Proyecto fin de carrera. Facultad de informática.
Universidad de Las Palmas de Gran Canaria

Diseño y creación de una interfaz para el desarrollo de aplicaciones que utilicen servicios de comunicaciones por radio

VICTOR OLIVARES GUEDES
Las Palmas de Gran Canaria, 20 de febrero de 2014

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

VICTOR OLIVARES GUEDES

Título del Proyecto : Diseño y creación de una interfaz para el desarrollo de aplicaciones que utilicen servicios de comunicaciones por radio.

Tutores : Francisco J. Alayón Hernández
Carmelo R. García Rodríguez

.

DEDICATORIA

Aquí va la dedicatoria

AGRADECIMIENTOS

Aqui van los agradecimientos

Índice de contenido

1.- Introducción

Hoy en día, no hay duda que las comunicaciones inalámbricas son el futuro. El abaratamiento de los costes de la tecnología y la facilidad de acceso a las redes de comunicación inalámbricas es cada vez más asequible y más rápido. La prestación de servicios en modo *online* provocan, que cada vez más, la vida cotidiana dependa de estar conectado a una red. De tal forma, que en la actualidad ya no es extraño poder llamar desde un móvil, recibir correo en un *smartphone*, consultar un mapa o localizar lugares vía GPS.

Existen una infinidad de comunicaciones por radio con sus bandas de frecuencias específicas para cada servicio, tales como las definidas para la Televisión, radio FM, radio AM , Wifi, Telefonía móvil, GPS , etc. Estas tecnologías, entre otras, además de ser las más conocidas por el usuario común, comparten espectro de frecuencias con las comunicaciones de radio. A su vez, y para desconocimiento de la mayoría de la población, existen soluciones de comunicación por radio para aplicaciones privadas, por ejemplo flotas de guaguas, cooperativas de taxi o servicios de emergencia. Como se puede observar, estas soluciones privadas se explotan en gran medida con empresas y servicios que requieren de gran movilidad.

En el ámbito del proyecto que se desarrolla este documento, se centra el estudio y desarrollo de un servicio de radio destinado principalmente para servicios de emergencia y seguridad llamado TETRA. Este protocolo de radio, con su frecuencia especifica, funciona de manera similar a la tecnología GSM, siendo, a su vez, una evolución de la tecnología de radio *Tunked.*

Este proyecto se encargará de generar una capa de conexión con dispositivos TETRA a través de un protocolo definido en el estándar de comunicaciones de los dispositivos. Al mismo tiempo, se describen los problemas encontrados durante el desarrollo, siendo en las primeras capas bastante graves, debido en mayor parte, a la escasa documentación y a la incompatibilidad de los dispositivos de pruebas en ciertos aspectos de los protocolos de comunicación con el dispositivo.

El desarrollo del mismo se orientará a una arquitectura por capas, definiendo el acceso básico a los dispositivos (leer y enviar datos), para luego, sobre la base de la comunicación desarrollar una serie de métodos de manejo de la información, ya sea para construir mensajes a enviar, como para interpretar los mensajes recibidos. Estas herramientas de primer nivel, se empaquetarán en una librería, siendo por tanto portables a cualquier solución de más alto nivel a desarrollar

Como ejemplo de la comunicación entre librería y terminal TETRA, se desarrollará en una segunda fase una aplicación, cuyo cometido será interconectar un servidor de correo y una serie de dispositivos TETRA clientes asociados a unos buzones. Este software desarrollado se encargará de retransmitir los correos que llegan a los buzones de los clientes como mensajes de texto corto a los terminales TETRA. Esta aplicación, creada con la ayuda del departamento de seguridad de la ULPGC, dispone también de una interfaz Web para su gestión, lo cual da un mayor realismo a la aplicación conceptual desarrollada.

2.- Estado actual del tema

En cuanto al desarrollo actual de la tecnología utilizada en el presente proyecto de fin de carrera, cabe mencionar que tras un largo tiempo de investigación en busca de referencias, trabajos anteriores o soluciones respecto a este tema ya implantadas en el mercado, se observa que la documentación encontrada es escasa, y no profundiza más allá de las especificaciones del estándar de ETSI.

La falta de información de este tipo de tecnología en las bibliotecas de la ULPGC, en trabajos desarrollados con anterioridad o investigaciones publicadas en este ámbito, demuestran que a pesar de tener un amplio uso con diversos propósitos, especialmente en para la comunicación en organismos públicos y, sobretodo, en los cuerpos de seguridad, se trata de una tecnología poco investigada y de recién implantación en Canarias.

En cuanto a referencias externas, solamente se han encontrado algunas indicaciones a ciertas soluciones creadas por empresas privadas pero que es imposible acceder a ellas ya que son parte del modelo de negocio de las mismas y por tanto su documentación no es pública.

3.- Objetivos

Este proyecto final de carrera está orientado a la realización de una librería de alto nivel para la intervención de dispositivos TETRA y otro tipo de periféricos, presumiblemente un PC.

El origen de la creación y definición de esta librería surge debido a la problemática encontrada en la especificación de la interconexión de dicho dispositivo, ya que actualmente está totalmente orientada a nivel de abstracción muy bajo, siendo la mayoría de los mensajes a nivel de bits. Este bajo nivel nos lleva a la necesidad de crear esta librería para poder desarrollar aplicaciones para esta tecnología de una forma más rápida y amigable de cara al programador convencional, el cual no está interesado en las peculiaridades del hardware, sino en una solución para su problema.

Como segundo objetivo, se plantea la realización de una aplicación de servidor que utilice las características más destacables del dispositivo, realizando una pasarela entre varios servicios, como puede ser una conexión a un sistema de correo electrónico o simplemente con otras interfaces de radio ya implantadas.

En esta aplicación la intención es ir más allá de una simple demostración técnica, y además de la posibilidad de hacer de servicio de interconexión, se hará un registro de todos los mensajes recibidos de los distintos servicios y de la interfaz TETRA. También este software se diseñará con una interfaz abierta, con el fin de permitir integrar en el futuro nuevos servicios en forma de *plugins*.

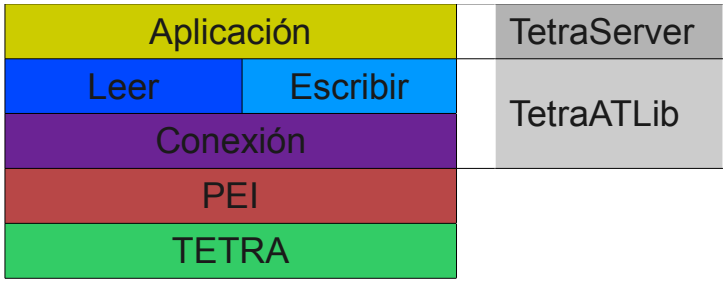

Figura 1.- Concepto de arquitectura a implementar.

Por último, es también objeto del presente proyecto la realización de una serie de herramientas que permitan el uso de esta librería de comunicación. En este ámbito se desarrollará una aplicación web que permita a un administrador ,el control de los dispositivos y parámetros a configurar para la interfaz de los servicios de comunicaciones por radio, y a un operador, las acciones para ejecutar los distintos servicios de mensajería.

4.- Metodología

La metodología seguida para la realización del proyecto de fin de carrera ha diferido según el tipo de objetivo a desarrollar. Por ello, se presenta a continuación una pequeña descripción de cada una de las metodologías de trabajo utilizadas: .

4.1.- Proceso Unificado de Larman

Para el desarrollo de la primera etapa “Librería TetraATLib” y la segunda etapa “Aplicación de pruebas TetraServer” del proyecto, se ha realizado un diseño orientado a objetos basándose en el proceso unificado de Ivar Jacobson, Grady Booch, James Rumbaugh.

El proceso unificado se basa en cuatro fases: iniciación, elaboración, construcción y transición. Cada una de estas fases esta sujeta a varias iteraciones, cuyo fin es refinar lo que se ha realizado anteriormente y aumentar las características del software.

Las características más importantes de esta metodología son:

- **Orientada a casos de uso:** En cada iteración se define los casos de uso que se implementarán durante la iteración.
- **Centrada en la arquitectura :** Al no existir un modelo único para todo un sistema, se pueden definir varios modelos y vistas que se adapten a los requisitos.
- **Enfocada al riesgo:** Se intenta en todo momento detectar los riesgos críticos y afrontarlos en las etapas más tempranas del proyecto con el fin de minimizar el impacto de una brecha en la arquitectura que retrase el desarrollo del producto.

En conjunción con el proceso unificado, se utiliza para modelar las interacciones de las distintas fases el lenguaje UML (*Unified Modeling Language*). UML es un lenguaje gráfico que se utiliza para visualizar, especificar, construir y modelar un sistema. En 1997 se estandarizó por el OMG (*Object Management Group*).

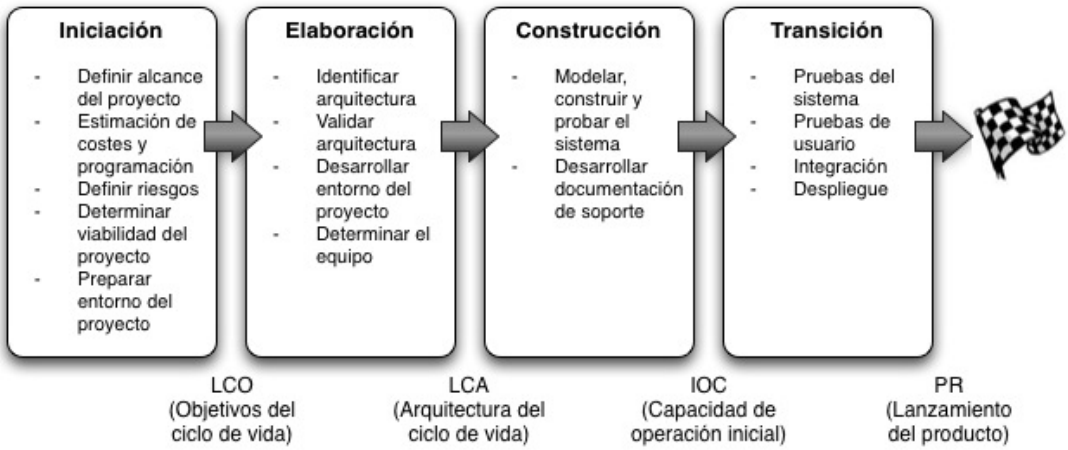

Figura 2.- Proceso unificado.

4.2.- Patrones de diseño GRASP y GoF

También durante la fase de diseño de esta etapa se hace uso de las sugerencias de los patrones de diseño GRASP y GoF (gang of four).

Grasp, de sus siglas en inglés *object-oriented design General Responsibility* Assignment Software Patterns, son una serie de herramientas, que se recomiendan usar a la hora de afrontar una serie de problemas muy frecuentes en el desarrollo de software. Estos patrones pertenecen al conjunto de recomendaciones de buenas prácticas.

GoF al igual que GRASP pertenecen también a una serie de recomendaciones. Se detallan en separado de GRASP por que, aunque su utilización es idéntica a la anterior, la primera aparición de estos patrones fue en el libro *Design Patterns*. Siendo sus cuatro autores conocidos como la banda de los cuatro (*Gang of Four*)

4.3.- Patrón Modelo-Vista-Controlador

Para definir la tercera etapa “Aplicación Web de Control TetraWeb” del proyecto de fin de carrera se prosigue la nomenclatura del *framework* CodeIgniter, el cual está basado en el patrón de desarrollo Modelo-Vista-Controlador, a partir de ahora MVC.

MVC es un enfoque de software que separa la lógica de la aplicación de la presentación. En la práctica, permite a las paginas web contener mínimo código ya que la presentación está separada del código PHP.

- El **Modelo** representa las estructuras de datos, donde las clases del modelo contendrán funciones que facilitarán la devolución, inserción y actualización de información de la base de datos.

- La **Vista** es la información que se presenta al usuario. Una vista es normalmente una página web, pero en CodeIgniter, una vista también puede ser un fragmento de página como el encabezado o pie de página. También puede ser una página RSS, o cualquier otro tipo de "página".

- El **Controlador** sirve como un intermediario entre el Modelo, la Vista y cualquier otro recurso necesario para procesar la solicitud HTTP y generar una página web.

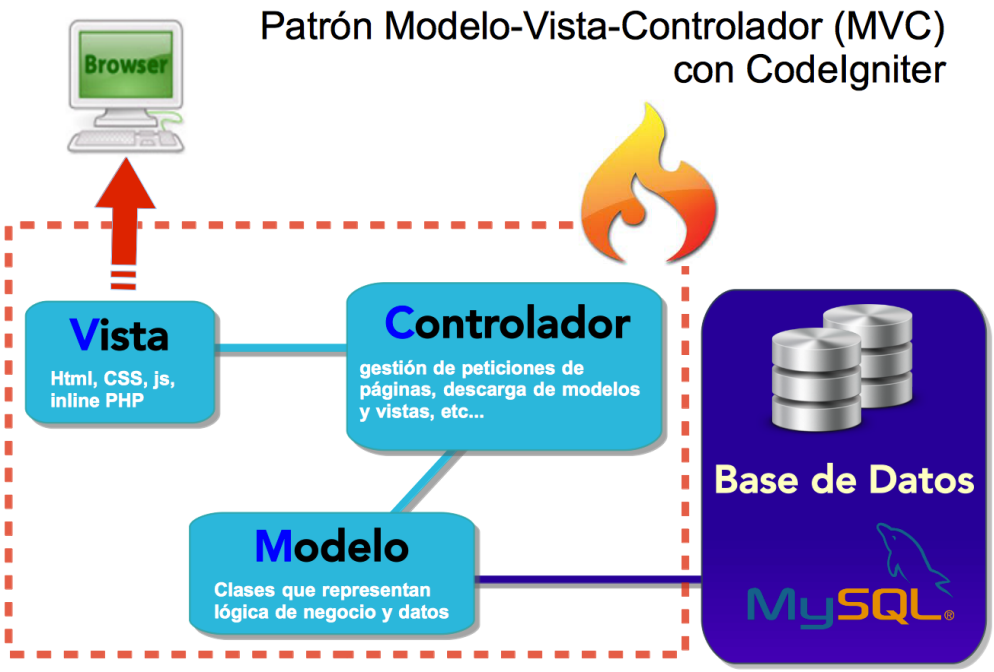

Figura 3.- Patrón Modelo-Vista-Controlador con CodeIgniter

5.- Recursos necesarios

Para la realización de este proyecto se ha dispuesto de una serie de recursos para llevar a cabo los objetivos. Estos recursos se pueden distinguir en tres grupos principales:

- Recursos Hardware
- Recursos Software
- Otros recursos

5.1.- Recursos hardware

Los recursos hardware para el desarrollo de este proyecto son los siguientes:

A) Dos PCs uno de ello suministrado por el departamento de Informática y Sistemas de la ULPGC utilizado para el desarrollo del software, tanto de la aplicación como de la librería y otro personal usado principalmente para el desarrollo de la documentación.

B) Dos terminales TETRA para las prueba suministrados por el servicio de seguridad de la ULPGC.

C) Un depurador de protocolos obtenidos del departamento de Informática y Sistemas.

D) Documentación de trabajos anteriores cedidos por los tutores del proyecto.

5.2.- Recursos software

Los recursos software utilizados en la definición de este proyecto han sido únicamente aquellos basado en Software Libre y bajo la Licencia Pública General de GNU.

Se detalla a continuación las aplicaciones utilizadas en este desarrollo:

- Sistema operativo GNU Linux 2.6 en distribuciones Ubuntu 12.04 y Centos 5.2. Todas las etapas de este proyecto han sido desarrolladas en este ámbito, así como la etapa de documentación.
- Editor NetBeans 7.3. Esta herramienta ha posibilitado la programación tanto de los *scripts* como las clases de las etapas de las aplicación de prueba y la aplicación Web.
- Editor de UML Umbrello 2.4.2. Herramienta gráfica que permite la elaboración de diagramas UML con gran facilidad. Pudiendo ademas desde la misma generar las cabeceras del programa diseñado.
- Suite ofimática LibreOffice 4.1. Paquete ofimático que ofrece un conjunto de herramientas al usuario bajo la licencia de Apache para el desarrollo de la memoria del trabajo, en concreto: *Apache LibreOffice Writer (*Procesador de textos), *Apache LibreOffice Calc (*Programa de hoja de cálculo) y *Apache LibreOffice Draw (*Herramienta de generación de diagramas).
- Un entorno básico de desarrollo web, siendo las aplicaciones usadas, el servidor web httpd de apache, la base de datos MySQL y por ultimo como lenguaje de programacion, las librerías necesarias para desarrollar en PHP.
- Uso del *MySQL Workbench* [13] para facilitar el modelado del trabajo. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de GNU.
- Se han utilizado cuatro principales navegadores, Internet Explorer, Mozilla Firefox, Google Chrome y Safari, para asegurar de que no hayan problemas a la hora de utilizar la plataforma en los distintos navegadores.

5.3.- Otras herramientas utilizadas

Además de los recursos descritos en los apartados anteriores, se definen a continuación otros recursos utilizados para llevar a cabo el proyecto:

- **Lenguaje de programación PHP**

  PHP es un acrónimo recursivo que significa PHP Hypertext Pre-processor (inicialmente PHP Tools, o, Personal Home Page Tools). Fue creado originalmente por Rasmus Lerdorf en 1994; sin embargo la implementación

principal de PHP es producida ahora por The PHP Group y sirve como el estándar de facto para PHP al no haber una especificación formal. Publicado bajo la PHP License, la Free Software Foundation considera esta licencia como software libre.

Puede ser desplegado en la mayoría de los servidores web y en casi todos los sistemas operativos y plataformas sin costo alguno. El lenguaje PHP se encuentra instalado en más de 20 millones de sitios web y en un millón de servidores, el número de sitios en PHP ha compartido algo de su preponderante sitio con otros nuevos lenguajes no tan poderosos desde agosto de 2005. Es también el módulo Apache más popular entre las computadoras que utilizan Apache como servidor web.

- **CodeIgniter, *framework* de PHP**

  CodeIgniter es un framework para desarrollo de aplicaciones - un conjunto de herramientas - para gente que construye sitios web usando PHP. Su objetivo es permitir desarrollar proyectos mucho más rápido que lo que podría hacer si escribiera el código desde cero, proveyéndole un rico conjunto de bibliotecas para tareas comunes, así como y una interfaz sencilla y una estructura lógica para acceder a esas bibliotecas. CodeIgniter permite enfocar creativamente un proyecto al minimizar la cantidad de código necesaria para una tarea dada.

- **HTML**

  HTML, siglas de *HyperText Markup Language* (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

  HTML también es usado para referirse al contenido del tipo de MIME text/html o todavía más ampliamente como un término genérico para el HTML, ya sea en forma descendida del XML (como XHTML 1.0 y posteriores) o en forma descendida directamente de SGML (como HTML 4.01 y anteriores).

- **CSS**

El nombre hojas de estilo en cascada viene del inglés Cascading Style Sheets, del que toma sus siglas. CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML (y por extensión en XHTML). El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

- **BootStrap**

Es una herramienta desarrollada por Twitter, liberada bajo licencia MIT en agosto de 2011, se trata de un *framework* CSS cuyo objetivo es fomentar la consistencia de las interfaces, tiene compatibilidad con HTML5 y CSS3, ademas proporciona un sistema de plantillas para desarrollo de interfaces *responsive*.

- **MySQL**

Es un sistema de gestion de bases de datos relacional de gran extensión, principalmente por esta una de sus versiones licenciado bajo GNU GPL. Ademas de tener una versión libre, MySQL ofrece una gran variedad e conectores con lenguajes de programación así como una fuerte comunidad y documentación esenciales en la resolución de problemas

- **BASH**

(Bourne-Again shell) Es un interprete de comando usado por defecto en la mayoría de las distribuciones de Linux. Creado como evolución de *Bourne Shell* (sh) Interpreta fichero con comandos internos, permitiendo desarrollar herramienta de alto nivel para tareas de administración básica.

- **C++**

Lenguaje de programación orientado a objetos, se creó como evolución del lenguaje C, es un lenguaje de nivel intermedio de abstracción, siendo idóneo para aplicaciones de sistemas, ya por un lado permite acceso a estructuras de alto nivel como la programación genérica o estructurada, y por otro lado puede

ejecutar código escrito en C y acercarse mas a los recursos hardware de la máquina.

6.- Plan de trabajo y temporización

Tal y como se ha descrito en los apartados anteriores, este proyecto de fin de carrera se ha separado en tres grandes etapas.

- **Etapa 1: Librería TetraATLib**. Análisis y diseño de la interfaz dde interconexión con un dispositivo TETRA. Estudio y desglose de las funcionalidades requeridas para proporcionar una librería altamente compatible con los requerimientos del cliente.

- **Etapa 2: Aplicación de pruebas TetraServer.** Análisis y desarrollo de una aplicación de pruebas de la librería TetraATLib. Estudio de integración de la librería en diversos módulos de prueba. Pruebas y verificación del funcionamiento de la librería de comunicación de radio en base a los objetivos establecidos.

- **Etapa3: Aplicación Web de Control TetraWeb.** Estudio y diseño de una aplicación web de control. Gestión de herramientas para facilitar el uso de la librería de comunicación TetraATLib. y la configuración de los parámetros y dispositivos involucrados en el sistema final del cliente.

Estas tres etapas han sido desarrolladas en un 70% de tiempo total dedicado en la realización de este proyecto de fin carrera, estableciendo alrededor de un 30% final en una última etapa de documentación y verificación del proyecto con los tutores asignados en este proyecto, así como con el cliente final.

Se presenta en los siguientes apartados el desglose del procedimiento llevado a cabo para cada una de las tres etapas aquí nombradas. A su vez, se detallan las distintos niveles de ciclo de trabajo llevados a cabo para completar de forma satisfactoria los requisitos solicitados del proyecto de fin de carrera con título "*Diseño y creación de una interfaz para el desarrollo de aplicaciones que utilicen servicios de comunicaciones por radio*"

7.- Etapa 1: Librería TetraATLib

La fase de análisis de esta librería difiere de los métodos normales de Ingeniería del software, esto es debido a que ya sabemos cuales son los requisitos de la librería, que no son más que la capacidad de acceder a toda la información que nos permite la definición del estándar ETSI EN 300 392-5.

7.1.- Introducción a TETRA.

Para conocer los requerimientos y funcionalidades proporcionados por TETRA, es objetivo del siguiente apartado describir el origen y las posibilidades ofrecidas por esta tecnología.

TETRA es un interfaz de radio digital diseñada por petición de la Unión Europea con el fin de unificar las distintas soluciones existentes en el mercado y especialmente diseñada para ser usada por la agencias gubernamentales, servicios de emergencia (policía, bomberos, ambulancias), servicios de transporte y el ejército.

La primera versión de TETRA fue publicada por el instituto ETSI en 1995. Este organismo también definió GSM, el cual es el estándar de telefonía móvil más extendido actualmente, es ahí la existencia de una gran variedad de similitudes entre TETRA y GSM. Su mayor diferencia radica en un mayor ancho de banda destinado a datos, y a la capacidad de alcanzar una mayor distancia de cobertura al poseer una banda de frecuencias menor.

TETRA tiene además la capacidad de establecer varios tipos de conexiones. El modo más común es el modo de llamada a grupos, que se comporta de la misma manera que un walkie-talkie convencional, transmitiendo la comunicación iniciada por un terminal en este modo a todos los terminales que estén suscritos al mismo grupo. También existe la posibilidad de hacer una llamada uno a uno, en cuyo caso los terminales involucrados se comportarán de un modo similar al de un teléfono móvil. Por último, tenemos las comunicaciones de emergencia, cuya utilización genera una parada de todas las demás actividades en ese momento ya que los planificadores de las pasarelas dan preferencias a estas comunicaciones.

Las comunicaciones por TETRA, aunque en casos de mala cobertura podrían llevarse a cabo en modo directo, están diseñadas para trabajar en modo *trunked*, a través de una infraestructura que les da servicio. A estas infraestructuras se les llama SwMI (*Switching and Management Infrastructure*)

Además de retransmitir señales de voz, TETRA soporta varios tipos de comunicaciones de datos. Si el control principal del sistema está preparado con un canal asignado para un circuito de datos, el sistema podrá transportar mensajes de estado y mensajes del tipo short data service (SDS).

Esta última parte es en la que vamos a centrar el estudio puesto que la capacidad de enviar y recibir datos de los terminales es la parte que nos interesa. Además de esta capacidad de enviar y recibir datos, todo dispositivo TETRA debe cumplir con un apartado de estándar que define la capacidad de conectar los terminales a periféricos, a esta parte del estándar se le llama PEI (*Peripheral Equipment Interface*). En ella están perfectamente especificados los tipos de parámetros de conexión, y los comando y estructuras de datos que se usarán como intercambio de mensajes. Para el acceso a los dispositivos, se han definido dos tipos de protocolos, AT y TNP1 ambos protocolos se describirán en secciones posteriores.

7.2.- SDS

Los *Short Data Service* (SDS) son mensajes cortos de texto similares a los SMS que se usan en la telefonía móvil convencional. Actualmente TETRA implementa cinco tipo de mensaje: mensaje de estado, SDS tipo 1, SDS tipo 2, SDS tipo 3 y SDS tipo 4. Se detalla a continuación cada uno estos tipos de mensaje:

- *Mensajes de estado*: Cadena de texto que devuelve un valor programado. Indicando en que estado se encuentra el dispositivo.

- *SDS tipo 1,2,3*: Este tipo de mensaje está compuesto de seis campos, el primer campo es un índice de mensaje con valores desde 0 a 65535 que nos permite

identificar el mensajes más allá del índice de la cola de mensajes, el siguiente campo es el campo que indica el destino o la fuente del mensaje, otro de los campos es un campo de estado que nos indica el estado del envío, el indicador del tipo de mensaje también añadiremos un campo con la marca de tiempo del servicio y por ultimo los datos del mensaje.

- *SDS tipo 4*: Además de llevar lo mismos campos que los SDS anteriores este tipo de SDS añade los siguientes, Id de protocolo TL y la cabecera del protocolo TL que incluye entre otro campos, la referencia del mensaje una petición de acuse de recibo, un periodo de validez y la dirección de retransmisión.

Internamente los SDS entrantes y salientes son almacenados en la pila de mensajes. Existen dos colas en todo dispositivo, una cola destinada para los SDS tipo 1,2, 3 y de estado y otra cola ideada para almacenar los SDS tipo 4. Ambas colas tienen un tamaño de 255. La codificación del texto es dependiente de las SIM y es un alfabeto de 8 bit ISO 8859-1. Este alfabeto predefinido incluye todos los caracteres especiales latino tales como vocales acentuadas, y letras con ñ o ç.

La diferencia entre el tipo de mensaje SDS 4 y el resto radica en que el mensaje de tipo 4 no tiene un tamaño fijo, lo cual lleva a una serie de precauciones a la hora de la implementación para evitar desbordamientos.

7.3.- PEI

En la arquitectura TETRA existe una capa llamada PEI *(Peripheral Equipment Interface*), la cual proporciona un protocolo de conexión entre un dispositivo TETRA y un periférico conectado al dispositivo.

En este protocolo es donde se va a albergar todo el peso del desarrollo de la librería, utilizando uno de los dos protocolos proporcionados por ETSI, durante todo el análisis se dio por supuesto que los terminales TETRA estarían operando en todo momento en modo *trunked*, ya que es así como se puede acceder a todos los servicios proporcionados por la red.

7.3.1.- Arquitectura Tetra PEI

En el diagrama de capas siguiente se puede observar en situación en que capa se encuentra el protocolo PEI y la representación de un periférico haciendo uso del mismo. En el diagrama se representa la conexión entre ambos con el símbolo RT

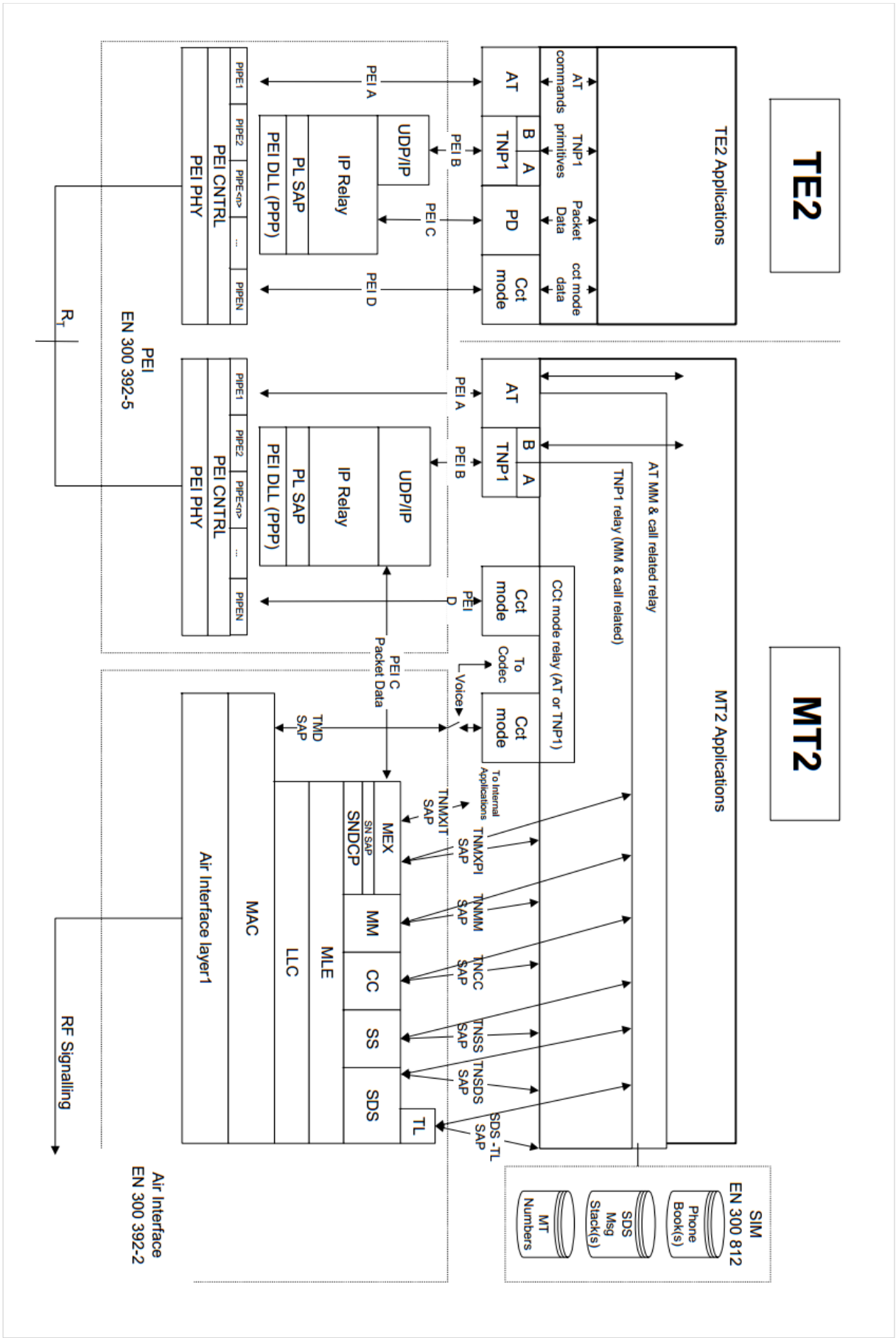
TE2
TE2 Applications
AT commands
TNP1 primitives
Packet Data
cct mode data
AT
B
A
TNP1
PD
Cct mode
PEI A
PEI B
PEI C
PEI D
UDP/IP
IP Relay
PL SAP
PEI DLL (PPP)
PIPE1
PIPE2
PIPE<n>
PIPEN
PEI CNTRL
PEI PHY
PEI
EN 300 392-5
R_T
MT2
MT2 Applications
AT MM & call related relay
TNP1 relay (MM & call related)
CCt mode relay (AT or TNP1)
To Codec
Voice
PEI C Packet Data
TMD SAP
To Internal Applications
TNMXIT SAP
TNMXPI SAP
TNMM SAP
TNCC SAP
TNSS SAP
TNSDS SAP
SDS -TL SAP
MEX
SNDCP
SN SAP
MM
CC
SS
SDS
TL
MLE
LLC
MAC
Air Interface layer1
RF Signalling
Air Interface
EN 300 392-2
SIM
EN 300 812
Phone Book(s)
SDS Msg Stack(s)
MT Numbers

Figura 4.- Lista de parámetros de configuración

7.3.2.- Capa física

La configuración de la conexión está basada en V.24 para la conexión física y el V.28 para las condiciones eléctricas. La transmisión estará formada por caracteres de 8 bits enviados de forma asíncrona con un start bit y un stop bit, en ningún caso se usara bit de paridad. La velocidad de conexión se recomienda que sea 9600kb/s aunque la mayoría de los terminales están preparados para aceptar una velocidad de conexión mayor.

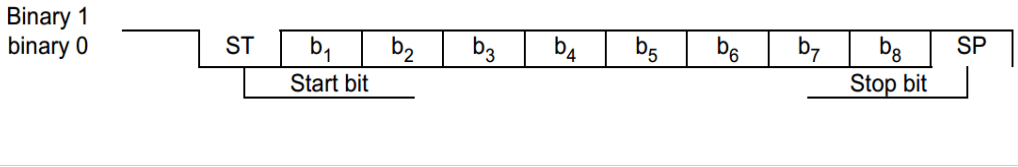

Figura 5.- Lista de parámetros de configuración

7.3.3.- Comando AT

Los comandos Hayes también conocidos como Comandos AT es un lenguaje desarrollado por la compañía Hayes Communications que se convirtió en estándar abierto para la interconexión de *modens*, con el fin de configurar su funcionamiento interno. EL prefijo AT abrevia la palabra *Atention* ya que estos comandos envían notificaciones al dispositivo para que éste las atienda.

La compatibilidad de interconexión a través de comandos AT en TETRA viene dada a la herencia de características de dispositivos GSM, en cuanto al conjunto de comando, como es de suponer no es idéntico, TETRA no usa todos los comandos de GSM y además añade una serie de comandos propios. Estos comandos son el modo por defecto de trabajar con los periféricos, por lo cual, en el momento de finalizar la configuración de la conexión, los dispositivos estarán preparados para aceptar comandos AT.

Mediante comandos AT podemos acceder a los servicios de voz, SDS, paquetes de datos y circuito de datos.

En cuanto a las limitaciones de este protocolo nos encontramos las siguientes:

- No soporta servicios suplementarios.
- El tamaño máximo de una linea está limitado.
- Solo se pueden usar los comandos AT cuando el dispositivo está configurado en este modo.
- El dispositivo periférico debe esperar un confirmación tras un envío de un SDS para poder enviar otro.

La sintaxis de un comando AT es la siguiente:
<Command Line>= <prefijo><cuerpo><carácter fin de linea>

Los tipos disponibles en el conjunto de comandos AT son

- *Set* (Establecer) : se envía un comando con los parámetros deseados para que el dispositivo remoto lo ejecute y modifique su configuración.
- *Read* (Leer) : envía un comando con el sufijo ? para que el dispositivo responda con el estado actual de la configuración solicitada. Esta operación se construye con el sufijo = más los datos del comando.
- *Test* : Envía un comando con el sufijo =? para preguntar por la sintaxis del comando enviado. La respuesta en caso de éxito suele ser una expresión regular BNF del comando.

El prefijo de todo comando es “AT” o “at”, si se da el caso que queremos repetir el carácter anterior,simplemente tendríamos que introducir en la linea de comandos los caracteres “A/” o “a/”. Toda linea acaba con un carácter de fin de línea. Para los comando AT este comando es el retorno de carro (*carry return*) codificado como el carácter 13 del código ASCII. En cuanto al cuerpo del comando, está compuesto de comandos individuales, en estos comando los espacio no influyen en el resultado, pudiendo ser usados para formatear el texto y hacerlo más legible.

7.3.4.- Protocolo TNP1.

Además del los comandos AT, TETRA nos da la posibilidad de interactuar con los dispositivos usando el protocolo TNP1(*TETRA Network Protocol type 1*) el cual permite a los periféricos tener acceso a los servicios de TETRA. Esto incluye el control del sistema móvil, control de las llamadas, SDS y servicios suplementarios. Como añadido TNP1 además permite acceder a los parámetros de configuración y almacenamiento. Prácticamente todos los servicios anteriores requieren de una infraestructura de servicios, para poder funcionar. Por ello en las primeras versiones de TNP1 era imposible habilitar el modo TNP1 si no se disponía de esta estructura, actualmente TNP1 define dos modos de trabajo, “local” o “wide” siendo el modo local el que nos permite conectarnos a los dispositivos y trabajar con todos los parámetros que no impliquen uso o configuración de los servicios.

TNP1 está basado en una conexión punto a punto (PPP), para la transmisión de mensajes TPN1 usa la capa de servicio UDP/IP. No hay que olvidar que aunque trabajemos sobre un protocolo IP, en la conexión solo estarán en todo momento el dispositivo TETRA y el periférico, por lo que para ahorrarnos el problemas de asignar direcciones IP a ambos dispositivos. ETSI ha definido que para las conexiones TPN1 habrán dos direcciones IP fijas para cada uno de los elementos, para el dispositivo TETRA la dirección IP es 10.0.0.100 y para el periférico es 10.0.0.101, para el uso del protocolo UDP se ha asignado el puerto UDP 4024 al protocolo TPN1.

Como se comentó anteriormente, los comandos AT es el modo de trabajo por defecto de los dispositivos. Para entrar en el modo AT una vez realizada la conexión física habrá que introducir una serie de comandos para cambiar el modo de trabajo. El primer comando que debemos introducir es para indicar si vamos a trabajar en modo “local” o “wide”, con el comando AT+WS46 enviando con este comando el valor 14 indicaremos modo wide y el valor 252 modo local. Todos los dispositivos TETRA asumen por defecto el modo wide. Tras indicar el modo, hay que comunicarle al dispositivo que entre en modo TNP1 enviando el comando AT#99*, la radio nos responderá con los parámetros de la conexión TNP1 y empezara a enviar tramas TLC para auto-configurarse con el demonio PPP del periférico.

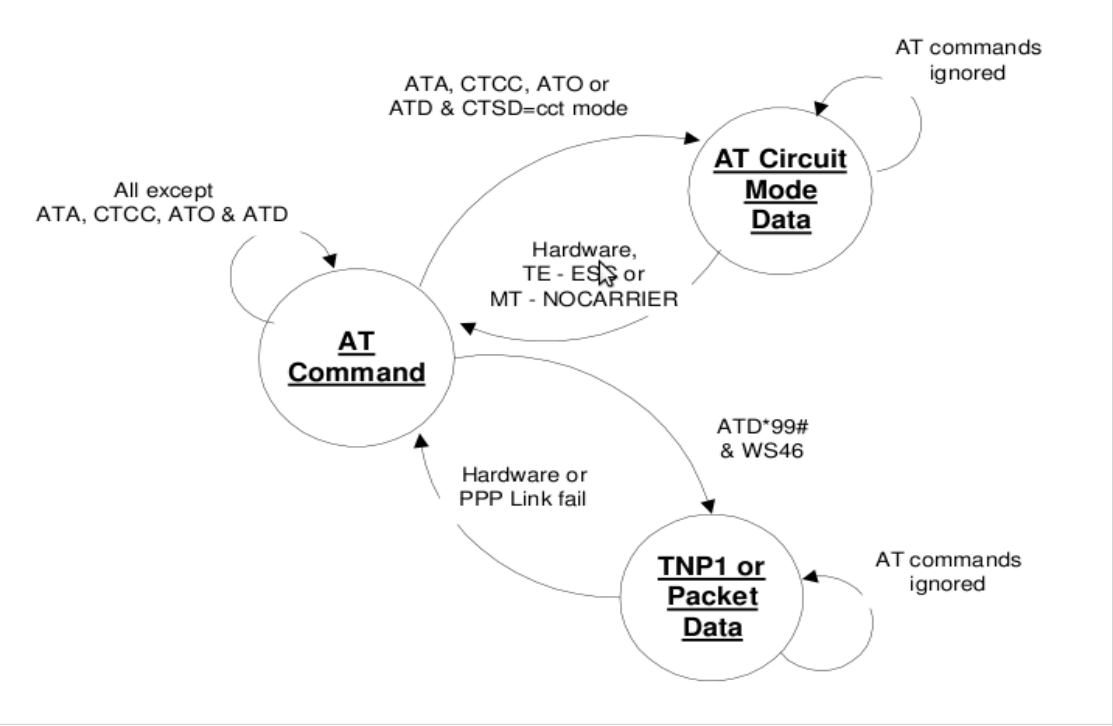

Figura 6.- Lista de parámetros de configuración

Tras arrancar el modo TNP1 y configurar el servidor ppp del servidor, después de abrir un *socket* la conexión ppp está creada y se puede empezar a enviar mensajes. Los mensajes TNP1 no son más que una serie de estructuras de datos llamadas PDU(*Packet Data Unit)* que se envían como contenido de una trama UDP por el *socket*. Cada PDU accede a uno o varios parámetros de configuración tanto para cambiarlos como modificarlos. Además podemos consultar números en la agenda de la SIM, acceder a las pilas de SDS o realizar llamadas entre otros.

Como ya se comento los PDU son estructuras de datos, cuyos campos son los parámetros de configuración o de consulta, alineados en octetos, para que sean mucho más fáciles de transmitir a través de una conexión serie.

Todo PDU tiene una esquema general, en todo PDU el primer campo es el PDU TYPE que indica que tipo de PDU se está llamando, seguido de esto viene los elementos de tipo 1 seguidos de tipo 2 y tipo 3. Estos tipos de elementos se diferencian entre sí por su tamaño, los tipo 1 y 2 de tamaño fijo y los tipo 3 de tamaño indeterminado. Los elementos tipo 1 se pueden enviar uno tras otro sin

indicar donde empieza uno o donde acaba otro, todo lo contrario sucede en los elementos tipo 2 y 3. Los elementos tipo 2 se suele utilizar cuando existen elementos opcionales, usando para indicar si este elemento opcional está presente un P-bit, octeto que precede a un elemento tipo 2 y que puede tener el valor 128 (el elemento opcional está presente) ó 0 (el elemento opcional no está presente) para no tener en cuenta el campo siguiente. Para los elementos tipo 3 se usa para indicar los tamaños y demás lo que ETSI llama el M-bit. El M-bit (more bits) siempre antes de un elemento tipo 3, es la estructura de 16bits representada en la siguiente figura.

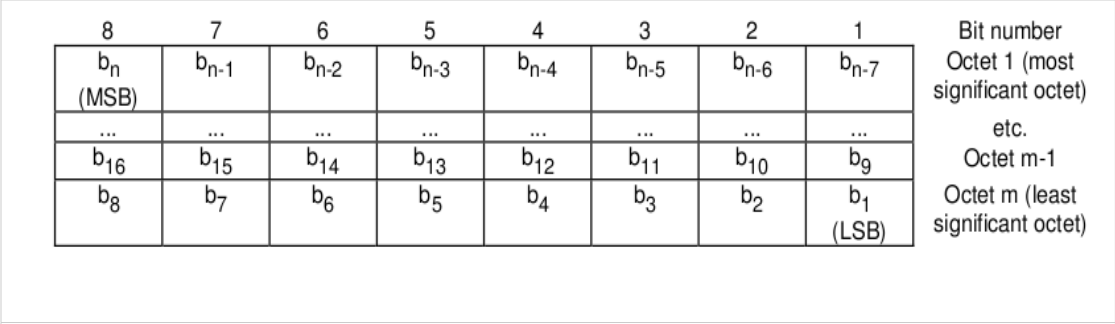

Figura 7.- Lista de parámetros de configuración

El M-bit de la estructura indica, si el campo al que precede está definido o no, el segundo campo *Type 3 element identifier*, identifica al set de PDU si que este elemento pertenece: CMCE,MM,MTA o SS. El resto del los bits se usan para indicar la longitud del elemento en bytes.

7.3.5.- Diferencias entre TNP1 y Comandos AT

Resumiendo lo comentado anteriormente, se expone a continuación una tabla comparativa entre las dos alternativas de comunicación:

Tetra PEI	
Comando AT	**TNP1**
Control de la movilidad	Control de la movilidad
Control de llamada de voz	Control de llamada de voz
SDS(Short data service)	SDS(Shot data service)

Circuit mode data	Circuit mode data
Configuración del estado del terminal	Configuración del estado del terminal
	Servicios suplementarios

7.4.- Primera toma de decisiones

Tras el estudio de las posibilidades de comunicación mediante periférico de TETRA se opto por la implementación de la capa que más posibilidades ofrecía, TNP1. Se realizaron varias pruebas de viabilidad, pero tras una primera impresión satisfactoria, se descubrió que los equipos de prueba de los que se disponía sólo implementaban un subconjunto muy reducido de PDUs TNP1, no incluyendo el manejo de SDS hacia el cual estaba dirigido principalmente este estudio.

Tras descartar el modo TNP1 , se comenzó a trabajar con comando AT, obteniendo mejores resultados. En este caso, además contábamos con un documento interno del fabricante, que especificaba mediante varios ejemplos como interactuaban estos comandos con los dispositivos que teníamos a nuestro alcance.

7.5.- Ejemplo de comunicación SDS

En este apartado se explicará como se comporta a nivel de comandos AT el envío y recepción con confirmación de los SDS.

Uno de los aspectos que se describió y que se considero fundamental para el funcionamiento del sistema es que los SDS funcionan con confirmación, y que el texto era escrito y leída del puerto en caracteres de números hexadecimales de 8 bit (2 caracteres leídos por carácter final) y cada dígito era el numero del código ASCII equivalente.

En esta operación, conceptualmente hay que diferenciar entre el terminal y el equipo al que esta conectado. Ya que la comunicación es entre ambos, siendo la red TETRA transparente para el equipo.

Acción	**Resultado**	**Explicación**
AT+CTSP=1,3,130<CR>	<CR><LF>OK <CR><LF>	El equipo 1 avisa al terminal 1 que desea enviar un SDS se registra para enviar texto
AT+CTSDS=12,0	<CR><LF>OK <CR><LF>	El equipo 1 indica al terminal 1 que quiere mandar un SDS tipo 4 (12)
AT+CMGS=102,88 <CR><LF> 8204E30148454C50204 D45 <Ctrl-Z>	<CR><LF>+CMGS:_0 <CR><LF> <CR><LF>OK <CR><LF>	EL equipo 1 enviar al terminal 1 el comando para que el terminal 1 envíe un SDS al terminal 2 con número 102 un SDS de 88 bits con el texto HELP ME
<CR><LF> +CMGS:_0,4,E3 <CR><LF>		El terminal 1 notifica un comando no solicitado que indica que se ha enviado el comando a la red
AT+CTSP=1,3,130<CR>	<CR><LF>OK <CR><LF>	El equipo 2 notifica al terminal 2 que desea recibir un SDS se registra para recibir texto
<CR><LF> +CTSDSR:_12,101,0,102 ,0,88 <CR><LF> 8204E30148454C50204 D45 <CR><LF>		El equipo 2 recibe del terminal 2 un comando no solicitado que indica que tiene un SDS del terminal 1 con numero 101 y de 88 caracteres que dice HELP ME
AT+CTSDS=12,0<CR>	<CR><LF>OK <CR><LF>	El equipo 2 comunica al terminal 2 que quiere enviar un SDS de tipo 4 (confirmación)
AT+CMGS= 101,32<CR><LF> 821000E3 <Ctrl-Z>	<CR><LF>+CMGS:_0 <CR><LF> <CR><LF> OK <CR><LF>	El equipo 2 enviar al terminal 2 un SDS tipo4 de 32 bit que es una confirmación de recepción del SDS del terminal 1
<CR><LF> +CMGS:_0,4,E3 <CR><LF>		El terminal 2 notifica un comando no solicitado que indica que se ha enviado el comando a la red
<CR><LF>+CTSDSR :_12,102,0,101,0,32 <CR><LF> 821000E3<CR><LF>		El terminal 1 recibe un SDS de tipo 4 del terminal 102 que es la confirmación de un mensaje que ha enviado anteriormente.

7.5.1.- Explicación de los Comandos AT

A continuación se explicará uno por uno, los comandos que intervienen en el proceso anterior, indicando sus parámetros, su propósito y su función dentro del ejemplo

AT+CTSP

Propósito: Lee o establece los servicios para los que el equipo conectado al dispositivo quiere tomar el control.

Sintaxis:

Tipo	Sintaxis	Respuesta
Establecer	AT+CTSP= <service_profile>, <service_layer1> [,<service_layer2>] <CR>	<CR><LF> OK<CR><LF>
		<CR><LF> +CME_ERROR :_<error report code> <CR> <LF>
Leer	AT+CTSP?<CR>	<CR><LF> +CTSP: _<service_profile>, <service_layer 1>, <service_layer 2> <CR><LF> [<service profile>, <service_layer 1> ,<service layer 2> <CR><LF>] <CR><LF> OK <CR><LF>
Test	AT+CTSP=?<CR>	<CR><LF> +CTSP :_(0-3),(0-4), (0-255)<CR><LF>

Parámetros:

Parámetro	Valor	Significado
<service profile>	0	Controlado por terminal
	1	Controlado por equipo
	2	Controlado por ambos
	3	Controlado por ninguno
<service layer 1>	0	Control de llamadas
	1	Control de movilidad
	2	SDS y estado
	3	SDS y TL
	4	Packet Data
<service layer 2>	0	Voice
	11	Group Management
	20	Status
	21	SDS tipo 1
	22	SDS tipo 2
	23	SDS tipo 3
	24	SDS tipo 4, PID entre 0 y 127
	25	SDS tipo 4, PID entre 128 y 255
	30	IP version 4
	0-255	PID cuando SDS-TL

Luego el comando usado en el ejemplo AT+CTSP=1,3,130 significa que se va a controlar por el equipo los SDS de tipo 4 con PID 130(el numero 130 se ha elegido al azar). Para que el destino sea notificado de la llegada del SDS ambos extremos tienen que controlar el mismo PID.

AT+CTSDS

Propósito: Lee o establece el envió y recepción de SDS.

Sintaxis:

Tipo	**Sintaxis**	**Respuesta**
Establecer	AT+CTSDS= <AI service>, <called party identity type> , [<area>] , [<access priority>], [<end to end encryption>] <CR>	<CR><LF> OK<CR><LF>
		<CR> <LF> +CME_ERROR :_<error report code> <CR> <LF>
Leer	AT+CTSDS?<CR>	<CR> <LF> +CTSDS: _<AI service>, <called party identity type> ,<area>, <access priority> ,<end to end encryption> <CR><LF> <CR> <LF> OK <CR> <LF>
Test	AT+CTSDS=?<CR>	<CR><LF>+CTSDS: _(9-13),(0-4), (0-15),(0-2),(0) <CR><LF> <CR><LF> OK <CR><LF>

Parámetros:

Parámetro	**Valor**	**Significado**
<AI service>	9	SDS tipo 1
	10	SDS tipo 2
	11	SDS tipo 3
	12(defecto)	SDS tipo 4
	13	Estado
<Called party identity type>	0(defecto)	SSI
	2	SNA
<area>	1(defecto)	Area no definida
	1-14	Area especifica
	15	Todas
<access priority>	0(defecto)	Baja
	1	Alta
	2	Emergencia
<end to end encryption>	0	Vacío
<priority level>	0(defecto)	Prioridad normal
	1	Alta prioridad

	2	Preventivo
	3	Emergencia
<comms type>	6	SDS sin confirmación
<importance factor>	1-6 (defecto 3)	Número de transmisiones por SDS sin confirmación

En el ejemplo () AT+CTSDS=12,0 , establece que se va a trabajar con SDS de tipo 4 y que la identificación de terminales es vía SSI.

AT+CMGS

Propósito

- Envía un SDS al terminal.
- Recepción de un SDS o de una confirmación.
- Cancela un SDS no enviado al media aéreo pero que se encuentra en la cola de enviados.

Sintaxis:

Tipo	**Sintaxis**	**Respuesta**
Establecer (envio)	AT+CMGS= <called party identity> ,<length> <CR><LF> <user data> <Ctrl-Z>	<CR><LF>+CMGS: _<SDS instance> <CR><LF> <CR><LF> OK<CR><LF>
		<CR><LF> +CME_ERROR :_<error report code> <CR><LF>
Establecer (Abortar)	AT+CMGS= <called party identity>, <length> <CR><LF> <user data> <ESC>	<CR><LF>OK<CR><LF>
		<CR><LF>+CME_ERROR:_<error report code><CR><LF>
No solicitado	No aplicable	+CMGS: <SDS Instance> ,[<SDS status>], [<message reference>]
Test	AT+CMGS=?<CR>	<CR><LF>+CMGS: _(1-1677214 ,00000001-9991638316777214, 1-255 ,0-99999999999999999999999) ,(8-1184) <CR><LF> <CR><LF> OK <CR><LF>

Parámetros:

Parámetro	**Valor**	**Significado**

<called party identity>	0–16777214	Para SSI
	00000001 – 999163831 6777214	Para TSI
	1–255	Para SNA
	Up to 24 digits	PSTN/PABX ESN
<SDS Instance>	0	Pila de SDS no soportada, solo una instancia permitida
<SDS Status>	4	Éxito en el envío
	5	Fallo en el envío
<length>	**Tamaño en bits**	
	16	Status
	16	SDS tipo 1
	32	SDS tipo 2
	64	SDS tipo 3
	8-1184	SDS tipo 4 si no incluimos TL el tamaño máximo es 1128

El comando utilizado en el ejemplo para enviar HELP ME a otro terminal (AT+CMGS=102,88 <CR><LF> 8204E30148454C50204D45 <Ctrl-Z>) significa que se envía un SDS de 88 caracteres al terminal con numero SSI 102.

El formato del campo <user data> no viene documentado en la especificación del comando pero su codificación es la siguiente. Pero aplicando ingeniería inversa a varios ejemplos y trazas se ha descubierto que su codificación es la siguiente:

Tipo	**Identificador**	**Secuencial**	**Datos**
8204(16 bits)	E3(8bits)	01(8 bits)	48454C50204D45
Especifica SDS tipo 4	Identificador del SDS entre los dos terminales utilizado para la confirmación	Numero de SDS, por si el SDS se rompen en varios trozos por tamaño	Mensaje, cada par de caracteres es la codificación ASCII del carácter original, p.e 48 = H , 45 = E

En el caso de que <user data> sea una confirmación de recepción el campo tiene los siguientes valores

Tipo	**Identificador**
821000	E3

Especifica un SDS de confirmación	Identificador del emisor

AT+CNUMF

Propósito: devuelve la identificación de red del terminal al que se esta conectado

Sintaxis:

Tipo	**Sintaxis**	**Respuesta**
Establecer	AT+CNUMF?<CR>	<CR><LF> OK<CR><LF>
		<CR><LF> +CME_ERROR :_<error report code> <CR><LF>
Leer	AT+CNUMF?<CR>	<CR><LF>+CNUMF:_ <num type>, <ITSI><CR> <LF> <num type> ,<PSTN Gateway> <CR><LF> <num type>, <PABX Gateway> <CR><LF> [<num type> ,<Service Centre ITSI> <CR> <LF>] <CR><LF>OK <CR> <LF>
Test	AT+CNUMF=? <CR>	<CR><LF>+CNUMF :_(0,2,3,4) , (000000000000000 - 999999916777214) <CR> <LF> <CR> <LF> OK<CR> <LF>

Parámetros:

Parámetros	**Valor**	**Significado**
<num type>	0	ITSI - ISSI
	2	PSTN
	3	PABX
	4	Centro de servicio ITSI
<ITSI>	00000000000 0000 - 9999999167 77214	Tres Primeros dígitos Código de país. 5 dígitos código de red, 8 dígitos ISSI
<PSTN Gateway>	00000000 - 16777214	8 dígitos de la pasarela PSTN
<PABX Gateway>	00000000-16777214	8 dígitos de la pasarela PABX
<Service Centre ITSI>	000000000 000000 - 99999991 6777214	Misma codificación que ITSI para servicios store and fordward
<ISSI>	1-13999999	Numero ISSI

Este comando se utilizará, ademas de para verificar que dispositivo de radio esta conectado al servidor, para enviarla a un terminal concreto si se detecta que ha estado mucho tiempo inactivo y se desea comprobar si sigue disponible.

7.6.- Análisis.

Una vez tomada la decisión de utilizar el protocolo de comando AT, el siguiente paso era un análisis en profundidad de lo que nos ofrecía la plataforma, nuestras necesidades y como adaptar todo en una solución que satisficiera los requisitos y a un posible cliente final de este desarrollo.

En la fase de desarrollo actual , debido a la naturaleza de la solución a implementar, se carece de un amplio repertorio de requisitos de usuario y por tanto, de casos de uso. Siendo esta parte más intensa en diseño que en análisis.

Hay que detallar que puesto que la librería se encargará de procesar tanto el envío como la recepción de comando AT desde el periférico hacia el terminal TETRA, así como como obtener y formatear los comando AT enviados por la radio hacia el dispositivo encargado de capturar los datos, se hace necesario desarrollar un método de comunicación libre de bloqueos y lo mas sencillo posible para el programador.

De cara al usuario se consideró que la forma más conveniente de trabajar con la librería seria mediante un objeto común que se encargara de enviar mensajes preprocesados al dispositivo TETRA y otro ente que estuviera en todo momento atento a la llegada de datos desde el terminal, así cada vez que se consultara devolver la información obtenida. El resto de clases groso modo sería una serie de clase que contengan las herramientas necesarias para trabajar con cada tipo de comando AT.

7.6.1.- Requisitos

Los siguiente requisitos, engloban todas las necesidades de la librería a desarrollar.

- Crear un mecanismo de envío de datos preformateados a los terminales conectados al equipo, ya sea de manera local, o remotamente mediante adaptadores LAN ↔ RS232.
- Proporcionar una interfaz que permita notificar cualquier tipo de información a objetos suscritos que puedan ser notificados de lo recibido por el terminal.
- Contener las herramientas necesarias para procesar un comando AT y obtener cualquier dato de un comando compatible.

7.7.- Diseño

En la fase de diseño se empezó dando soluciona a cada uno de los requisitos, para luego encargarlos todos juntos en una solución global.

El primer requisito, indica la necesidad de crear una herramienta de comunicación directa con la capa hardware, en este caso un puerto serie. Extrapolando esta idea a programación orientada a objetos, se considera que la mejor arquitectura para este problema es la aplicación de un patrón *facade*.

Al utilizar el patrón de fachada proporcionamos una interfaz única de acceso de escritura al puerto de comunicación. El objeto *facade* puede implementar toda la lógica necesaria para interaccionar con el subsistema al que enmascara. Esto permite reducir la complejidad de los objetos que hacen uso de el, reduciendo a su vez el acoplamiento de la solución. Añadir que este tipo de diseño es idóneo para la creación de un sistema por capas.

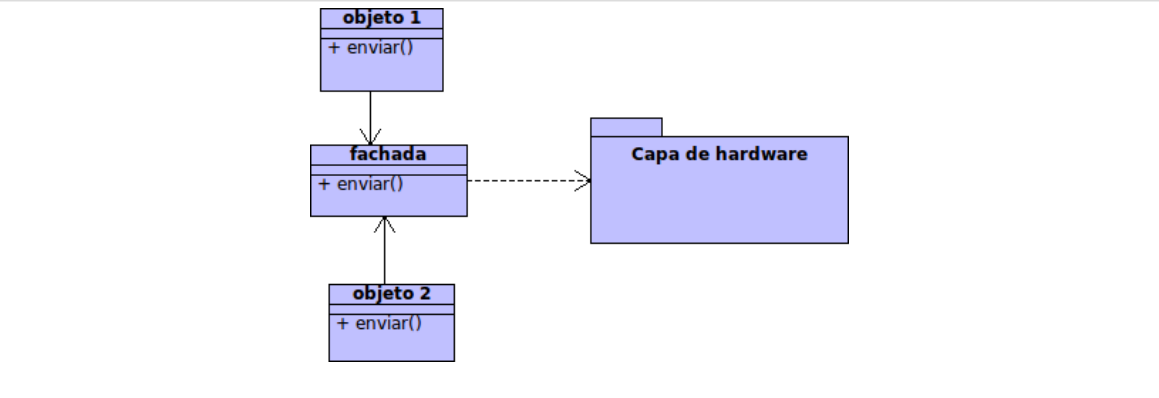

Figura 8.- Patrón fachada

Como segundo requisito se incluye la petición de un sistema que notifique todo lo leído del buffer de un *socket* de comunicación. En este caso, la arquitectura a implementar se basará en el patrón publicador-subscriptor.

El patrón publicador-subscriptor, es un concepto básico de intercambio de mensajes. En este modelo, existe una clase que obtiene datos de alguna fuente y lo desea

retransmitir a una serie de subscriptores que están a la espera de algún tipo de mensaje que tengan que procesar.

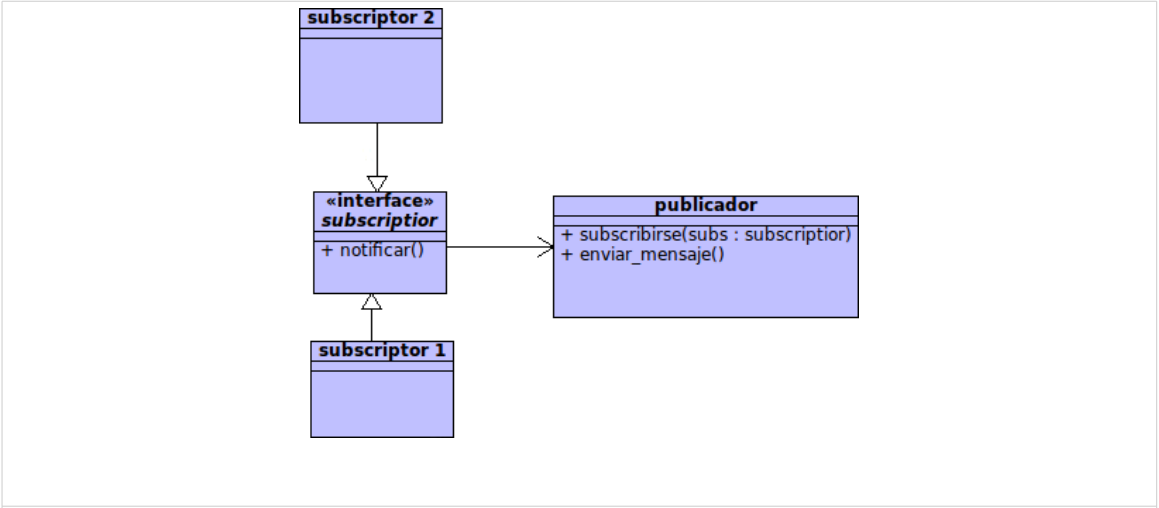

Figura 9.- Patrón publicador-subscriptor

Como se observa en la figura anterior, existe una clase interfaz, que todo subscriptor ha de implementar. Estas clases luego han de subscribirse usando el método *subscribir.* En este punto, ha de existir en la clase *publicador* una lista de subscriptores, que serán notificados cuando llegue un mensaje con el método notificar. Este método recorre la lista y envía el mensaje por el método *aviso.*

El tercer requisito del sistema, no lleva consigo ninguna consideración especial de diseño, simplemente se tratan de clases con herramientas para interpretar los mensajes y sus campos.

7.7.1.- Diagramas de colaboración

ConectaAT

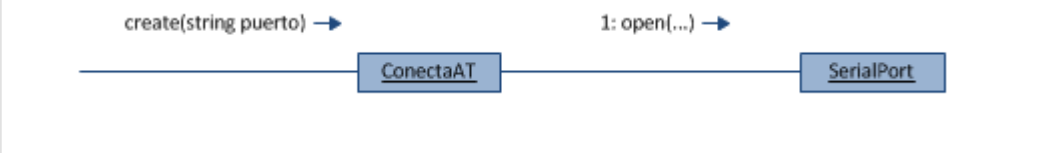

Figura 10.- Abrir conexión

En esta operación se hace uso de la librería *SerialPort* para conectar a un puerto serie especifico. En Linux hay que pasar la ruta del fichero especial que representa al puerto serie, normalmente /dev/tty*. La clase tendrá almacenada los parámetros de conexión que especifica el estándar ETSI EN 300 392-5 (ver 7.2.1 capa física)

Leer

Figura 11.- Leer del puerto

Al igual que conectar, la operación de leer del puerto se hace a través de una instancia *SerialPort* que tenga un puerto abierto. en este caso, se leerá linea a linea, siento el carácter salto de linea (\r) el que según la capa física simboliza el final de comando.

Enviar

Figura 12.- Enviar comando AT

Para enviar un comando por el puerto, se utiliza la clase *ATSimpleCommand* que para simplificar la arquitectura sólo envía una ristra de caracteres por el puerto, siendo competencia de las clases de nivel superior el encargarse de formatear correctamente el comando.

Esta operación sólo se ejecutara de forma satisfactoria sobre un puerto abierto.

Notificar

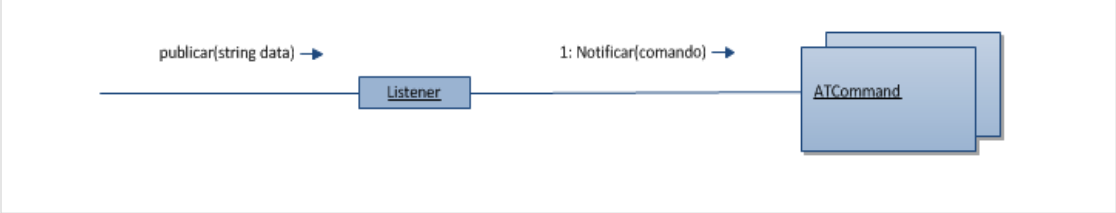

Figura 13.- Notificar comando AT

La función publicar, notifica a todas las clases suscritas a un *Listener* los datos que va leyendo por el puerto al que está asociado.

Suscribirse

Figura 14.- Suscribirse

Esta operación incrementa la lista de clases que implementen la interfaz comandoAT que se suscribirán al *Listener*. Las clases pueden ser suscritas en cualquier momento de la ejecución del programa.

7.7.2.- Diagrama de clases

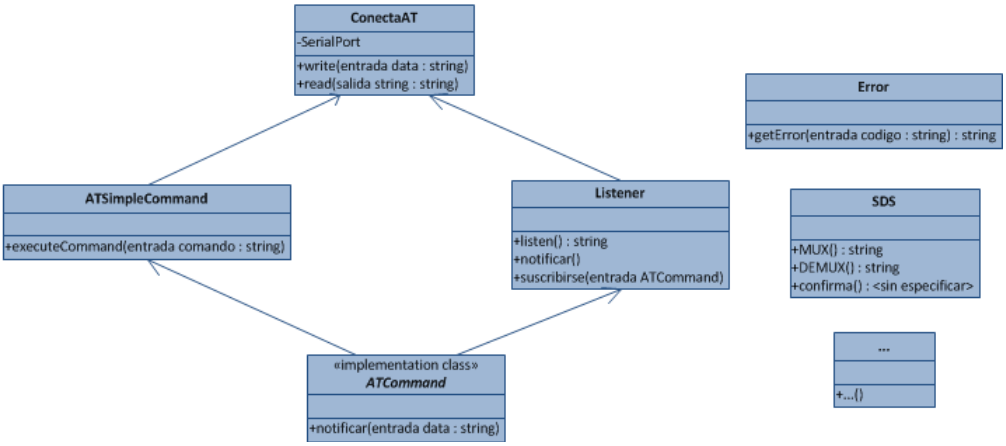

Figura 15.- Diagrama de clases

En el diagrama de clases podemos ver de forma global el acoplamiento de las clases y que a su vez sirve para entender mucho mejor el funcionamiento de la librería de conexión a TETRA.

Se puede observar que las clases de ayuda, tales como error o SDS, son independientes y no están acopladas en el diseño. Esta situación se debe a que sus instancias no dependen de un comando concreto, por ello también podrían declarase como estáticas.

7.8.- Problemas encontrados durante esta fase de desarrollo

Esta fase del proyecto, aunque en la visión global, parece la más sencilla de implementar y diseñar, en realidad es todo lo contrario, más allá de los problemas surgidos en el diseño con la falta de experiencia a la hora de diseñar soluciones de este tipo, puesto que durante todo el proceso de aprendizaje, las prácticas sólo están orientadas al desarrollo de métodos y clases sueltas, sin cohesión con un entrono mayor.

La falta de documentación por parte del fabricante y los distribuidores, que como se ha comentado con anterioridad, los mismos distribuidores de hardware, son a su vez distribuidores de software, añaden la dificultad de encontrar documentación y ejemplos de comunicación. Este problema de escasez de recursos para el aprendizaje puede observarse comúnmente en los recursos *online* durante la búsqueda de documentación, que obviamente, siendo la tecnología utilizada para este proyecto una tecnología destinada a la comunicaciones de organismos gubernamentales, no es muy extensa en esos medios.

Por estas razones, todo el proceso de ingeniería inversa requerido para obtener toda la información necesaria para manejar algo como el servicio de SDS que nos ofrece TETRA, ha sido un proceso extenso, especialmente en horas de prueba y error.

8.- Etapa 2: Aplicación de pruebas TetraServer

Para la etapa 2, se especifica paso a paso el análisis y el diseño realizado para llegar a la construcción de la solución de ejemplo de TetraATLib.

8.1.- Introducción a TetraServer

Como ejemplo de aplicación de la librería desarrollada, se propone realizar una pequeña aplicación de servido que sea capaz de comunicarse con dispositivos TETRA y usar uno o varios terminales como pasarela entre un servidor de correo y la capa de mensajes cortos (SDS) de TETRA.

Esta aplicación sera capaz de leer buzones de un servidor de correo local y enviar por mensajería SDS los correos electrónicos recibidos. La aplicación también sera capaz de realizar la operación inversa. Enviará correos electrónicos a los destinatarios, si el usuario introduce una dirección de correo a la hora de redactar un SDS.

Esta aplicación de ejemplo se ha desarrollado bajo la supervisión del departamento de seguridad de la ULPGC, siendo este departamento el encargado de dar los requisitos del mismo. Por ello, la arquitectura de la base de datos y modelo del dominio esta adaptado a las necesidades de este departamento.

8.2.- Análisis

En la fase de análisis de esta etapa, al ser una aplicación con prácticamente nula interacción del usuario, es decir, todas las operaciones serán analizadas desde el punto de vista de eventos del sistema, la elaboración de requisitos y casos de uso, se especificarán siempre desde el punto de vista del sistema, siento este el único actor de todos los procesos.

8.2.1.- Requisitos

Se detalla a continuación los requisitos estudiados con el cliente y el tutor del proyecto para llevar a cabo la pasarela de comunicaciones TetraServer:

- Ofrecer una capa de mayor nivel a la intercomunicación entre dispositivo TETRA y un periférico.
- El software desarrollado ha de funcionar como una aplicación de servidor.
- El software desarrollado ha de tener un funcionamiento 24/7
- El programa ha de estar capacitado para poder trabajar con más de un dispositivo TETRA, tanto para enviar como para recibir.
- El programa hará *pool* a un buzón de correo de usuarios del servicio TETRA y reenviará a los terminales el contenido de los mensajes que tenga pendiente el usuario.
- Los mensajes SDS transmitidos tendrán un máximo de 999 caracteres, si el mensaje es superior a este tamaño, el mensaje se enviará troceado.
- Los mensajes se retransmitirán en orden de llegada.
- Se esperará la confirmación de todo mensaje enviado, marcando como confirmado si ésta se produce.
- Se creará una confirmación a todo mensaje recibido.
- La aplicación ha de funcionar con la menor latencia posible, ya que está ideada como prototipo de una aplicación de mensajes de emergencia.
- El sistema procesará los mensajes que tengan la expresión regular “email;<dirección de correo>;mensaje” y reenviará el contenido al buzón de correo solicitado.
- La aplicación debe ofrecer una interfaz, para poder agregar servicios extra con la mayor facilidad posible.
- Se debe de almacenar en base de datos todo el contenido que se reciba o transmita por los terminales asociados.
- Cada usuario de dispositivo TETRA del sistema tendrá una dirección de correo asociada.

- La aplicación será capaz de detectar el estado del hardware asociado a ella, ya sea tanto por envío de mensajes a los terminales TETRA, como por envío de solicitudes *echo* a los dispositivos de red.
- A todo hardware asociado se le asignará uno o varios responsables.
- En el caso de detectar una desconexión de algún hardware asociado notificará por *email* a su responsable.

8.2.2.- Modelo del dominio

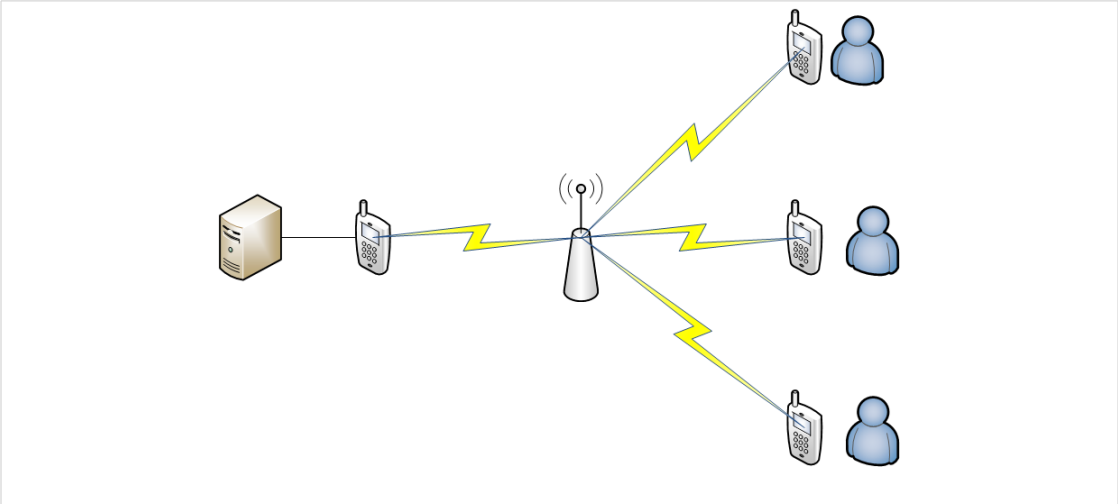

Figura 16.- Diagrama conceptual de TetraServer.

Conceptualmente, la aplicación a desarrollar, debe de realizar las funciones de una pasarela entre una matriz de dispositivos TETRA y una serie de servicios. Estos servicios han de funcionar mediante una interfaz común, siendo capaces con ayuda de la librería TetraATLib de procesar los comandos AT que provengan de los terminales.

Más concretamente, se pide como primer servicio el servir a la aplicación de una serie de buzones de correo electrónico, que previamente se hayan asociado a los terminales y que portan los usuarios del terminales TETRA en sus distintos puestos. Estos buzones servirán para que desde el exterior, se envíen correos a estos

buzones y el sistema los lea y los retransmita a los portadores de dispositivos en forma de SDS.

Los mensaje que provengan de los terminales SDS han de estar formateados mediante una expresión regular, la cual, será lo más reducida posible, puesto que debido a la construcción la mayoría de dispositivos no tienen una interfaz de entrada de datos muy eficiente.

La aplicación solicitada, como prototipo para una aplicación de *relay* de mensajes destinada a la seguridad, se ha de enfocar en todo momento al mejor rendimiento posible, nunca sacrificando la seguridad, esto quiere decir, que los tiempos de respuesta entre el envío de un mensaje por un servicio asociado y su retransmisión por SDS al destinatario, ha de ser el menor posible. Por otro lado, como consecuencia del futuro uso del desarrollo, el tránsito de la información ha de quedar registrado en el sistema, ya sean mensajes, comandos pertenecientes a otros servicios o confirmaciones.

En cuanto al hardware que se conectará a la aplicación, llamados por la nomenclatura de ETSI TE (*Terminal Equipment*), podrán estar conectados de forma local, mediante la interfaz RS232 de la máquina que esté ejecutando el programa o de forma remota mediante pasarela *RS232* ↔ *Ethernet*. En el caso de usar el segundo tipo de conexión, se añade el problema del posible mal funcionamiento de la pasarela. La forma más sencilla que existe de controlar esto es proveer de un temporizador que consulte a las pasarelas cada periodo y ver si alguna no responde. También habrá que dotar al software de una herramienta que sea capaz de detectar la inactividad prolongada de alguno de los dispositivos TETRA asociados.

Para los recursos hardware existirá la posibilidad de asignar responsables para en el caso de detectar una inactividad, generar una alarma que notifique de su correcto funcionamiento. La alarma será un correo electrónico que informará del dispositivo que causó el error.

Como medida de protección adicional, se plantea el requisito de funcionalidad 24/7, para ello, habrá que proveer un mecanismo básico de auto-mantenimiento en el caso

de que se detecte que el proceso principal por alguna razón desconocida deje de responder.

El diseño de la arquitectura de la aplicación ha de dar la opción a incluir nuevas funcionalidades a este software de manera sencilla, aunque a ultima instancia haga falta un desarrollador con un perfil alto en programación.

Destacar, que una aplicación de este calibre, ha de tener la propiedad de ser parametrizable, pudiendo, por ejemplo, ajustar los temporizadores de espera entre envíos de SDS o de latencia de los terminales. Se dispondrá, para ello, de una interfaz *GUI* vía web para gestionar la configuración básica de los parámetros del software inicial, pudiendo ser reutilizable también en el caso de añadirle nuevos modos de funcionamiento. Al ser el desarrollo de esta interfaz gráfica de un gran peso dentro del proyecto se le ha dedicado a la misma la tercera etapa de desarrollo.

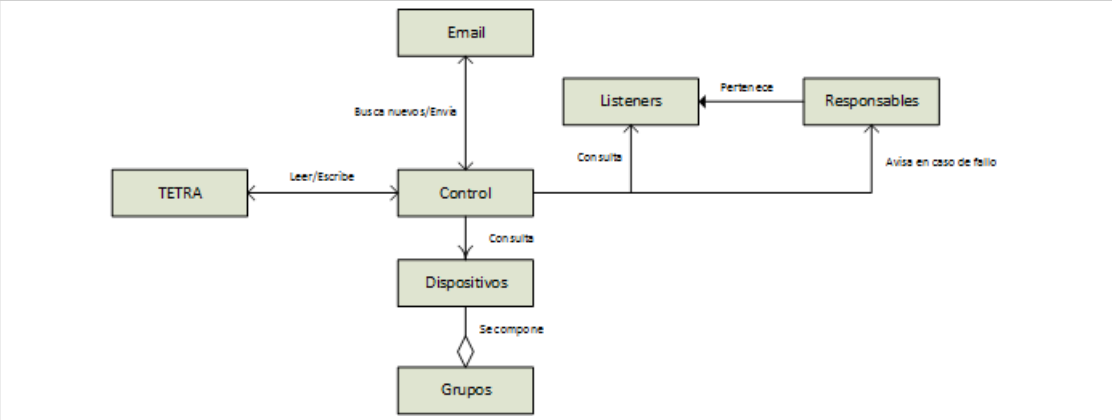

Figura 17.- Modelo del dominio

8.3.- Casos de uso

En la siguiente fase de análisis se expondrán los casos de uso obtenidos a partir de los requisitos. Primeramente se darán en forma de diagrama de casos de uso, para luego detallarlos en formato completo según indica el proceso unificado.

8.3.1.- Diagramas de casos de uso

Tras el análisis de las operaciones a realizar por la aplicación se obtuvieron los casos de usos que se incluyen en la imagen siguiente.

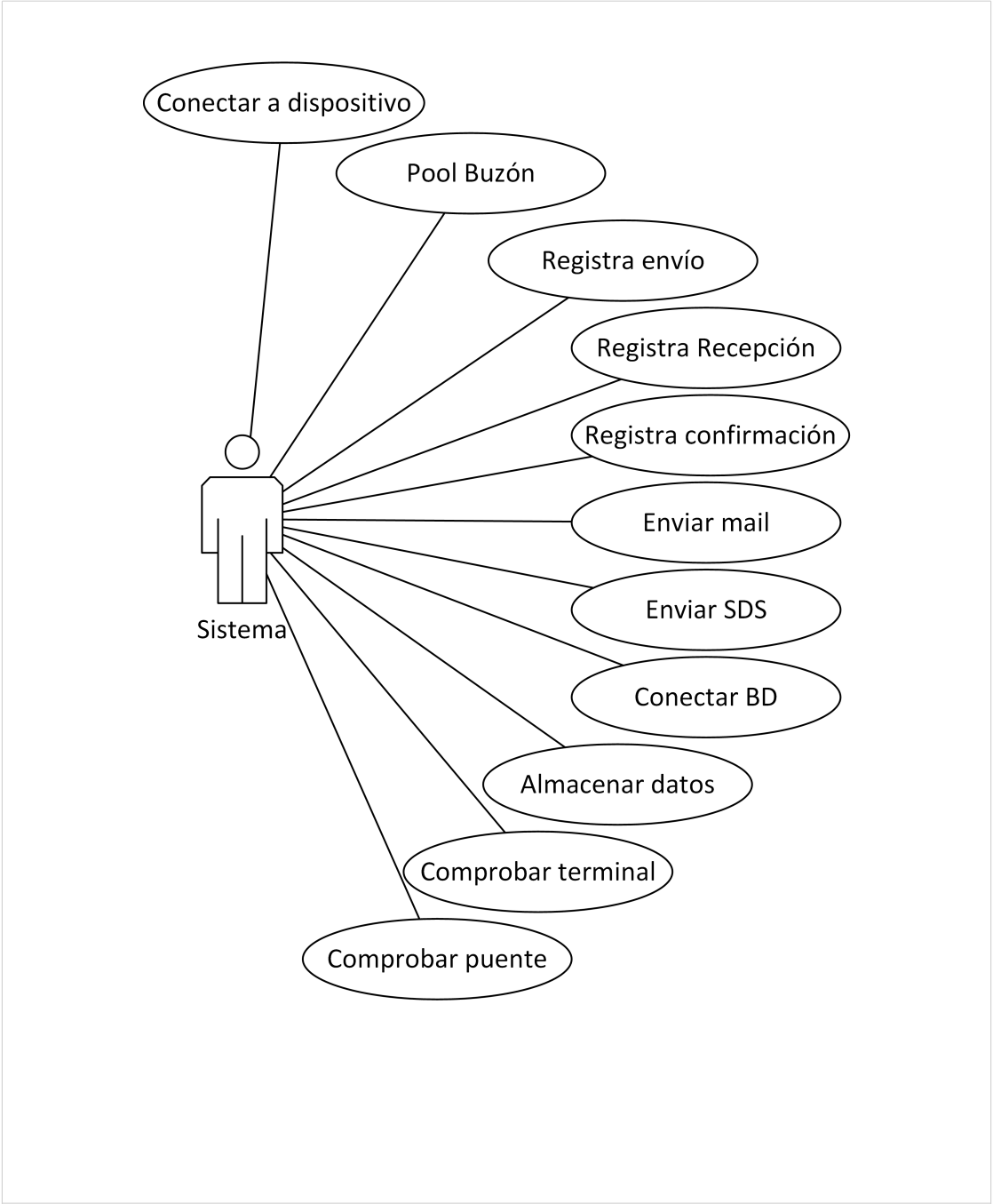

Figura 18.- Diagrama de casos de uso

8.3.2.- Casos de uso formato completo.

A continuación se detallarán los casos de uso detectados en el sistema en formato completo.

8.3.2.1.- Caso de uso 1 : Conectarse a dispositivo

Actor	Sistema
Personal involucrado	*Sistema*: Tiene que crear las conexiones de los dispositivos almacenados en la base de datos.
Precondiciones	Los datos de los terminales conectados deben estar en la base de datos. Los terminales tienen que estar físicamente conectados al servidor.
Garantías de éxito	Se obtendrá el comando AT que confirma la conexión
Escenario principal	1. El sistema obtiene los datos de todos los terminales. 2. El servidor crea una instancia de conexión con un terminal TETRA. 3. El terminal responde con un OK a la conexión. *Repetir paso del 1 al 3 hasta que se conecte con todos los dispositivos.*
Flujos alternativos	*a. En cualquier momento el sistema falla: 1.El disparador avisa al sistema de que algo va mal. 2.El sistema intenta reconectar el dispositivo. 2a. El error persiste tras reintento. 1. Generar fichero de estados. 2.a Tras un tiempo no se recibe respuesta del dispositivo. 1. Se notifica al responsable del error en el dispositivo.
Frecuencia	Al iniciar la aplicación.

8.3.2.2.- Caso de uso 2: Pool buzón

Actor	Sistema
Personal involucrado	*Sistema*: quiere estar enterado de todos los mensajes entrantes en los buzones de correo electrónico
Precondiciones	El servidor de correo debe almacenar los correos en buzones accesible por el sistema
Garantías de éxito	Se leen todos los mensajes pendientes de los buzones de los usuarios del sistema.
Escenario principal	1. El sistema cargar la ruta de los buzones 2. El sistema consulta un buzón 3. El sistema busca en el buzón los mensajes no leídos. 4. El sistema coge un mensaje no leído y lo manda a la cola de salida. 5. El disparador marca el mensaje como leído. *Repetir los pasos 2-5 hasta que estén todos los mensajes leídos* 6. El disparador salta al siguiente buzón de la lista. *Repetir 2-6 hasta que no queden buzones que leer*
Flujos alternativos	*a en cualquier momento el sistema falla. 1. Abortar la ejecución de la tarea y repetirlo más adelante. 2. Marcar error en un registro de log 1a. No se encuentra ruta para buzones validos. 1.La tarea es abortada por el disparador y escribe un registro de log con dicho error. 2a. No se encuentra ningún buzón que corresponda a un usuario del sistema de radio. 1.El disparador termina su ejecución sin encolar ningún mensaje. 3a. No encuentra mensajes nuevos para ese usuario.

	1.El disparador salta al siguiente buzón. 4a. No se puede encolar el mensaje. 1. Abortar la tarea y esperar a una nueva ejecución. 2. Reintentar pasado un tiempo. 2a. Tras varios reintentos no se puede insertar en la cola. 1. Escribir en el registro de log un mensaje advirtiendo de este error repetido.
Frecuencia	Periódica, indicada por un parámetro que indique la periodicidad de la acción

8.3.2.3.- Caso de uso 3: Registra envío

Actor	Sistema
Personal involucrado	*Sistema*: quiere un histórico de todos los mensajes que envía y recibe.
Precondiciones	Se ha enviado un mensaje por la interfaz de radio.
Garantías de éxito	Se leen todos los mensajes pendientes de los buzones de los usuarios del sistema.
Escenario principal	1. El sistema conecta con la base de datos. 2. El sistema ejecuta una sentencia SQL para inserta el mensaje en la tabla correspondiente. 3. El sistema marca el envío como pendiente.
Flujos alternativos	*a : El sistema falla en cualquier punto. 1. Abortar al operación y volver al estado anterior. 1a. El sistema no puede conectar a la base de datos. 1. El sistema escribirá en el log este evento y continuara su ejecución.

Frecuencia	Cada vez que se realice un envía, muy frecuente.

8.3.2.4.- Caso de uso 4: Registra recepción

Actor	Sistema
Personal involucrado	*Sistema*: quiere un histórico de todos los mensajes que envía y recibe.
Precondiciones	Se ha recibido un mensaje por la interfaz de radio.
Garantías de éxito	Se envía a su destinatario de correo el mensaje recibido.
Escenario principal	1. El sistema conecta con la base de datos. 2. El sistema ejecuta una sentencia SQL para inserta el mensaje en la tabla correspondiente.
Flujos alternativos	*a : El sistema falla en cualquier punto. 1. Abortar al operación y volver al estado anterior. 1a. El sistema no puede conectar a la base de datos. 1. El sistema escribirá en el log este evento y continuara su ejecución.
Frecuencia	Cada vez que se realice un envía, muy frecuente.

8.3.2.5.- Caso de unos 5: Registra confirmación

Actor	Sistema
Personal involucrado	*Sistema*: tiene que marcar todas las confirmaciones recibidas
Precondiciones	Se ha recibido un mensaje de confirmación por la interfaz de radio.
Garantías de éxito	El sistema marcar un mensaje previamente enviado como recibido y almacenara la confirmación en otro registro.
Escenario	1.El sistema detecta que el comando recibid es un mensaje

principal	de confirmación 2.El sistema procesa el mensaje y obtiene los campos de cabecera 3.El sistema busca en la base de datos el SDS enviado previamente y en el caso de existir lo marca como recibido. 4. El sistema almacenara a confirmación completa en la base de datos.
Flujos alternativos	*a : El sistema falla en cualquier punto. 1. Abortar al operación y volver al estado anterior. 1a. El sistema no puede procesar el SDS. *1. Saltar directamente al paso 4.*
Frecuencia	Cada vez que se realice un envía, muy frecuente.

8.3.2.6.- Caso de uso 6: Enviar mail

Actor	Sistema
Personal involucrado	*Sistema*: Tiene el deber de retransmitir los mensajes entrantes.
Precondiciones	Se ha recibido un mensaje entrante para mandar por correo
Garantías de éxito	El sistema envía correctamente el mail al usuario al que va destinado el mail.
Escenario principal	1.El sistema conecta son un relayer de correo 2.El sistema envía el correo al usuario destinatario. 3.El sistema marca el correo enviado como enviado.
Flujos alternativos	*a : El sistema falla en cualquier punto. 1. Abortar al operación y volver al estado anterior. 1a. El sistema no puede conectar con el *relayer*. 1. El sistema reintentara enviar pasado un tiempo.
Frecuencia	Cada vez que envíe un correo, muy frecuente.

8.3.2.7.- Caso de uso 7: Enviar SDS

Actor	Sistema
Personal involucrado	*Sistema*: Tiene el deber de retransmitir los SDS salientes
Precondiciones	Hay en la base de datos un SDS pendiente de enviar.
Garantías de éxito	El sistema envía correctamente el SDS al usuario.
Escenario principal	1. El sistema recoge la información y la empaqueta según la especificación de datos para los mensajes SDS. 2. El sistema encapsula el mensaje en un datagrama de radio 3. El sistema envía el mensaje completo por la interfaz de radio y espera por la a confirmación 4. El sistema recibe de la radio la confirmación de emisión del mensaje. 5. El sistema marca como enviado el SDS en la base de datos. 6. El sistema actualiza el mensaje en la base de datos y lo marca como enviado.
Flujos alternativos	*a. el sistema falla en algún momento. 1. abortar la operación y dejar las colas como están para volver al estado actual tras solucionar errores. 3.a Se detecta un error al enviar el mensaje. 1.El sistema reintentara el envío una serie de veces 2.si el problema persiste el sistema escribirá en el log un registro informando de esto y se parara a la espera de que se le indique que vuelva a realizar al operación en curso. 3b. No se recibe confirmación del mensaje. 1.El sistema eliminara de la cola el mensaje y lo marcara en la base de datos como, sin confirmación.

Frecuencia	Cada vez que se realice un envío, muy frecuente.

8.3.2.8.- Caso de uso 8: Conectar con la base de datos.

Actor	Sistema
Personal involucrado	*Sistema*: Quiere conectarse con la base de datos.
Precondiciones	Ninguna.
Garantías de éxito	La base de datos está preparada para recibir sentencias SQL.
Escenario principal	1. El sistema carga la configuración de un fichero de configuración. 2. El sistema conecta con la base de datos. 3. La base de datos confirma la conexión.
Flujos alternativos	Cuando se inicie la aplicación. Poco frecuente.
Frecuencia	

8.3.2.9.- Caso de uso 9: Almacenar datos

Actor	Sistema
Personal involucrado	*Sistema*: Quiere almacenar todos los datos que envíe la radio.
Precondiciones	El sistema ha de estar conectado a un dispositivo.
Garantías de éxito	Los comando serán almacenados en la base de datos.
Escenario principal	1. El sistema lee un comando AT que no va relacionado con ningún servicio 2. El sistema lo almacena en la base de datos para que otros servicios externos lo puedan interpretar

	3. La base de datos confirma la conexión.
Flujos alternativos	Cada vez que se lea un comando, muy frecuente.
Frecuencia	

8.3.2.10.- Caso de uso 10: Comprobar estado terminal Tetra.

Actor	Sistema
Personal involucrado	*Sistema*: Quiere ver si un dispositivo que lleva tiempo sin notificar nada sigue funcionando correctamente
Precondiciones	El dispositivo se conecto con anterioridad
Garantías de éxito	El dispositivo está funcionando correctamente
Escenario principal	1. El sistema envía un comando AT de identificación al dispositivo. 2. El sistema lee la información de la respuesta AT 3. El sistema actualiza en la base de datos la información del dispositivo.
Flujos alternativos	*a. el sistema falla en algún momento. 1. abortar la operación y dejar las colas como están para volver al estado actual tras solucionar errores. 2a. No se recibe respuesta del dispositivo. 1. Caso de uso 1. 1a. No se puede conectar. 1. Escribir en el log del sistema el error. 2. Enviar email a responsable.
Frecuencia	Después de un *timeout* especifico.

8.3.2.11.- Caso de uso 11: Comprobar estado de puente.

Actor	Sistema
Personal involucrado	*Sistema*: Quiere ver si un dispositivo puente está funcionando de manera correcta
Precondiciones	El dispositivo se conecto con anterioridad
Garantías de éxito	El dispositivo está funcionando correctamente
Escenario principal	1. El sistema carga los parámetros de conexión IP del dispositivo. 2. El sistema envía un comando ICMP de tipo ECHO al dispositivo 3. El comando registra la respuesta del puente. *Repetir para todos los puentes*
Flujos alternativos	*a. el sistema falla en algún momento. 1. abortar la operación y dejar las colas como están para volver al estado actual tras solucionar errores. 2a. No se recibe respuesta del dispositivo. 1. Se marca como inactivo a el y todos los terminales que dependen de el. 2. Se notifica al responsable del error.
Frecuencia	Periódica, indicada por un parámetro que indique la periodicidad de la acción.

8.4.- Diseño

En el apartado de diseño de TetraServer, se detallará en primer lugar el diseño de la base de datos, según el modelo conceptual de la aplicación, seguidamente se expondrán los diagramas de colaboración de las diferentes clases del sistema, para luego explicar el diagrama de clases y la visión global del programa a desarrollar.

8.4.1.- Diseño base de datos

Como punto de apoyo a la aplicación TetraServer se hace uso de una base de datos MySQL. En dicha base de datos, almacenaremos las capturas de los terminales, la configuración y los usuarios.

8.4.1.1.- Configuración

Para almacenar los parámetros de configuración se hará uso de dos tablas. En la que una almacenaras principalmente los parámetros en modo clave-valor, además de estos valores tendrá una clave ajena a otra tabla maestra que categorizará los parámetros por categorías. Pudiendo estas categorías tener tantos registros clave valor como desee.

Esta estructura de tabla da la libertad a usuarios posteriores poder crear sus ámbitos de parámetros, sin tener que hacer cambios en la estructura de la base de datos.

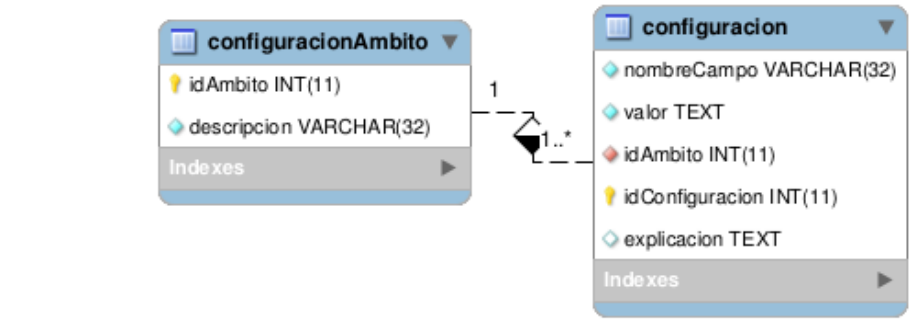

Figura 19.- Parámetros de configuración

8.4.1.2.- Dispositivos y puestos.

Debido al tipo de topología de la infraestructura de radio, es más que probable que los dispositivos TETRA tengan más de un usuario, ya que por ejemplo un vigilante al final del su turno puede dejar su material de trabajo y un compañero sustituto hacerse cargo de los pertrechos del anterior. También es muy común el caso de un usuario, que durante el desempeño de sus funciones, disponga de más de un terminal TETRA para su uso, un ejemplo claro sería aquellos vigilantes

que patrullen con un vehículo, teniendo a su disposición su radio personal y el terminal que incorpora un vehículo.

La casuística descrita hace que el mejor sistema para almacenar estos conceptos sea mediante una relación de tres tablas, donde se puedan asociar múltiples terminales TETRA a múltiples puestos de trabajo.

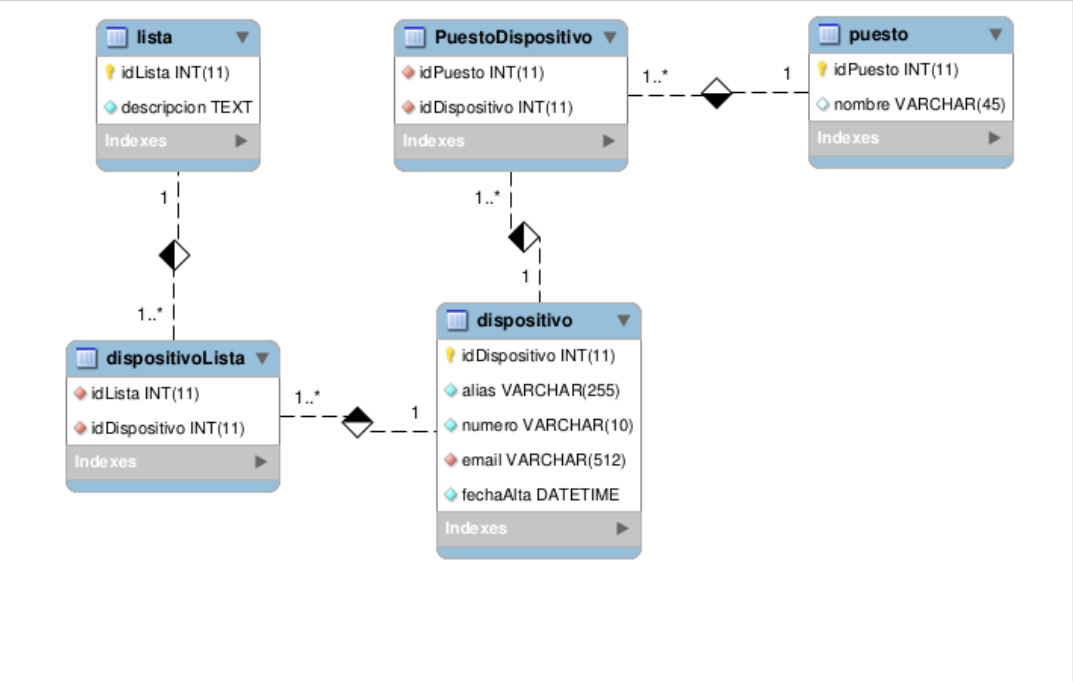

Figura 20.- Puestos y dispositivos

8.4.1.3.- Capturas

Para el almacenamiento de las capturas de los distintos dispositivos primero hay que tener en cuenta, que en ultima instancia esta tabla tendrá el comportamiento de un *log*, pudiendo a través de ella llevar un registro del comportamiento de mensajes, así como un registro del resto de comando recibidos desde los terminales. Así que esta tabla, aparte de un campo de texto libre donde estará el comando tal y como se captura desde la interfaz, existe la posibilidad de que otro servicio lo use para sus propósitos. En este caso deberá almacenar los extremos de comunicación de los mensajes (receptor , emisor), no olvidar los campos básicos de cualquier log, fecha , hora y estado.

También como herramienta extra para la conversión a correo electrónico. Se añade una tabla auxiliar para capturas, que almacenara los datos extra que se generen en el envío y recepción de correos electrónicos, asunto, dirección de correo de destino y texto formateado.

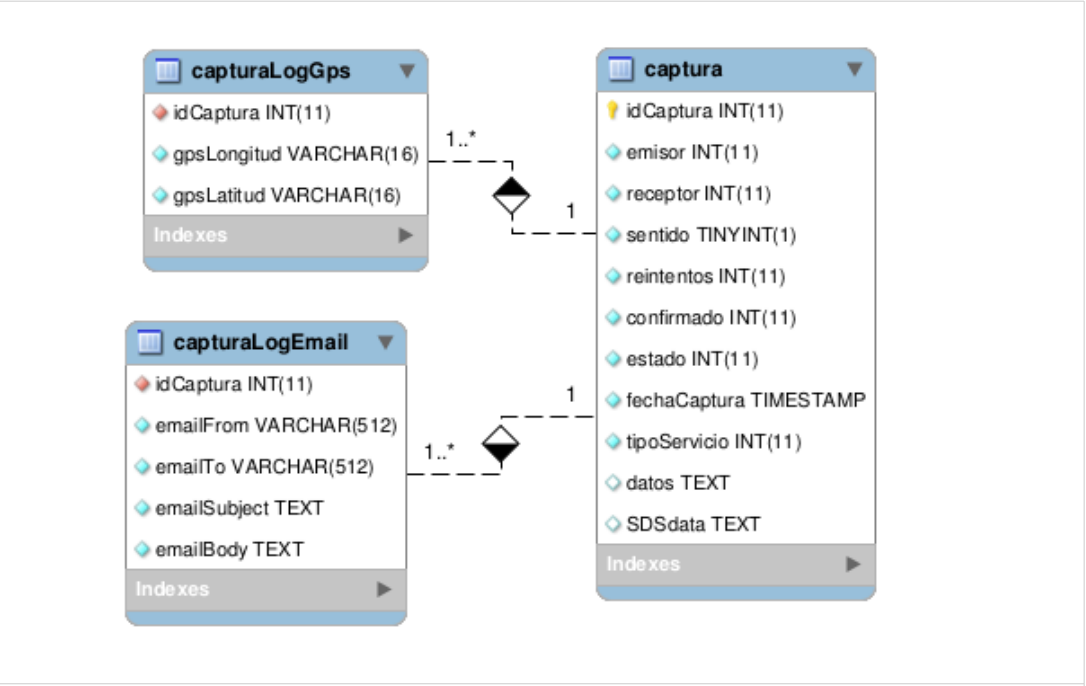

Figura 21.- Capturas y log de capturas

Puntualizar que estas tablas debido a su naturaleza de funcionamiento tendrán asociadas un plan de mantenimiento especial puesto que crecerán muy rápido en tamaño

8.4.1.4.- Puentes y responsables

Como indican los requisitos de la aplicación, el software ha de poder controlar la disponibilidad de su hardware asociado y en el caso de detectar error, avisar a su responsable mediante correo electrónico.

Al poder haber dos tipos de hardware a controlar, los puentes RS232 ↔ Ethernet y los terminales TETRA de escucha, se hace necesario una estructura de base de

datos en las que cualquier responsable creado pueda tener asociado un puente o un terminal.

Por estas causas, se crea el concepto recurso, del que un responsable se hace cargo. Cada recurso identifica la tabla a la que hace referencia mediante un campo tipo.

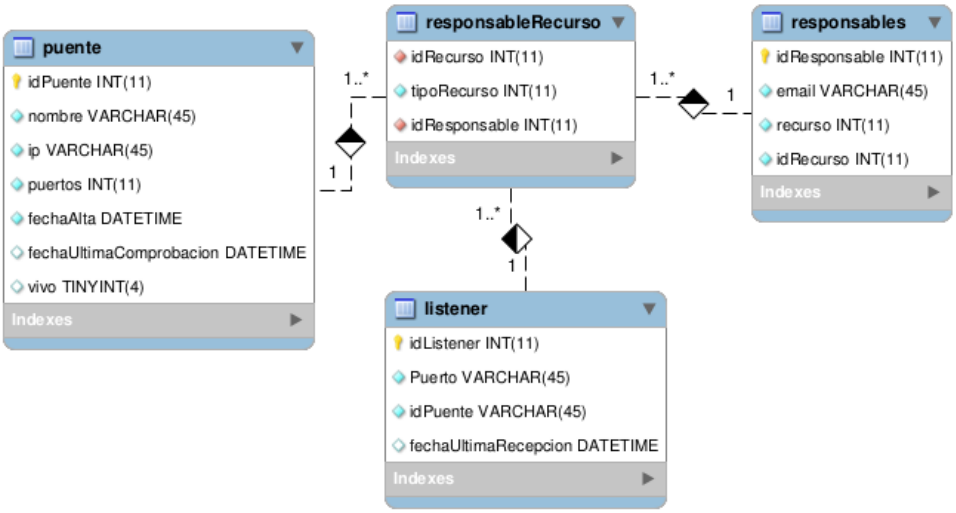

Figura 22.- Responsables, puertos y listeners

8.4.2.- Diagramas de colaboración

Los diagramas de colaboración siguientes detallarán el comportamiento y el movimiento de la información dentro del programa. También aparecen en los diagramas siguientes, elemento de colaboración extra, los cuales, han sido añadido por restricciones técnicas y decisiones de diseño que no se contemplan en la fase de análisis. Aclarar también que pueden haber diagramas que engloben mas de un caso de uso, por ello existen menos diagramas de colaboración que casos de uso.

8.4.2.1.- Caso de uso 1, 3, 4 ,5, 9. Buscar mensaje en la radio.

En este flujo, el bucle principal invoca el método, *getPrimermensaje* que se encarga de buscar los mensajes disponibles en el *buffer* de la emisora, este proceso a su vez invoca al *listener* que se encarga de notificar el comando AT a

todas las clases que se han suscrito a el. En este caso se detalla el comportamiento de la clase SDS_sever que detecta un mensaje, lo confirma y lo almacena en la base de datos. Esta clase a su vez hace uso de elementos de la librería TetraATLib, como la clase SDS que es usada para procesar el SDS y la clase ATSimpleCommand que se usa para enviar la confirmación

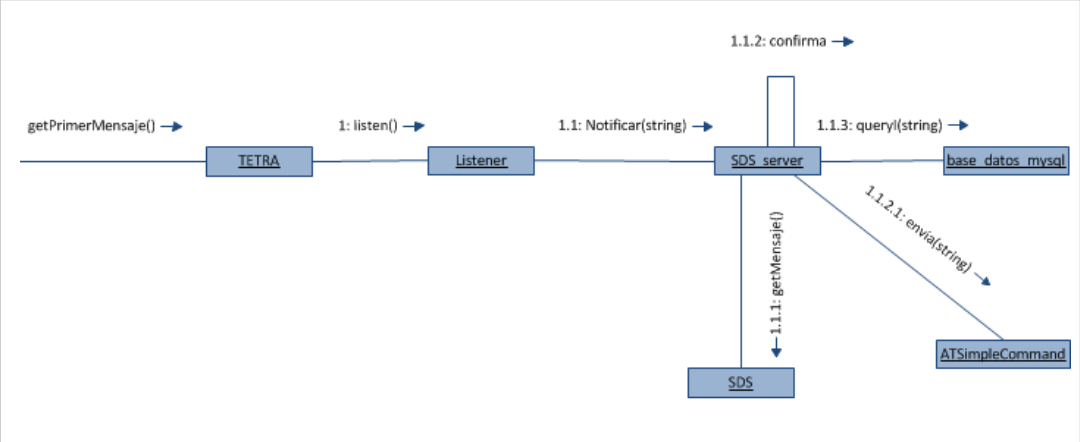

Figura 23.- Caso de uso 1.

8.4.2.2.- Caso de uso 2. Buscar mensaje servicio.

En este caso de uso, el adaptador de correo, consultara la base de datos de usuarios para saber los buzones activos y luego leerá, haciéndose uso de la librería *vmime* los buzones de correo. Por último insertara en la base de datos los mensajes que haya detectado para enviar por TETRA

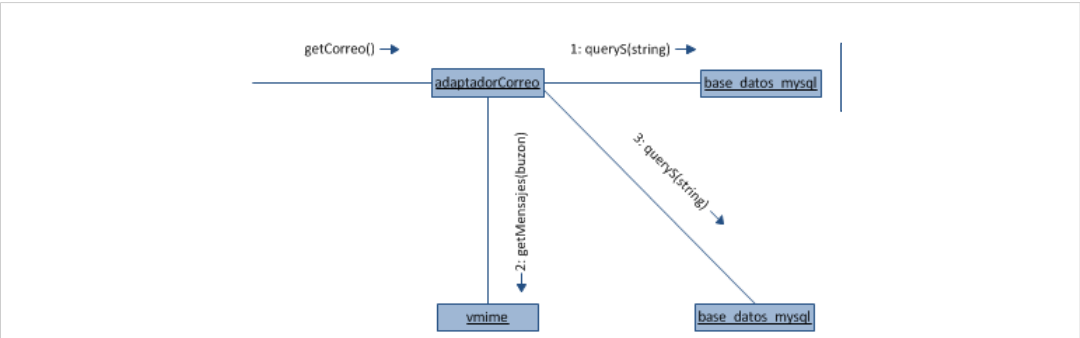

Figura 24.- Caso de uso 2.

8.4.2.3.- Caso de uso 6 Enviar por servicio

En este caso, el sistema solo tendrá que obtener de la base de datos los SDS pendientes de retransmitir, enviarlos y marcarlos como guardados, para enviar correos, se hará uso de la herramienta de correo del sistema.

Figura 25.- Caso de uso 6.

8.4.2.4.- Caso de uso 7. Enviar SDS.

En el envío de SDS primero hay que consultar la base de datos y obtener todos los mensajes pendientes de enviar, una vez se ha obtenido la información de mensaje, emisor y receptor. El sistema hace uso de las utilidades de la librería TetraATLib para convertir el mensaje a formato SDS.

Una vez convertido los mensajes, se envían a los terminales mediante una instancia de *ATSimpleCommand* válida.

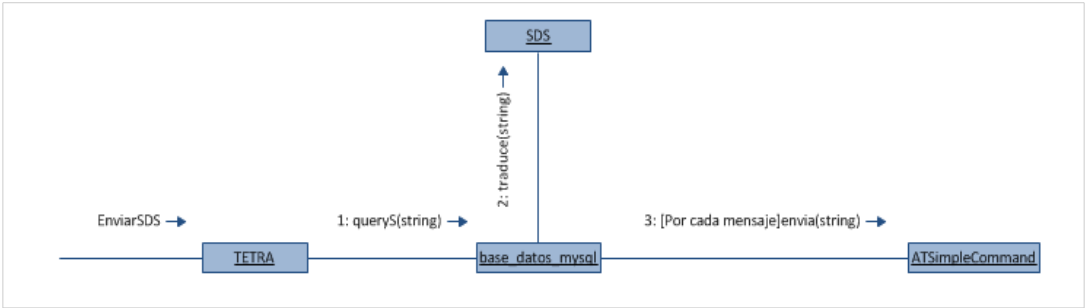

Figura 26.- Caso de uso 7.

8.4.2.5.- Caso de uso 1. Conectar Radio.

En el caso de conectar los terminales del sistema. Se consultará en primer lugar a la base de datos, todos los terminales dados de alta en el sistema. Luego por

cada terminal se creará una instancia de conectaAT a la que se le enviara una solicitud de identificación del terminal.

El hecho de mandar un mensaje de identificación al terminal tiene dos funciones básicas. La primera se utiliza para comprobar que la conexión es correcta, la segunda causa es saber el numero de terminal que esta conectado, así se podrá saber el numero de emisor de los SDS y registrar correctamente esta información en la base de datos.

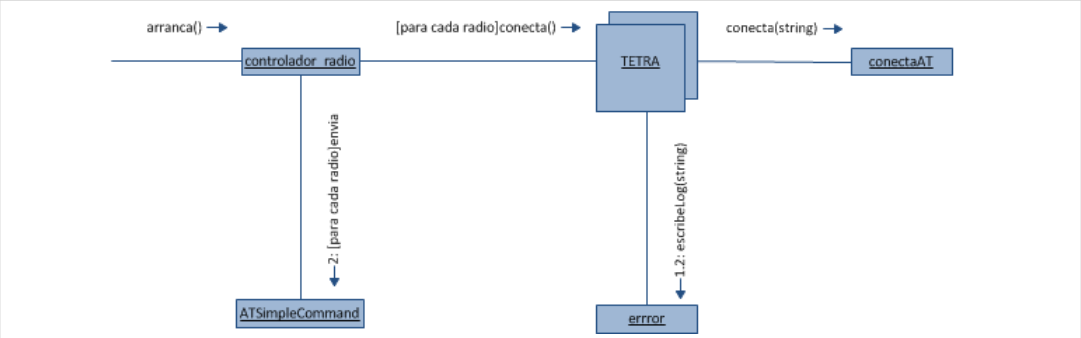

Figura 27.- Caso de uso 8.

En caso de error, se registrará esto en el *log* principal de la aplicación.

8.4.2.6.- Caso de uso 10. Comprobar estado

Cada cierto periodo de tiempo, el sistema enviará un mensaje de registro a los terminales. Esto servirá para detectar caídas en el sistema.

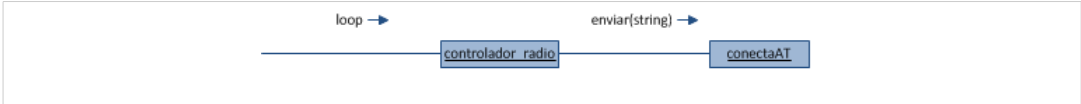

Figura 28.- Caso de uso 8.

8.4.3.- Diagrama de clases

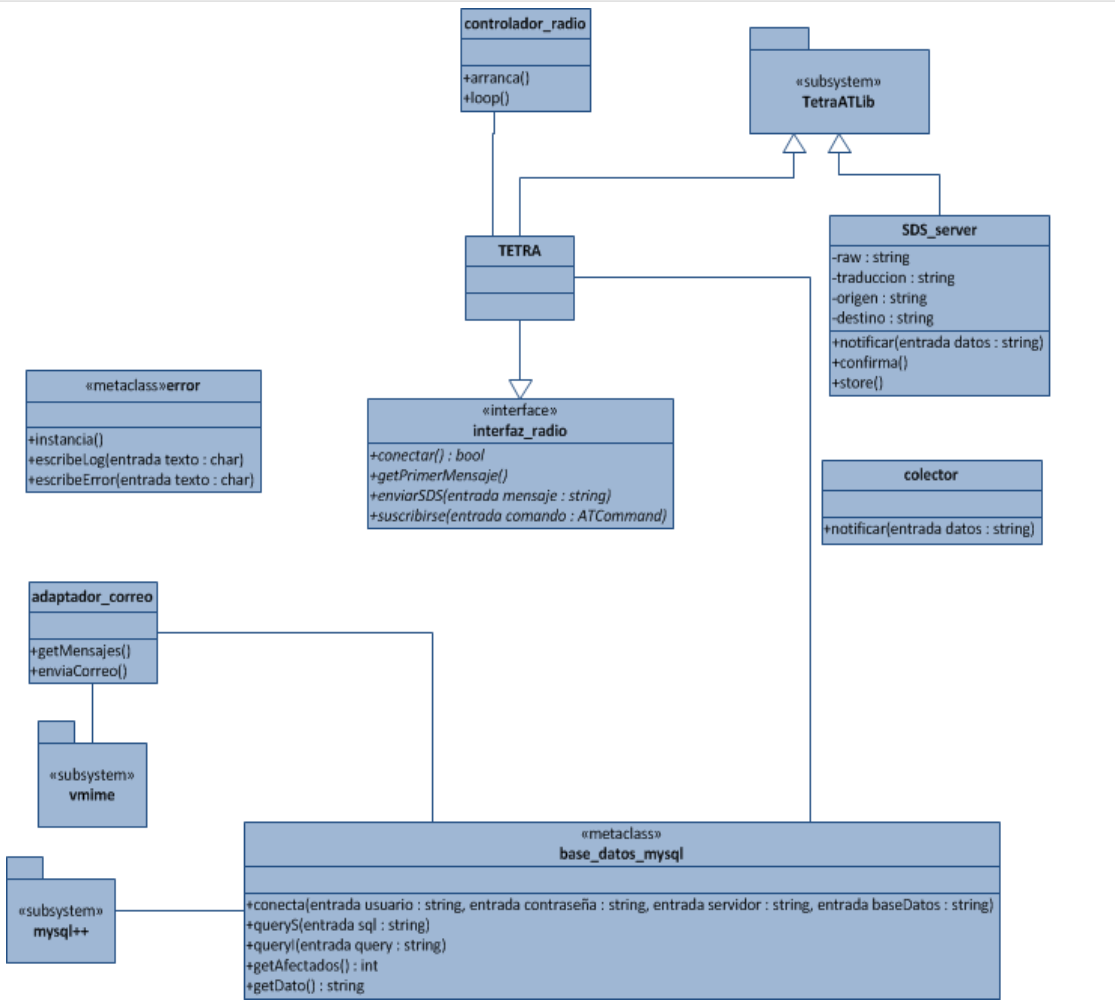

Figura 29.- Diagrama de clases TetraServer

En el diagrama de clases el mayor peso dentro de la aplicación lo tienen el control de la base de datos y la librería TetraATLib, esto es claramente lógico ya que el fundamento de la aplicación es interactuar con terminales TETRA y almacenar toda la información en una base de datos.

En el diseño se puede observar la existencia de dos clases que heredan de comandoAT, TETRA y SDS_server, la primera encargada de interactuar de una manera más directa con el envío de comandos a la radio y la segunda se encarga de leer los comando AT que tengan información relacionada con SDS. Groso modo, podemos decir que la clase TETRA se encarga de escribir, mientras la clase SDS_server se encarga de leer y confirmar.

Existe una tercera clase que hereda de comandoAT, llamada colector, su funcionamiento es almacenar todo registro que provenga de la matriz de emisoras TETRA y almacenarlo en la base de datos.

También hay que prestar atención a la clase base_datos_mysql, la cual se ha declarado como un miembro estático. Esto nos confiere la posibilidad de poder usar un objeto declarado de la misma en cualquier momento durante la ejecución del programa y pudiendo olvidarnos en todo momento de su inicialización y destrucción, Esto va más allá de la comodidad que supone en el momento de la implementación, ahora muchas conexiones a la base de datos que como es sabido es una operación bastante costosa.

La implementación de la clase base_datos_mysql se hará haciendo uso de la librería MySQL++, la cual nos ofrece una capa de abstracción mucho más cómoda en la implementación de la misma. Aunque no impide que en cualquier momento si se desea acoplar la aplicación a un motor de base de datos diferente, poderlo hacer, puesto que la clase se ha declarado como una interfaz de la clase base_datos, esto nos permite hacer cualquier cambio en la implementación y seguir funcionando siempre que se mantenga la interfaz.

Para la lectura y escritura de correo se ha creado un adaptador de correo,que hará uso de la librería *vmime*. Esta clase controlador recogerá todo el correo electrónico que esté sin leer y lo almacenará en la base de datos y marcará como pendiente de retransmitir. De la misma manera leerá la base de datos en busca de mensaje de TETRA pendientes de retransmitir.

8.5.- Problemas encontrados durante el desarrollo de esta fase

En la fase de desarrollo que se acaba de explicar, los problemas más graves encontrados fueron de cara al diseño, ya que al igual que en la primera fase, la

inexperiencia a la hora de desarrollar soluciones tan completas, ha complicado bastante todo el proceso de ingeniería del software.

Por otro lado, aunque el servidor de pruebas cedido por el departamento de seguridad de la ULPGC ha sido de gran utilidad, se ha encontrado muchos problemas a la hora de instalar la solución, teniendo CentOS 5 bastantes librería desactualizadas, generando por tanto el problema de buscar dichas librería, compilarlas e instalarlas.

En esta fase, también se detectaron problemas de estabilidad de los terminales TETRA de prueba, pues por razones desconocidas, cada cierto tiempo su sistema procedía a apagarse, perdiendo por tanto total conectividad. Debido a esto, se agrego la funcionalidad extra de la asignación de responsables de los dispositivos, acompañado de las notificaciones de inactividad.

9.- Etapa 3: Aplicación de Web de control TetraWeb

9.1.- Análisis

Como herramienta a la gestión de la aplicación TetraServer se incluye un GUI (*Graphic User Interface*) vía web que ayudará esencialmente a la gestión y la configuración de TetraServer.

La principal opción de esta web es la de proporcionar una manera sencilla y eficaz de modificar o crear nuevas opciones de parametrización para TetraServer, además de dar la opción de crear nuevos parámetros en el caso de querer añadir nuevas funcionalidades al sistema.

También podremos crear y modificar grupos de correo ya sea de uno en uno o mediante la carga de un fichero CSV.

Por otro lado también tendrá una gestión de hardware mediante la cual se podrá asignar responsables a los elementos externos del ecosistema. En cuanto al hardware, aquí habrá que hacer una puntualización, tendremos dos tipos de hardware, el que llamaremos puentes que no es más que una interfaz Ethernet<--> RS232 que junto con un *driver* crea ficheros especiales en el sistema los cuales funcionan de idéntica manera que un puerto serie convencional, pero dando la posibilidad de tener un terminal en una localización remota, también es muy útil en el caso de no tener capacidad física de expansión de puestos RS232. El otro tipo de dato que tenemos son los terminales TETRA llamados *listeners* en este ámbito.

Debido a que algunos de los parámetros modificables no surtirá efecto hasta que el servicio no se reinicie, se añadirá un control para esto mismo, poder reiniciar el servicio cada vez que el usuario lo necesite. Esto se realizará mediante la generación de una tarea en el planificador del sistema operativo.

Por último llevará un registro de los últimos mensajes leídos por el sistema y podrá enviar un SDS a un usuario o a un grupo predefinido.

9.1.1.- Requisitos

Tras analizar la problemática, según las notas aportadas por el departamento de seguridad de la ULPGC, para esta fase del proyecto.

- Desde la interfaz se ha de poder crear grupos de configuración.
- Se dispondrá agregar parámetros nuevos a cualquier grupo de configuración
- Se podrá borrar y actualizar cualquier parámetro.
- La utilidad sera accesible mediante una sesión accesible con usuario y contraseña.
- Se podrá ver de forma jerarquizada la disposición de los elementos de hardware asociados.
- Se podrá ver el estado de cada elemento y la ultima vez que se obtuvo una notificaron del mismo
- Desde el listado de hardware asociado podemos agregar más puentes o añadir puertos (terminales tetra) a los puentes creados.
- Agregar un responsable a un hardware del sistema.
- Consultar la lista de responsables de un hardware concreto.
- Creación de nuevos usuarios en el sistema.
- Crear dispositivos mediante formulario
- En el caso de tener que crear una gran cantidad de dispositivos, se podrá realizar pasándole un fichero CSV al sistema.
- El sistema dará la opción de crear nuevos grupos de envió de SDS.
- Se podrá asignar o borrar nuevos dispositivos ya creados de los grupos existentes.

9.1.2.- Diagramas de casos de uso.

En primer lugar se distinguen dos actores principales del sistema, un operario, el cual manejara la información de los usuarios que van a recibir SDS y envió de los

mismo, y por otro lado un administrador que ser encargara de la gestión de usuarios web,

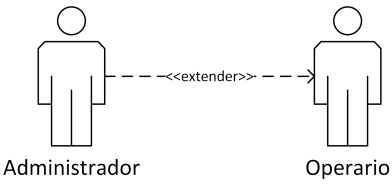

Figura 30.- Diagrama de casos de uso: Actores

Los casos de uso específicos del administrador son los siguientes.

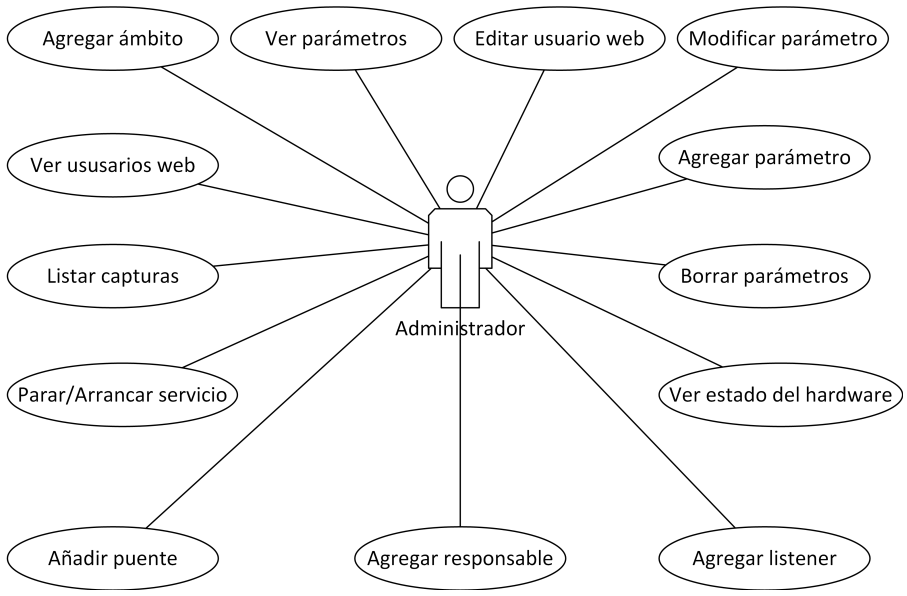

Figura 31.- Diagrama de casos de uso: Administrador.

El operario, tiene unas acciones más reducidas que el administrador, pudiendo en todo momento el administrador tener también acceso a las acciones del operario.

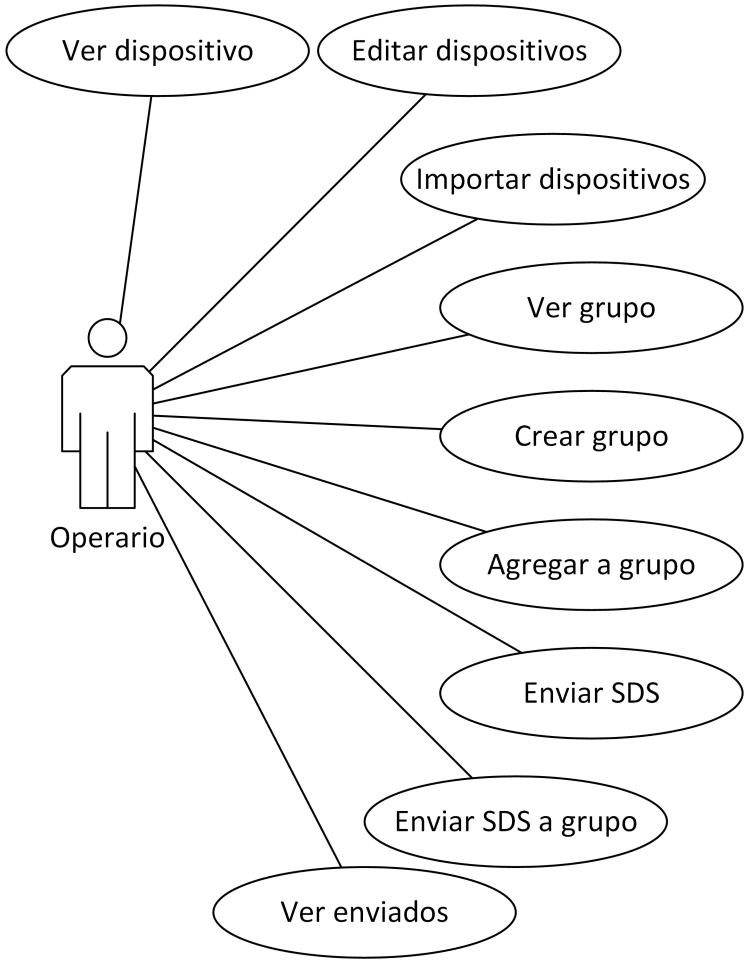

Figura 32.- Diagrama de casos de uso: Operario.

9.1.3.- Casos de uso

La descripción de los casos de uso enumerados en los diagramas anteriores, se detallan a continuación en formato completo.

9.1.3.1.- Caso de uso 1 : Autenticarse

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario no ha iniciado sesión.

Garantías de éxito	El usuario habrá iniciado la sesión en el sistema.
Escenario principal	1. El usuario accede a la dirección web del servicio. 2. El servidor detecta que no hay sesión iniciada y le muestra un formulario de autenticación. 3. El usuario introduce los datos de su cuenta. 4. El sistema lo redirige al menú principal.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*. 3a. El usuario introduce datos incorrectos. 1. El sistema vuelve al paso 2.
Frecuencia	Al entrar por primera vez, al caducar la sesión, tras haber cerrado la sesión.

9.1.3.2.- Caso de uso 2 : Ver parámetros

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión.
Garantías de éxito	El administrador tendrá una representación visual de los parámetros del sistema.
Escenario principal	1. El administrador entra desde el panel central al modulo de parámetros. 2. El servidor muestra los parámetros del TetraServer agrupados por ámbitos de configuración.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 1a. El administrador accede al modulo de parámetros a través del menú superior de la aplicación.

	1. El sistema va al paso 2.

9.1.3.3.- Caso de uso 3 : Agregar ámbito

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	Se habrá creado un nuevo ámbito.
Garantías de éxito	El sistema tendrá registrado un nuevo ámbito de configuración.
Escenario principal	1. El administrador entra desde el panel central al modulo de parámetros. 2. El servidor muestra los parámetros del TetraServer agrupados por ámbitos de configuración 3. El administrador entra en la opción de crear ámbito. 4. El sistema le muestra el formulario de creación. 5. El administrador rellena los datos solicitado crea el ámbito.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*. 4a. El sistema no puede crear la entrada del nuevo ámbito. 1. El sistema vuelve al listado de parámetros y avisa de la eventualidad al administrador.

9.1.3.4.- Caso de uso 4 : Modificar parámetro

Actor	Sistema.
Personal	Administrador.

involucrado	
Precondiciones	El administrador ha iniciado sesión
Garantías de éxito	Un parámetro o varios parámetros estarán actualizados.
Escenario principal	1. El administrador entra desde el panel central al modulo de parámetros. 2. El servidor muestra los parámetros del TetraServer agrupados por ámbitos de configuración 3. El administrador elige el ámbito que desea modifica y marca la opción editar. 4. El sistema transforma el tipo de elemento y permite editar los campos. 5. El administrador valida las modificaciones. 6. El sistema actualiza los parámetros del ámbito modificado.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log.* 6a. El sistema no es capaz de aplicar los cambios. 1. El sistema muestra el error.

9.1.3.5.- Caso de uso 5 : Agregar parámetro

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión.
Garantías de éxito	Un parámetro o varios parámetros serán creados.
Escenario principal	1. El administrador entra desde el panel central al modulo de parámetros. 2. El servidor muestra los parámetros del TetraServer

	agrupados por ámbitos de configuración 3. El administrador elige el ámbito al que le desea añadir la entrada y selecciona la opción editar. 4. El sistema transforma el tipo de elemento y permite editar los campos y muestra opciones de añadir y guardar. 5. El administrador elige la opción añadir. 6. El sistema crea un par de inputs para rellenar los parámetros de clave valor. 7 El administrador rellena estos campos y valida los datos. 8. El sistema almacena las entradas en la base de datos.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 6a. El sistema no es capaz de aplicar los cambios. 1. El sistema muestra el error.

9.1.3.6.- Caso de uso 6 : Borrar parámetro

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El usuario ha iniciado sesión
Garantías de éxito	Un parámetro o varios parámetros serán creados.
Escenario principal	1. El administrador entra desde el panel central al modulo de parámetros. 2. El servidor muestra los parámetros del TetraServer agrupados por ámbitos de configuración. 3. El administrador buscar el parámetro a eliminar y marca la opción eliminar. 4. El sistema borra los registros de ese parámetro.

Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 6a. El sistema no es capaz de aplicar los cambios. 1. El sistema muestra el error.

9.1.3.7.- Caso de uso 7 : Ver dispositivos

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario podrá ver la lista de dispositivos del sistema.
Escenario principal	1. El usuario entra desde el panel central al modulo de dispositivos. 2. El servidor muestra los dispositivos de TetraServer.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*

9.1.3.8.- Caso de uso 8 : Editar dispositivos

Actor	Sistema
Personal involucrado	Usuario
Precondiciones	El usuario ha iniciado sesión .
Garantías de éxito	El usuario habrá modificado al menos un parámetro de dispositivo.
Escenario principal	1. El usuario entra desde el panel central al modulo de dispositivos.

	2. El servidor muestra los dispositivos de TetraServer. 3. El usuario selecciona un dispositivo y marca la opción de editar. 4. El sistema le muestra un formulario con los campos editables. 5. El usuario modifica los parámetros deseados y los valida. 6. El sistema actualiza los parámetros y devuelve al usuario al listado de dispositivos.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*. 6a. El sistema no es capaz de aplicar los cambios. 1. El sistema muestra el error.

9.1.3.9.- Caso de uso 9 : Importar dispositivos

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario habrá creado tantos dispositivos como lineas de datos el fichero importado.
Escenario principal	1. El usuario entra desde el panel central al modulo de dispositivos. 2. El servidor muestra los dispositivos de TetraServer. 3. El usuario selecciona en la cabecera la opción, importar desde fichero. 4. El sistema le muestra un formulario con los campos de selección de fichero. 5. El usuario selecciona el fichero y lo valida. 6. El sistema crea los registros en la base de datos.

Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 6a. El sistema no es capaz de aplicar los cambios. 1. El sistema muestra el error.

9.1.3.10.- Caso de uso 10 : Ver estados del hardware

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión.
Escenario principal	1. El administrador entra desde el panel central al modulo de control del hardware. 2. El servidor muestra el hardware, bajo el control de TetraServer.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log.

9.1.3.11.- Caso de uso 11 : Añadir puente

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión.
Garantías de éxito	El administrador habrá creado un nuevo dispositivo puente.
Escenario principal	1. El administrador entra desde el panel central al modulo de control del hardware.

	2. El servidor muestra el hardware, bajo el control de TetraServer. 3. El administrador selecciona la opción de crear nuevo puente. 4. El sistema le muestra un formulario con los campos necesario. 5. El administrador rellena los campos y lo valida. 6. El sistema añade el dispositivo y redirige al usuario hacia el listado de *hardware*.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 6a. El sistema no puede crear el dispositivo. 1 Notifica al usuario del error y le muestra el listado del *hardware*.

9.1.3.12.- Caso de uso 12 : Agregar listener

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión
Garantías de éxito	El administrador habrá creado un nuevo *listener* en el sistema.
Escenario principal	1. El administrador entra desde el panel central al modulo de control del *hardware*. 2. El servidor muestra el *hardware*, bajo el control de TetraServer. 3. El administrador selecciona la opción de crear nuevo *listener* de la lista. 4. El sistema le muestra un formulario con los campos

	necesario. 5. El administrador rellena los campos y lo valida. 6. El sistema añade *listener* y redirige al usuario hacia el listado de *hardware*.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*. 6a. El sistema no puede crear el dispositivo. 1 Notifica al usuario del error y le muestra el listado del *hardware*.

9.1.3.13.- Caso de uso 13 : Agregar responsables

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión.
Garantías de éxito	El administrador habrá asignad un responsable a un dispositivo puente o *listener* existente.
Escenario principal	1. El administrador entra desde el panel central al modulo de control del hardware. 2. El servidor muestra el *hardware*, bajo el control de TetraServer. 3. El administrador selecciona la opción agregar responsable de la lista de persistente y puentes. 4. El sistema le muestra un formulario con los campos necesario. 5. El administrador rellena los campos y lo valida. 6. El sistema asigna el responsable y redirige al usuario hacia el listado de *hardware*.
Flujos	*a. En cualquier momento el sistema falla:

alternativos	1. El sistema crea una entrada en el registro de *log.* 6a. El sistema no puede crear el dispositivo. 1 Notifica al administrador del error y le muestra el listado del *hardware*.

9.1.3.14.- Caso de uso 14 : Ver grupos

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Escenario principal	1. El usuario entra desde el panel central al modulo de control de grupos. 2. El servidor muestra el listado de grupos existentes.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*.

9.1.3.15.- Caso de uso 15 : Crear grupo

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario habrá creado un nuevo grupo de.
Escenario principal	1. El usuario entra desde el panel central al modulo de control de grupos. 2. El servidor muestra el listado de grupos creados. 3. El usuario selecciona la opción crear nuevo grupo.

	4. El sistema le muestra un formulario con los campos necesario. 5. El usuario rellena los campos y lo valida. 6. El sistema crea el grupo y muestra al usuario la lista de grupos creados.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log*. 6a. El sistema no puede crear el dispositivo. 1 Notifica al usuario del error y le muestra el listado de grupos.

9.1.3.16.- Caso de uso 16 : Agregar dispositivos a grupo

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario habrá asignado un nuevo dispositivo al grupo.
Escenario principal	1. El usuario entra desde el panel central al modulo de control de grupos. 2. El servidor muestra los grupos creados. 3. El usuario selecciona el grupo a asignar dispositivo. 4. El sistema le muestra el listado de dispositivos asignados. 5. El usuario elige la opción de asignar nuevo dispositivo. 6. El sistema muestra el formulario para asignar el dispositivo. 7. El usuario rellena los campos necesarios y los valida. 8. El sistema asigna el nuevo dispositivo y muestra al usuario el listado de dispositivos creados.
Flujos	*a. En cualquier momento el sistema falla:

alternativos	1. El sistema crea una entrada en el registro de log. 6a. El sistema no puede asignar el dispositivo. 1 Notifica al usuario del error y le muestra el listado de grupos.

9.1.3.17.- Caso de uso 17 : Enviar SDS

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario habrá creado para enviar un nuevo mensaje SDS.
Escenario principal	1. El usuario entra desde el panel central al modulo de SDS. 2. El servidor muestra el formulario de envío de SDS. 3. El usuario rellena los campos necesarios y los valida. 4. El sistema encola el mensaje y devuelve al usuario al formulario de envío.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 4a. El sistema no puede encolar el mensaje. 1 Notifica al usuario del error y le muestra de nuevo el formulario de envío.

9.1.3.18.- Caso de uso 18 : Enviar SDS a grupo

Actor	Sistema.
Personal	Usuario.

involucrado	
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario habrá creado para enviar un nuevo mensaje SDS a grupo.
Escenario principal	1. El usuario entra desde el panel central al modulo de SDS a grupo. 2. El servidor muestra el formulario de envío de SDS a grupo. 3. El usuario rellena los campos necesarios y los valida. 4. El sistema encola el mensaje y devuelve al usuario al formulario de envío.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log* 4a. El sistema no puede encolar el mensaje. 1 Notifica al usuario del error y le muestra de nuevo el formulario de envío.

9.1.3.19.- Caso de uno 19 : Listar capturas

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario podrá visualizar las ultimas capturas del sistema.
Escenario principal	1. El usuario entra desde el panel central al modulo de capturas. 2. El servidor muestra el listado de las ultimas capturas registradas.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log.

	4a. El sistema no puede encolar el mensaje. 1 Notifica al usuario del error y le muestra de nuevo el formulario de envío.

9.1.3.20.- Caso de uso 20 : Ver enviados

Actor	Sistema.
Personal involucrado	Usuario.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El usuario podrá visualizar los ultimos SDS enviados.
Escenario principal	1. El usuario entra desde el panel central al modulo de SDS. 2. El servidor muestra el listado de los ultimo SDS. *Este listados se recarga automáticamente pasado un tiempo.*
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log.*

9.1.3.21.- Caso de uso 21 : Parar/Arrancar servicio

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El administrador ha iniciado sesión.
Escenario principal	1. El administrador entra desde el panel central al modulo de control de control de la instancia de TetraServer. 2. El servidor muestra el estado de ejecución de TetraServer. 3. El administrador hace *click* sobre el icono para cambiar el estado.

Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log.*

9.1.3.22.- Caso de uso 22 : Ver usuarios web

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El usuario ha iniciado sesión.
Escenario principal	1. El administrador entra desde el panel central al modulo de control de usuarios. 2. El servidor muestra el listado de usuarios existentes.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de *log.*

9.1.3.23.- Caso de uso 23 : Editar usuarios web

Actor	Sistema.
Personal involucrado	Administrador.
Precondiciones	El usuario ha iniciado sesión.
Garantías de éxito	El administrador habrá modificado al menos un parámetro de dispositivo.
Escenario principal	1. El administrador entra desde el panel central al modulo de usuarios. 2. El servidor muestra los dispositivos de TetraServer. 3. El administrador selecciona el usuarios y marca la opción de editar. 4. El sistema le muestra un formulario con los campos

	"editables". 5. El administrador modifica los parámetros deseados y los valida. 6. El sistema actualiza los parámetros y devuelve al administrador al listado de usuarios.
Flujos alternativos	*a. En cualquier momento el sistema falla: 1. El sistema crea una entrada en el registro de log. 6a. El sistema no es capaz de aplicar los cambios. 1. El sistema muestra el error.

9.2.- Diseño

Se decidió durante el periodo de diseño de la aplicación web, realizar todo el desarrollo mediante una arquitectura de software modelo-vista-controlador, para ello se utilizo como base un framework PHP. El framework elegido fue CodeIgniter, las características principales de CodeIgniter son:

- Pequeña huella
- Enfocado al rendimiento
- compatible con la mayoría de configuraciones de los *hosting*
- configuración prácticamente nula
- poco restrictivo en el desarrollo
- no precisa de un aprendizaje de un motor de renderizado de plantillas.
- Baja complejidad.
- Facilidad de aprendizaje
- Publicado bajo licencia libre.
- Fuerte comunidad que respalda el futuro del framework.

9.2.1.- Diagramas de colaboración

La construcción del diagrama de clases final, se realizó mediante la elaboración previa de diagramas de colaboración, en estos diagramas, se muestra como se maneja la información dentro del sistema. Con estos diagramas se puede ver de una manera clara la visibilidad y dependencia entre clases.

9.2.1.1.- Parámetros de configuración

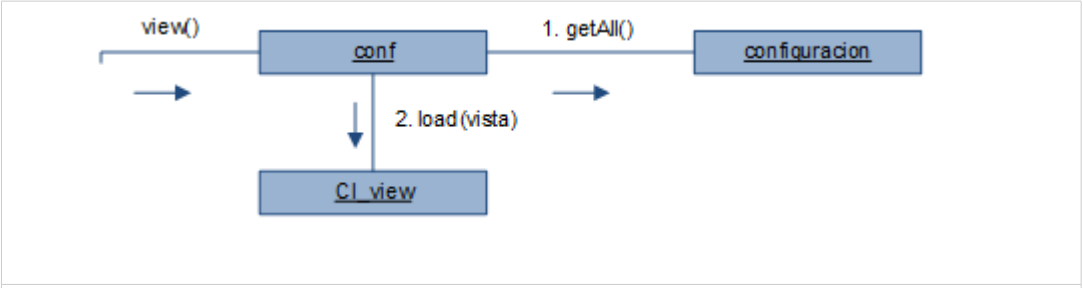

Figura 33.- Lista de parámetros de configuración

Este flujo muestra como se accede al modelo *configuración* para obtener la lisa de ámbitos y parámetros de la aplicación. Finalmente el controlador una vez preparada toda la configuración procesa la vista correspondiente.

9.2.1.2.- Crear grupo de configuración

Figura 34.- Crear grupo de configuración

En este proceso, se crea un nuevo ámbito de configuración, se puede observar en este método y en prácticamente todos los siguientes, como el controlador no llama a la vista, esto es así ya que al terminar la tarea de creación el proceso ejecutara el procedimiento *redirect* y volverá a la lista original de parámetros.

9.2.1.3.- Actualizar parámetro

Figura 35.- Actualizar parámetro

Es este caso, al actualizar cualquier parámetro de una lista, tampoco invocamos a la vista para que renderice un resultado, pero al contrario que el caso anterior, no usamos un *redirect*, sino una llamada por *Ajax* que devuelve una respuesta en JSON sin usar ninguna vista.

9.2.1.4.- Borrar parámetro

Figura 36.- Borrar parámetro

El borrado de un parámetro, también se realiza mediante una llamada en AJAX. Por lo cual tras la operación en el modelo, no invoca una vista.

9.2.1.5.- Ver estado del hardware

Figura 37.- Listar estado del hardware

Se consulta modelo la infraestructura sobre todos los puentes y terminales que tenemos creados en el sistema, la información devuelta, indica si un puente está

activo, o la ultima notificación de un dispositivo TETRA. Como ultimo paso, se invoca una vista que muestra toda esta información de forma jerárquica.

9.2.1.6.- Agregar puente

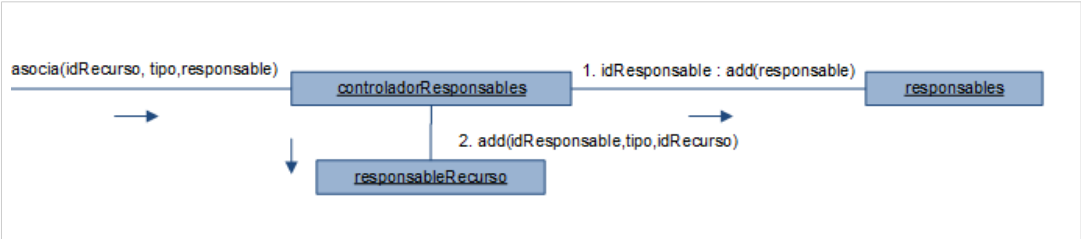

Figura 38.- Agregar puente

Al crear un *listener*, damos un nombre común, por el cual sea identificable de una forma más humana y la ip a la que habrá que consultar el estado.

9.2.1.7.- Borrar recurso

Figura 39.- Borrar recurso

Al borrar un recurso hemos de indicar el tipo de recurso y su id mediante el campo tipo el modelo será capaz de identificar si ha de borrar un puente o un *listener*. Si el campo tipo es 0 quiere decir que se trata de un puente, si es 1 borrara el *listener* que coincida con el campo id pasado como parámetro.

9.2.1.8.- Agregar puerto

Figura 40.- Agregar puerto

Al agregar un puente, se crea una entrada en la base de datos que especifica el fichero especial que simboliza el puerto donde está conectado el terminal. Este controlador no muestra vista, ya que hace uso de un método *redirect*

9.2.1.9.- Ver responsable

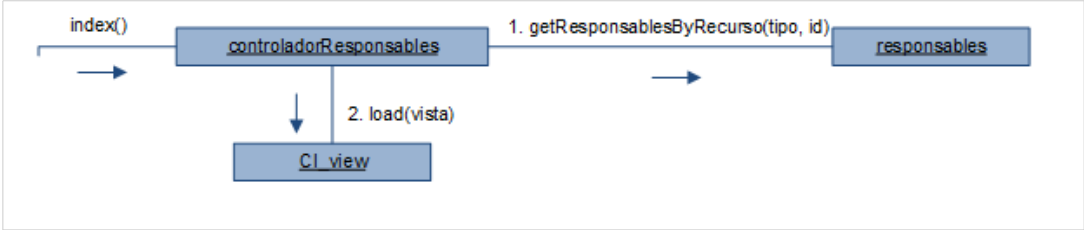

Figura 41.- Ver responsable

En este diagrama se muestra como es posible ver la lista de responsables de un recurso, como indica diagrama de modelo del domino, el recurso se identifica por un tipo de recurso y un identificador. Para terminal el controlador invoca la vista que muestra el listado de responsable del recurso solicitado.

9.2.1.10.- Agregar responsable

Figura 42.- Agregar responsable

Al agregar un responsable se crea un registro en la base de datos que asocia un email dado con un recurso de un tipo concreto. Al terminar el controlador no invoca una vista, pues usa un método *redirect*

9.2.1.11.- Login

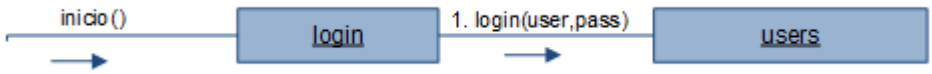

Figura 43.- Login

El controlador de *login*, se encarga de validar el par nombre de usuario y contraseña en el modelo de usuario, en el caso afirmativo, crea una sesión valida y redirige al usuario al la pagina principal.

9.2.1.12.- Logout

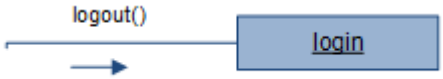

Figura 44.- Logout

Este proceso solo se encarga de destruir la sesión activa y redirigir al usuario a la pantalla inicial de *login*.

9.2.1.13.- Crear usuario

Al crear un usuario, se ha de introducir los datos de nombre de usuario, contraseña y perfil de permisos del nuevo usuario.

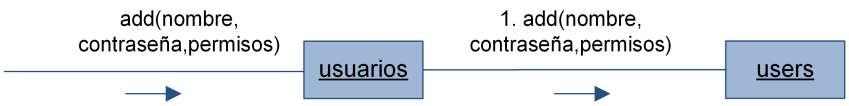

Figura 45.- Crear usuario.

9.2.1.14.- Desactivar usuario

Desactivar al usuarios, es un cambio de estado del campo *active* en la base de datos. En ningún caso se borraran datos. Al terminar la operación se redirigirá al usuario a la lista de usuario

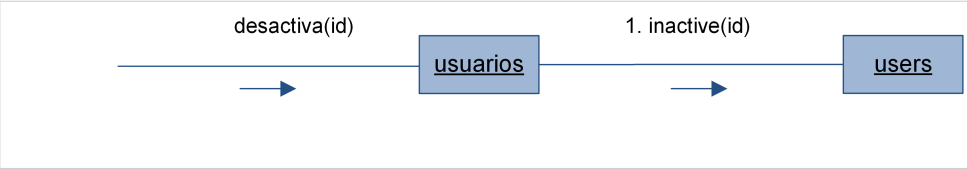

Figura 46.- Desactivar usuario.

9.2.1.15.- Actualizar usuario

Actualizar usuario tiene dos partes, la primera operación es obtener los datos actuales del usuario, para luego confirmar los datos modificados y almacenarlos en la base de datos.

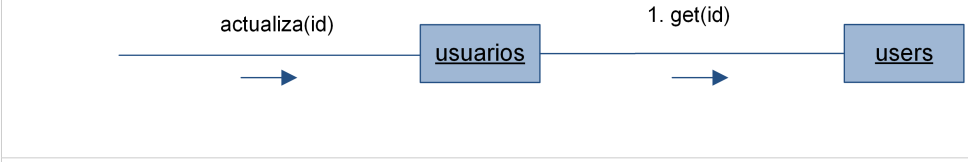

Figura 47.- Obtener datos del usuario.

El método actualizar redirige a la lista de usuarios.

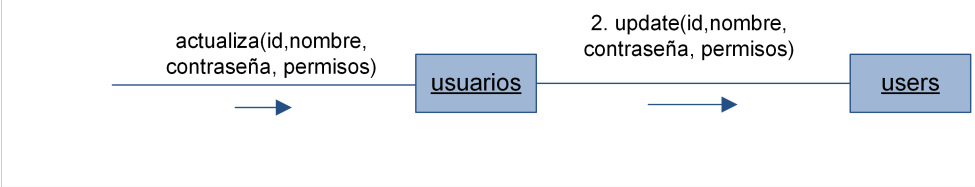

Figura 48.- Actualizar usuario.

9.2.1.16.- Listar usuarios

Se obtiene la vista principal del modulo de usuarios, la lista de los usuarios activos.

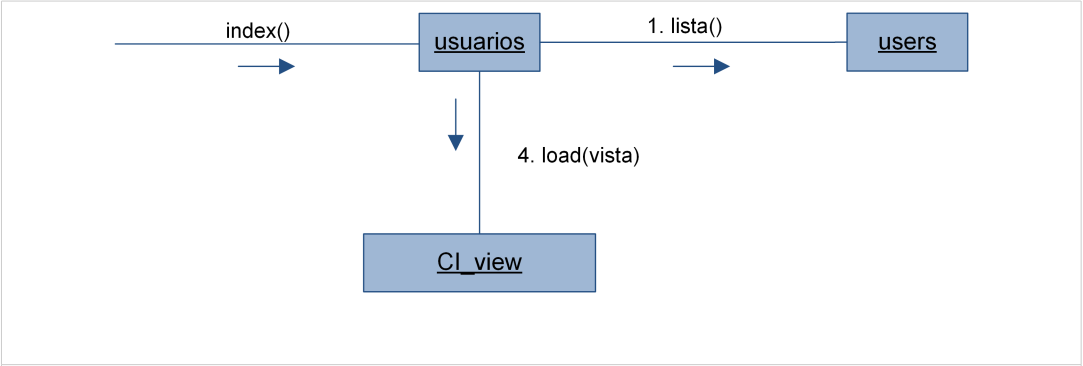

Figura 49.- Lista de usuarios.

9.2.1.17.- Lista de correo

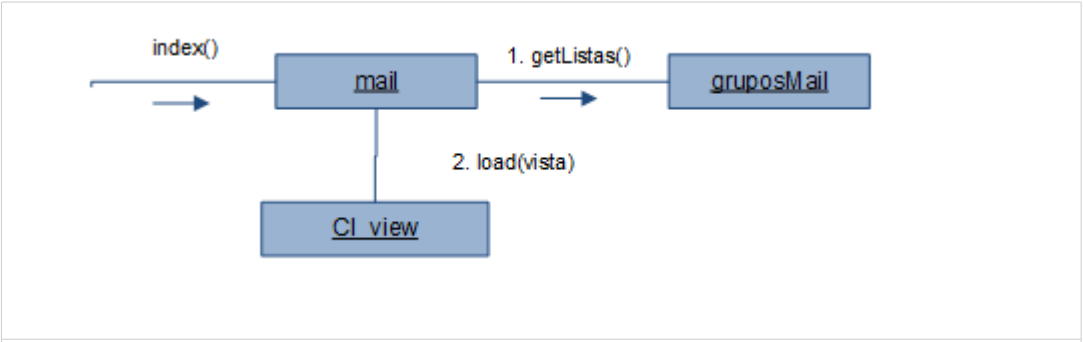

Figura 50.- Ver listas de correo.

El primer paso en el modulo de grupos de correo es mostrar el listado de los grupos existentes. Esto se hace obteniendo la información necesaria del modelo *gruposMail.*

9.2.1.18.- Crear grupo de correo

Figura 51.- Ver listas de correo.

La creación de un grupo se realiza mediante la inserción de una entrada en la base de datos, en la tabla lista por parte del modelo gruposMail. El controlador al terminal vuelve a la lista de grupos mediante un *redirect*.

9.2.1.19.- Borrar grupo de correo

Figura 52.- Borrar grupo de correo

Para borrar un grupo de correo el controlador invoca el método *delete* del modelo *gruposMail*. El método solo ejecuta el borrado del registro en la tabla lista, el resto de las asociaciones es borrado de forma automática por el motor de base de datos ya que tiene definida una clave ajena con la propiedad de borrado en cascada activada.

9.2.1.20.- Ver usuarios de un grupo de correo

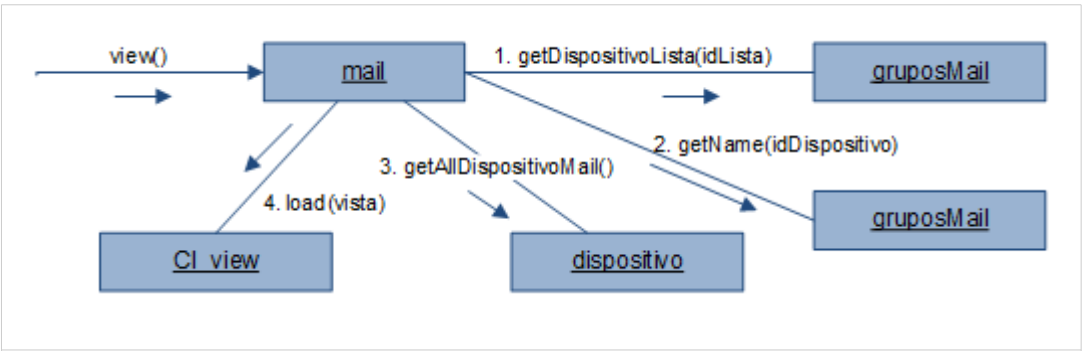

Figura 53.- Ver usuarios de un grupo de correo

Para ver los usuarios de un grupo, se necesita del modelo *gruposMail*, el nombre del grupo y los dispositivos de la lista, además de esto, se requiere información de los dispositivos asociados, que es solicitada al modelo dispositivo. Para terminar se crea una lista formateada con todos los dispositivos pertenecientes al grupo en cuestión.

9.2.1.21.- Agregar usuario a un grupo de correo

Figura 54.- Agregar usuario a un grupo de correo

Es este se paso se asocia en la base de datos una lista con un dispositivo. Al terminar se ejecuta un *redirect* al listado de dispositivos del grupo.

9.2.1.22.- Borrar grupo de correo

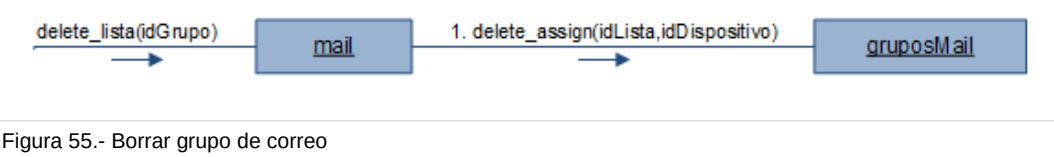

Figura 55.- Borrar grupo de correo

El borrado de una asociación a un grupo de envío se realiza ejecutando la sentencia SQL de borrado con el id de dispositivo y la lista dada.

9.2.1.23.- Enviar SDS a dispositivo

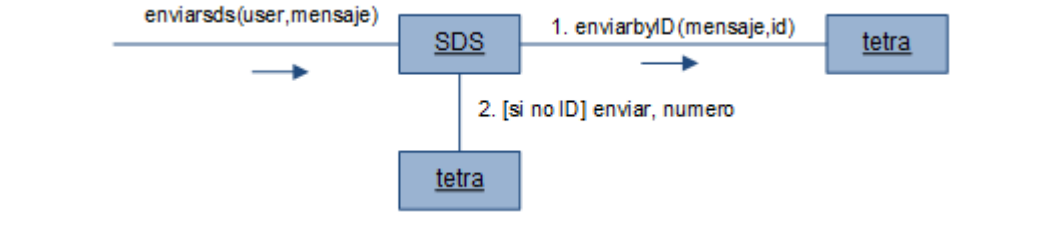

Figura 56.- Borrar grupo de correo

Para el envío de un SDS desde la interfaz web, existe la posibilidad de enviarlo seleccionado un disposiontrolador tendrá que ir a buscar el numero detivo de una lista de dispositivos creados en el sistema, por lo cual, el c este dispositivo a un modelo y este modelo a la base de datos, o también si se desea enviar un SDS a algún usuario que no este en el sistema, es posible enviarlo introduciendo un numero directamente, en este caso el controlador no tendrá que buscar el numero

y le pasara toda la información necesaria al modelo de un solo paso. Tras realizar el envío, el controlador vuelve a mostrar el formulario de envío de SDS.

9.2.1.24.- Enviar SDS a grupo

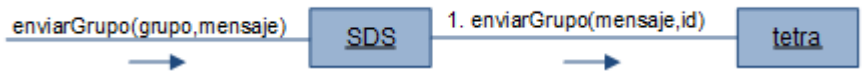

Figura 57.- Enviar SDS a grupo

En el envío de SDS a grupo, solo se dispone de la opción de enviar a un grupo ya formado con anterioridad, será el modelo en encargado de buscar los números de los usuarios de un grupo y enviar a todos el mensaje dado.

9.2.1.25.- Lista de dispositivos

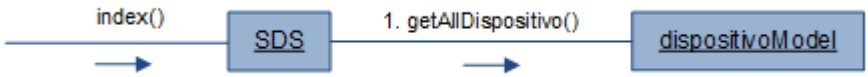

Figura 58.- Lista de dispositivos

Al entrar en el formulario de envío de SDS el sistema muestra un listado seleccionable de los dispositivos creados, esto dispositivos los obtiene mediante el controlador *dispositivoModel.*

9.2.1.26.- Crear dispositivo

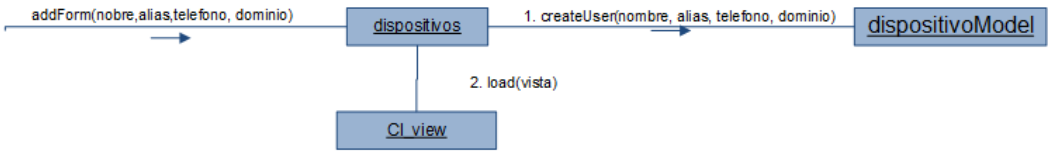

Figura 59.- Crear dispositivo

Para crear un nuevo dispositivo se precisa de un nombre de puesto donde se localizara un dispositivo, el alias del dispositivo y el numero que tiene creado, el resto de procesos se realiza mediante la ejecución de un proceso almacenado en la base de datos que se encarga de controlar la lógica de creación en las dos bases de datos del sistema.

9.2.1.27.- Crear dispositivo desde fichero

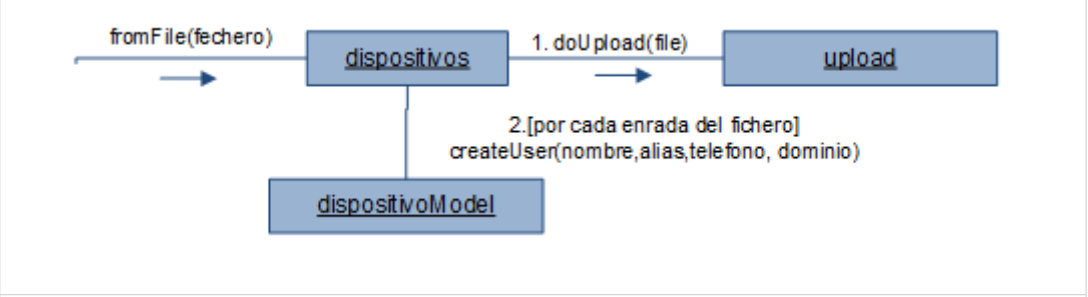

Figura 60.- Crear dispositivo desde fichero

El controlador preprocesar un fichero subido, saltando la primera linea ya que considerara que se trata de una linea de títulos e ira creando de uno en uno los usuarios, haciendo uso de un proceso almacenado en la base de datos.

9.2.2.- Diagrama de clases

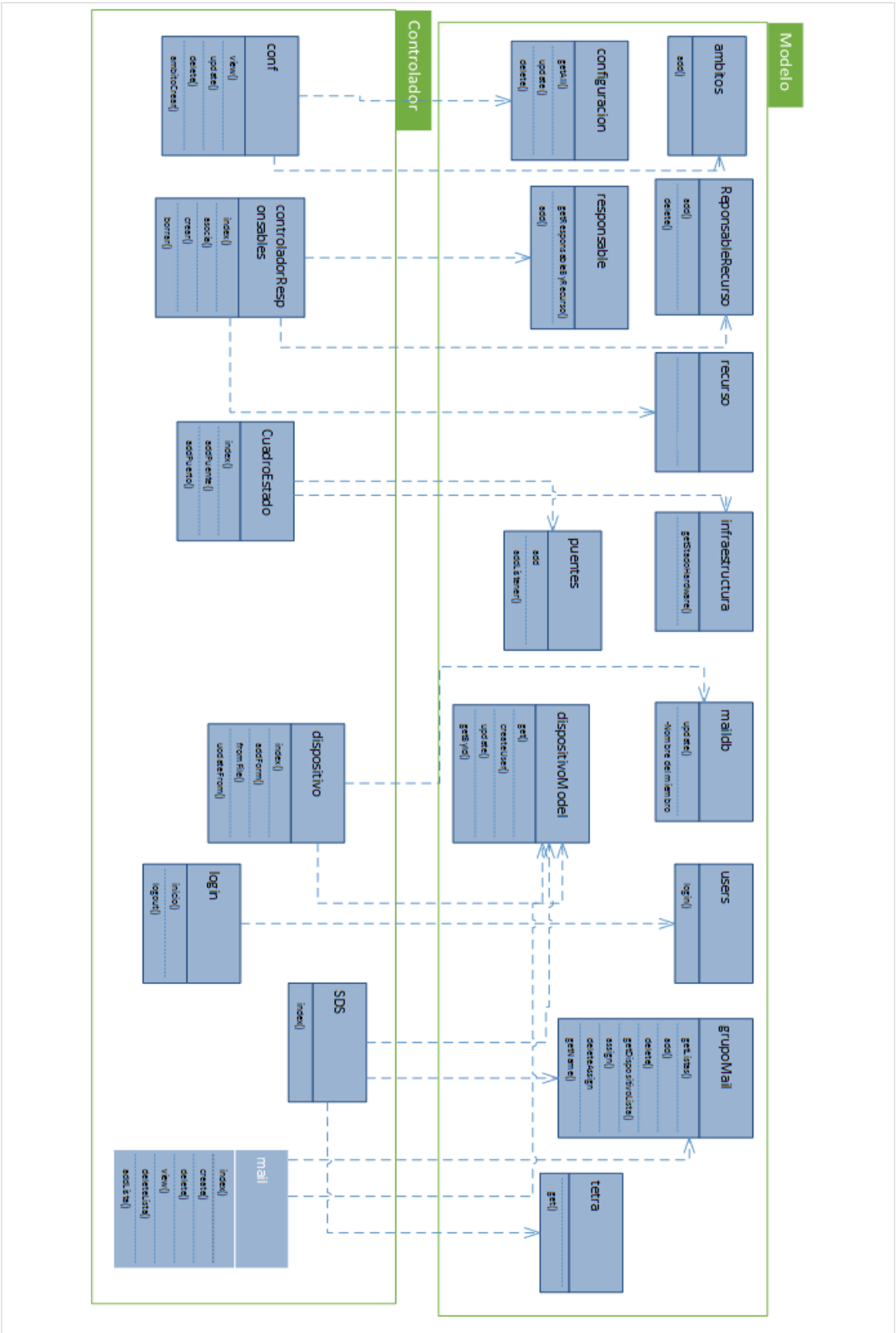

Figura 61.- Diagrama de clases TetraWeb

Al estar orientada la estructura según el patrón de diseño modelo-vista-controlador, en el diagrama deberían aparece tres subconjuntos de clases, en nuestro caso, se ha hecho uso de un *framework* de alto nivel, el cual ya da una interfaz para cada una de las capas, en este caso, solo se ha desarrollado las capas de modelo y controlador, quedando la capa de vista como una simple representación de los datos, no existiendo en ésta ningún control de lógica más allá del que proporciona el *framework*, por esto no se ha incluido la capa de vista en el diagrama.

En todas las clases se ha tenido especial cuidado en no salirse del patrón escogido. El modelo en todo momento se encarga del trabajo con la información, cediendo a ella manipulándola y devolviendo al controlador la información solicitada lo más preparada posible para su utilización en la vista. Por su parte las clases controladoras se encargan de que la información vaya de la vista al modelo y viceversa.

9.3.- Problemas encontrados durante el desarrollo de esta fase

Esta etapa de desarrollo, ha sido la más rápida y menos problemática de todas, ésta se debe mayormente al tener ya todas las funcionalidades descritas en TetraServer, la interfaz web solo tenía que hacer una capa de vista sobre todo. La única problemática encontrada, fue controlar mediante la interfaz web, el comportamiento de una aplicación de servidor. Al final, esto se resolvió combinando el planificador de tareas del sistemas operativo y un *script* de consola de comandos.

Otro aspecto es la familiaridad de este equipo de aplicación, puesto que ya se había tenido experiencia con esta problemática y la tecnología también era bastante familiar.

10.- Resultados y conclusiones

Como sucede con cualquier producto software, el resultado mas visibles, de cara a un usuario final, es la aplicación y sus funcionalidades, en este caso, el resultado ha sido una aplicación con interfaz web que es capaz de conectar a un servidor de correo y hacer las funciones de *relayer* entre este servidor de correo y el servicio de mensajería de texto de TETRA.

Aunque, a grandes rasgos se pueda interpretar todo el desarrollo como una aplicación que lee correos electrónicos y los reenviá a unos terminales, que se comportan en apariencia, como teléfonos móviles. En el fondo, como se ha venido comentando en las distintas fases de desarrollo del proyecto y más concretamente en las etapas 1 y 2, el desarrollo se ha basado en dos grandes pilares, el aprendizaje de una tecnología prácticamente desconocida para la gran mayoría y la creación de una solución que funcione mas allá del fin de este proyecto final de carrera, siendo extensible en cualquiera de sus aspectos.

Por un lado se ha creado una librería de herramientas que proporciona un mecanismo de cola y acceso al recurso compartido de cada terminal TETRA conectado. Este mecanismo permite crear objetos con una interfaz determinada y que serán notificados directamente por la librería cuando existan comandos que procesar. Esto supone una gran ventaja, pues se puede crear una aplicación que haga uso de esta cola de notificación para agregar tanto servicios de interconexión como se desee. Como método de escritura en los terminales, se provee de una interfaz común de envió de comandos secuencial en formato ristra.

Junto con la capa de acceso, se ha creado una serie de clases alternativas que harán que el desarrollo de las aplicaciones para procesar los mensajes AT de TETRA sea más rápido. Estas herramientas ayudaran a decodificar e interpretar los campos de los comandos AT, también poseen de una serie de métodos para construir correctamente comando para luego enviarlos a los terminales.

El resultado de la segunda fase, fue el desarrollo de una aplicación de cuyo objetivo de cara al usuario, como ya se a dicho antes, es interconectar correo electrónico y mensajería

TETRA, pero más allá de la funcionalidad, el software desarrollado se puede tomar también como una batería de pruebas y ejemplos a la librería de comunicación, que a fin de cuentas es el corazón de este proyecto.

Por ultimo, la fase 3 del proyecto, se destino al desarrollo de una interfaz web para la aplicación desarrollada. Esta última etapa se desarrollo sirve fundamentalmente para darle un acabo de distinción y utilidad a la solución, ya que con esta interfaz se pierde la necesidad de incluir a un usuario con perfil alto de informática como operario de la herramienta. Con esta interfaz de usuario, la cual aparte de servir para poner en práctica las nociones de usabilidad en interfaces de usuario aprendidas durante la carrera, sirve para darle un toque más comercial al producto desarrollado, puesto que no hay que perder la perspectiva de que por mucha investigación, conclusiones y resultados, si no hubiera un trasfondo comercial, esto no sería más que una aplicación académica sin futuro.

Para terminar y como opinión personal, he de decir que el desarrollo de este proyecto final de carrera me ha parecido muy gratificante y completo, por un lado en todas las fases de desarrollo se ha tocado ramas que se han ido aprendiendo durante todo el proceso de aprendizaje. Configuración, automatización del sistema operativo, configuración de servidores, análisis, diseño y programación de una aplicación completa destinada en principio a un usuario final. El poner en práctica todas las disciplinas aprendidas durante los años previos de carrera y hacerla converger en una sola solución da una mayor satisfación por los resultados obtenidos. Por otro lado, realizar una aplicación de servidor y las horas dedicadas al estudio del hardware, como se comunicaba con otros dispositivos y el análisis de los protocolos, me han parecido apasionantes, ya que al venir previamente de la carrera de Ingeniería Técnica en Informática de Sistemas, todos estas tareas son de gran interés para mi.

11.- Apéndice I. Detalles sobre la implementación TetraATLib

Como detalles de implementación de TetraATLib se presenta la descripción de los patrones aplicados para la realización de la arquitectura, así como un ejemplo muy básico de uso de las herramientas de comunicación con los terminales.

11.1.- Patrón suscriptor

Para manejar de la mejor manera posible los mensaje que provienen de un terminal y que según el tipo de mensaje se procese por el controlador deseado, se ha implementado un patrón observador en la librería.

Este patrón consta de dos partes, una clase que se encargue de enviar el mensaje a todos los objetos suscritos y una interfaz común para toda clase que se desee suscribir. En la implementación la clase encargada de notificar a todos los elementos suscritos es la case *Listener.*

```
struct listaComandos;

struct listaComandos{
    ATCommand *comando ;
    struct listaComandos *siguiente;
};


class Listener {
public:
    Listener();
    Listener(const Listener& orig);
    virtual ~Listener();
    int listen(conectaAT *ConexionT);
    ATCommand* getCommand();
    void publicar(string dato);
    bool suscribirse(ATCommand *com);
private:
    string respuesta;
    int tipo;
    conectaAT *Conexion;
    struct listaComandos *subscripciones;
};
```

Esta clase consta en sus atributos una lista de objetos del tipo ATCommand. ATCommand es el nombre de la interfaz que todo subscriptor ha de implementar.

Junto con la lista ATCommand implementa el método suscribirse, necesario para manejar esa lista.

El método más importante de esta clase es publicar, el cual enviar un dato leído a todos los objetos suscritos.

```
void Listener::publicar(string dato){
         struct listaComandos *aux = subscripciones;
            while (aux != NULL){
                aux->comando->notificar(dato);
                aux = aux->siguiente;
            }
    }
```

La implementación del método publicar es muy sencilla, simplemente recorre la lista de objetos suscriptor y ejecuta su método notificar. Esta lista no tiene ningún control y aunque el primer objeto sea el único encargado de tratar un comando determinado, este comando se enviará a todos los subscriptores. Esto se decidió así por que existe la posibilidad que se añadan subscriptores que traten los mismos comando AT.

11.2.- Interfaz ATCommand

Como se detalla en el apartado de diseño de la librería TetraATLib. Para poder hacer uso del patrón observador. Toda clase que desee suscribirse al listener de cada puerto ha de implementar la interfaz ATCommand. Esta interfaz tiene los siguientes métodos.

```
class ATCommand {
public:
    ATCommand();
    ATCommand(const ATCommand& orig);
    virtual ~ATCommand();
    virtual  void    executecomand(const char* comando);
    virtual  void  executecomand();
    virtual  int getType();
```

```
    virtual void notificar(string nota);
};
```

Cada objeto ATCommand es avisado de un nuevo dato leído de la radio por la invocación del método notificar. Este método será el encargado de manejar toda la lógica de proceso de un comando AT en el caso de que la notificación coincida con el tipo de mensaje AT que su implementación trata.

11.3.- Ejemplo de uso

A continuación se detalla un ejemplo de uso de la librería TetraATLib. Este ejemplo dará los detalles más importantes de su uso. El ejemplo desarrollara una clase llamada colector, cuyo único funcionamientos será almacenar todo comando AT que se le notifique.

Para empezar se detalla la clase colector que extiende a la clase ATCommand

```
class Colector : public ATCommand {
public:
    Colector();
    Colector(const Colector& orig);
    virtual ~Colector();
    void notificar(string nota);
private:
    string numero;
};
```

Esta clase tendrá un método notificar que se encargará del manejo del comando notificado. No se incluirá la implementación de este método para no hacer más complejo el ejemplo.

Para controlar la conexión al terminal y de los suscriptores se crea una clase llamada controlador radio.

```
class controlador_radio {
public:
    controlador_radio();
```

```
    controlador_radio(const controlador_radio& orig);
    virtual ~controlador_radio();
    void arranca();
    void loop();

private:
   listener  *escuchador;
};
```

Esta clase principalmente tiene dos métodos, uno para inicializar los objetos y otro para llamar al listener que notifique lo que ha leído.

```
void controlador_radio::arranca(){

    escuchador = new Listener();

    col = new Colector();
    escuchador->suscribirse(col);
}
```

El método listen de un listener realiza una lectura del buffer de la conexión y la notifica a los subscriptores. Como ya se ha explicado, cada subscriptor se encarga de leer el paquete y procesarlo en caso de que corresponda.

```
controlador_radio::loop() {

     escuchador->listen();

}
```

Para terminar la invocación de estos dos métodos es muy sencilla, bastaría con inicializar el entorno con el comando arranca e ir iterando e invocando al método loop.

```
int main (){

    controlador_radio * cont = new controlador()

    cont->arranca();

    while(1){

           cont->loop();
    }
```

```
}
```

12.- Apéndice II. Detalle de la implementación de TetraServer

Se analizara en este apartado los elementos mas destacables de TetraServer.

Entre estos elementos se incluye, el manejo de la base de datos y el bucle principal, Para terminar se incluirá la explicación del método de notificación de la caída de terminales y su reconexión en caso de perdida.

12.1.- Base de datos

El acceso a la base de datos, se hace mediante la clase base_datos_mysql. Esta clase esta construida de forma que se pueda usar como objeto *singleton.*

```
extern string db_host , db_user , db_pass , db_name ;
class base_datos_mysql {

private:
    static base_datos_mysql *pinstancia;
    mysqlpp::Connection *con;
    mysqlpp::StoreQueryResult res;
    mysqlpp::SimpleResult sres;
    string q;


public:
    base_datos_mysql();
    base_datos_mysql(string host , string name , string user , string
pass);
    ~base_datos_mysql();
    void setQuery(string  q);
    void queryS(string q);
    bool queryI(string q);


    int filas();
    int columnas();
    int afectados();

    string getDato(int row , int col);
    string getDato(int row , string col);
    static base_datos_mysql *instancia();
}
```

Para que esta clase, funcione de esta manera, se ha de declarar como atributo de la clase un puntero estático a un objeto de su misma clase, junto a este puntero, se ha de declarar también un método estático que devuelva dicho puntero. Este método ademas de devolver el puntero, se ha de encargar de que el puntero devuelto corresponda a un objeto inicializado, en el caso de que no sea asi, ha de inicializarlo antes de crearlo.

```
base_datos_mysql* base_datos_mysql::instancia()
{

    if (!base_datos_mysql::pinstancia)
    {
        base_datos_mysql::pinstancia = new base_datos_mysql();
    }
    return base_datos_mysql::pinstancia;
}
```

Aplicando este patrón al objeto de base de datos, se evita el tener que estar conectando a la base de datos cada vez que se realizar una operación desde un método, sirviendo la misma y única instancia ya inicializada para todos los casos. Otra ventaja en la ganancia de rendimiento, puesto que se evita estar realizando costosas operaciones de conexión en cada uso.

Aunque el objeto *base_datos_mysql* se pueda utilizar en como objeto *singleton*, en el caso de necesitarlo, se podría usar como objeto dinámico, inicializándolo de manera convencional cuando se precise.

12.2.- Bucle principal

Lo que se detalla aquí como bucle principal son paso a paso las operaciones en secuencia que realiza TetraServer. TetraServer esta desarrollado como una aplicación que se inicializa y luego empieza a iterar consultado en cada iteración los servicios que interconecta.

```
while (1) {

    radio->loop();

    correo->getCorreo();

    correo->enviarCorreo();


    if ((time(NULL) - _LAST_MONITOR_TS_) > _MONITOR_TIMER_ ){
            registro->signal();
            registro->inactivos();
            _LAST_MONITOR_TS_ = time(NULL);
    }
    if ((time(NULL) - _LAST_PING_TS_) > _PING_TIMER_ ){
            mon->checkbridges();
            _LAST_PING_TS_ = time(NULL);
    }

    sleep(_MAIN_LOOP_TIMER_);
}
```

Los pasos que realizar TetraServer son los siguientes:

- **controlador_radio::loop() :** En esta operación el controlador radio consulta los bufferes de los terminales asociado, enviándolo a los subscriptores, también en este paso, se recolecta desde la base de datos los mensajes a retransmitir a los dispositivos.

- **adaptador_correo::getCorreo() :** En esta operación el adaptador de correo, recolecta de los buzones de los dispositivos mensajes pendiente de enviar. Un vez obtenidos los mensajes, se insertan en la base de datos para enviarlos en la siguiente iteración

- **adaptador_correo::enviarCorreo()** : La operación enviar correo, enviar todos los SDS pendientes de retransmitir obtenidos en el primer paso de la iteración

- **ActiveReg::signal()** : La operación *signal*, envía un byte de información a la aplicación monitor, notificándole así que funciona correctamente. Esta operación

se ejecuta cuando se ha pasado un *timeout*, indicado por el parámetro _MONITOR_TIMER_

- **ActiveReg::inactivos() :** Esta operación se ejecuta tras un tiempo determinado por _MONITOR_TIMER_, en esta operación se obtiene la lista de terminales, que no han notificado actividad por un periodo prolongado, si se detecta alguno, se envía un correo de notificación a los responsables

- **monitoring::checkbridges() :** Al igual que el método anterior, *checkbridges*, realiza una operación de *ping* sobre todos los puentes del sistema y notifica a sus responsable en caso de perdida.

- **sleep(_MAIN_LOOP_TIMER_) :** La última operación a realizar es un tiempo de espera, se realiza esta operación para no saturar la CPU del servidor.

12.3.- Control de terminales

Para controlar el estado de los terminales, se estable una consulta periódica a los terminales. La periodicidad de este mensaje se establece mediante el comando AT AT+CNUMF?, este comando tiene como función solicitar el número del terminal conectado.

La periodicidad de este comando se establece la variable *tty_notice,* Declarada en el ámbito de temporizadores de la aplicación.

En combinación con el comando de registro se establece un disparador en la base de datos, el cual, se encargar de actualizar la fecha de última recepción de un comando en el terminal, la periodicidad de la de identificación, en cierta manera es sólo útil en casos de inactividad prolongada en los comandos AT.

```
create trigger actualiza3  on captura
```

```
For each row
BEGIN
      update listener set FechaUltimaRecepcion = CURRENT_TIMESTAMP
      where puerto = (
            select
                Valor
            from configuracion c
            where c.NombreCampo = 'puerto'
              and c.IdAmbito = (select
                   idAmbito
                from configuracion c
                where c.NombreCampo = 'numero'
                  and c.Valor = New.receptor)
             );
 END;
```

Para enviar una notificación se establece en el bucle principal una consulta a terminales inactivos, por definición, se ha establecido, considerar un terminal desconectado si ha pasado dos veces el tiempo establecido por el parámetro *tty_notice.* En el caso de detectar una inactividad prolongada, se activa una alarma que enviá un email a todos los responsables asociados.

13.- Apéndice III. Detalles de implementación de TetraWeb

Los siguientes puntos, resaltan lo elementos mas importantes y complejos del desarrollo de la aplicación TetraWeb.

13.1.- Parada y arranque de TetraServer desde la interfaz web

Durante el análisis de TetraWeb, se concretó el requisito de tener que incluir la posibilidad de parar y arrancar la instancia de TetraServer asociada, esto entro en conflicto directo con las políticas de seguridad del servidor web, puesto que seria un gran riesgo que desde una aplicación web, ser pudiera arrancar o parar un servicio del sistema.

Para solventar la problemática anterior, se elaboró un mecanismo de parada y arranque mediante notificación a una tarea programada del sistema.

Concretamente lo que se creó fue una tarea del planificador de tareas del sistema operativo (cron), la cual se ejecutaría cada pocos segundo y que se encargaría de consultar un mensaje dejado en un fichero por la aplicación web, este fichero se localizará en una ruta segura controlada por el servidor web.

```
#!/bin/bash

if [ -f "/var/www/html/tetraserver/notificaciones/nota.txt" ]; then
while read line
do

        modo=`echo "$line" | cut -f 1 -d " "`

        dominio=`echo "$line" | cut -f 2 -d " " | tr -d "\r"`

        user="user"
        dbna="controller"
        pass="****"

        if [ $modo == "start" ]; then
```

```
                sql="select conf_file from stock where
dominio='$dominio'"

                if [ -z $pass ]; then
                        conf_file=`mysql $dbna -u$user -s -N -e
"$sql"`
                else
                        conf_file=`mysql $dbna -u$user -p$pass -s -N
-e "$sql"`
                fi
                /usr/sbin/tetraserver $conf_file &
                pid2=$!
                sql3="update stock set pid = $pid2 where
dominio='$dominio'"
                if [ -z $pass ]; then
                        pid=`mysql $dbna -u$user -s -N -e "$sql3"`
                else
                        pid=`mysql $dbna -u$user -p$pass -s -N -e
"$sql3"`
                fi


        else
                sql="select pid from stock where dominio='$dominio'"
                if [ -z $pass ]; then
                        pid=`mysql $dbna -u$user -s -N -e "$sql"`
                else
                        pid=`mysql $dbna -u$user -p$pass -s -N -e
"$sql"`
                fi
                kill -9 $pid
                mysql $dbna -p$pass -s -N -e "update stock set pid = 0
where dominio='$dominio'"
        fi

done < /var/www/html/tetraserver/notificaciones/nota.txt
rm -f /var/www/html/tetraserver/notificaciones/nota.txt
fi
```

En el script de bash mostrado a continuación, se aprecian varios elementos nuevos que no se han comentado anteriormente.

Un de estos cambios es la aparición de la base de datos *controller*, Esta base de datos se creó como consecuencia de la petición de poder incluir múltiples instancias de TetraServer en ejecución a la vez en la misma maquina, cada instancia con sus recursos independientes. Esto trajo consigo la necesidad de un ente que controle los parámetros de todas las posibles instancias en ejecución en la maquina, el elemento creado fue una base de datos llamada *controller*, que se encarga de controlar, las instancias creadas, su identificador de procesos, los ficheros de configuración y los parámetros de conexión a la base de datos.

El script de inicio/parada, como se aprecia, lee el fichero creado por el servidor web, el cual le indica un nombre de dominio y la operación a realizar. En el caso de realizar una operación de parada, primeramente se busca el PID (*process identifier*) de la instancia del proceso a parar, para realizar sobre el una operación *kill*. Tras matar el proceso se borra el PID de la base de datos *controller.* En el caso de querer arrancar la instancia, se ha de obtener de la base de datos *controller,* la ruta del fichero donde esta la configuración de arranque de TetraServer. Una vez arrancada la instancia de TetraServer, se obtiene su PID y actualiza este valor en la base de datos.

13.2.- Configuración base de datos Multi-instancia

Debido a la necesidad de ejecutar mas de una instancia de TetraServer por maquina y que dichas instancias no relacionen sus datos de ninguna manera. Se provee a TetraWeb de un mecanismo multi base de datos. En otras palabras, la instalación de TetraWeb será única y los datos mostrados serán en función del usuario que se haya registrado.

Para que esto sea posible. Se incluye en la operación de *login*, un parámetro más que indique a que dominio se esta autenticando el usuario. Con este tercer parámetro, aparte de lo de usuario y contraseña, se puede hacer una consulta a la base de datos correctas, ya que al crear cada instancia, se incluye en el fichero de configuración de bases de datos una entrada al *array* de bases de datos usando como clave el nombre de domino de la instancia.

```
$db['default']['hostname'] = 'localhost';
$db['default']['username'] = 'user';
$db['default']['password'] = '****';
$db['default']['database'] = 'controller';
$db['default']['dbdriver'] = 'mysqli';
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
```

```
$db['default']['stricton'] = FALSE;


//base de datos para la instancia con dominio tetraserver
$db['tetraserver']['hostname'] = 'localhost';
$db['tetraserver']['username'] = 'user';
$db['tetraserver']['password'] = '****';
$db['tetraserver']['database'] = 'tetraserver';
$db['tetraserver']['dbdriver'] = 'mysqli';
$db['tetraserver']['dbprefix'] = '';
$db['tetraserver']['pconnect'] = TRUE;
$db['tetraserver']['db_debug'] = TRUE;
$db['tetraserver']['cache_on'] = FALSE;
$db['tetraserver']['cachedir'] = '';
$db['tetraserver']['char_set'] = 'utf8';
$db['tetraserver']['dbcollat'] = 'utf8_general_ci';
$db['tetraserver']['swap_pre'] = '';
$db['tetraserver']['autoinit'] = TRUE;
$db['tetraserver']['stricton'] = FALSE;
```

Para mantener durante toda la sesión el dominio al que se hará referencia y por ello a la base de datos especifica, se crea en el momento de autenticación una entrada en el *array* de parámetros de sesión una entrada que incluya el dominio actual, mediante funciones de CodeIgniter se realiza de la siguiente manera.

```
$sess_array = array(
      'id' => $row->id,
      'nombre' => $row->firstname,
      'perfil' => $row->profile,
      'dominio'=> $domain
      );

      $this->session->set_userdata('logged_in', $sess_array);
      //Se inicia una sesión con los datos obtenidos del login
```

Junto a lo anterior, en cada momento que se desee consultar cualquier dato a la base de datos, habrá que primeramente, acudir a la variable de sesión *logged_in->dominio.*

```
  $this->load->library('session');
  $login = $this->session->userdata('logged_in');

  $db_actual = $this->load->database($login['dominio'], true);
```

13.3.- BootStrap Responsive

Para reducir el tiempo en el desarrollo de las vistas de TetraWeb se recurrió a la framework *CSS* BootStrap, este framework provee al desarrollador una serie de componentes visuales bastante completos, ahorrando por tanto los detalles tediosos del diseño de la interfaz.

Otra gran característica de esta herramienta y la cual se ha usado en este proyecto es la capacidad de realizar plantillas de estilo *responsive* de una manera bastante sencilla. Estando por tanto la interfaz de TetraWeb adaptada para dispositivos móviles como SmartPhones o tabletas.

13.4.- Procesos almacenados CodeIgniter

Como último apunte comentar el uso de procesos almacenados de base de datos para las operaciones mas críticas de TetraWeb, lamentablemente en primera instancia, CodeIgniter no da soporte especifico para estas operaciones si se usan con su Active Record. Para poder utilizar ambas herramientas, ya que ambas tienen cualidades muy interesantes se ha de hacer uso del *driver* de base de datos *mysqli.*

La especificación del *driver* se hace en el fichero de configuración de base de datos. Se puede observar un ejemplo de uso de este parámetro en el apartado anterior.

14.- Apéndice IV. Configuración de herramientas

14.1.- Pasarelas RS232-Ethernet

Como se comento antes, la conexión distribuida de los terminales se hará a través de internet. Esto supone un problema ya que ETSI definió el estándar de la comunicación mediante puerto serie, para solventar esto se ha optado por usar una pasarela RS232-Ethernet que estará físicamente junto a cada terminal. Estos terminales se conectaran a Internet y se les especificara un puerto y una dirección IP en la que el el servidor que ejecuta TetraServer escuchara y enviará los datos de los terminales.

En dispositivo seleccionado para este fin, cedido por el servicio de seguridad de la ULPGC es un MOXA nPort 5110.

Los pasos necesarios para configurar este dispositivo se encuentran en perfectamente detallados en el manual de configuración. Aquí solo se detallarán los pasos en la configuración del servidor.

Para que un equipo sea capaz de trabajar con estos dispositivos, es necesaria la utilización de un *driver*, que cree una interfaz RS-232 virtual en el sistema. En este caso el *driver* ha de ser para un sistema operativo GNU/Linux. Para este dispositivo el fabricante distribuye el *driver* en formato código fuente, luego el primer paso para hacerlo funcionar es compilarlo. Las dependencia del *driver* son las siguientes:

- kernel >= 2.6.x
- fuentes del kernel
- gcc >= 2.7.2.1
- ld.so >= 1.7.14
- libc.so >= 5
- binutils >= 2.7.0
- make >= 3.74
- gunzip >= 1.2.4

- gawk >= 3.1.1.9
- openssl >= 0.9.8a

Para compilar o instalar el *driver* el comando necesario es el siguiente:

x86

```
$ /ruta/al/código/fuente/mxinst
```

x86_64

```
$ /ruta/al/código/fuente/mxinst m64
```

Una vez terminado el *script* de instalación se crearan en /usr/lib/npreal2/driver las utilidades para conectar nuestro equipo con el dispositivo. En esta caso, el comando más interesante es mxaddsvr el cual crea una interfaz serie en sistema. Estas nuevas interfaces se pueden encontrar en el sistema en la carpeta /dev/ y los ficheros especiales creados responden al nombre ttyrXX donde X es un dígito del 0-9. La sintaxis del comando es la siguiente:

```
mxaddsvr [NPort IP Address] [Total Ports] ([Data port] [Cmd port])
```

Para más información sobre las posibilidades del *driver* visitar:
http://www.moxa.com/doc/man/NPort_5110_QIG_v3.pdf

14.2.- Instalar Postfix

Para instalar PostFix, hemos optado porel paquete genérico que trae la distribución CentOS 5. Ejecutando el siguiente comando como usuario *root* , se instala postfix con compatibilidad MySQL.

```
$ yum install postfix postfix-mySQL
```

Una vez el gestor de paquetes ha terminado su tarea, se creara la carpeta /etc/postfix que contendrá todos los ficheros de configuración de PostFix.

Dentro de esta carpeta existen varios ficheros importantes:

- main.cf : Almacena la configuración principal de servidor de correo.
- transport.cf : Especifica los destinos de los distintos dominios.
- virtual.cf : Contiene la información sobre los dominios virtuales.
- uids.cf : Establece los id de usuarios de sistemas y dominios virtuales.
- gids.cf : Funciona de igual maneras que uids.cf pero con grupos de usuarios.
- mysql_virt.cf : Especifica los parámetros de la base de datos que almacena la información de los usuarios.
- master.cf : Define como un programa cliente se conecta al servicio y que demonio activar cuando se solicita el servicio.

main.cf

Aquí se especifican los parámetros más importantes del servidor de correo, como los ficheros de configuración extra y los datos de los relayers.

Para especificar el relayer de la ULPGC y el dominio de nuestra máquina se incluyen estos parámetros.

```
relayhost = smtp.ulpgc.es
myhostname = tetra.ulpgc.es
mydomain = $myhostname
mydestination = $mydomain, $myhostname, $transport_maps
Para especificar el tamaño de los buzones en nuestro caso 100MB
virtual_mailbox_limit=102400000      # 100MB
#Para terminar, se añaden los parámetros de los ficheros para
```

```
la configuración con MySQL y usuarios virtuales.

transport_maps=mySQL:/etc/postfix/transport.cf

virtual_mailbox_maps=mySQL:/etc/postfix/mySQL_virt.cf

virtual_uid_maps=mySQL:/etc/postfix/uids.cf

virtual_gid_maps=mySQL:/etc/postfix/gids.cf

virtual_mailbox_base=/var/spool/postfix/virtual

virtual_maps=mySQL:/etc/postfix/virtual.cf

virtual_minimum_uid=100
```

master.cf

En este fichero se activara la opción de trabajar con usuarios virtuales. El fichero ha de quedar de la siguiente manera.

```
smtp      inet  n       -       n       -       -       smtpd

virtual   unix  -       n       n       -       -       virtual
```

La linea de virtual en casi todas las instalaciones suele venir por defecto comentada con el carácter #.

transport.cf

Este es el primer fichero en el que aparece de manera directa la configuración de MySQL. Especifica la tabla de transporte, que indica los dominios del sistema.

```
user=usuario #usuario para el login en la base de datos

password=contraseña #contraseña para el login en la base de datos.

dbname=maildb  #nombre de la base de datos.

table=transport  # nombre de la tabla que almacena la información de
transporte

select_field=transport   #nombre del tipo de transporte, virtual o
```

```
local
where_field=domain  # nombre del dominio.
hosts=localhost #host de la base de datos.
```

Como se puede observar, en estos ficheros, se especifica una conexión a la base de datos con los valores user, password, dbname y host y una sentencia SQL con table, select_field y where_field. En este caso, se ve claramente que se va a consultar el tipo de transporte para un dominio dado (SELECT transport FROM transport WHERE domain = '[dominio_consulta]').

mysql_virt.cf

Como todos los ficheros que especifican una acción con la base de datos, indica mediante valores una sentencia SQL y una conexión a una base de datos. En este caso consulta la carpeta donde depositar un correo para una dirección de correo

```
user=usuario
password=contaseña
dbname=maildb
table=users
select_field=maildir
where_field=address
hosts=localhost
```

virtual.cf

Con este fichero , PostFix es capaz de determinar a donde distribuir un correo según la dirección de destino. En el apartado de diseño de la base de datos maildb se explican varias características interesantes que entran en conjunción con este fichero.

```
user=usuario
password=contraseña
dbname=maildb
table=virtual
select_field=goto
where_field=address
hosts=localhost
```

uids.cf

PostFix con la sentencia que incluye este fichero, obtiene el uid del usuario al que pertenece una dirección.

```
user=usuario
password=contraseña
dbname=maildb
table=users
select_field=uid
where_field=address
hosts=localhost
```

uids.cf

El fichero incluye una sentencia para obtener el gid de una dirección de correo.

```
user=usuario
password=contraseña
dbname=maildb
table=users
select_field=gid
```

```
where_field=address
hosts=localhost
```

14.2.1.- Base de datos maildb

En esta sección no se explicara como instalar la base de datos, pues en el apartado anterior ya se ha mostrado este proceso.

Para este sistema, como se puede intuir a raíz de los ficheros de configuración anterior, es necesario crear tres tablas.

transport

Es la tabla que mapea el transporte. Indica a PostFix que agente usar en cada dominio. No es necesaria si no queremos tener todos los usuarios en la base de datos. En nuestro caso, todos los usuarios están almacenados aquí, esto nos ahorra tener que definir manualmente en el fichero main.cf el valor *mydestinations* con todos los dominios del sistema explícitamente. Simplemente se pone “mydestinations = $transport_maps” y podemos añadir mediante sentencias SQL nuevos dominios virtuales. También nos ahorra tener que reiniciar PostFix cada vez que se añada un cambio a los dominios.

La sintaxis SQL para la creación de esta tabla es la siguiente:

```
CREATE TABLE transport (
        domain varchar(128) NOT NULL default ’’,
        transport varchar(128) NOT NULL default ’’,
        UNIQUE KEY domain (domain)
) TYPE=MyISAM;
```

Descripción de los campos:

- *domain*: Cualquier dominio de nuestro servidor, ya sea de tipo local o virtual.
- *transport*: el método de transporte, los valores permitidos son "local:" para correo local o "virtual:" para un agente de correo virtual

users

La tabla de usuario es el corazón de toda la configuración. La primera cosa a tener en cuenta es la diferencia entre el campo id y el campo *address* que en muchas ocasiones pueden tener valores idénticos. En los casos en los que los usuarios para hacer *login* precisen usar la dirección de correo como *login* para el sistema (nombre@direcion.dom) los valores serán idénticos, en caso contrario en el campo id irá el nombre de *login* del usuario. Otro aspecto importante es el algoritmo de cifrado de la contraseña en este caso se usa la función de mySQL encrypt(). Para mejorar la encriptación de la función se puede incluir un segundo parámetro que se usara como clave extra para cifrar.

Sentencia para la tabla users:

```
CREATE TABLE users (
        id varchar(128) NOT NULL default '',
        address varchar(128) NOT NULL default '',
        crypt varchar(128) NOT NULL default '',
        clear varchar(128) NOT NULL default '',
        name varchar(128) NOT NULL default '',
        uid smallint(5) unsigned NOT NULL default '1000',
        gid smallint(5) unsigned NOT NULL default '1000',
                    home   varchar(128)   NOT   NULL   default
'/var/spool/postfix/virtual',
        domain varchar(128) NOT NULL default '',
        maildir varchar(255) NOT NULL default '',
        PRIMARY KEY  (id),
        UNIQUE KEY id (id),
        UNIQUE KEY address (address),
        KEY id_2 (id),
        KEY address_2 (address)
) TYPE=MyISAM;
```

Descripción de los campos:

- *id*: nombre de usuario ejemplo : “usuario@tetra.ulpgc.es”.
- *address*: dirección de correo ejemplo : “usuario@tetra.ulpgc.es”.

- *crypt*: contraseña cifrada del usuario. para almacenar aquí la contraseña la mejor manera es usar la función encrypt('contraseña') en la inserción.
- *clear*: Contraseña sin cifrar. Usada sólo solo durante la fase de desarrollo para fines de depuración.
- *name*: Nombre real del usuario del sistema o alias en el sistema por ejemplo (terminal 1).
- *uid*: virtual uid.
- *gid*: virtual gid. (por motivos de sencillez todos los usuarios virtuales pertenecen al mismo grupo) .
- *home*: ruta de la carpeta donde están los buzones de postfix por defecto /var/spool/postfix/virtual .
- *domain*: dominio al que pertenece el usuario.
- *maildir*: nombre de la carpeta del buzón de correo.

Para aclarar, postfix hallara la ruta del buzón de un correo entrante mediante la conjunción de los campos home+maildir.

virtual

Esta tabla hace la función de mapa virtual, similar a *aliasses* pero con la salvedad que una entrada en la tabla no apunta a un ejecutable o aun fichero sino a una dirección.

Esto no quiere decir que no se pueda tener una dirección que apunte a una sola dirección de destino. separando por comas el campo goto, se puede definir una lista de destinatarios.

Sentencia SQL para crear esta tabla:

```
CREATE TABLE virtual (
        address varchar(255) NOT NULL default '',
        goto varchar(255) NOT NULL default '',
        UNIQUE KEY address (address)
) TYPE=MyISAM;
```

Descripción de los campos:

- *address*: Dirección virtual la cual sera retransmitida a la dirección del campo goto. También puede ser el nombre de un dominio. Útil en el caso de que se quieran almacenar en un buzón concreto todos los *email* de un dominio que no tengan buzón concreto.
- *goto*: Direción de destino de una dirección virtual, si no se quiere reenviar a otra dirección diferente, poner el mismo valor que en el campo *address*

Una vez terminada la configuración solo habría que reiniciar el servicio de correo con la siguiente sentencia.

```
$ service postfix restart
```

Ejemplos

Ahora especificaremos una serie de ejemplo de como insertar un usuario nuevo de correo, mediante sentencias SQL ayudado por el interprete de MySQL.

Insertar una entrada en transporte.

Con la siguiente sentencia SQL creamos un nuevo dominio virtual (recurso.tetra.ulpgc.es).

```
mySQL> INSERT INTO transport (
                domain ,
                transport)
        values (
                'recurso.tetra.ulpgc.es',
                'virtual:'
        );
```

Nota: Al usar postfix una base de datos para gestionar los dominios no es necesario reiniciarlo cada vez que se añade uno

Insertar un nuevo usuario.

Con el siguiente ejemplo se crea el usuario *admin*, con la contraseña abc en el sistema y se le asigna un buzón.

```
mySQL> INSERT INTO users
        VALUES (
        'admin',
        'admin@recurso.tetra.ulpgc.es',
        encrypt('abc'),
        'abc',
        'Administrador',
        '5000',
        '5000',
        '/var/spool/postfix/virtual',
        'recurso.tetra.ulpgc.es ',
        'recurso.tetra.ulpgc.es/admin/',
        );
```

Insertar una ruta para virtual

Por último creamos la ruta para que PostFix asigne todos los correos del dominio a admin@recurso.ulpgc.es.

```
mySQL> INSERT INTO virtual VALUES ('@recurso.tetra.ulpgc.es' ,
admin@recurso.tetra.ulpgc.es);
```

15.- ANEXOS

ANEXO I

Instalación de TetraServer

ANEXO II

Configuración de TetraServer

ANEXO III

Manual de usuario de TetraServer

ANEXO IV

Instalación Monitor

ANEXO V

Manual de Monitor

ANEXO VI

Instalación de TetraWeb

ANEXO VII

Configuración TetraWeb

ANEXO VIII

Mantenimiento MySQL

ANEXO IX

Instalador

ANEXO X

Instalación de la base de datos

ANEXO XI

Manual de usuario TetraWeb

16.- Trabajo Futuro

Como se ha remarcado en todo momento durante la redacción de este documento, todo el trabajo que se ha realizado, ha sido el inicio de un camino a seguir.

Entre las ideas de extensión a este ecosistema sobre TETRA se encuentran.

- Extender TetraServer para que sea capaz de capturar la localización GPS de los dispositivos asociados. Con estos datos se podría generar un modulo en TetraWeb que genere una serie de mapas con la localización de las unidades. Esto sería altamente útil en el caso de coordinación en situaciones de emergencia.

- Otra posible ampliación seria la capacidad de solicitar mediante plantillas de mensajes SDS, información. Por ejemplo, solicitar información relativa al estado administrativo de un vehículo enviado un SDS con el numero de matrícula. Ahorrando en este caso costes de personal y establecimiento de comunicaciones de voz.

- Como tercera opción de desarrollo, se puede incluir la vía de seguir investigando con el protocolo TNP1 en el caso de tener dispositivos que sean compatibles con el mismo.

17.- Bibliografía

- **Craig Larman.**

  UML y patrones de diseño

- **Erich Gamma, Richard Helm, Ralph Johnson, John Blissides**

  Design Patterns: Elements of Reusable Object-Oriented Software

- **ETSI TS 100 392-5 V1.3.1 (2007-03)**

  Title: Terrestrial Trunked Radio (TETRA);Voice plus Data (V+D);Part 2: Air Interface (AI)

- **ETSI TS 100 392-5 V1.3.1 (2007-03)**

  Terrestrial Trunked Radio (TETRA);Voice plus Data (V+D);Part 5: Peripheral Equipment Interface (PEI)

- **Wikipedia**

  Terrestrial Trunked Radio. 2010 http://en.wikipedia.org/wiki/TETRA

- **Wikipedia**

  Short message service 2010 http://en.wikipedia.org/wiki/SMS

- **Wikipedia**

  Concatenated SMS 2011 http://en.wikipedia.org/wiki/Concatenated_SMS

- **Wikipedia**

  Global System for Mobile Communications 2010 http://en.wikipedia.org/wiki/GSM