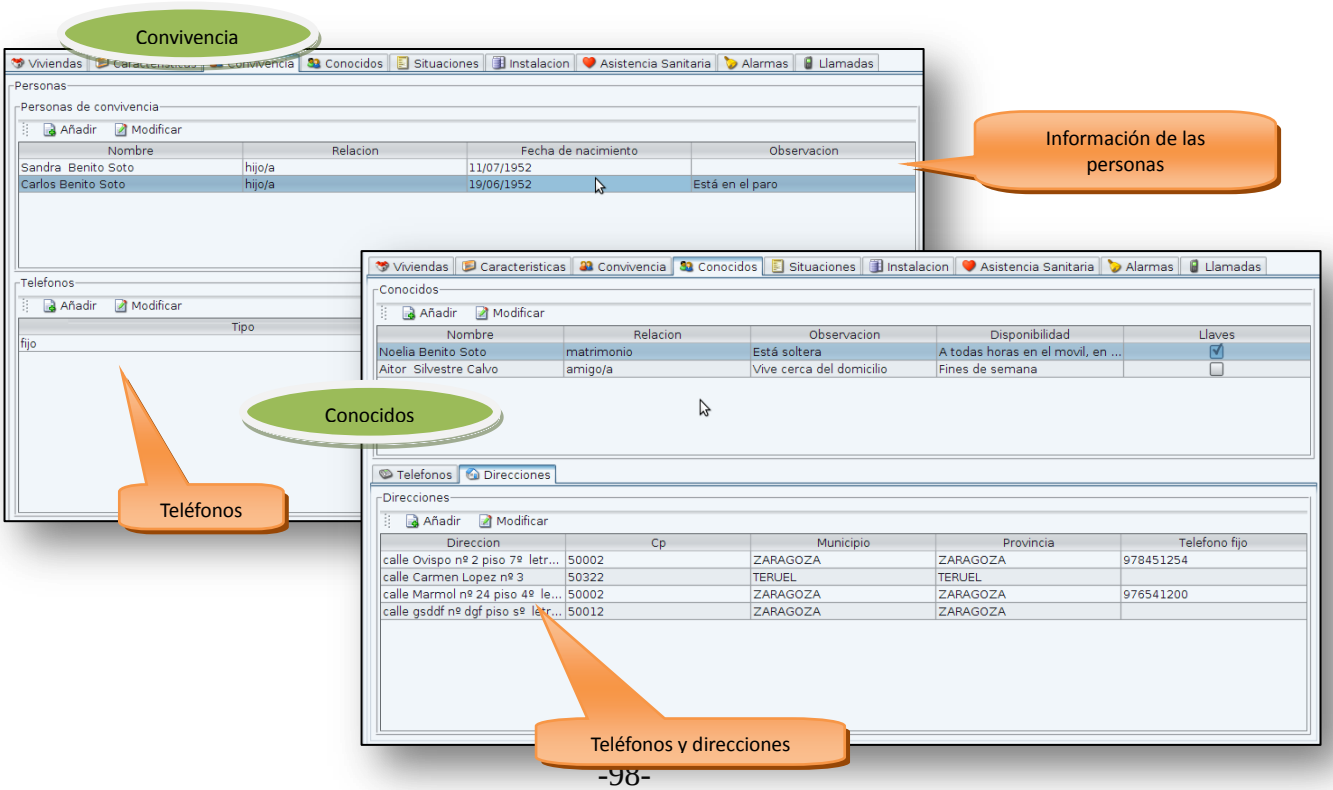
Al acceder a una pestaña, aparece la información del cliente con los datos que disponga.
La operativa es similar en todas:

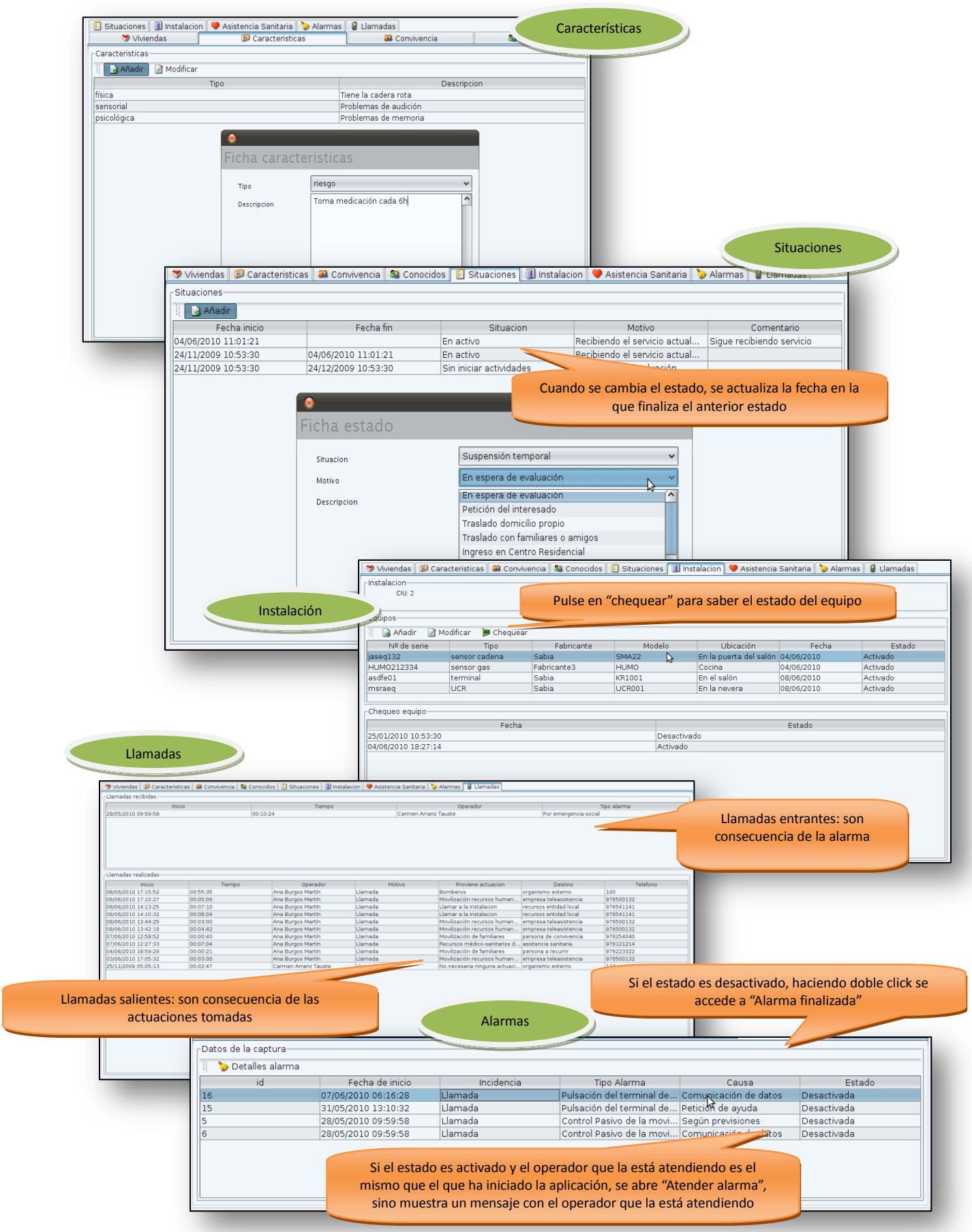
Añadir: aparece una ficha vacía con nueva información a introducir.
Modificar: aparece una ficha con la información a modificar. Puede borrarla o guardar los cambios.
Asociar: aparece en una ficha una lista desplegable con las posibles asociaciones que aun no tiene el cliente.
Seleccione la opción y pulse guardar o salga sin guardar los cambios.
Desasociar: elimina la asociación de la información. Operación inversa a asociar.

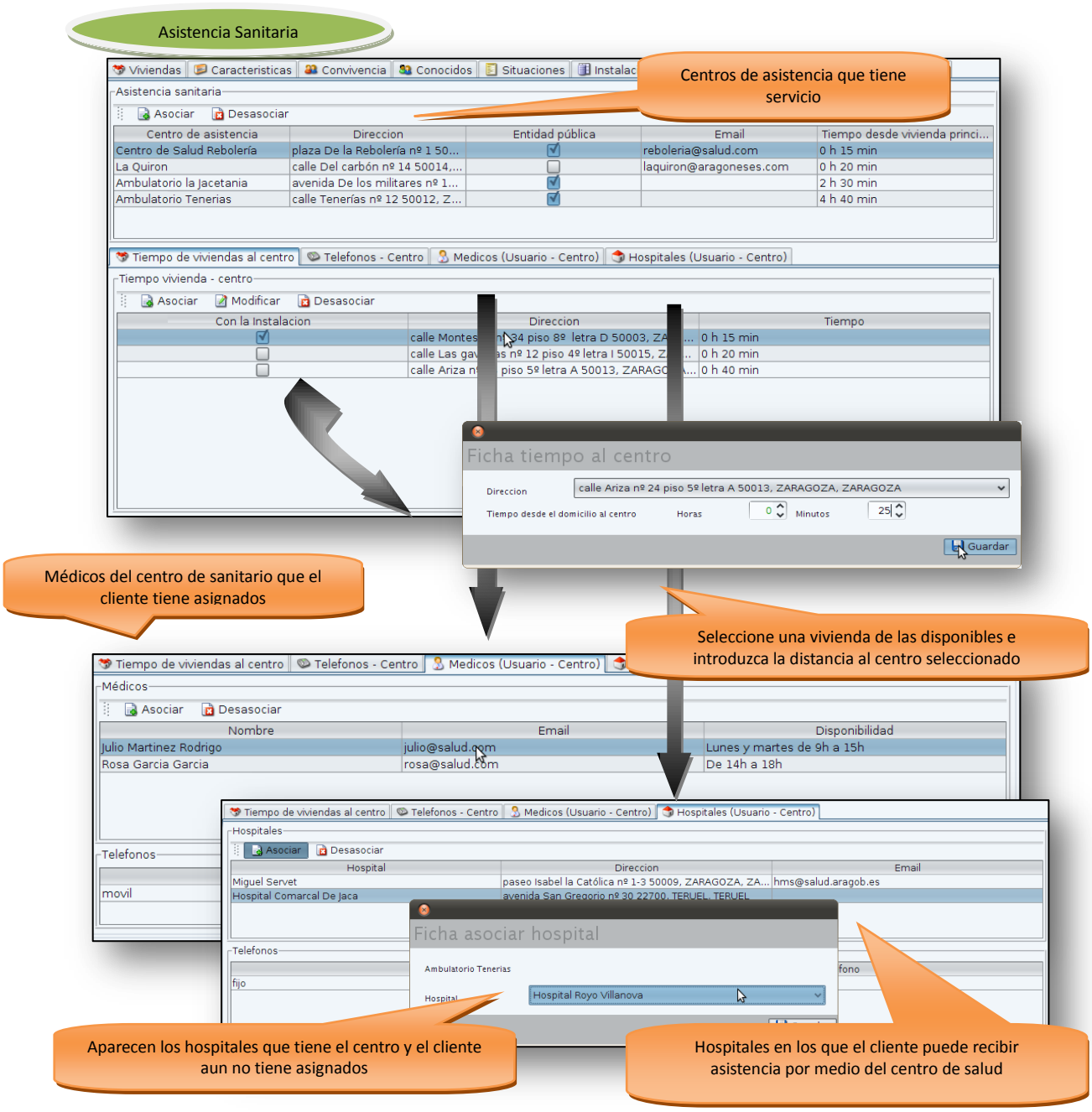


El teléfono fijo que se introduce en la ficha vivienda, corresponde con el número de teléfono del terminal.

- ✓ Al seleccionar una fila de una tabla superior, las tablas inferiores se actualizan con la información dependiente. Por ejemplo, al seleccionar una vivienda, se mostrará las empresas de servicios contratadas por esa vivienda y en el tercer nivel, los teléfonos de la empresa.
- ✓ Pulse doble "click" sobre la fila de una tabla y podrá editar la información.




Características
Situaciones
Cuando se cambia el estado, se actualiza la fecha en la que finaliza el anterior estado
Instalación
Pulse en "chequear" para saber el estado del equipo
Llamadas
Llamadas entrantes: son consecuencia de la alarma
Llamadas salientes: son consecuencia de las actuaciones tomadas
Alarmas
Si el estado es desactivado, haciendo doble click se accede a "Alarma finalizada"
Si el estado es activado y el operador que la está atendiendo es el mismo que el que ha iniciado la aplicación, se abre "Atender alarma", sino muestra un mensaje con el operador que la está atendiendo

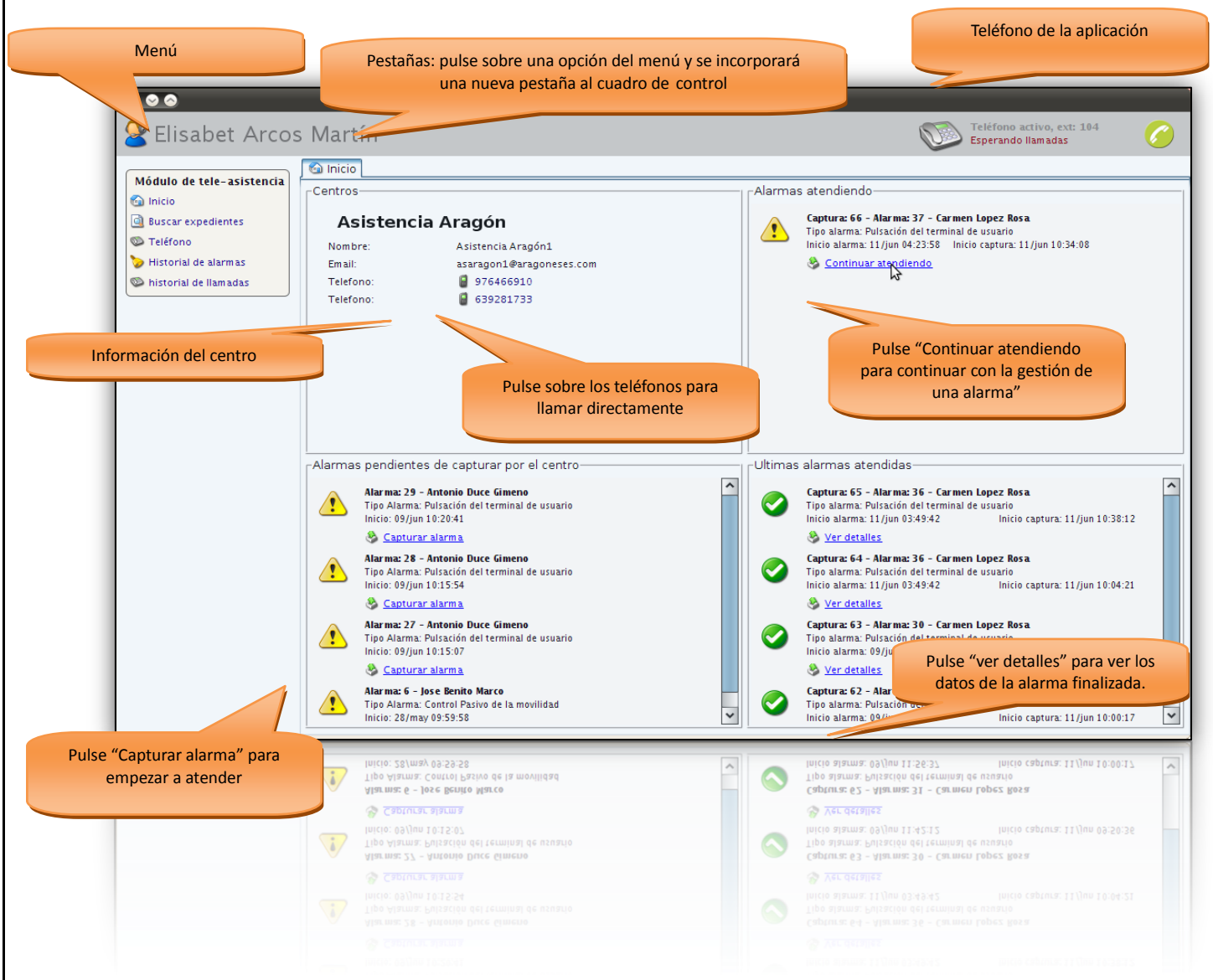


Las asistencias sanitarias, están ordenados por distancia a la vivienda principal.
En caso de cambiar la vivienda principal, la tabla se actualiza con las distancias a la nueva vivienda.

Gestión de alarmas

Inicio

La pantalla “Inicio” siempre está visible para un usuario con perfil operador.
El teléfono siempre está visible “ext:104” es la extensión del teléfono del centro.



En caso de que cierre la pestaña de una alarma, o la propia aplicación, accede a “Inicio” y aparecerá en el cuadro de “Alarmas atendiendo”.
Si se desea reactivar una alarma, accede a “ver detalles” en “Últimas alarmas atendidas”. En la pantalla de “alarma finalizada” podrá reactivarla.
Comprueba que los cascos y micrófono están conectados al equipo.

Teléfono

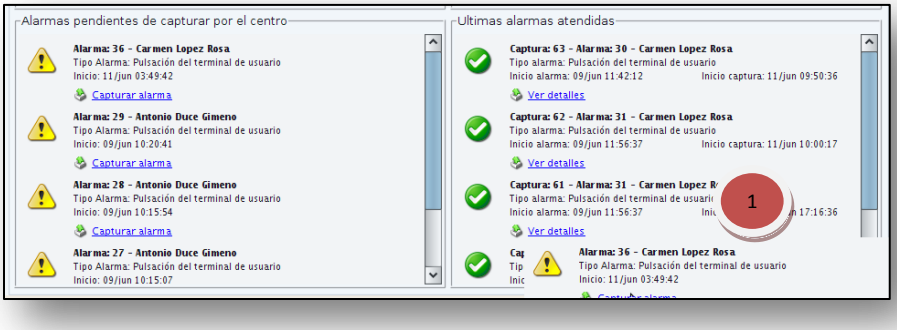
El teléfono mostrará diferentes mensajes según la operativa:

- “Esperando llamadas”: una vez que el teléfono se ha registrado correctamente y se ha terminado una llamada.
- “Registrando teléfono”: ha lanzado la petición para conectarse y espera confirmación.
- “Recibiendo llamadas”: el *softphone* tiene una llamada entrante.

- “Marcando <número_al_que_marca>”: ha iniciado la llamada pero el teléfono del destinatario, aún no está sonando.
- “Teléfono no registrado”: cuando los datos de configuración del usuario no son correctos.
- “Hablando…”: el canal de audio ha sido abierto.
- “Estableciendo conexión”: cuando esté conectado con la centralita IP
- “Llamada <número de teléfono>”: en el caso que esté realizando una llamada o recibiendo. Aparece la información del número de teléfono y del Tiempo actualizado de la llamada.
- “Sonando”: cuando tiene una llamada entrante.
- “Denegado”: cuando la centralita deniega establecer una llamada con dicho número.
- “Prohibido”: cuando no está permitido llamar a dicho número.
- Sólo puede realizar llamadas si la aplicación ofrece un link o es a consecuencia de una actuación

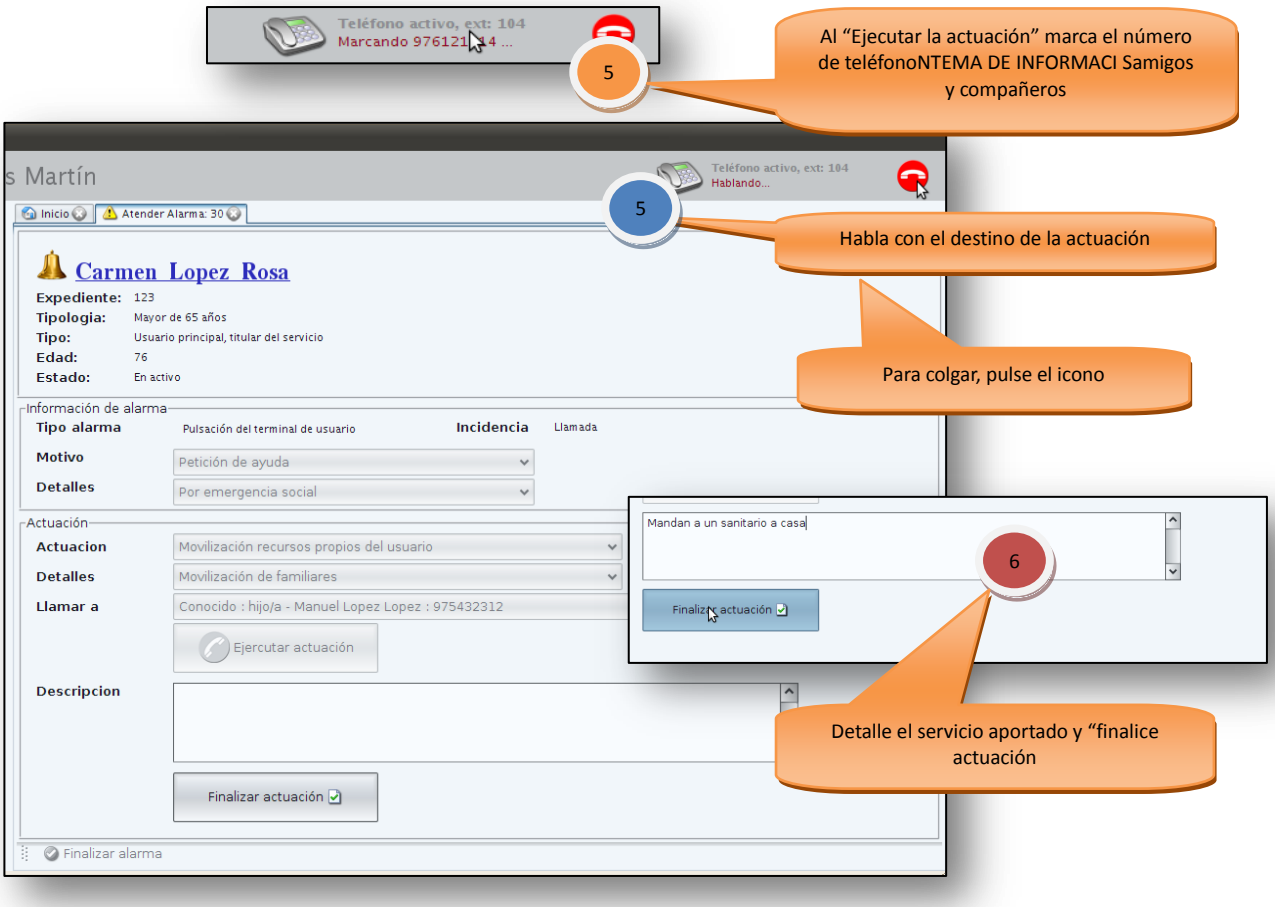
Gestión de una alarma

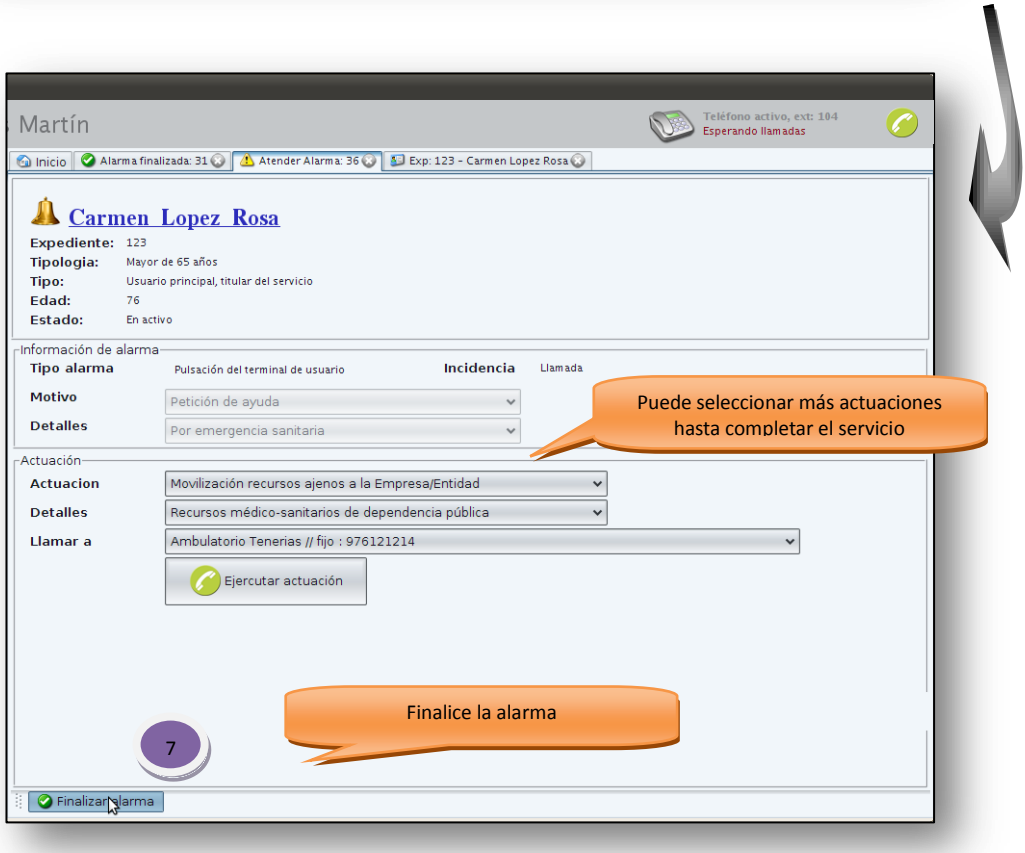
Al tener la aplicación encendida, aparecerá siempre por pantalla un aviso de alarma en la esquina inferior derecha.



Pantalla atender alarma

Seguimos en la misma pantalla, atendiendo la alarma


Teléfono activo, ext: 104
Marcando 976121214 ...
5
Al “Ejecutar la actuación” marca el número de teléfonoNTEMA DE INFORMACI Samigos y compañeros
s Martín
Teléfono activo, ext: 104
Hablando...
Inicio
Atender Alarma: 30
5
Habla con el destino de la actuación
Carmen Lopez Rosa
Expediente: 123
Tipologia: Mayor de 65 años
Tipo: Usuario principal, titular del servicio
Edad: 76
Estado: En activo
Para colgar, pulse el icono
Información de alarma
Tipo alarma
Pulsación del terminal de usuario
Incidencia
Llamada
Motivo
Petición de ayuda
Detalles
Por emergencia social
Actuación
Actuacion
Movilización recursos propios del usuario
Detalles
Movilización de familiares
Llamar a
Conocido : hijo/a - Manuel Lopez Lopez : 975432312
Ejercutar actuación
Descripcion
Finalizar actuación
Finalizar alarma
Mandan a un sanitario a casa
6
Finalizar actuación
Detalle el servicio aportado y “finalice actuación


Martín
Teléfono activo, ext: 104
Esperando llamadas
Inicio
Alarma finalizada: 31
Atender Alarma: 36
Exp: 123 - Carmen Lopez Rosa
Carmen Lopez Rosa
Expediente: 123
Tipologia: Mayor de 65 años
Tipo: Usuario principal, titular del servicio
Edad: 76
Estado: En activo
Información de alarma
Tipo alarma
Pulsación del terminal de usuario
Incidencia
Llamada
Motivo
Petición de ayuda
Detalles
Por emergencia sanitaria
Puede seleccionar más actuaciones hasta completar el servicio
Actuación
Actuacion
Movilización recursos ajenos a la Empresa/Entidad
Detalles
Recursos médico-sanitarios de dependencia pública
Llamar a
Ambulatorio Tenerias // fijo : 976121214
Ejercutar actuación
Finalice la alarma
7
Finalizar alarma

Pantalla alarma finalizada, la anterior pantalla desaparece.

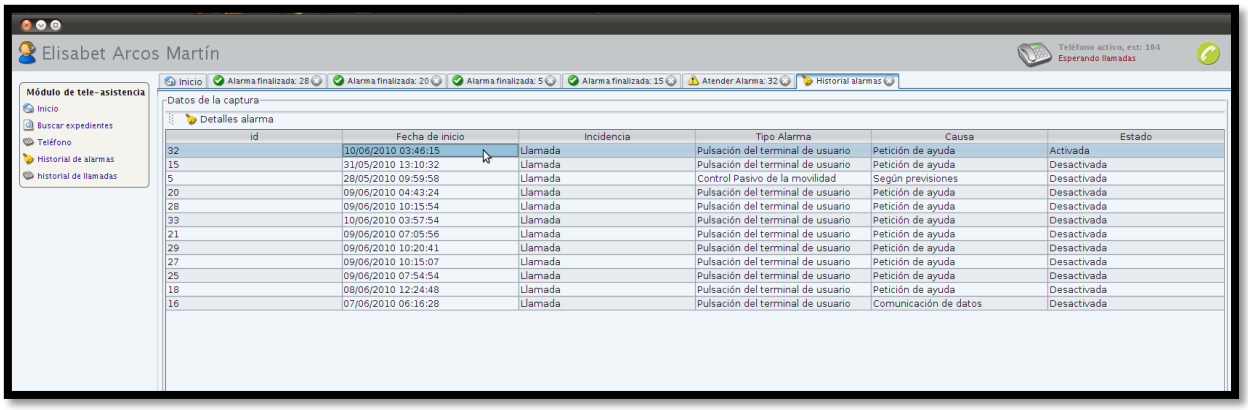
"Reactivar" es lo mismo que volver a capturar una alarma cuyo "tipo de alarma" está predefinido.

- ✓ Es de utilizad reactivar la alarma, en caso de que se haya finalizado la alarma y se desee volver a llamar al cliente.

Historiales

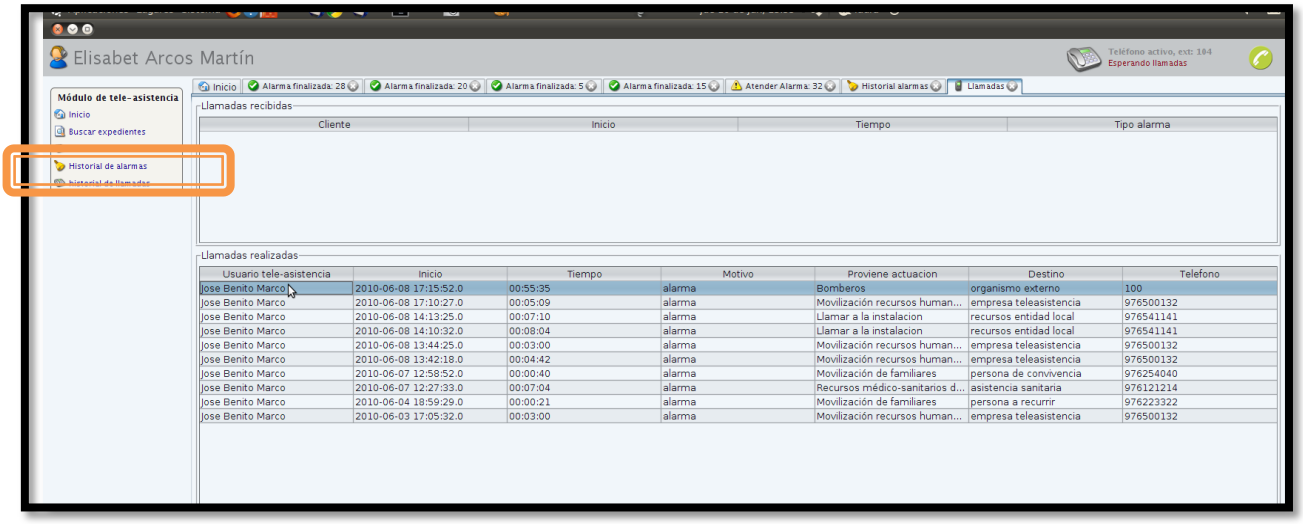
Historial de alarmas

Muestra las alarmas atendidas por un operador.
Si se hace doble “click sobre la fila y el estado es “activado” se abre la pestaña de “Atender alarma”, en caso de ser “desactivada” se abre la pestaña, “Alarma finalizada”.
Las alarmas están ordenadas de la última atendida, a la primera



Historial de alarmas

Muestra las últimas llamadas entrantes y salientes, con los datos de las mismas.



ACRÓNIMOS

API	Application Programming Interface
BD	Base de Datos
CIU	Código de Identificación de Usuario
CPD	Centro de Proceso de Datos
CTI	Computer Telephony Integration
DAO	Data Access Object
DFD	Diagrama de Flujo de Datos
DTMF	Dual-Tone Multi-Frecuency
FEMP	Federación Española de Municipios y Provincias
IMSERSO	Instituto de Mayores y Servicios Sociales
GPS	Global Position System
IP	Internet Protocol
JDBC	Java Database Connectivity
JMS	Java Messaging Service
PBX	Private Branch Exchange
PDF	Portable Document Format
RTB	Red de Telefonía Básica
SAAS	Sotfware As A Service
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UDP	User Data Protocol
UCR	Unidad de Control Remoto
URI	Uniform Resource Identifier
VO	Value Object
XML	Extensible Markup Language

BIBLIOGRAFÍA

Referencia	Título
[1]	JAIN http://java.sun.com/products/jain/
[2]	SIP: RFC 3261 http://www.ietf.org/rfc/rfc3261.txt
[3]	IMSERSO http://www.imserso.es/imserso_01/index.htm
[4]	FEMP http://www.femp.es/
[5]	Normativa de teleasistencia aportada por IMSERSO y FEMP: http://www.imsersomayores.csic.es/documentos/documentos/imserso-programateleasistencia-01.pdf
[6]	Java : http://www.java.com/en/
[7]	Comunidad Open Source: http://www.opensource.org/
[8]	Sistema opertaivo Linux Ubuntu: http://www.ubuntu-es.org/
[9]	MySQL: http://www.mysql.com/
[10]	Postgree SQL http://www.postgresql.org/
[11]	Oracle: Gestor de bases de datos http://www.oracle.com/global/es/index.html
[12]	Ley juridical de protección de datos: http://noticias.juridicas.com/base_datos/Admin/lo15-1999.html
[13]	NetBeans: Herramienta de programación http://netbeans.org/
[14]	Aptana: Herramienta de programación utilizada para PHP http://www.aptana.org/
[15]	JDBC Driver para MySQL http://dev.mysql.com/downloads/connector/j/3.0.html
[16]	Substance: https://substance.dev.java.net/
[17]	Herramienta de debuging para java: http://logging.apache.org/log4j/1.2/download.html
[18]	Librería JMS para paso de mensajes: http://java.sun.com/products/jms/docs.html Java message service / Richard Monson-Haefel, David A. Chappell
[19]	Librería JUnit pruebas del sistema: www.junit.org/
[20]	Design Patterns: Elements of Reusable Object-Oriented Software Escrito por Gamma
[21]	Fundamentos de voz sobre IP / Jonathan Davidson, James Peters
[22]	AsterisK: Software de gestión de la centralita IP http://www.asterisk.org/
[23]	Properties claas: http://java.sun.com/j2se/1.5.0/docs/api/
[24]	Cola de mensajería Active MQ: http://activemq.apache.org/
[25]	Cola de mensajería OpenJMS http://openjms.sourceforge.net/
[26]	Cola de mensajería Rabbit MQ http://www.rabbitmq.com/
[27]	Gestor de almacenamiento InnoDB http://dev.mysql.com/doc/refman/5.0/en/innodb.html
[28]	Análisis y diseño orientado a objetos de sistemas usando UML / Simon Bennet, Steve McRobb, Ray Farmer
[29]	Object-oriented analysis / Peter Coad and Edward Yourdon
[30]	Métrica 3:Metodología de Software http://www.csi.map.es/csi/metrica3/index.html
[31]	Pencil v1.0: herramienta de modelado http://www.evolus.vn/pencil/Downloads.html
[32]	PHP http://php.net/index.php

Referencia	Título
[33]	DIA : Herramienta de diagramas http://live.gnome.org/Dia
[34]	GIMP: Herramienta de edición de imágenes http://www.gimp.org/
[35]	Umbrello: Herramienta de modelado UML http://uml.sf.net,
[36]	OpenOffice: Procesador de texto y diagramas http://es.openoffice.org/
[37]	Subversion http://subversion.tigris.org/
[38]	Label: Clase Label de Java http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Label.html
[39]	Clase JOptionPane http://java.sun.com/j2se/1.4.2/docs/api/javax/*swing*/JOptionPane.html
[40]	Event Listener http://java.sun.com/docs/books/tutorial/ui*swing*/events/index.html
[41]	Timer Task http://java.sun.com/j2se/1.4.2/docs/api/java/util/TimerTask.html
[42]	Mozilla Firefox: navegador web http://www.mozilla.com/en-US/
[43]	ActiveMQ Web console http://activemq.apache.org/web-console.html
[44]	MySQL Query Browser http://dev.mysql.com/doc/query-browser/en/index.html
[45]	Empresa Verklizan: Servidor UMO http://www.verklizan.org/exec/verklizanweb.exe?lang=SP&page=Pagina/UMO52.html
[46]	Empresa Sabia: http://bioingenieria.es/Teleasistencia/Teleasistencia.htm
[47]	.Net http://www.microsoft.com/net/
[48]	Select for update http://dev.mysql.com/doc/refman/5.0/en/innodb-locking-reads.html
[49]	XML http://www.w3.org/XML/
[50]	*Swing* http://java.sun.com/j2se/1.4.2/docs/api/javax/*swing*/package-summary.html
[51]	Aplicación Flash http://www.adobe.com/es/
[52]	Applet Java http://java.sun.com/applets/
[53]	Active X http://msdn.microsoft.com/en-us/library/ms537508.aspx
[54]	AGI http://www.voip-info.org/wiki/view/Asterisk+AGI
[55]	GPS http://www.gps.gov/
[56]	API JCalendar https://jcalendar.dev.java.net/
[57]	Wikipedia http://www.wikipedia.org

GLOSARIO

Teleasistencia domiciliaria: la teleasistencia domiciliaria es un servicio que permite a las personas mayores y/o personas discapacitadas, recibir asistencia en un centro, atendido por personal específicamente preparado para dar respuesta adecuada a la necesidad. El servicio es prestado a través de la línea telefónica y con un equipamiento de comunicaciones e informático específico, ubicado en un centro de atención y en el domicilio de los usuarios. Los clientes con sólo accionar el dispositivo que llevan constantemente puesto o el terminal de su hogar, entran en contacto verbal, "manos libres", durante las 24 horas del día y los 365 días del año. La empresa que ofrece el servicio presta atención, bien movilizando los recursos propios que dispone o movilizando otros recursos humanos o materiales, propios del usuario o existentes en la comunidad [5].

Centro de atención: también llamado centro de operadores o "call-center" es el lugar en el que los empleados de una empresa de teleasistencia dan servicio a los clientes [5].

Clientes: son personas mayores o discapacitadas, que contratan el servicio de teleasistencia a una empresa [5].

El titular del servicio de teleasistencia domiciliaria: persona que reúne todos los requisitos para ser cliente y el terminal de la instalación pertenece a su nombre [5].

Servicio de teleasistencia domiciliaria sin unidad móvil: el servicio básico de teleasistencia domiciliaria se presta exclusivamente desde el centro de atención, tratando las distintas situaciones que se presentan [5].

Servicio de teleasistencia domiciliaria con unidad móvil: equipo propio de la empresa de teleasistencia, encargado de incrementar la eficacia del servicio, por medio de una atención presencial en los domicilios de los usuarios. Esta actividad, será realizada por el personal de las unidades móviles [5].

Entidad local: una entidad local es una zona que puede ser definida como: municipio, código postal, barrio, distrito, etc. que solicita como miembro público del estado, realizar las tareas de empresa de teleasistencia, bajo su jurisdicción [5].

Usuarios - operadores: son los encargados de recibir, en primera instancia, las alarmas y llamadas, interviniendo según instrucciones y protocolos establecidos. Pueden solicitar la intervención directa del responsable del centro cuando por su complejidad no puedan o no sepan resolver la situación por sí mismos.

Terminales de usuario: emisores de alarmas con sus correspondientes unidades de control remoto (UCR). Dispondrá de un botón bien diferenciado del resto, si es que los hubiese, cuya sola pulsación permita la activación del sistema y puesta en contacto, en modo conversación "manos libres" para el usuario, con el centro receptor. Integrará un altavoz y un micrófono controlables, en caso de alarma, desde la central receptora. Será capaz de comunicarse con el centro receptor utilizando más de un protocolo de comunicaciones [5].

La unidad de control remoto (UCR): dispositivo asociado al terminal de usuario, el cual dispone de un botón cuya sola pulsación desencadene la activación del sistema y puesta en contacto, en modo conversación "manos libres" para el usuario, con el centro receptor [5].

Sistema de atención de alarmas: sistema destinado a la recepción y gestión de las llamadas de

alarma recibidas por el centro receptor de alarmas [5].
Discapacitado: persona con minusvalías permanentes o temporales, así como a quienes padezcan cualquier tipo de limitación física, psíquica o sensorial [5].

Control pasivo de la movilidad: tipo de servicio disponible sólo para usuarios de alto riesgo. Son todos aquellos que requieran de un seguimiento más intenso [5].

Entidad de asistencia sanitaria: establecimiento médico dependiente del sistema de sanidad pública o privada, en el que se presta asistencia médica y farmacéutica a pacientes sin ingresarlos en él [5].

Estados de un usuario: dependiendo de su situación respecto al proyecto, pueden generarse las siguientes situaciones:

- En activo: recibe actualmente el servicio.
- Ausencia domiciliaria inferior a 24 horas: estado en activo. El usuario ha salido del domicilio y lo comunica.
- Suspensión temporal: habiendo estado en activo, deja provisionalmente de recibir el servicio por periodo superior a 24 horas.
- Baja definitiva: habiendo estado en activo, causa baja definitivamente en el servicio.
- Sin iniciar actividades: servicio solicitado y asignado, aún no ha causado alta.
- Baja definitiva de solicitantes que nunca estuvieron en activo [5].

Solicitud: es la previa autorización de los servicios sociales, familiares o del propio usuario, explicando de forma clara las consecuencias de su utilización. Cada usuario sólo podrá presentar una solicitud [5].

Empresa de servicios: unidad de organización dedicada a actividades del hogar como Gas, Electricidad, etc [5].

Empresa de Terminales: unidad de organización dedicada a la fabricación de terminales de teleasistencia [5].

Recursos de una entidad: son servicios de urgencia propios de una entidad [5].

Organismos: son servicios públicos que ofrecen servicios de urgencia, como bomberos, ambulancias, etc [5].