ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Configuración de una red telefónica empresarial mediante el uso del protocolo VoIP. Implementación, configuración y administración de la centralita Asterisk y sus funcionalidades.

AUTOR: María del Carmen Llorente Sánchez
DIRECTOR: Juan Carlos Sánchez Aarnoutse
Septiembre de 2013

Autor	María del Carmen Llorente Sánchez
Email autor	mcarmen_llorente83@hotmail.com
Director(es)	Juan Carlos Sánchez Aarnoutse
Email director	juanc.sanchez@upct.es
Codirector(es)	
Título del PFC	Configuración de una red telefónica empresarial mediante el uso del protocolo VoIP. Implementación, configuración y administración de la centralita Asterisk y sus funcionalidades.
Descriptores	Telefonía IP, VoIP, voz sobre IP, Asterisk, software libre.
Resumen	La expansión de la voz sobre IP y la mejora de las características que ofrece frente a las redes de telefonía tradicional, como reducción de costes y versatilidad de aplicaciones hacen de VoIP una tecnología de conocimiento obligatorio. El objetivo del proyecto es implementar soluciones prácticas bajo un sistema abierto de comunicaciones como es Asterisk.
Titulación	Ingeniero Técnico de Telecomunicaciones, especialidad en Telemática
Departamento	Tecnología de la información y de las comunicaciones
Fecha presentación	01/09/13

AGRADECIMIENTOS

Me gustaría dar las gracias en primer lugar a Juan Carlos por su paciencia y por permitirme divagar y dar rienda suelta a todo lo que se me ocurría y que he intentado plasmar aquí.

Sobre todo quería agradecer a mis padres su espera, su silencio, sus tiempos, su esfuerzo y sobre todo su ánimo cuándo lo he necesitado. A ellos se lo debo todo.

A mi hermana, por hacerme ver el lado positivo de las cosas, por darme energía y por estar ahí siempre.

A mi abuela, por hacerme reír, por ser un soplo de aire fresco todos los días, por hacerme soñar y por dejarme claro que todo con esfuerzo se consigue.

A mis amigos, que siempre están ahí, incluso cuándo están lejos. A ellos, a los que un día se les ocurrió regalarme un puñado de arena para hacerme ver que existía el exterior. Por conseguir distraerme de mi estado binario y enseñarme mitología. Sin sus risas y sus ánimos no sería lo que soy hoy.

A Karl, por su empuje, por su positividad y por su paciencia infinita, por estar siempre dispuesto a ayudar. Sin su tenacidad no habría llegado hasta aquí

A Chispi, que ha vigilado cada palabra que he escrito, que me ha alegrado los días con sus juegos.

A los que no están, pero que siempre van conmigo. A los que me enseñaron a caminar.

GRACIAS

íNDICE DE CONTENIDOS

Capítulo 1: INTRODUCCIÓN A VOIP

La historia de la telefonía ha variado mucho desde que en 1876 Alexander Graham Bell hiciera la primera transmisión de voz en un circuito ring-down que ni siquiera permitía la marcación de un número telefónico. Se trataba de un sistema unidireccional formado por un sólo cable.

Posteriormente, la mejora de la transmisión por cable usando un micrófono de carbón, una batería, un electroimán y un diafragma de hierro, permite ampliar el concepto de comunicación.

Tan rápido fue el avance de este sistema de telefonía, en el que era necesario un cable para cada conexión, que inevitablemente surgieron soluciones para poder gestionar un número de llamadas y de clientes es aumento constante.

Es en ese momento cuándo se plantean los switches y las primeras centrales telefónicas, que permiten ahorrar líneas y aumentar el número de posibles clientes.

Inicialmente humanos, éstos switches solucionarán la problemática de la necesidad de un número de líneas: N(N-1)/2, para comunicar a todos los clientes.

1.1 Comparativa de VoIP con telefonía tradicional.

Mientras que las historia de la telefonía tradicional se basa en la conmutación de circuitos, estableciendo un canal dedicado, la historia de Internet toma otros derroteros usando la conmutación de paquetes para transportar información.

Conmutación de circuitos:

La conmutación de circuitos es un tipo de comunicación que usa un canal dedicado (o circuito) durante la duración de una sesión. Después de que finalice la sesión (ej: una llamada telefónica) se libera el canal y éste podrá ser usado por otro par de usuarios.

El ejemplo más típico de este tipo de redes es el sistema telefónico, el cuál usa conexiones dedicadas de cable para crear un circuito único durante la duración de una llamada. Los sistemas de conmutación de circuitos son ideales para comunicaciones que requieren que los datos/información sean transmitidos en tiempo real.

Las características que definen a los sistemas de conmutación de circuitos son las siguientes:

- Tráfico constante
- Retardos fijos
- Sistemas orientados a conexión
- Sensibles a pérdidas de la conexión.
- Orientados a voz u otras aplicaciones en tiempo real

Conmutación de paquetes:

En los sistemas basados en conmutación de paquetes, la información que va a ser transmitida previamente es encapsulada en paquetes. Cada paquete es transmitido individualmente y éste puede seguir diferentes rutas hacia su destino. Una vez que los paquetes llegan a su destino, los paquetes son otra vez re-ensamblados.

Mientras que la conmutación de circuitos se usa un único canal para cada sesión, en los sistemas de conmutación de paquetes el canal se comparte por muchos usuarios a la vez. La mayoría de los protocolos de WAN tales como TCP/IP, X.25, Frame Relay, ATM, se basan en la conmutación de paquetes.

La conmutación de paquetes es muy eficiente para datos de gran volumen o que no necesitan de una llegada en tiempo real.

Las características de éstos sistemas son:

- Tráfico en ráfagas
- Retardos variables
- No orientados a conexión
- Sensibles a pérdidas de datos
- Orientados a aplicaciones de datos.

1.2. Por qué VoIP. Características y evolución

El desarrollo de la telefonía analógica se expande hacia un nuevo camino y es la posibilidad de usar en la misma línea voz digitalizada.

Éste cambio produce una mejora en cuánto a la calidad de sonido entre los interlocutores.

Definiendo el ruido de línea como las interferencias añadidas en una red de voz, y el ruido acumulado como el fruto del paso por varios amplificadores, se pueden observar las diferencias entre una señalización analógica y una digital.

La señalización analógica contendrá una señal que a lo largo de la comunicación por cable, va aumentando el nivel de ruido de la señal debido al ruido de línea. Si el receptor se sitúa a una distancia tal dónde es necesario amplificar la señal, éste se amplificará aumentando tanto la señal analógica propia de la llamada, como también el ruido asociado a los amplificadores.

Sin embargo, aunque una señal digital también puede contener ruido de línea, serán los repetidores los encargados de limpiar la señal reconstruyéndola y eliminando el ruido acumulado.

Para poder implementar la digitalización de la voz será necesario recurrir a una serie de procesamientos, como modulación por impulsos codificados (PCM), la conversión digital, la detección de actividad de voz y la codificación.

Con el incremento de las necesidades en cuánto a tráfico de datos en detrimento del tráfico de voz, es lógico incorporar en las redes de telefonía los datos que se comienzan a demandar por parte de los clientes.

Las razones del cambio de las redes PSTN a las redes de voz y datos son las siguientes:

- Los datos comienzan a tener una gran carga de tráfico en redes inicialmente construidas para el tráfico de voz. Los datos necesitan un gran volumen de ancho de banda, que sin embargo es variable, por tanto el hecho de tener circuitos dedicados durante el tiempo necesario para el transporte de datos supone un incremento de precio demasiado alto. Sin embargo es posible realizar una diferenciación en las aplicaciones para distinguir el tipo de tráfico que se está usando.
- La PSTN no crea ni desarrolla prestaciones con la rapidez necesaria de demanda. Son los fabricantes de los equipos quiénes se encargan del desarrollo de las aplicaciones.
- Los datos, la voz y el vídeo no pueden converger en la PSTN actual. Con una sola línea analógica no se puede tener acceso a Internet, telefónico y de vídeo a través de un módem de 56 Kbps, es necesario un mayor ancho de banda.
- La arquitectura construida para un sistema de voz no es suficientemente flexible para abarcar también el tráfico de datos necesario.

Sin embargo, un sistema de telefonía de paquetes reúne los requisitos necesarios para dar solución a las crecientes necesidades en el mercado.

Un cambio en la infraestructura y en la metodología permite un desarrollo más rápido de nuevas funciones, así como la posibilidad de aplicaciones software independientes.

Además provee un modelo con estándares abiertos en las capas que permite la creación de una infraestructura de paquetes para el tráfico de voz, así como una capa de control de llamadas y una interfaz de programación de las aplicaciones (API)

Con este sistema de telefonía por medio de paquetes se consigue transmitir un mayor número de información en mejor tiempo, así como integrar las soluciones de voz y datos, mejorando además la calidad de voz.

La telefonía IP aprovecha la infraestructura de telecomunicaciones ya existente, sin embargo necesita nuevos elementos. En primer lugar necesita transformar las ondas de voz en datos digitales y que además los divida en paquetes que puedan ser transmitidos haciendo uso del protocolo IP. Será el procesador de señal digital (DSP), que está integrado en los teléfonos IP o en los gateways encargados de la transmisión, el que se encargue de empaquetar la voz.

El principal problema de la voz sobre IP es que al compartir el ancho de banda con otros datos, y que los paquetes puedan tomar distintas rutas a través de la red, puede derivar en retardos y variaciones de retardos o jitter.

1.3 Funcionamiento

Muestreo digital.:

Aunque la comunicación analógica es la ideal en la comunicación humana, la retransmisión de dicha información no es robusta frente al ruido de línea. Al pasar a través de los distintos amplificadores que estarán repartidos a lo largo de la red, no sólo se incrementa la voz, sino también el ruido de la línea.

Por tanto, es necesario una mejora de la calidad de audio. Las señales digitales, formadas a partir de una serie de modificaciones que parten de una señal analógica, son el medio de transmisión perfecto.

La modulación por impulsos codificados (PCM) convierte el sonido analógico en formas digitales cogiendo muestras de ella 8000 veces por segundo, y convirtiendo cada muestra en un valor numérico.

El teorema de Nyquist afirma que si se muestrea una señal analógica a una velocidad dos veces superior a la frecuencia de interés más alta, se puede reconstruir de nuevo de manera exacta esa señal en su forma analógica.

Teniendo en cuenta que el contenido de voz suele estar por debajo de 4 Khz, se requiere una velocidad de muestreo de 8000 veces por segundo (125 ms entre muestras).

En telefonía cada muestra se compone por 8 bits, por tanto la velocidad de muestreo de serían 64 Kbps (8 bits x 8000 Hz).

Cuantificación:

La cuantificación no es otra cosa que el proceso de redondeo de estos valores muestreados al valor discreto predefinido más próximo, permitiendo representar el valor del impulso como un flujo binario de bits.

Se deberá tener en cuenta tanto el número de niveles de cuantificación como la distribución de los niveles.

El número de niveles de cuantificación es el resultado de la compensación entre la calidad de señal requerida y la proporción de bits de la salida digitalizada.

Un incremento de los niveles de cuantificación produce que la señal digital sea más próxima a la señal analógica de partida.

La distribución de niveles de cuantificación es la resolución digital en los diferentes rangos del valor de la señal analógica.

Puede ser lineal, dónde la calidad de la señal es igual para todos los rangos de valores. O puede ser una distribución asimétrica, que permite optimizar la calidad de la señal de las aplicaciones deseadas. Se usa habitualmente en redes de telefonía, dónde se incrementa el nivel de cuantificación en la zona más cercana al silencio.

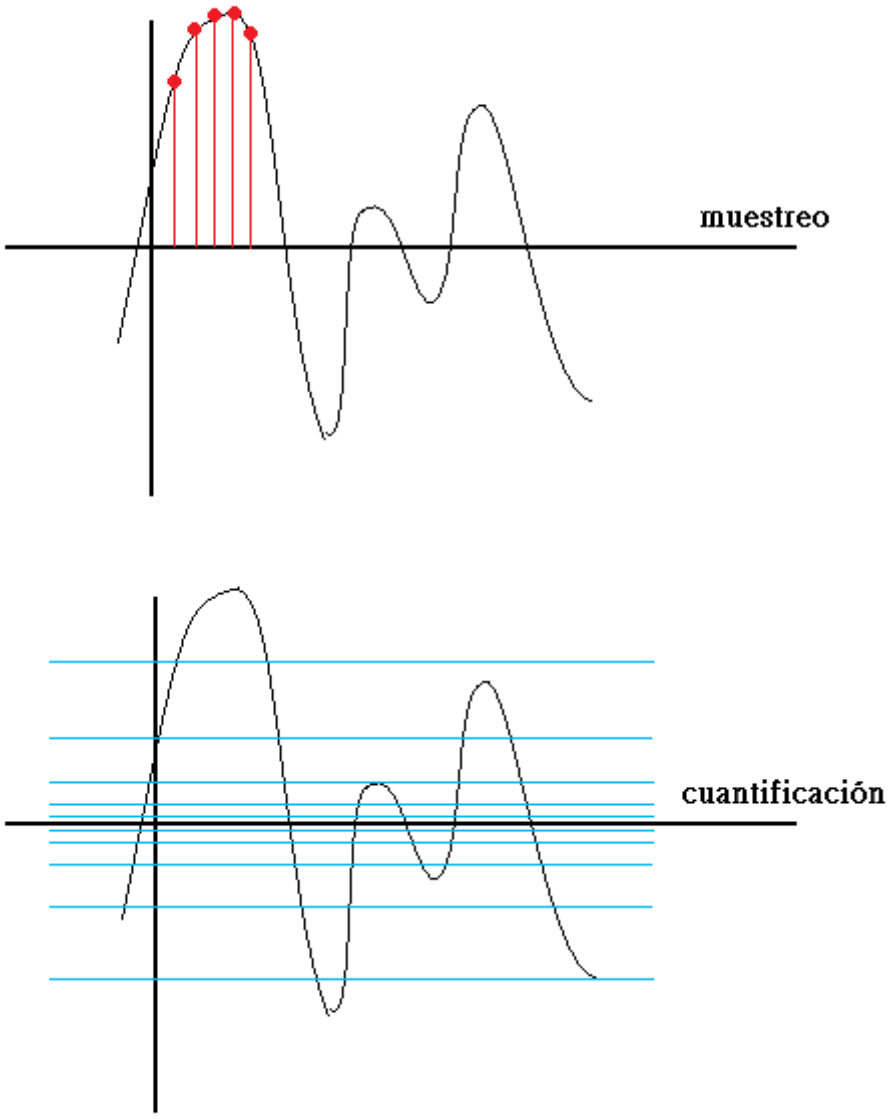


Figura 1: “Muestreo y cuatificación”

Detección de la actividad de la voz (VAD):

En una conversación de voz normal, uno de los interlocutores habla mientras el otro escucha. Teniendo en cuenta que las redes de voz actuales tienen canales bidireccionales de 64Kbps dedicados, supone una pérdida al menos del 50% del total del ancho de banda.

En realidad, esta cifra es mayor si se consideran las pausas necesarias en una conversación.

VoIP permite utilizar éste ancho de banda perdido para otros propósitos si tiene activa la detección de la actividad de voz o VAD. Éste sistema detecta la magnitud de la voz en decibelios para saber cuándo comenzar a empaquetar la voz.

En caso de que detecte una disminución de al amplitud de onda, espera un tiempo (hangover) con una duración de 200 ms antes de dejar de empaquetar las tramas.

Sin embargo, suele tener problemas para determinar el comienzo y el fin de la voz y para distinguir la voz del ruido de fondo. Para solucionar esto se establece un umbral de señal/ruido el cuál permite identificar como voz todo aquello que sobrepase dicho umbral.

Después de un tiempo en silencio, VAD puede detectar de nuevo la voz al cabo de muy poco tiempo, a ésto se le conoce como recorte de voz frontal y suele ser imperceptible.

Codificación:

Una vez tomadas las muestras oportunas de la señal analógica original, es necesario codificar los datos con el fin de que puedan ser empaquetados posteriormente.

Aunque hay muchos tipos de códecs, el objetivo de todos ellos es desarrollar códecs de audio que proporcionen una mejor calidad de conversación con una proporción más baja de bits, de retraso y de complejidad de implementación.

Los códecs tienen que cumplir una doble función, codificar una señal digitalizada en una forma más eficaz para la transmisión o el almacenamiento, además de poder restaurar la señal a la forma original.

Existen diferentes algoritmos de codificación, como son los códecs de forma de onda, los códecs fuente y los híbridos.

Los códecs de forma de onda reconstruyen una señal de entrada sin modular el proceso que creó la señal de entrada. Son poco complejos, y su principal ventaja es que se pueden replicar sonidos de muchas fuentes. Sin embargo no están optimizados para la codificación a baja proporción de bits como los específicos de una conversación normal.

En función del dominio del tiempo existen: PCM (recomendación g.711 de ITU-T, ley α para América del Norte y ley μ para Europa) y ADPCM (recomendación G.726 de ITU-T). Y en el dominio de la frecuencia: SBC y ATC.

Los códecs de fuente intentan replicar el proceso físico de la creación del sonido. Dividen la señal en sonidos sordos, en los cuales no intervienen las cuerdas vocales, similar al sonido blanco; y en sonidos sonoros, considerando que las cuerdas vocales se abren y cierran a distintas frecuencias, modulando el aire que pasa por ellas. El codificador determina la frecuencia de impulso de la modulación de las cuerdas vocales.

El tracto vocal se calcula según una función algebraica de frecuencia de señal.

El tamaño de la trama es variable, y contiene los coeficientes de la ecuación lineal, un bit del tipo de fuente de estímulo y la frecuencia del estímulo sonoro. Es necesaria una agrupación de muestras en tramas para el análisis y la codificación.

Producen señales de muy baja tasa de bits, pero tienen potencial limitado de calidad de voz, normalmente se han usado en aplicaciones militares.

Los códecs híbridos consiguen una mayor calidad de conversación que los códecs de fuente con proporciones de bits más bajos que los códecs de forma de onda.

Son bastante complejos, puesto que combinan un modelado de fuente y de análisis de forma de onda.

Operan en el dominio del tiempo usando técnicas de predicción lineal de análisis por síntesis (LPAS)

A la hora de seleccionar un códec es necesario conocer una serie de características del rendimiento en función de:

- Tasa de bits codificados
- Retraso algorítmico
- Complejidad de procesado
- Calidad de conversación
- Rendimiento de señales que no son propias de la conversación.

Entre las distintas normativas que hay en el mercado caben destacar las siguientes:

ITU-T G.711	PCM de 64 Kbps
	Permite la entrega de voz digital en telefonía pública y como intercambio privado a través de un PBX
ITU-T G.726	Codificación ADPCM de 40, 32, 24 y 16 Kbps
	Permite el intercambio de voz digital tanto en telefonía pública como en redes PBX
ITU-T G.728	Codificación CELP de 16 Kbps
	Permite variación de bajo retraso
ITU-T G.729	Codificación CELP de 8 Kbps
	Proporciona una distinta complejidad de computación y una calidad de voz similar al ADPCM de 32 Kbps
ITU-T G.723.1	Posee una técnica de compresión de voz a baja velocidad de bit.
	Pertenece a la familia de estándares H.324. Las velocidades varían entre 5'3 Kbps (CELP) y 6'3 Kbps (MP-MLQ)

Capítulo 2: TELEFONÍA IP

En 1980, la Organización Internacional de estándares crea el modelos de referencia OSI que fragmenta la comunicación entre máquinas en 7 capas distintas.

Cada capa se ocupa sólo de hablar con su correspondiente capa situada en la otra máquina.

A su vez cada capa proporciona servicios a la que está por encima de ella y solicita servicios de la capa directamente por debajo.

7	Aplicación	NFS
6	Presentación	XDR
5	Sesión	RPC
4	Transporte	TCP, UDP
3	Red	IP, ICMP, Protocolos de enrutamiento
2	Enlace	ARP, RARP
1	Física	No especificados

Debido a la necesidad de transmitir las tramas en un tiempo los más inmediato posible, se recurre al protocolo UDP para el transporte de voz.

Sin embargo, no es suficiente con el uso de UDP, la IETF desarrolla el RTP como protocolo para transmitir tráfico sensible al retraso en las redes basadas en conmutación de paquetes. Este nuevo protocolo contiene una información esencial para determinar los tiempos de llegada y el orden de los paquetes.

Además consta de una parte de datos y una parte de control (protocolo RTCP)

La voz sobre IP se encapsula dentro de RTP, que a su vez se encapsula dentro de UDP, por tanto la cabecera del paquete VoIP será RTP/UDP/IP

Figura 2: “Encapsulamiento VoIP”

2.1. Protocolos relacionados con VoIP

IP

El protocolo de Internet o protocolo IP, se denomina sin conexión y pertenece a la capa 3. No proporciona mecanismo de actuación para dotar al sistema de fiabilidad, control de flujo, secuenciación o reconocimiento.

Sin embargo se puede ejecutar IP a través de cualquier medio dada su posición en el modelo OSI. Es posible escribir una aplicación y tenerla para un conjunto de medios en cualquier sitio.

Está definido en el RFC 791.

Existen distintos tipos de paquetes IP:

- Unidifusión: Sólo identifica una dirección de destino específica. Permite que dos estaciones se comuniquen independientemente de su ubicación.
- Difusión: Paquetes enviados a todos los usuarios de una subred local. Pueden atravesar puentes y switches, pero no routers. Se usan para la comunicación de toda la subred.
- Multidifusión: Usan una gama especial de direcciones que permiten la comunicación entre usuarios de redes distintas. Permiten aplicaciones que tienen un transmisor y múltiples receptores, como por ejemplo en una videoconferencia.

UDP

En la capa de transporte se ha optado por el uso del protocolo UDP entre las posibles opciones para el transporte de IP.

Ha sido elegido en detrimento de TCP por la necesidad de una alta velocidad en la transmisión de los datos. Sin embargo, no proporciona seguridad en la entrega de los paquetes.

Aunque TCP es fiable y garantiza la entrega, es demasiado lento para transmitir señales de audio que deben llegar con un retardo mínimo.

El protocolo de datagrama de usuario (User Datagram Protocol, UDP) está definido en el RFC 768.

Se trata de un protocolo más sencillo que TCP.

Se define como protocolo sin conexión y contiene una cabecera más pequeña, lo que produce un menor coste temporal en la transferencia.

Los campos UDP son:

- puerto origen
- puerto destino
- longitud (longitud de cabecera + longitud de datos)
- suma de comprobación (opcional)

No hay retransmisión de paquetes, por tanto existe la posibilidad de que se puedan perder algunos por el camino.

Se usa en VoIP para transportar el tráfico de voz real, puesto que en VoIP controlar la latencia es más importante que asegurar la entrega fiable de cada paquete.

RTP

RTP es un protocolo de transporte de audio usado por UDP para el transporte de información en tiempo real en redes de conmutación de paquetes.

Proporciona a las estaciones receptoras un número de secuencia, que permite ver si los paquetes llegan en orden, y una marca de temporización, que determina el tiempo de llegada entre paquetes.

Contiene una parte de datos y una de control.

La parte de datos es el protocolo TCP propiamente dicho, da soporte a aplicaciones con propiedades de tiempo real gestionando la reconstrucción de la temporización, la detección de pérdidas y la identificación de contenidos.

La parte de control RTP está gestionada por el protocolo RTCP

Los paquetes VoIP finales contendrán una o más muestras de códec de voz o tramas encapsuladas en cabeceras IP/UDP/RTP.

UDP proporciona los servicios de entramado y multiplexación de la aplicación para VoIP (a través de los puertos UDP). Y RTP proporciona los servicios adicionales que se necesitan para el transporte de datos en tiempo real.

La cabecera RTP tendrá la siguiente forma:

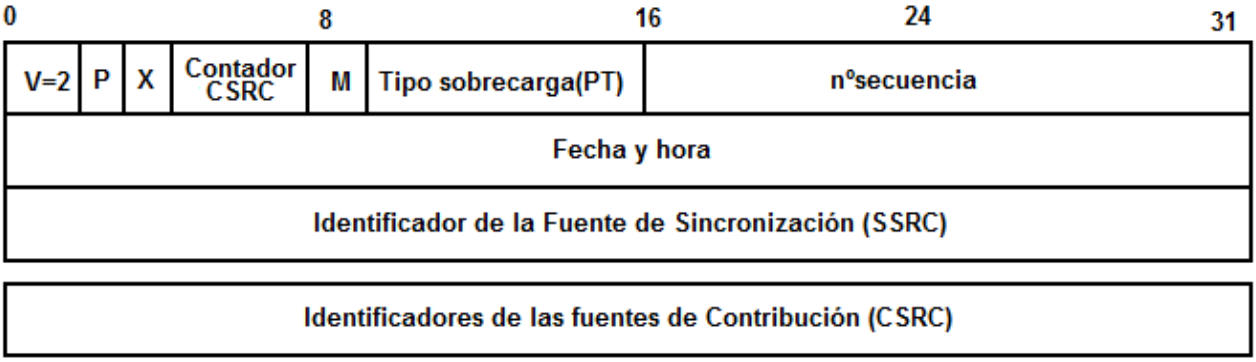


Figura 3: "Cabecera RTP"

Los campos que conforman la cabecera son los siguientes:

- V: versión de RTP con valor 2, compatible con RFC 1889
- P: Padding, existencia de bits de relleno
- X: Extensión, indica si hay extensión de cabecera de longitud variable
- CC: Contador CSRC (CC), número de campos de cabecera CSRC que sigue a la cabecera, 12 bytes
- M: Marker, identifica el comienzo de un paquete de información speech burst para VoIP
- PT: Tipo de sobrecarga, identifica los códecs de audio
- nº secuencia: Permite detectar paquetes perdidos
- fecha y hora: Permite corregir el tiempo de temporización de la sobrecarga
- SSRC: Fuente de envío, identificador de emisor durante sesión
- CSRC: Fuentes de contribución, todos los SSRC que se mezclan en una conferencia.

El tamaño total de la cabecera RTP podrá variar entre 8, 12 y 20 bytes. Esto supone un transporte ineficaz de mensajes VoIP, puesto que las cabeceras tienen el doble de bits que la información transportada.

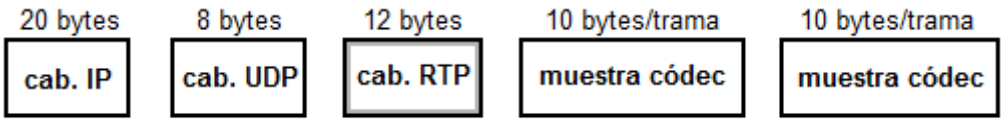


Figura 4: "Tamaño de cabeceras IP/UDP/RTP"

Una solución a este problema es comprimir las cabeceras a 2 ó 4 bytes, usando las compresión de cabecera RTP (CRTP)

Los servicios proporcionados son:

- Distinción entre los emisores múltiples en un flujo multidifusión RTP (gracias a SSRC y CSRC)
- Conserva la relación de temporización entre los paquetes. El bit M identifica el comienzo de una conversación entrecortada, mientras que el buffer playout controla las ráfagas de paquetes.
- Posibilita la sincronización de temporización entre los flujos de medios gracias al campo fecha y hora.
- Secuencia los datos para identificar los paquetes perdidos. Para ello usa el número de secuencia, que aunque no son retransmitidos, pueden ser devueltos gracias a los informes de RTCP.
- Identifica los tipos de medios. El tipo de sobrecarga (PT) refleja el códec de audio usado.
- No asegura la calidad de servicio.

RTCP

RTCP es un protocolo que trabaja junto a RTP complementando algunas de sus funciones como:

- Envío de información periódica sobre la calidad de servicio.
- Control de la sesión con el uso del paquete BYE
- Identificación. Los paquetes incluyen los datos de todos los participantes de la sesión.
- Sincronización entre medios. Provee la información necesaria para que se reproduzcan a la vez el audio y el vídeo.
- Permite administrar informes
- Es posible el uso de conferencias RTP multimedia.

Está definido en el RFC 1889.

Es especialmente útil en VoIP punto a punto para proporcionar retroalimentación de calidad de servicio desde el receptor al emisor.

Existen varios tipos de paquetes RTCP:

** SR (Sender Report):*

Es enviado desde el origen a todos los participantes multidifusión.

Contiene informes de emisión y recepción.

Es una mezcla de información del emisor RTCP y un informe de recepción del RTCP respecto a cada emisor desde el que recibe los flujos de medios.

En VoIP, cada punto final actúa como emisor, de forma que ambos forman los SR

Aprenden las siguientes estadísticas de la red: tiempo de ida y vuelta (RTT), tasa de paquetes perdidos y fluctuación de fase.

** RR (Receiver Report):*

Enviado por participantes "sin orígen" a todos los participantes multidifusión.

Contiene informes de recepción

Informe de recepción del RTCP respecto a cada emisor desde el que recibe los flujos de medios.

Los participantes de la sesión pueden conocer las tasas de paquetes perdidos examinando el campo RR de los RTCP.

** SDES (Source Description):*

Proporciona información acerca de cada emisor de medios RTP en una sesión multidifusión.

El elemento CNAME refleja el host de la aplicación.

Cada host envía un paquete sencillo relacionado con su propia identificación SSRC en un paquete RTCP de descripción de origen.

Un dispositivo mezclado (puente de conferencia VoIP o MCU) envía varios bloques de elementos SDES en un paquete RTCP.

** BYE*

Paquete enviado por los participantes de una conferencia RTP multimedia a cada participante cuándo se desconectan.

Es importante, ya que cada participante limita la velocidad de las transmisiones RTCP en función del número de participantes.

Sin los mensajes BYE el número de participantes continuaría aumentando siempre.

** APP:*

Capacitan al protocolo para la experimentación y las extensiones sin necesidad de nuevos tipos de paquetes que registren un tipo de paquete/sobrecarga.

2.2 Protocolos de control y señalización de llamadas.

Los protocolos asociados a VoIP se pueden dividir en dos grupos: los que transportan la ruta de audio, y los de señalización de llamadas y funciones de control.

Los que transportan la ruta de audio contienen información de temporización para asegurar una reproducción de audio consistente en la recepción. Además permiten la retroalimentación del rendimiento de la Calidad de Servicio (QoS) respecto a la red subyacente.

RTP y RTCP son protocolos de transporte de audio.

Aquellos que se encargan de la señalización de llamadas y las funciones de control permiten configurar y cancelar la llamada. Además incluyen las distintas formas de direccionamiento y enrutamiento así como servicios de información adicionales y métodos para trabajar con otros tipos de señalización.

Entre ellos se encuentran: H.323, SIP, H.248, Megaco y MGCP.

2.2.1 Tipos y definición:

Entre los más comunes en el tratamiento de la voz sobre IP nos encontramos con:

H.323

Especifica cómo se transporta el tráfico multimedia sobre las redes de paquetes.

Utiliza los estándares existentes.

Es un protocolo bastante complejo creado para permitir la ejecución de aplicaciones multimedia sobre redes de datos “no fiables”.

La aplicaciones multimedia de vídeo y datos ocupan la mayor parte del protocolo. Dichas aplicaciones requieren un gran trabajo para ser escalables con H.323, necesitan especificaciones separadas.

SIP

Protocolo de control de la capa de aplicación definido en el RFC 254.

Se usa para crear, modificar y terminar sesiones con uno o más participantes.

Las sesiones multimedia incluyen audio vídeo y datos.

Además, permite que los participantes sean invitados a una conferencia improvisada.

Permite sesiones multimedia de multidifusión y unidifusión.

Está poco implementado a pesar del interés de los fabricantes y clientes.

SGCP y MGCP

Permiten que un dispositivo central (agente de llamadas, MGC) pueda controlar los puntos finales (MG)

Usualmente se les conoce como xGCP.

Se pueden desarrollar aplicaciones mediante el uso de API basadas en estándares que comunican con los MGC y ofrecen funcionalidad adicional.

IAX2

Protocolo para la comunicación entre PBXs Asterisk en reemplazo de IAX.

Incorpora mejoras al protocolo original.

Permite minimizar el ancho de banda usado en las transmisiones de control y multimedia de VoIP.

Evita problemas de NAT (Network Address Translation)

Proporciona un soporte para transmitir los planes de marcación.

2.2.2 H.323

H.323 es el estándar creado por la Unión Internacional de Telecomunicaciones (ITU-T) que se compone por un protocolo sumamente complejo y extenso, el cuál además de incluir la voz sobre IP ofrece especificaciones para videoconferencias y aplicaciones en tiempo real.

El H.323 es una familia de estándares definidos para las comunicaciones multimedia sobre redes LAN. El hecho de que se defina específicamente para redes LAN no garantiza una calidad de servicio óptima.

Algunos ejemplos de uso de éste protocolo es sobre TCP/IP sobre Ethernet, Fast Ethernet o Token Ring. Sin embargo la más común dónde se implementa es IP.

Elementos H.323:

Terminal

Los terminales o puntos finales son teléfonos con soporte opcional para vídeo interactivo y aplicaciones de datos compartidas. Permiten conferencias punto a punto y multipunto para audio, vídeo y datos.

Permite controlar el sistema con un seguimiento del intercambio de la capacidad, control de llamadas, mensajería y señalización de comandos.

En cuánto a la transmisión de medios permite dar formato y recibir todo tipo de mensajes desde la interfaz de red. Además codifica y descodifica la señal de audio.

Gestiona servicios de unidifusión y multidifusión de extremo a extremo tanto para TCP como para UDP.

Las funciones de control que realizan los terminales son:

- H.245 para negociación del canal
- H.225.0 (Q.931) para señalización y control de llamada
- H.225.0 (RAS) para comunicación con el gatekeeper

Gatekeeper

El gatekeeper o GK regula los puntos finales dentro de su zona que pueden iniciar o recibir llamadas.

Además controla el procedimiento de las llamadas, permitiendo comunicación directa entre los puntos finales o actuando como intermediario para transmitir la señalización de llamada.

Deben desarrollar las siguientes funcionalidades:

- Traducción de direcciones: convierte los alias H.323 o las direcciones E164 (teléfonos tradicionales) en direcciones IP de punto final.
- Control de admisiones: da acceso autorizado a H.323 usando los mensajes ARQ/ACF/ARJ
- Control de ancho de banda: administra los requisitos de ancho de banda usando los mensajes BRQ/BCF/BRJ
- Administración de zona: para terminales, gateways y MCU registrados

Además, el gatekeeper ofrece un mecanismo centralizado para administrar planes de conexión y enrutamiento de llamada en una red VoIP. Administrando tanto llamadas como ancho de banda.

Facilitan además el control de llamadas a terceros, esencial en entornos Call Center, y otras aplicaciones especializadas en llamadas.

Gateway

Un gateway H.323 (GW) es un extremo que proporciona comunicaciones bidireccionales entre terminales H.323 y otros terminales o gateways.

Su objetivo es transportar y traducir los formatos necesarios para que los equipos H.323 puedan operar con otras redes.

Traducen señalización, información de control e información de usuario, posibilitando la comunicación entre distintos terminales y permitiendo integrar sus servicios con múltiples plataformas.

Funciones:

- conversión de audio y vídeo
- traducción entre medios cuándo se conecta una red H.323 a otra de distinto tipo.
- Conversión de señalización de llamada y de señalización de medios.

Pueden actuar como terminal H.323 o MCU en la red, o como terminal SCN o MCU en la red de circuito conmutado (SCN)

MCU

La Unidad de Control Multipunto es un punto final que soporta conferencias multipunto entre terminales, llevando además la negociación entre terminales para determinar las capacidades comunes para el procesamiento de audio y vídeo.

Se divide en: un controlador multipunto (MC) y un procesador multipunto (MP).

- Controlador Multipunto (MC): Soporta conferencias entre varios puntos finales. Transmite el conjunto de capacidades de cada punto final en la conferencia. Pueden revisar las capacidades durante la conferencia. Utiliza la señalización H.245 para el control de medios.
- Procesador Multipunto (PM): Envía y recibe flujos de medios hacia y desde los participantes en la conferencia. Puede convertir los medios entre distintos formatos y combinarlos desde múltiples orígenes. Sus funciones dependen de la ubicación y el punto de conferencia.

Protocolos de H.323

Señalización RAS:

Permite un intercambio de mensajes entre el gatekeeper y el punto final usando UDP. Se abre antes de que se establezca ningún otro canal. Es independiente de la señalización de control de llamadas y de los canales de transporte de medios. Los mensajes RAS realizan: registro, admisiones, cambios de ancho de banda, comprobación del estado y procedimientos de finalización.

Existen los siguientes procedimientos:

- *Descubrimiento del gatekeeper:*

Permite que el punto final descubra a su gatekeeper a través de un mensaje multidifusión.

GRQ	Un punto final envía un mensaje con el fin de encontrar un gatekeeper, y de paso proporciona información sobre sí mismo.
GCF	El gatekeeper confirma que el punto final puede proceder con el registro y a su vez proporciona información como identidad, dirección IP y puerto.
GRJ	El gatekeeper puede también indicar que el punto final debería buscar otro gatekeeper distinto. Proporciona identificador y razón de rechazo.

- *Registro y sin registro*

Proceso que permite que los gateways, puntos finales y MCU alcancen una zona e informen al gatekeeper de sus direcciones IP y alias. Se realiza después del proceso de descubrimiento y antes de la realización de ninguna llamada.

Registro:

RRQ	Proporciona información sobre el punto final al gatekeeper
RCF	El GK confirma el registro y da información propia
RRJ	El GK rechaza el registro, indica identificador de GK y motivo de rechazo.

Sin registro:

URQ	El punto final informa al GK de los alias H.323 que dejará de utilizar. También puede informar el GK al punto final de los alias que no debe seguir usando.
UCF	Enviado desde el punto final o el GK para confirmar la cancelación
URJ	Enviado desde el punto final o el GK para informar de un fallo en URQ, normalmente debido a que el punto final no estaba registrado con el GK.

- *Localización de punto final.*

Los usan los puntos finales y GK para obtener información de contacto cuándo sólo está disponible la información de alias.

LRQ	El punto final lo envía para solicitar información al GK sobre una o varias direcciones E164
LCF	Indica una resolución con éxito de los alias H.323. Contiene la dirección del canal propia.
LRJ	Indica un fallo para resolver el alias H.323

- *Admisiones y ancho de banda*

Proporcionan las bases para la admisión de llamadas y control de ancho de banda.

ARQ	Se envía cuándo el punto final recibe la petición de una llamada entrante o cuándo quiere ubicar una llamada.
ACF	Autorización de petición de llamadas entrantes o salientes por parte del GK
ARJ	Deniega la petición del punto final para acceder a la red.
BRQ	Enviado por el punto final al GK pidiendo un incremento o disminución en el ancho de banda

BCF	Enviado por el GK confirmando la petición de cambio
BRJ	Enviado por el GK rechazando la petición de cambio, normalmente si el ancho de banda solicitado no está disponible.

- *Información de estado*

Un gatekeeper puede usar un canal RAS para obtener información de estado desde un punto final. Se usa para monitorizar si el punto final está en línea o no.

IRQ	Se envía desde el GK al punto final para solicitar estado
IRR	Se envía desde el punto final al GK como respuesta a IRQ. También se envía desde el punto final si el GK solicita actualizaciones periódicas de estado.
IACK	Reconocimiento de información. Respuesta a IRR negociada durante el registro inicial.
INAK	Reconocimiento negativo de información.

Señalización de control de llamadas H.225.0 (Q.931):

La recomendación H.225.0 de la ITU-T especifica la utilización y soporte de los mensajes de señalización Q.931.

Se usa el puerto 1720 de TCP para inicializar los mensajes Q.931 entre dos puntos finales con el propósito de conectar, mantener y desconectar las llamadas.

En caso de que no haya gatekeeper en una red, la señalización se realiza en los dos extremos de la llamada, asumiendo direcciones conocidas de transporte de señalización. Ademas se creará un canal de control con el protocolo H.245 entre los puntos finales.

Mensajes:

- Setup: Enviado por el punto final para establecer conexión con el GK. Se usa el puerto TCP 1720 de H.225
- Call Proceeding: Enviado por el GK para indicar que se han iniciado los procedimientos de establecimiento de llamada.
- Alerting: Enviado por el GK para avisar que el sonido de llamada se ha iniciado.
- Connect: Enviado por el GK para indicar que se ha respondido a la llamada. Puede contener la dirección de transporte UDP/IP para la señalización de control H.245
- Release Complete: Enviado por el punto final que inicia la desconexión. Indica que la llamada ha sido liberada.
- Facility: Mensaje Q.932 usado para solicitar o confirmar servicios suplementarios.

Control y transporte de medios (H.245):

El canal de control de medios H.245 se establece dinámicamente sobre una conexión TCP fiable. Se basa en los parámetros del mensaje CONNECT (Q.931) del control de llamadas H.225.0

Se establece un canal H.245 para cada llamada entre puntos finales.

Permite el intercambio de capacidades, apertura y cierre de canales lógicos, modos de preferencia y control de los mensajes, así como negociación de funciones.

Sus funcionalidades son:

- Intercambio de capacidades:

Los puntos finales negocian un conjunto común de formatos de medios antes de establecer cualquier sesión con medios.

Indican la capacidad del terminal para transmitir y recibir mensajes de audio, vídeo y datos.

En las redes VoIP, el conjunto de capacidades del códec es una lista de los códecs de audio que soporta.

Entre la información que comparten se encuentra: tipos de carga que soporta RTP, si soporta RSVP, tipos de canales de medios soportados, si soporta retransmisión DTMF, etc.

- Elección de maestro y esclavo:

Es necesario determinar qué punto final es el maestro y qué punto final es el esclavo para una llamada determinada.

La relación establecida estará habilitada durante toda la llamada.

Se usa para resolver conflictos entre puntos finales, como cuándo se solicitan acciones similares a la vez.

El punto final que tenga mayor capacidad de medios afirma la autoridad y se establece como esclavo para así poder tener unos requisitos comunes.

- Retraso de ida y vuelta:

Procedimientos usados para determinar el retraso entre los puntos finales de origen y terminación.

Mide el retraso y verifica si la entidad remota del protocolo H.245 está activa.

- Señalización lógica de canal:

Abre y cierra el canal lógico que transporta la información de audio, vídeo y datos.

El canal se prepara antes de la transmisión real para asegurar que los terminales están preparados y son capaces de recibir y descodificar la información.

Una vez establecida la señalización del canal lógico, se pasa el puerto UDP desde el punto final de terminación al punto final origen.

2.2.3 SIP

El Protocolo de Inicio de Sesión (SIP) es un protocolo de señalización de la capa de aplicación.

Está definido en el RFC 2543

Se usa para establecer, mantener y terminar sesiones multimedia.

Soporta unidifusión y multidifusión, así como llamadas punto a punto y multipunto.

Puede operar junto a otros protocolos de señalización, como H.323.

Posee cabeceras versátiles (se pueden registrar funciones adicionales)

Permite servicios telefónicos avanzados.

Puede establecer y terminar comunicaciones obteniendo las siguiente características: localización de usuario, capacidad de usuario, disponibilidad de usuario, configuración de la llamada, y manejo de la llamada.

Entre las principales características de SIP se encuentran:

- Simplicidad: Uso de mensajes de texto plano que se pueden leer. Uso de formatos estándares como HTTP 1.1 y "mailto". Fácil integración y resolución de problemas.
- Eficiencia: Las funciones de señalización consumen poco ancho de banda en relación con el flujo de medios. La información del establecimiento de llamada se incluye en el mensaje inicial.
- Escalabilidad: Los servidores no mantienen la información de estado de las sesiones SIP, por lo que un servidor puede manejar muchos clientes. SIP detecta y previene los bucles de enrutamiento, lo que aumenta el rendimiento en redes extensas.
- Flexibilidad: Como SIP usa SDP para negociar los códecs, se puede usar cualquiera de los registrados por la IANA, mientras que en H.323 están definidos de forma explícita.
- Soporte de movilidad: SIP está dirigido a los usuarios que se mueven de un terminal a otro. Proporciona soporte para proxying y redirección, los usuarios pueden ocultar o proporcionar su verdadera ubicación.
- Programación del usuario: Además de las funciones de telefonía tradicional, SIP puede usar el lenguaje de procesamiento de llamada (CPL) que permite que los usuarios proporcionen reglas complejas a un servidor.
- Extensión: Creación de una arquitectura modular y flexible, mejorando el incremento y las extensiones de protocolo. También permite retirar las opciones de protocolo no usadas.

Componentes del sistema SIP:

Agentes de usuario (UA):

Son aplicaciones de punto final que envían y reciben peticiones SIP.

Cada usuario puede tener varios agente de usuario. Se asocia una dirección SIP con cada agente.

Los agentes de usuario se dividen en:

- cliente de usuario-agente (UAC): Inicia las peticiones SIP y actúa como agente usuario del llamante
- servidor de usuario-agente (UAS): Recibe las peticiones y devuelve las respuestas en nombre del usuario. Actúa como agente de usuario llamado.

Servidores:

- Servidor proxy:

Actúa en nombre de otros clientes y contiene funciones de cliente y de servidor.

Interpreta y puede reescribir cabeceras de peticiones antes de pasarlas a servidores, para ello se identifica al proxy como iniciador de la petición, ésto asegura que las respuestas sigan la misma ruta de vuelta.

Reciben peticiones SIP e inician nuevas peticiones hacia los agentes de usuario de destino. Pueden tener un reconocimiento local de los agentes de usuario desde un registrador SIP.

Pueden usar varias alternativas para localizar a un agente de usuario, de forma paralela o secuencial.

Pueden proporcionar una dirección de redirección para los clientes solicitantes en lugar de reenviar la solicitud SIP.

- Servidor de redirección:

Acepta peticiones SIP y envía una respuesta al cliente con la dirección del siguiente servidor.

No acepta llamadas ni procesa o reenvía peticiones SIP.

La principal diferencia es que el servidor proxy queda formando parte del camino entre el UAC y el (o los) UAS, mientras que el servidor de redirección una vez indica al UAC cómo encaminar el mensaje ya no vuelve a intervenir.

Un mismo servidor puede actuar como redirector o como proxy dependiendo de la situación.

- Servidor registrador:

Acepta registros de clientes que indican las direcciones de localización y guarda dicha información para suministrar un servicio de localización y traducción de direcciones.

Se combinan con un proxy o con un servidor de redirección.

Actúan como un proceso lógico separado, y están fuera del ámbito de SIP.

Direccionamiento

Las direcciones SIP son localizadores universales de recursos SIP (URL SIP).

Sólo los usuarios y agentes de usuario tienen direcciones SIP. Los servidores SIP se identifican por una dirección IP y los puertos TCP/UDP, usando normalmente 5060.

El formato viene definido en RFC 2369 y RFC 2543, y se expresa como:

```
usuario@host
```

Dónde el usuario puede ser un nombre de usuario o un número de teléfono, y el host un nombre de dominio o una dirección de red.

El formato de una URL SIP básica es:

```
“sip: ”[user[“:”password]”@”](hostname|IP_address)[:port]
```

Unos ejemplos serían: `sip:bob:secret@company.com:5060`
`sip:company.com`

Además, una URL SIP también puede incluir parámetros e información de cabecera, añadida al URL SIP básico:

```
[“;”param”=”value][“?”1st_header”=”value[“&”other_headers”=”value]]
```

Ejemplos: `sip:bob@company.com;transport=udp`
`sip:bob@company.com;transport=tcp;user=ip?subject=test%20TCP%20call`

Además, SIP proporciona servicio de traducción de direcciones para E.164, y proporciona soporte para movilidad actualizando las direcciones SIP de dispositivos genéricos, para ello el registrador SIP se comunica con el servidor proxy para que cualquier petición que llegue a una dirección SIP concreta se desvíe al dispositivo correspondiente.

Localización de servidor y usuario.

Localización de un servidor:

Para localizar un servidor por parte de un cliente es necesario enviar una petición SIP.

Pueden darse dos casos:

- Enviar la petición a un servidor proxy configurado localmente:

  Es una solución fácil, ya que la aplicación de sistema final conoce al servidor proxy.

- Enviar la petición a la dirección IP y número de puerto de la URL SIP.

  Para ello, el cliente tiene que determinar la dirección IP y el número de puerto del servidor al que va destinada la petición.
  Si el número de puerto no está definido en la URL SIP se toma por defecto el 5060.
  Si el protocolo no está definido, el cliente intenta conectar primero con UDP y luego con TCP.
  Además, el cliente consulta al servidor DNS para obtener la dirección IP, si no la encuentra el servidor DNS no se puede conectar con el servidor buscado.

Localización de un usuario:

La parte llamada puede desplazarse a varios sistemas finales a lo largo del tiempo, como un cambio de LAN en la empresa, una conexión de casa, etc.

La localización del sistema final es posible si:

- Está registrada con el servidor SIP
- Está registrada con servidores de localización fuera del ámbito de SIP. El servidor SIP almacena la lista de localizaciones basadas en el servidor exterior que está devolviendo múltiples posibilidades de host.

El servidor de redirección devuelve la lista completa de localizaciones y permite que el cliente localice directamente al usuario.
El proxy, puede probar las direcciones en paralelo hasta que la llamada tenga éxito.

Estructura del mensaje.

Un mensaje SIP se compone de tres partes:

- Línea de inicio: indica el propósito del mensaje
- Cabeceras: proporcionan los detalles del mensaje
- Cuerpo (opcional): suministra detalles añadidos que no encajan en las cabeceras

Línea de inicio:

* Solicitudes SIP: `<Método><Solicitud-URI><Versión SIP>`

Se envían de cliente a servidor.

- Método: permiten que los agentes de usuarios y servidores de red localicen, inviten y administren llamadas

Tipos:

- INVITE: usuario o servicio invitado a participar en una sesión
- ACK: confirmación final, termina la operación iniciada por INVITE
- OPTIONS: permite consultar y reunir posibilidades de agentes de usuario y servidores
- BYE: se usa por las partes llamantes y llamadas para liberar una llamada
- CANCEL: permite que los agentes de usuario y servidores de red cancelen cualquier petición que esté en progreso
- REGISTER: utilizado por los clientes para registrar información de localización con los servidores SIP

- Solicitud-URI: URL SIP del receptor, ya sea un servidor proxy o UAS

- Versión SIP: SIP/2.0

* Respuestas SIP: `<Versión SIP><Código de estado><Razón>`

Los servidores contestan a las solicitudes de un cliente con una o más respuestas.

- Versión SIP: SIP/2.0

- Código de estado:

Tipos:
- 1XX: Informational
- 2XX: Success
- 3XX: Redirection
- 4XX: Request Failure
- 5XX: Server Failure
- 6XX: Global Failures

Los códigos de respuesta SIP del rango 1XX son respuestas provisionales, por tanto el servidor producirá respuestas adicionales a la solicitud del cliente.

Si un cliente recibe un código de respuesta no reconocido, negocia la respuesta como una respuesta X00 para una categoría determinada.

La categoría 6XX distingue entre las respuestas de error que pueden producirse en cualquier servidor y las respuestas de error únicas para uno en concreto.

Asignación de rangos, cualquier código con formato menor que X80 será identificado como HTTP 1.1, los rangos X80 a X99 serán identificados como propios de SIP

- Razón: Frase asociada al código

Cabeceras SIP:

```
<nombre cabecera>:<valor cabecera>
<continuación de valor de cabecera>
```

Las cabeceras SIP representan un valor variable que se transporta a través de la red. Especifican la parte llamante, la parte llamada, la ruta y el tipo de mensaje.

Hay cabeceras obligatorias y otras que no lo son.

El orden de aparición no es importante, sin embargo aquellas que se deben procesar en los servidores proxy deberían aparecer antes que las cabeceras para los agentes de usuario.

Los tipos de cabeceras se organizan en cuatro grupos:

- Cabeceras generales: Accept, Accept-Encoding, Accept-Language, Call-ID, Contact, CSeq, Date, Encription, Expires, From, Record-Route, Timestamp, To, Via

    - Call-ID: sólo identifica una invitación específica o todos los registros de un cliente determinado
    - Expires: identifica la fecha y hora a la que expira el contenido del mensaje
    - From: indica quién ha iniciado la petición
    - To: indica el receptor de la petición
    - Via: indica la ruta tomada por una petición

- Cabeceras de entidad: Content-Encoding, Content-Length, Content-Type

    - Content-Length: indica el tamaño del cuerpo del mensaje en octetos
    - Content-Type indica el tipo medio del cuerpo del mensaje

- Cabeceras de petición: Authorization, Contact, Hide, Max-Forwards, Organization, Priority, Priority-Authorization, Proxy-Require, Route, Require, Response-Key, Subject, User-Agent

    - Route: indica la ruta tomada por una petición
    - Subject: describe la naturaleza y tema de la llamada

- Cabeceras de respuesta: Allow, Proxy-Authenticate, Retry-After, Server, Unsupported, Warning, WWW-Authenticate

Cuerpo de los mensajes SIP:

Los contenidos del cuerpo de un mensaje varían dependiendo del tipo de solicitud o respuesta SIP.

- Solicitudes SIP:
    - método BYE: no incluye cuerpo
    - métodos INVITE, ACK y OPTIONS: codifican el cuerpo de acuerdo a SDP

- Respuestas SIP:
  - rango 1XX: puede incluir información de advertencia sobre la solicitud.
  - rango 2XX: descripción de la sesión que se ha negociado
  - rango 3XX: información sobre los servicios alternativos o los destinos para completar la solicitud.
  - rangos 4XX a 6XX: pueden contener información legible sobre errores.

Protocolo de descripción de la sesión SDP: Está diseñado para identificar completamente todos los atributos de una sesión (información administrativa, sobre el programa y sobre los medios). El orden de los atributos SDP está estrictamente especificado, se basa en minimizar el tamaño y complejidad del analizador sintáctico del protocolo.

2.2.4 IAX

El protocolo IAX (Inter-Asterisk eXchange protocol) fue diseñado como un protocolo de conexiones VoIP entre servidores Asterisk.

Hoy en día, además sirve para conexiones entre clientes y servidores que soporten el protocolo.

Actualmente la versión original ha quedado obsoleta en favor de la versión IAX2, que toma el nombre del protocolo para referirse a ella.

Se trata de un protocolo robusto, y simple si se compara con otros protocolos. Permite un manejo de un gran número de códecs y de streams, por lo que puede ser utilizado para transportar cualquier tipo de dato. Por lo tanto, resulta especialmente útil para videoconferencias.

Usa el puerto UDP 4569 para comunicación entre puntos finales, para señalización y datos. El uso de UDP evita los problemas relacionados con NAT, ya que tanto la información de señalización como los datos viajan conjuntamente (a diferencia de SIP), y por tanto lo hace más robusto frente a problemas con NAT.

Es prácticamente transparente a los cortafuegos.

Soporta además trunking, permitiendo que un sólo enlace pueda enviar datos y señalización por múltiples canales. Esto proporciona que los paquetes IP puedan viajar sin retrasos sustanciales.

Los objetivos de IAX son:

- Minimizar el ancho de banda usado en las transmisiones de control y multimedia de VoIP
- Evitar problemas de NAT
- Proporcionar un soporte para transmitir planes de marcación.

Mensajes IAX

El protocolo IAX tiene cuatro fases en cuánto a la gestión de una llamada. Son las siguientes:

Registro de llamada:

Para que dos terminales puedan comunicarse es necesario un registro previo. Para ello deben conocer la dirección de red del otro terminal. IAX2 permite que un terminal registre su dirección y datos necesarios con el otro terminal que hace las funciones de servidor de registro.

El registro comienza con una solicitud de registro REGREQ, que es contestada por una autenticación REGAUTH. Una vez autenticado es necesario confirmar el registro con el comando REGACK, que es respondido con un ACK para informar que el registro se ha cursado con éxito.

Establecimiento de llamada:

El terminal A inicia una conexión y manda un mensaje NEW.

El terminal llamado responde con un ACCEPT y el llamante le responde con un ACK.

A continuación el terminal llamado da las señales de RINGING y el llamante contesta con un ACK para confirmar la recepción del mensaje.

Por último, el llamado acepta la llamada con un ANSWER y el llamante confirma ese mensaje.

Flujo de datos o flujo de audio:

Se mandan las tramas M y F en ambos sentidos con la información vocal.

Las tramas M son mini-tramas que contienen solo una cabecera de 4 bytes para reducir el uso en el ancho de banda.

Las tramas F son tramas completas que incluyen información de sincronización.

Es importante volver a resaltar que en IAX este flujo utiliza el mismo protocolo UDP que usan los mensajes de señalización evitando problemas de NAT.

Liberación de la llamada o desconexión:

La liberación de la conexión es tan sencilla como enviar un mensaje de HANGUP y confirmar dicho mensaje con un ACK.

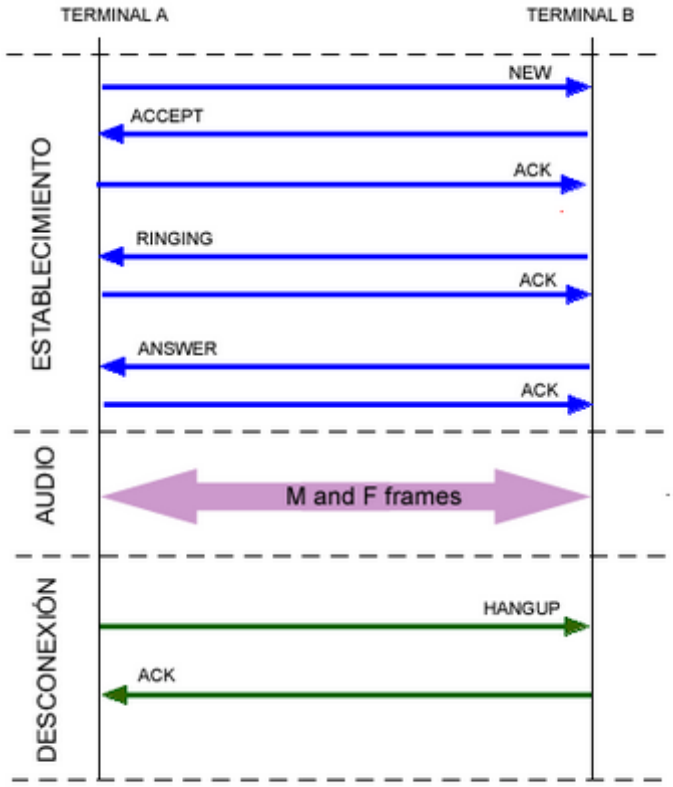


Figura 5: "Intercambio de mensajes IAX"

Tipos de tramas:

Los mensajes o tramas que se envían en IAX2 son binarios y por tanto cada bit o conjunto de bits tiene un significado.

Como se ha indicado anteriormente existen dos tipos de mensajes:

Tramas F o Full frames:

La particularidad de las tramas o mensajes F es que deben ser respondidas explícitamente.

Es decir, cuándo un usuario manda a otro una trama F (full frame) el receptor debe contestar confirmando que ha recibido ese mensaje.

Estas tramas son las únicas que deben ser respondidas obligatoriamente.

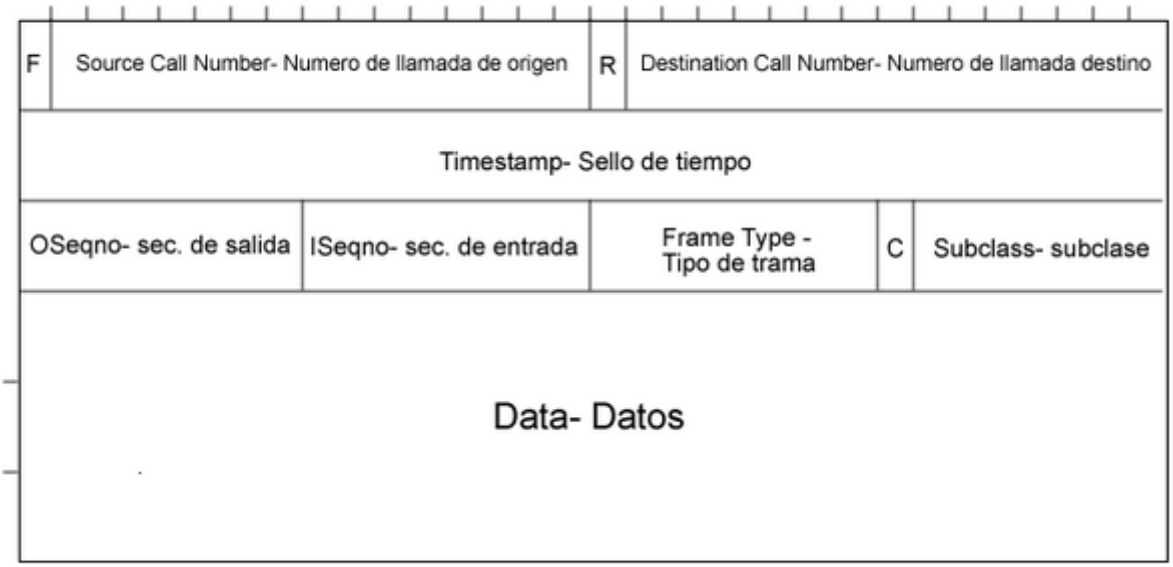


Figura 6: "Formato de trama F"

El significado de cada uno de los campos es:

- F: bit que indica si la trama es F o no. En caso de que sea F o full frame contendrá 1.
- Source Call Number: Número de llamada de origen. 15 bits que identifican la conversación de origen, ya que pueden haber varias multiplexadas por la misma línea.
- R: bit de retransmisión. Contiene 1 cuándo la trama es retransmitida.
- Destination Call Number: Número de llamada de destino. Identifica el destino de la llamada.
- Timestamp: Sello de tiempo. Marca el tiempo en cada paquete
- OSeqno: Secuencia de salida. Número de secuencia de salida con 8 bits. Comienza en 0 y se incrementa en cada mensaje.
- ISeqno: Secuencia de entrada.
- Frame Type: Indica qué clase de trama es.
- C: Puesto a 0 indica que el campo subclase debe tomarse como 7 bits (un solo mensaje). Puesto a 1 indica que el campo subclase se obtiene con 14 bits (dos mensajes consecutivos)
- Subclass: Subclase del mensaje
- Data: Datos que se envían en formato binario.

Tramas M o Mini Frames:

Las tramas M o mini frames se usan para mandar la información con la menor cabecera posible.

Estas tramas no tienen porqué ser respondidas y si alguna de ellas se pierde se descarta sin más.

El formato es el siguiente:

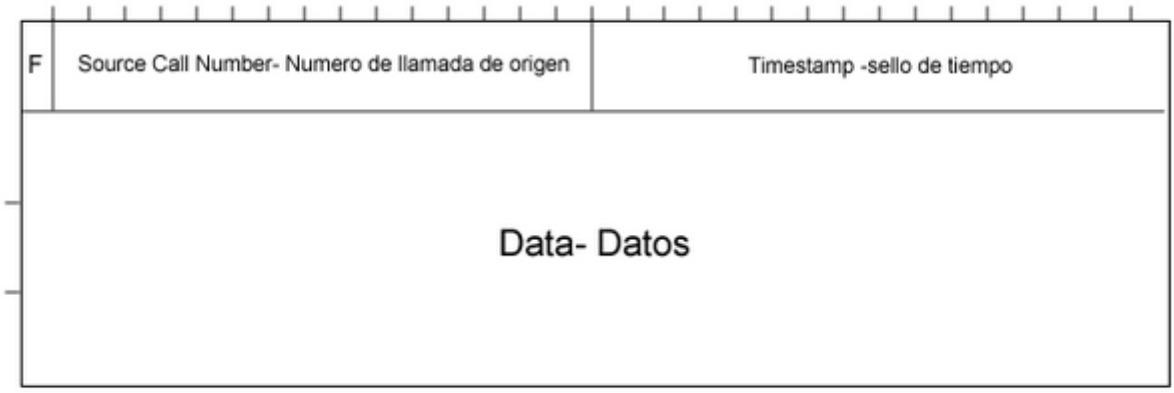


Figura 7: "Formato de trama M"

El significado de los campos es similar al de las tramas F o full frame.

- F: El bit contiene el valor 0
- Source Call Number: Número de llamada de origen. Identifican la conversación de origen. 15 bits.
- Timestramp: Sello de tiempo. Truncado en 16 bits para aligerar la cabecera. Son los clientes los que deben encargarse de llevar un timestamp de 32 bits si lo desean y para sincronizarlo mandar una trama F.

2.3 Calidad

Retraso, latencia:

Se denomina retraso o latencia en VoIP al tiempo que tarda la voz en salir de la boca del que está hablando y el tiempo en llegar al oído del que está escuchando.

La recomendación G.114 de la ITU-T define que para que exista una buena calidad de voz no debe darse un retraso mayor de 150 ms en un sentido y de extremo a extremo. Por lo tanto, para que exista una buena calidad de voz la suma de los retrasos tiene que ser menor a 150 ms.

Existen tres principales causas de retraso en las redes telefónicas:

Retraso de propagación:

El retraso de propagación depende de la velocidad de la luz en un medio determinado, como fibra óptica o cable de cobre.

La velocidad de la luz en el vacío es de 300.000 Km/s, sin embargo, tanto en cobre como en fibra óptica puede alcanzar unos 200.000 Km/s.

Suponiendo que exista una red que viaje alrededor del mundo (21.000 Km), este retraso de sentido único sería de 70 ms, y por lo tanto prácticamente imperceptible para el ser humano.

Retraso de manejo:

El retraso de manejo está provocado por los dispositivos durante el envío de la trama.

El retraso de manejo atiende a distintas causas, empaquetado, compresión y switching de paquetes.

Es un gran problema en entornos de paquetes.

El procesador digital de señal (DSP) genera una muestra cada 10 ms si se usa el códec G.729.

Retraso en gestión de colas o serialización:

Se define el retraso de serialización como el tiempo que se tarda en colocar un bit en una interfaz. O también como el tiempo que se necesita para mover un paquete hasta la cola de salida.

En la gestión de colas, el retraso puede referirse al envío de más paquetes que puede manejar la interfaz en un momento dado.

Variación de retardo (jitter):

Es la variación del tiempo de llegada de los paquetes al receptor. Depende del retraso sufrido en la red, por lo tanto es variable.

La calidad de voz es totalmente dependiente de la temporización con la que las muestras de sonido se ejecutan en el oyente.

Es un retraso exclusivo de las redes de conmutación de paquetes.

Si la temporización entre tramas en el emisor es distinta que en el receptor, éste no oirá una representación exacta del sonido original.

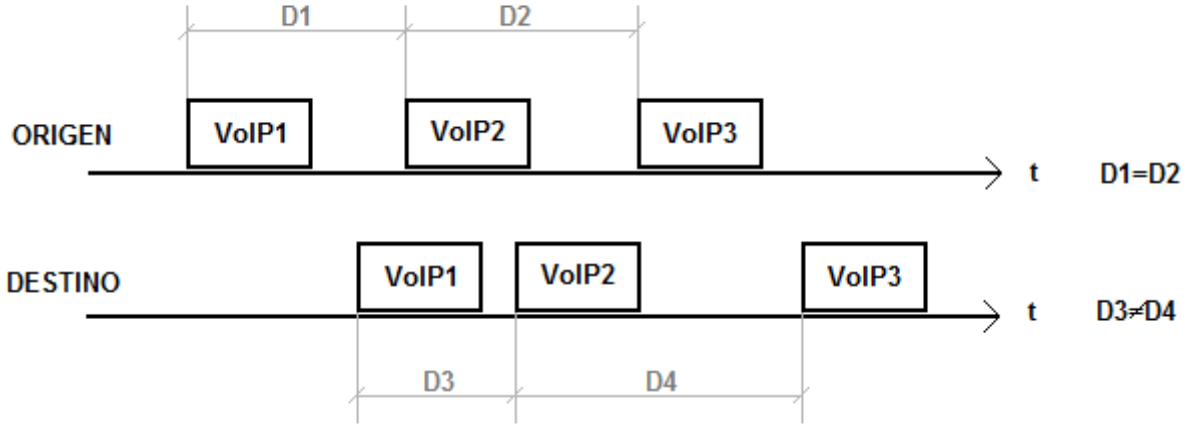


Figura 8: "Variación de retardo (jitter)"

Para solucionar los problemas de la fluctuación de fase es necesario usar búferes de reproducción, los cuáles almacenan un suministro de tramas códec por un breve tiempo para asegurarse que puede ejecutar las tramas a una velocidad constante.

El número de tramas almacenadas es variable, disminuye cuándo no llega ningún paquete de voz desde la red, y aumenta cuándo llega una ráfaga de paquetes de voz.

La velocidad de vaciado del búfer es constante.

Ya que los búferes de reproducción sólo pueden eliminar una mínima cantidad de fluctuación de fase en los sistemas de telefonía por paquetes, se debe diseñar una red que presente muy poca fluctuación de fase para los paquetes en tránsito.

Existen distintas estrategias para minimizar y gestionar la fluctuación de fase, como priorizar el tráfico de voz por delante de otros tráficos de datos, disminuir el tamaño máximo de paquete para minimizar el retraso por serialización variable, evitar rutas de transmisión paralelas con distintos elementos de retraso, o aumentar la longitud de los búferes de reproducción para alojar la alta variación de retraso.

Es preferible un sistema con un retraso global alto que uno con una fluctuación de fase en un flujo de audio.

Compresión de voz:

Existen dos métodos de codificación de la voz por forma de ondas: PCM y ADPCM

PCM de 64 Kbps:

Contiene en forma de leyes la forma de codificación:

- ley μ: Proporciona un mejora de rendimiento en relación señal/ruido.
  Es usada en América del Norte
  Para llamadas de larga distancia que requieren un cambio entre la ley μ y la ley α, es responsabilidad del país de ley μ.
- ley α: Es usada normalmente en Europa

Ambas se diferencian en detalles de compresión relativamente menores. Usan compresión logarítmica para alcanzar de 12 a 13 bits de calidad PCM lineal en 8 bits.

ADPDM:

La codificación Adaptative Differential Pulse Code Modulation (ADPCM) está definida en el ITU-T G726.

Realiza una codificación usando muestras de 4 bits.

Tiene una velocidad de transmisión de 32 Kbps.

Mide las diferencias de amplitud y la velocidad de cambio de la amplitud.

Usa una predicción lineal rudimentaria.

Además de estos dos tipos, existen nuevas técnicas desarrolladas en los últimos 15 ó 20 años que realizan una codificación basada en la vibración y la modulación.

Los procedimientos de procesamiento de señales que comprimen la voz se basan en un envío de información paramétrica simplificada (vibración y modulación de la voz original).

Necesitan menos ancho de banda para transmitir información.

Existen una serie de códecs que usan este sistema, como son LPC, CELP y MP-MLP.

Entre las normas existentes caben destacar:

- G.711: PCM de 64 Kbps
- G.726: ADPCM de 40, 32, 24 y 16 Kbps
- G.728: CELP de 16 Kbps
- G.729: CELP de 8 Kbps
- G.723.1: CELP de 5'3 Kbps y MP-MLQ de 6'3 Kbps

Eco:

En una red tradicional de telefonía, el eco está provocado por un desajuste en la impedancia de la conversión del switch de red de cuatro cables al bucle local de dos cables.

Para solucionar este problema, la Red Pública de Telefonía Conmutada (PSTN) hace uso de:

- canceladores de eco: tapan la impedancia en un circuito. No es la mejor forma de suprimir el eco, ya que provoca otros problemas, como que no se pueda usar la RDSI porque se corta el radio de acción de su frecuencia.
- control de desajustes de la impedancia en los puntos de reflexión común.

En las redes basadas en paquetes se pueden construir canceladores de eco en códecs de velocidad de transmisión baja, y hacerlos funcionar en cada DSP.

Están limitados por la cantidad de tiempo que esperan a que llegue la palabra reflejada (echo tails). Cisco tiene echo tails configurables de 16, 24 y 32 ms, es importante configurarlos correctamente, ya que una configuración insuficiente provocaría eco, y una configuración exagerada aumentaría el tiempo de conversión del cancelador de eco.

La ubicación de estos canceladores de eco varían en función del fabricante, mientras que en algunos está situada en el software, los VoIP de Cisco introducen toda la cancelación en el DSP.

Para eliminar el eco, el usuario transmisor guarda una copia inversa de la conversación durante un tiempo determinado. El cancelador de eco oye el sonido que viene del receptor y sustrae la ganancia para eliminar el eco.

Pérdida de paquetes:

En las redes de datos una pérdida de paquetes es común y esperada, además de una medida para conocer las condiciones de la red y poder reducir el número de paquetes que se están enviando.

Sin embargo, una pérdida de paquetes de voz es algo que merece la pena mejorar. Es importante construir una red que transporte con éxito la voz de manera fiable y oportuna.

Es necesario crear un mecanismo para hacer que la voz sea resistente a la pérdida periódica de paquetes.

Cisco System ha desarrollado herramientas de calidad de servicio que permiten a los administradores clasificar y administrar el tráfico.

La implementación G.729 de VoIP permite al router de voz responder a la pérdida periódica de paquetes. Si un paquete de voz no es recibido cuándo se esperaba, la estación receptora espera un periodo de tiempo (búfer de fluctuación de fase) y ejecuta una estrategia de ocultación. Es decir, el receptor vuelve a repetir el último paquete recibido, por lo que el oyente no aprecia el silencio. Sólo se puede usar para un único paquete y el grado de tolerancia a la pérdida de paquetes es de un 5%.

Para terminar cabe destacar qué problemas de calidad son difíciles de minimizar y qué problemas son minimizables con una planificación y diseño sólidos.

- Difíciles de minimizar:	compresión-descompresión de la trama de voz
	retraso de propagación
- Minimizables:	fluctuación de fase
	latencia
	retraso de manejo
	velocidades de muestreo
	codificación tándem
	diseño del plan de marcación

2.4 Ventajas e inconvenientes:

Ventajas:

Las ventajas de VoIP frente a la telefonía tradicional se pueden dividir en los siguientes rasgos principales:

Ahorro económico:

El ahorro puede variar dependiendo de la zona geográfica. Por ejemplo en países que no pertenecen a Estados Unidos, la comparación del coste minuto a minuto entre VoIP y PSTN justifica de sobra el gasto del uso de una nueva red.

También es necesario considerar el ahorro en mantenimiento que supone una red así. Una sola persona puede controlar sin problemas una red amplia.

Mejora de la funcionalidad:

Una infraestructura IP requiere menor número de añadidos, menos desplazamientos que una PSTN y menos cambios. Ya que se pueden usar alternativas como el Protocolo de Configuración Dinámica del Host (DHCP), que permite que un dispositivo reciba dinámicamente una dirección IP, por tanto un teléfono configurado con DHCP se puede llevar a cualquier sitio y mantiene el número de teléfono.

Nuevas aplicaciones:

El mejor aprovechamiento de las nuevas tecnologías, hace que VoIP pueda ofrecer servicios que para PSTN están limitados.

La PSTN basa su funcionamiento en redes manejadas por centrales telefónicas y conmutadores, que aunque aportan una gran fiabilidad apenas admiten ampliaciones de funciones.

VoIP, al usar servidores y routers que están en constante desarrollo y que además se adaptan con facilidad a los cambios da como resultado que el transporte de información resulte más barato y más rápido que en una red tradicional.

Interoperatibilidad:

Compatibilizar el uso de una tecnología entre distintos fabricantes ha supuesto un gran avance en cuánto a la propagación de VoIP.

Para ello se han estandarizado protocolos y se han creado normas de uso que permiten la interconexión entre distintos fabricantes.

Integración:

VoIP permite integrarse con distintos servicios y aplicaciones que hay actualmente en el mercado, y además permite integrar cualquier programación privada y personal.

Inconvenientes:

El principal problema de la voz sobre IP es la calidad de servicio que ofrece.

Las pérdidas de paquetes de información y el retardo entre ellos hacen de VoIP una mala alternativa en cuánto a calidad en las llamadas de audio y vídeo.

Aunque mejora la velocidad de transmisión, la fiabilidad de la llegada de los paquetes está bastante limitada.

De momento su escenario es una red privada y no muy amplia, ya que los niveles de calidad empeoran con el número de usuarios y participantes de los servicios.

Las posibles soluciones se basan en hacer una diferenciación de paquetes de voz frente a los de datos, priorizar los paquetes de voz y disminuir los retardos de transmisión.

Las propuestas actuales de mejora de la calidad de servicio se basan en:

- Uso del protocolo de reserva de recursos (RSVP):

Permite a los routers que destinen dinámicamente anchos de banda para flujos específicos que necesitan un servicio especial. RFC 2205

- Políticas de encolamiento:

Gestionan las prioridades en las colas mediante la asignación de un ancho de banda disponible, una prioridad a las colas y al tráfico y un control de enrutamiento.

- Compresión de cabeceras mediante el protocolo CRTP:

Aumentan el ancho de banda disponible para datos. Se efectúa la compresión en la capa de enlace. Cada interfaz debe tener esta función activa para evitar errores y atascos de las tramas.

- Fragmentación de tramas e intercalado:

Realizada en la capa de enlace permite fragmentar paquetes grandes para que un paquete VoIP no tenga que esperar un tiempo demasiado alto. Se intercalan paquetes VoIP entre los fragmentos del paquete grande.

Capítulo 3:
SOLUCIÓN PROPUESTA

Éste capítulo expone de forma detallada la solución implementada para cumplir los objetivos planteados en el proyecto.

Haciendo referencia al proyecto final de carrera de Federico Arroyo Almagro con título "Estudio, Evaluación, Implantación y Configuración de Aplicaciones VoIP. Software Libre. Casos Prácticos", se ha hecho un estudio posterior con una ampliación de las funcionalidades de la centralita Asterisk y su forma de uso.

Basándose en dicho proyecto, éste amplía funcionalidades como llamada a grupos, uso de respuesta interactiva de voz (IVR), parqueamiento de llamadas y transferencia, uso de archivos callfiles externos y expansión y uso a través de Dundi.

Para llevar a cabo los objetivos se ha optado por la instalación de las siguientes aplicaciones:

- Centralita software: Asterisk PBX versión 1.8.4
- Softphones: Ekiga (protocolo SIP) y Kiax (protocolo IAX)

Todo el software utilizado es libre y funciona bajo el sistema operativo GNU/Linux.

Se ha elegido Asterisk como la centralita software PBX debido a las múltiples ventajas que ofrece, que son:

– Flexibilidad de código: ofrece posibilidades de programación y de creación de códigos independientes que permitirán desarrollar las aplicaciones deseadas, permitiendo así una versatilidad de la que no disponen otras centralitas del mercado como pueden ser OpenPBX, YATE o Freeswicht.

– Interoperabilidad: Asterisk incorpora además la mayoría de los estándares de telefonía del mercado tanto tradicionales como de Voz sobre IP, permitiendo la integración con las centralitas tradicionales.

– Multiplataforma: A pesar de que está pensada para su uso en Linux también permite su implantación en otros sistemas operativos como Windows o MacOS. Aunque será en Linux dónde más posibilidades de desarrollo se obtengan.

– Funcionalidad: Incorpora todas las funcionalidades de las centralitas que se ofrecen en el mercado, desde las más simples a las más avanzadas. Como pueden ser uso de buzones de voz, llamadas de respuesta interactiva, conferencias múltiples, etc.

– Coste: Por el hecho de ser un sistema Open Source y con unos requisitos mínimos de hardware muy fáciles de conseguir, el coste de la centralita es el más barato de todas las opciones del mercado.

Los softphones elegidos han sido Ekiga y Kiax, que vienen instalados por defecto en la distribución de Linux OpenSUSE 11.0. Éstos dos clientes utilizan los protocolos SIP e IAX respectivamente y tienen una gran compatibilidad con Asterisk

3.1 Servidor VoIP. Centralita Asterisk

Asterisk es un programa de software libre y de código abierto para la creación de aplicaciones de comunicaciones.

Hace uso de un ordenador para crear un servidor, ofreciendo funcionalidades como una centralita o PBX, un gateway VoIP, un servidor de conferencias, respuesta de voz interactiva y muchas más.

Es posible usar determinadas interfaces que permitan conectar teléfonos analógicos (FXS) o conectarse a la red telefónica tradicional (FXO) o la RDSI (BRI/PRI) de forma transparente

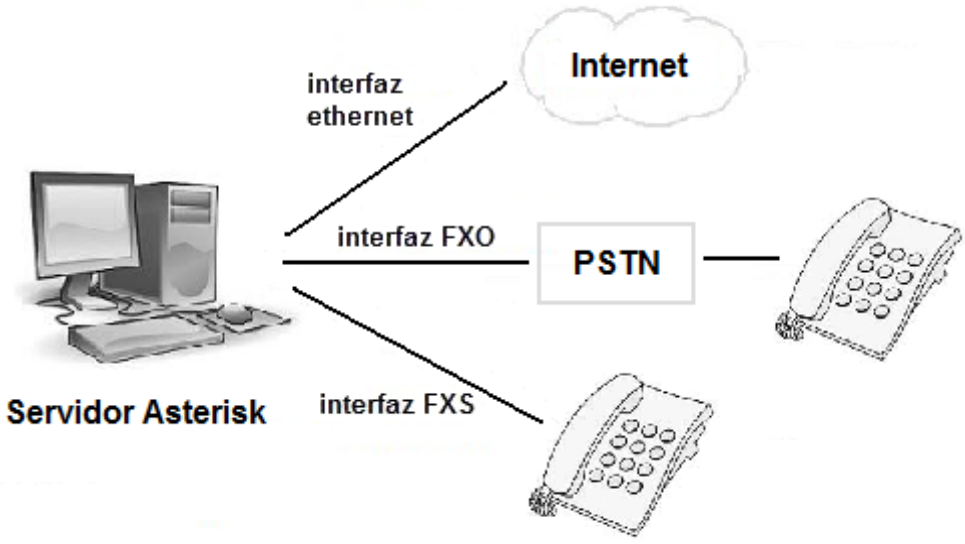


Figura 9: “Conexión de interfaces con Asterisk”

La facilidad de uso y de modificación hacen que Asterisk sea una de las principales opciones a la hora de elegir una centralita telefónica. Cualquier usuario puede crear nuevas funcionalidades escribiendo un dialplan en lenguaje script de Asterisk o añadiendo módulos escritos en C o cualquier lenguaje soportado por Linux.

Las ventajas que pueden hacer que los usuarios finales elijan Asterisk por encima de otras centralitas telefónicas son su facilidad para controlar y gestionar comunicaciones de cualquier tipo, ya sean analógicas, digitales o VoIP; además de la versatilidad, la seguridad y la calidad que ofrece.

Ésta flexibilidad que ofrece Asterisk se debe al funcionamiento modular que posee. Los módulos o APIs permiten una mayor abstracción entre las funciones básicas e integran fácilmente nuevas funcionalidades sin que el núcleo de la centralita tenga que preocuparse de los detalles propios de ese módulo.

Los módulos de funcionamiento de Asterisk están compuestos por:

- API de canal: maneja el tipo de conexión al cual el cliente está conectado (VoIP, RDSI, PRI o cualquier otro tipo de tecnología)
- API de aplicaciones: permite a los módulos cumplir varias funciones como conferencias, uso del buzón de voz durante la llamada, etc.
- API del formato de archivos: maneja la lectura y escritura de varios formatos de archivos para almacenarlos.
- API de traducción de códecs: se encarga de traducir los códecs para que la comunicación se pueda llevar a cabo

Aunque la idea principal es hacer uso de terminales VoIP software por la facilidad de uso y el bajo coste que tienen, también es posible conectar teléfonos analógicos a la red digital que gestiona Asterisk, para ello serán necesarios adaptadores ATA o interfaces FXS que convierten y comprimen la señal a la vez en paquetes de datos.

Dichas tarjetas o adaptadores son fabricadas por Digium o por otras empresas cuyo hardware de conexión con Asterisk es totalmente compatible.

Fue Mark Spencer de la compañía Digium quién creó Asterisk y quien sigue contribuyendo junto a otros programadores a corregir errores y añadir nuevas funcionalidades.

Originalmente se desarrolló para el sistema operativo GNU/Linux, aunque en la actualidad se ofrece también para los sistemas operativos BSD, Mac OS X, Solaris y Windows en versiones anteriores y con peor soporte.

Otro de los factores determinantes a la hora de elegir Asterisk es la versatilidad de protocolos que ofrece, ya que se puede hacer uso de SIP, H.323, IAX y MGCP.

Entre los códecs de audio nos encontramos con muchas opciones, como pueden ser: G.729, GSM, ILBC/Speech, G.722/G.723, G.711a/G.711u.

De la página de Asterisk (http://www.asterisk.org/) se puede descargar de forma gratuita la última versión de la centralita y de los módulos que la acompañan. La versión estable de Asterisk está formada por los siguientes módulos:

– **DAHDI (Digium Asterisk Hardware Device Interface)**: contiene una estructura de drivers de código libre que definen la interfaz de comunicación entre la aplicación y el hardware telefónico. Anteriormente conocido como Zaptel.

– **LibPRI**: librería que encapsula los protocolos utilizados para las comunicaciones sobre interfaces digitales (T1, E1, J1). Es una dependencia de Asterisk si la señalización PRI está activada

– **LibSS7:** es una librería de código libre que se usa para proveer los servicios de protocolo SS7 a aplicaciones como Asterisk. Usa MTP2, MTP3 e ISUP para ITU y ANSI. Es una dependencia de Asterisk si la señalización SS7 está activada.

– **Sounds**: por defecto, Asterisk incluye una recopilación de fases pregrabadas en inglés usando el códecs GSM. Éste paquete incluye frases en otros idiomas y grabadas con otros códecs distintos.

– AsteriskNOW: es la primera distribución "ready-to-run" de Asterisk. Se trata de una imagen ISO que permite instalar Linux, Asterisk y una interfaz gráfica de usuario con el objetivo de configurar una centralita PBX lo más completa posible de una forma sencilla y rápida

La configuración de Asterisk se realiza a través de la edición de archivos de texto o mediante la línea de comandos CLI, no dispone de una interfaz gráfica que permita una configuración más directa. Sin embargo, existen interfaces gráficas desarrolladas por la comunidad Asterisk.

3.2 Softphones: Ekiga y Kiax.

Ekiga

Figura 10: "Softphone Ekiga"

Ekiga, anteriormente conocido como GnomeMeeting es un softphone de código abierto que permite conferencias y mensajería instantánea a través de Internet para GNOME

Presta una alta calidad de sonido e imagen y está disponible tanto para Unix como para Windows.

Permite hacer llamadas de audio y vídeo a usuarios con un hardware y software compatibles con los protocolos SIP o H.323.

Viene instalado por defecto en la distribución de Linux, pero también se puede obtener de la página: http://www.ekiga.org/

Características:

- Facilidad de uso con una moderna interfaz gráfica de usuario
- Llamadas de audio y vídeo gratuitas a través de Internet
- Mensajería instantánea a través de Internet
- Llamadas de audio (y vídeo) a móviles y fijos con soporte de proveedores de servicio
- Alta definición de sonido y calidad de vídeo similares a la calidad de DVD
- Elección libre del proveedor de servicios
- Envío de SMS a móviles si el proveedor soporta el servicio
- Características de la telefonía estándar como las llamadas en espera y transferidas
- Multiplataforma GNU/Linux y Windows
- Interoperabilidad: Ekiga usa los principales protocolos de telefonía (SIP y H.323) y ha sido probado en un amplio rango de softphones, hardphones, PBX y proveedores de servicios
- Soporte de DTMF: imprescindible cuándo se usan servicios que solicitan números de teléfono
- Monitorización de llamadas: estadísticas sobre el tráfico de la red causado por Ekiga
- Soporte de libreta de direcciones remota: Ekiga puede cargar la lista de usuarios de un directorio LDAP remoto (con autentificación)
- Posibilidad de registro de varias cuentas: se puede registrar cuántas cuentas SIP o H.323 como se quiera, y es posible usarlas simultáneamente
- Intercambio de capacidades SIP: Ekiga automáticamente selecciona los códecs comunes entre clientes
- Códecs de audio: G.711-Alaw, G.711-uLaw, Speex, G.722, iLBC, GSM-06.10, MS-GSM, G.726, G.721
- Códecs de vídeo: THEORA (SIP), H.264 (SIP), H.263 (SIP), H.263+ (SIP), H.261 c (SIP y H323), MPEG4(SIP)
- Detección de cambios en las direcciones IP dinámicas y en la adición o eliminación de interfaces de red.
- Rango de puertos configurables (SIP y H323): Ekiga usa puertos estándar, en algunos casos es posible cambiar los puertos (Configuración Avanzada)

Kiax

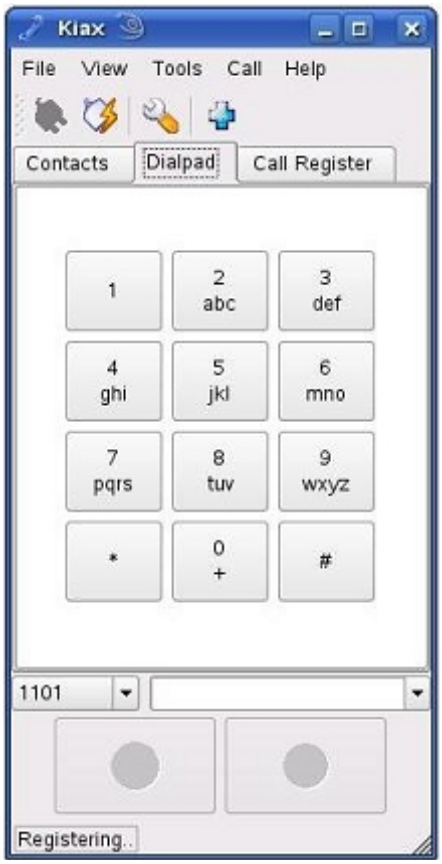


Figura 11: "Softphone Kiax"

Kiax se diseña en 2004 como un pequeño programa para crear una interfaz de usuario que permita hacer llamadas VoIP con Asterisk. Ésto cubriría los requisitos de facilidad de uso, código abierto y libre que el mercado demandaba.

Actualmente ha sido descargado por más de 60.000 usuarios y está disponible para instalar directamente de los repositorios de las distribuciones Linux más usadas como pueden ser Ubuntu y SuSE.

Kiax ver.2 es una nueva y reescrita versión de softphone que tiene como objetivo limpiar los fallos de diseño y proveer una mayor flexibilidad de arquitectura para hacerla más extensible y más personalizable.

El código fuente de Kiax ver.2 es publicado bajo licencia open source. Contiene tres partes principales: la interfaz gráfica de usuario (GUI), el núcleo, y las librerías necesarias.

La última versión es Kiax 2.1, que está disponible para Windows, Linux y Mac OS X, se puede descargar de: http://sourceforge.net/projects/kiax/files/

La arquitectura ha sido diseñada pensando en la extensibilidad y la personalización.

La siguiente figura muestra el modelo por capas usado:

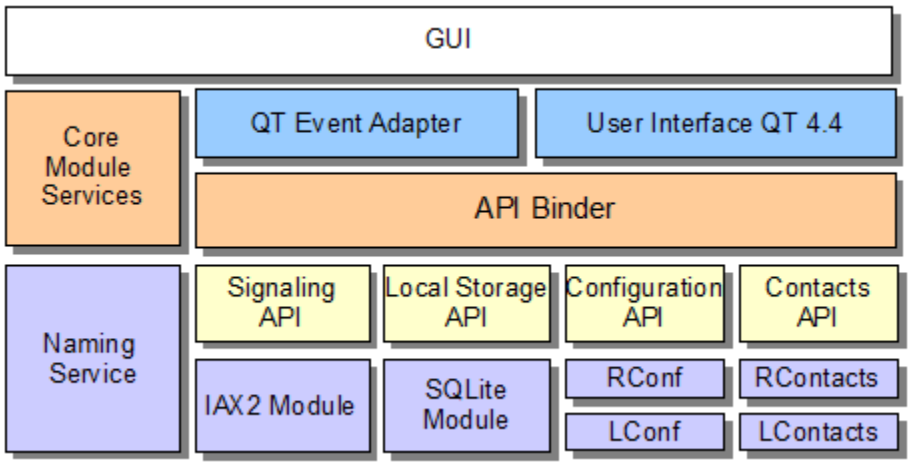


Figura 12: Arquitectura de Kiax

La ventaja clave de ésta organización es la habilidad de extender la funcionalidad del núcleo.

El API Binder proporciona la interfaz para todos los aspectos del sofhphone (mantenimiento de contactos, mantenimiento de llamadas, mantenimiento de cuentas, señalización y configuración).

La implementación actual reside en módulos que están localizados y cargados al iniciar el sofhphone. The Naming Service proporciona los métodos para localizar las implementaciones de los módulos.

El almacenamiento API y la implementación asociada para el almacenamiento de módulos proporciona un punto de acceso para la recuperación de objetos.

La interfaz gráfica de usuario (GUI) implementa una interfaz de usuario intuitiva basada en QT 4.4.

El puenteo de señalización de eventos es archivado a través del adaptador de eventos QT , que tiene en cuenta el envío de eventos entre hilos separados.

Características:

- Licencia de núcleo LGPL y licencia GUI GPL
- Desacople de señal, almacenamiento y aspectos de visualización
- Capa de núcleo modular y liberada
- Código GCC4 activo
- Código base para Linux, Windows y MacOS
- SQLite como almacenamiento por defecto
- QT4.4 como una GUI
- Integración Web
- Interfaz de usuario más simple que la versión anterior
- Completamente encadenable

- Configuración remota
- Integración simplificada con proveedores de servicios
- Soporte de varios proveedores de servicios
- Soporte de varias llamadas simultáneas
- Soporte de registro de fallos
- Búsqueda de contactos y CDR
- Códecs: g711, iLBC, GSM, Speex
- Filtro de reducción de ruido
- Soporte I18n

3.3 Distribución del laboratorio

Para la realización del proyecto se ha hecho uso del laboratorio docente IT-2 del departamento TIC de la UPCT.

El laboratorio está compuesto por 15 ordenadores conectados mediante una red LAN a otro ordenador que hace la función de router para la comunicación con el exterior.

Todos los ordenadores tienen la misma configuración hardware y software. Las características hardware principales son: procesador Intel Core 2 Duo a 6400 a 2.13GHz y 2GB de RAM.

Cada ordenador tiene instalado los sistemas operativos Linux OpenSUSE 11.0 así como Windows XP. Siendo usado para el proyecto exclusivamente Linux durante todos los casos de uso.

La disposición es de 3 ordenadores por banco de trabajo. Haciendo uso de ésta distribución se ha optado por crear una red por banco, dando lugar a un total de 5 redes distintas con las direcciones de red que van desde 192.168.1.0 a 192.168.5.0.

Cada red está compuesta por dos ordenadores con los clientes de VoIP: Kiax y Ekiga, así como uno con la centralita Asterisk v1.8.3.2 instalada.

El esquema de distribución es el siguiente:

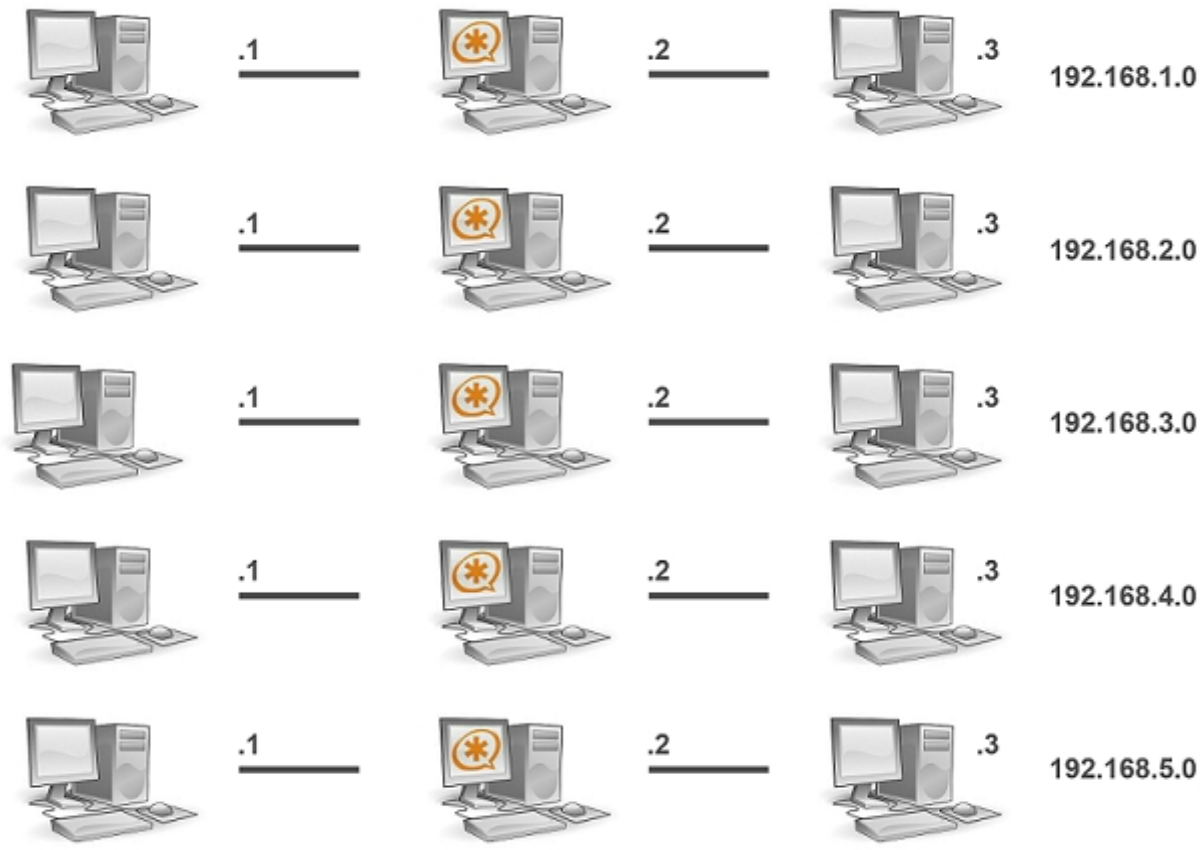


Figura 13: “Distribución del laboratorio”

3.4 Instalación y configuración

3.4.1 Asterisk

3.4.1.1. Instalación

Para la realización de los casos prácticos de los que está compuesto el proyecto se hará uso de la centralita Asterisk v1.8.4, cuya instalación y configuración se detalla a continuación en tres sencillos pasos

Comprobación de librerías e instalación

El primer paso a realizar será comprobar la existencia de las librerías: *OpenSSL, ncurses, newt, libxml2 y Kernel headers* en los ordenadores que harán las veces de servidor y dónde se ejecutará Asterisk, ya que serán necesarias para el correcto funcionamiento de la centralita.

Para comprobar si están instaladas o para instalarlas se hará uso de la herramienta *YaST2* que viene por defecto en la versión SUSE de los ordenadores del laboratorio.

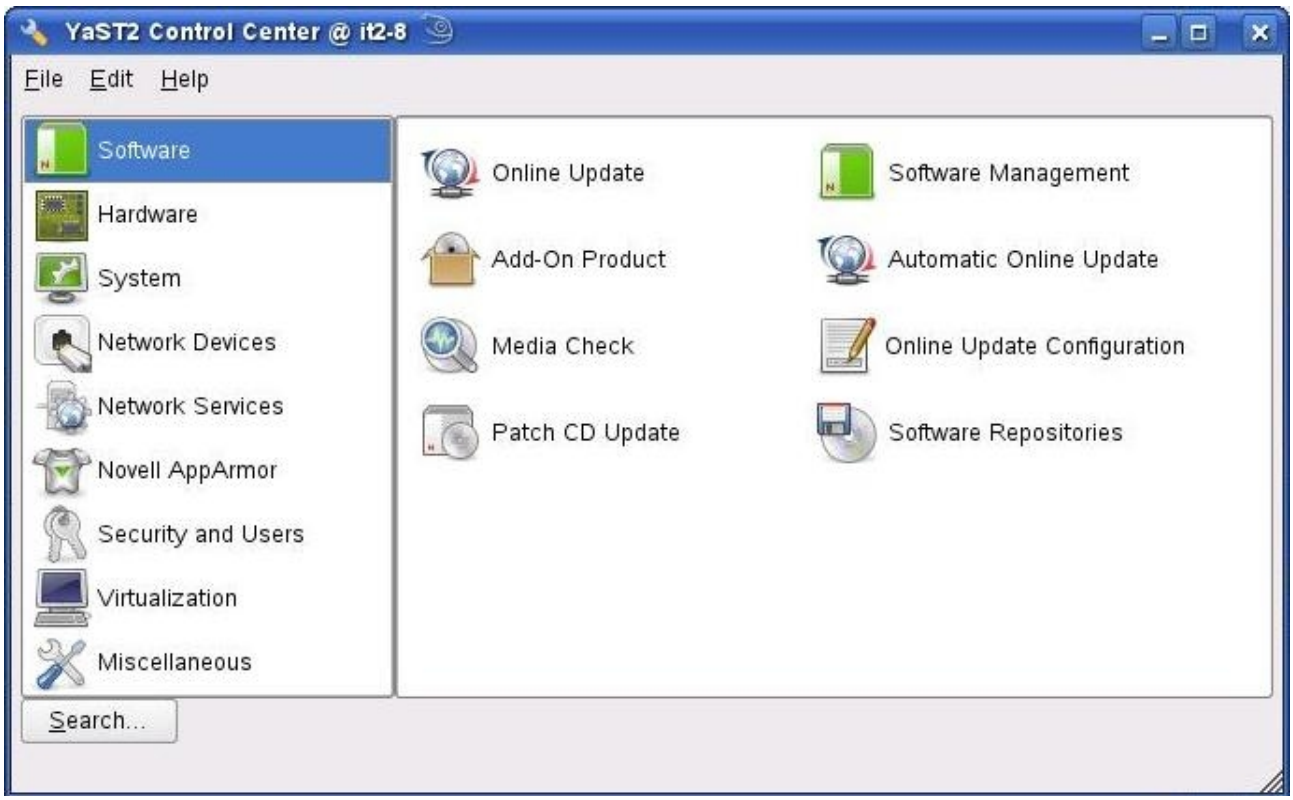


Figura 14: "Instalación de Asterisk: YaST2 - librerías"

Descarga de Asterisk y de los módulos DAHDi y libPRI

El paso siguiente será la descarga de la centralita de Asterisk y de los módulos *DAHDI* y *libPRI*.

La página oficial de Asterisk, en su sección descargas, contiene todo lo necesario para poder instalar correctamente la centralita.

http://www.asterisk.org/downloads

Se hará uso de la versión más reciente de Asterisk, ya que es la ofrece mejores resultados y prestaciones, descargando así el código fuente de la versión 1.8.4

También se descargará el paquete de herramientas *DAHDI* que contiene una estructura de drivers de código libre que definen la interfaz de comunicación entre la aplicación y el hardware telefónico. A su vez éste paquete contiene utilidades para el mantenimiento y monitorización de los dispositivos *DAHDI*.

En éste caso cobra importancia ya que contiene un temporizador, anteriormente llamado *Zaptel*, que será necesario en la aplicación *meetme()* para conseguir realizar conferencias grupales.

El siguiente módulo a descargar será *libPRI*, una librería de código abierto que encapsula los protocolos usados para la comunicación a través de las interfaces primarias ISDN (T1, E1, J1). *LibPRI* es una dependencia de Asterisk y *DAHDI* si se usa la señalización PRI.

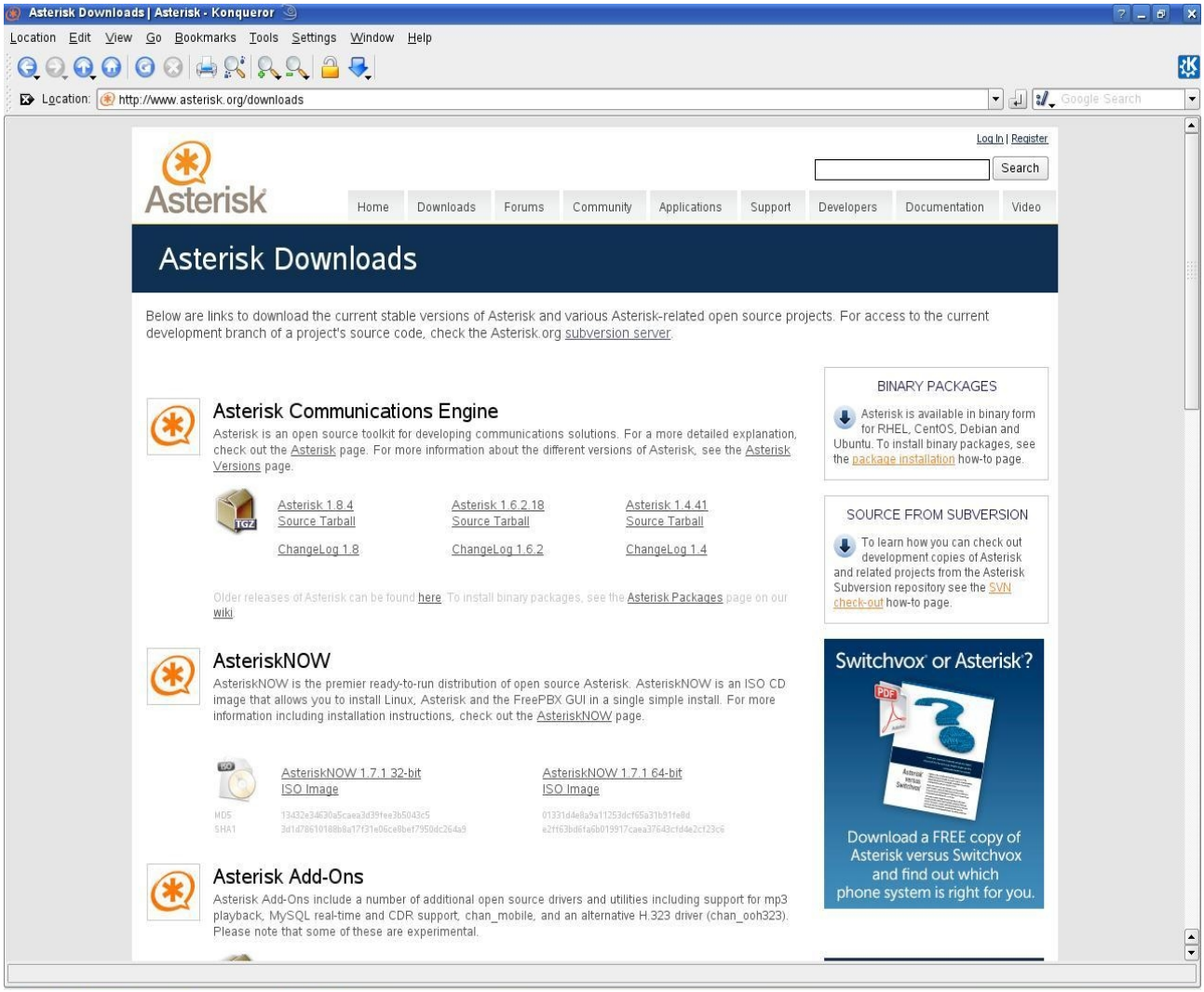


Figura 15: “Instalación de Asterisk: downloads”

Descompresión e instalación

Una vez descargados los paquetes

- Asterisk 1.8.4 Source Tarball
- DADHDI Complete 2.6.0+2.6.0 Source Tarball
- LibPRI 1.2.12 Source Tarball

Se descomprimirán en el directorio */usr/local/src* mediante el comando `tar`:

```
# tar -zxvf libpri-1.4.12.tar.gz
# tar -zxvf dahdi-linux-complete-2.6.0+2.6.0.tar.gz
# tar -zxvf asterisk-1.8.4.tar.gz
```

Para la configuración e instalación del módulo *libPRI* se hará de la siguiente forma:

```
# cd libpri-1.4.12
# make
# make install
```

Para la configuración e instalación del módulo *DAHDI*:

```
# cd dahdi-linux-complete-2.6.0+2.6.0
# make
# make install
# make config
```

Para la configuración de la centralita Asterisk, el procedimiento a seguir será el siguiente:

```
# cd asterisk-1.8.4
# ./configure
# make menuselect
# make
# make install
# make samples
# make config
# make install-logrotate
```

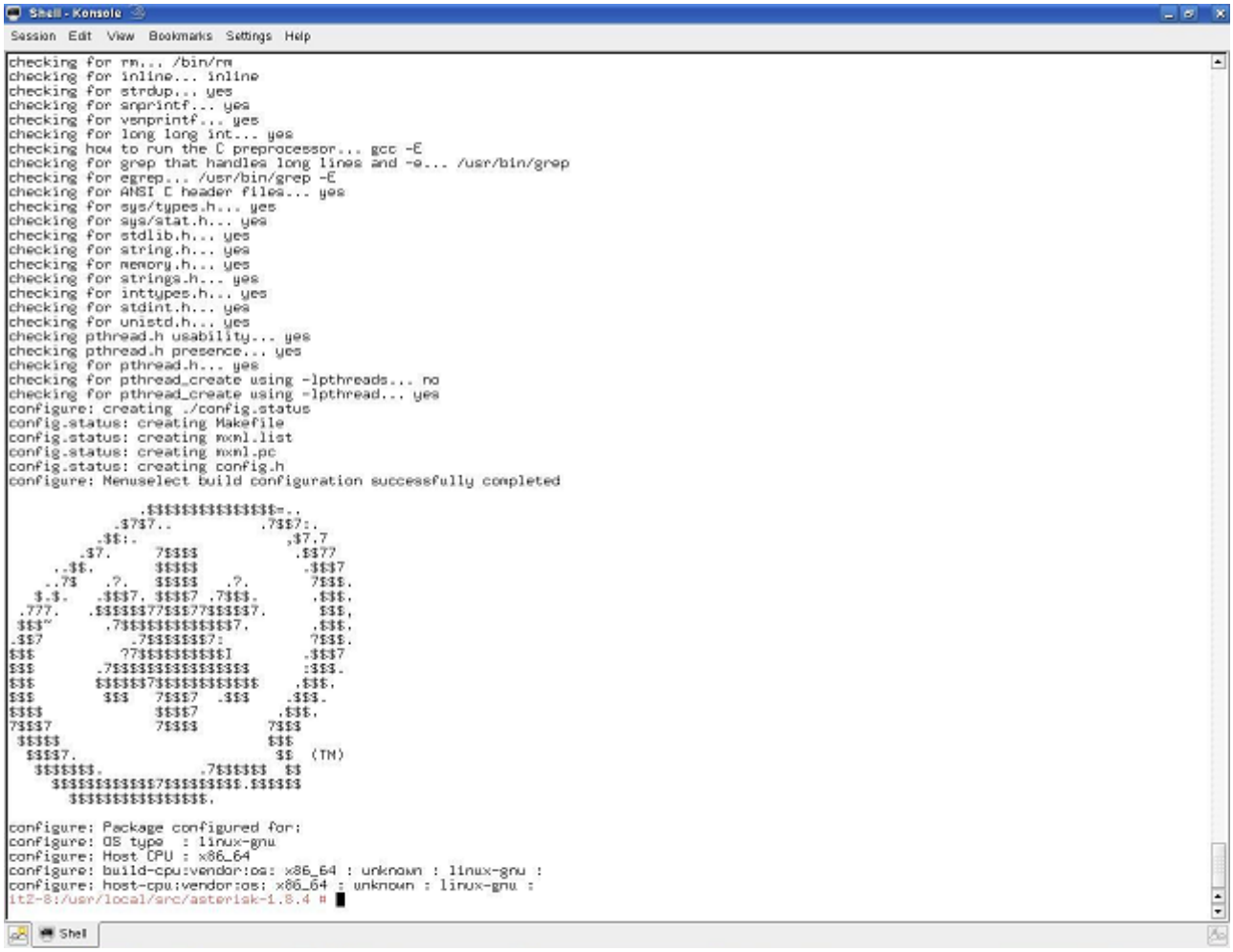


Figura 16: “Instalación de Asterisk: comando ./configure”

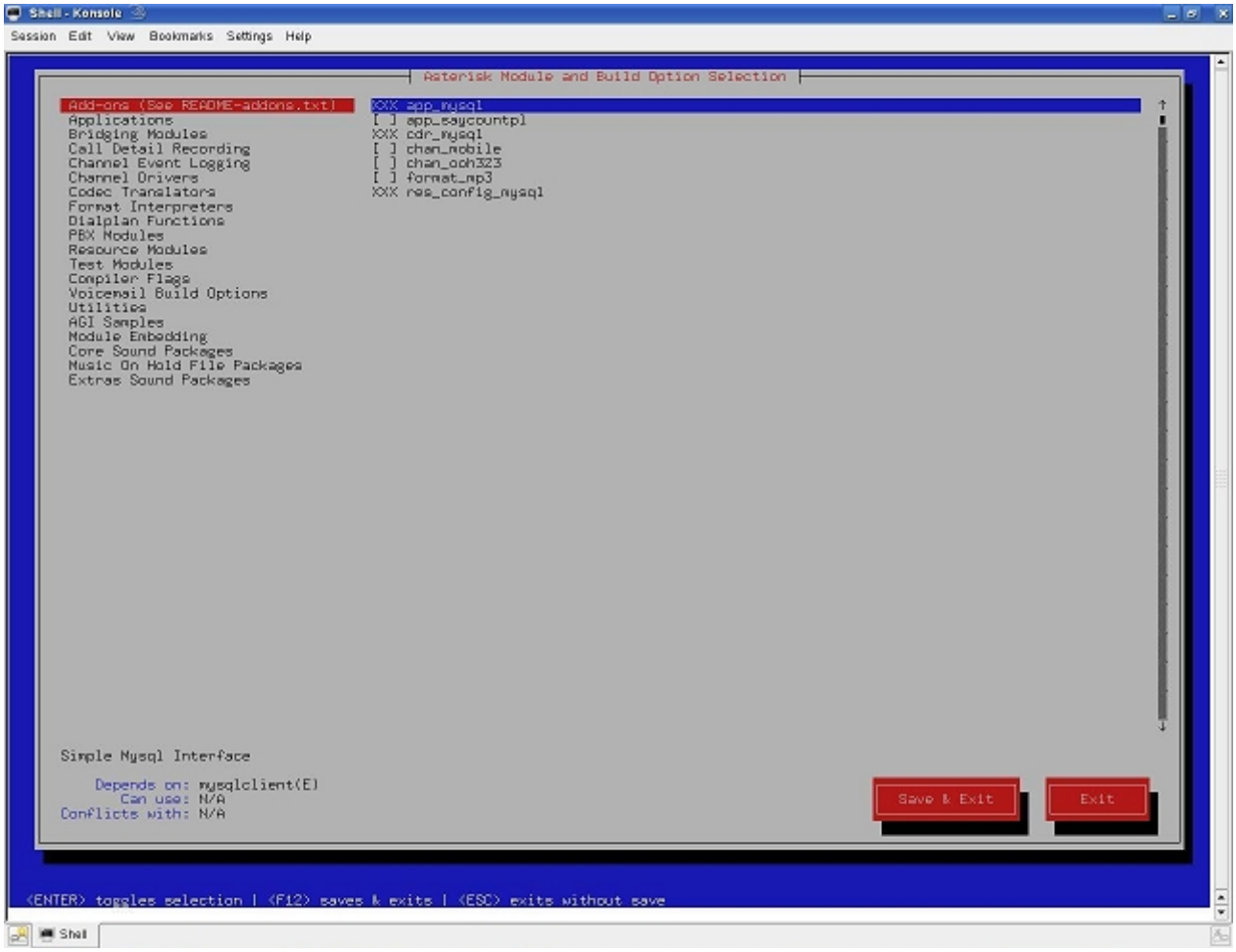


Figura 17: “Instalación de Asterisk: comando make menuselect”

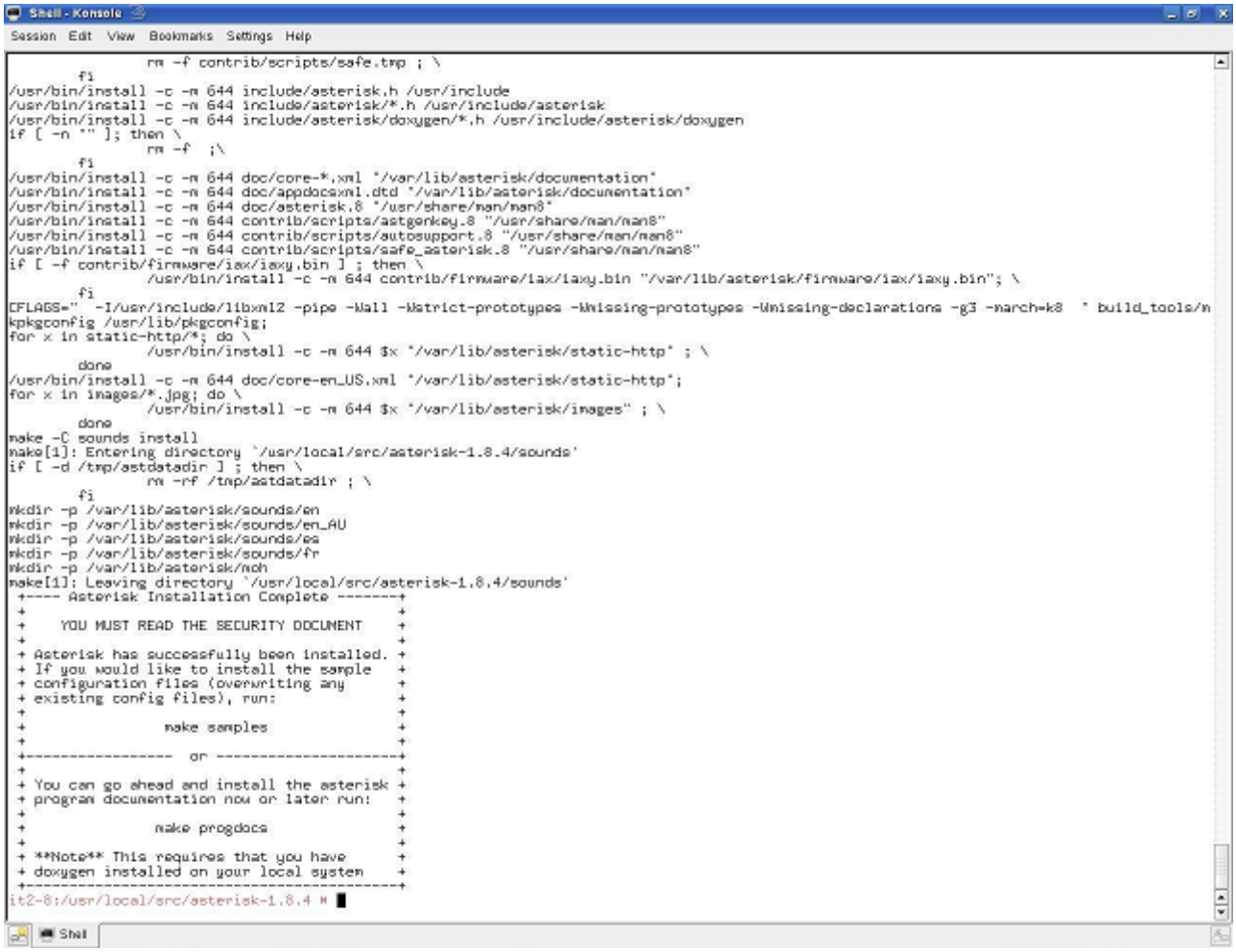


Figura 18: "Instalación de Asterisk: comando make install"

3.4.1.2 Administración

Arranque:

Tanto DAHDI como Asterisk son demonios que se ejecutan en segundo plano, para comprobar que estén activos se ejecutará en el shell:

```
#lsmod |grep dahdi
#lsmod|grep asterisk
```

En caso de no mostrar nada significará que no están activos por lo tanto habrá que arrancarlos, desde el directorio */etc/init.d* ejecutaremos:

```
#./dahdi start
#./asterisk start
```

Conexión al CLI:

Para conectar con la línea de comandos de Asterisk (CLI) se hará mediante

```
#asterisk -r
```

Al arrancar Asterisk se puede añadir las opciones `debug` y `verbose` que añaden información adicional de la centralita y su funcionamiento.

La opción `verbose` muestra información detallada de los eventos generales de la centralita, se configura añadiendo un número determinado de 'v' al arrancar, a mayor número más información se obtiene.

Para conectar a la línea de comandos con el modo `verbose` activo se escribirá en el shell:

```
#asterisk -vvvr.
```

También se puede modificar desde el CLI mediante el comando: `set verbose` y un número que indique el grado correspondiente de información.

La opción `debug` muestra información de bajo nivel que puede resultar útil para detectar fallos de Asterisk, se usa añadiendo las 'd' deseadas al conectarse al CLI, por lo tanto en el shell se escribirá:

```
#asterisk -ddddr.
```

También se puede modificar desde el CLI usando el comando: `set debug` y el número correspondiente que indique el grado de mensajes que se mostrarán por pantalla para depurar errores.

Las opciones `debug` y `verbose` son compatibles, por lo que se pueden seleccionar juntas a la hora de conectarse a la línea de comandos. Se escribirán en el shell de la forma:

```
#asterisk -dddvvvvr
```

El intérprete de comandos o CLI permite comprobar, configurar y monitorizar la centralita. Entre las distintas opciones de comandos tenemos las siguientes más utilizadas:

- `quit`*:* se desconecta el CLI y Asterisk sigue trabajando en segundo plano
- `restart now`*:* reinicia automáticamente Asterisk
- `restart gracefully`*:* reinicia cuándo no hay llamadas en el sistema, no deja entrar nuevas
- `restart when convenient`*:* reinicia cuándo no hay llamadas activas, pero no previene que entren nuevas
- `!<comando>`*:* ejecuta el comando deseado en el shell de Linux, devuelve el resultado y vuelve inmediatamente al CLI de Asterisk
- `help`*:* muestra una lista de ayuda o la ayuda del comando específico

- `show application`*:* describe la aplicación especificada
- `show channel`*:* muestra la información del canal
- `show conferences`*:* muestra el estado de las conferencias
- `show dialplan`*:* muestra el dialplan
- `show parkedcalls`*:* muestra las llamadas que están parqueadas
- `iax2 show peers`*:* muestra el estado de los clientes IAX que están definidos en Asterisk
- `sip show peers`*:* muestra el estado de los clientes SIP definidos en Asterisk
- `meetme list`*:* muestra la lista de salas de conferencias y el número de usuarios que participan en cada una de ellas

Organización:

Los archivos instalados de Asterisk se encuentran distribuidos en los siguientes directorios:

- Binarios Asterisk: */usr/sbin/asterisk*
- Módulos ejecutables de Asterisk: */usr/lib/asterisk/modules*
- Voces pregrabadas: */var/lib/asterisk/sounds*
- Ficheros de configuración: */etc/asterisk/*.conf*
- Otros servicios (buzón de voz, etc): */var/spool/asterisk/*
- Proceso activo: */var/run*

3.4.2 Ekiga

Ekiga viene instalado por defecto en la distribución OpenSUSE 11.0 que corre en los ordenadores del laboratorio IT-2, así que no necesita de una instalación previa.

Para arrancar el softphone podremos hacerlo desde un terminal con el comando:

```
#ekiga &
```

O bien desde el menú de KDE de la forma: aplicaciones, Internet, teléfono.

La configuración puede realizarse de forma manual o con una configuración rápida guiada, optaremos por la configuración manual del teléfono.

Lo primero que se realizará es la configuración de los parámetros hardware, como el dispositivo de sonido o el controlador para videoconferencia. El dispositivo de sonido es HDA Intel, que es el usado por los ordenadores del laboratorio. Para la videoconferencia usaremos el que se muestra por defecto ya que en el proyecto no se hará uso de dicho servicio.

A continuación se configurará la interfaz IP usada para comunicarse con la centralita Asterisk, para ello se habrá de acceder a: “Edit”, “Preferences”, “Network Settings” y ahí se seleccionará la interfaz 192.168.X.Y, siendo X el número de red, e Y dicho ordenador.

Para finalizar se crearán la o las cuentas de Ekiga que se hayan configurado previamente en Asterisk. En el menú “Edit” se seleccionará la opción “Accounts” y posteriormente, seleccionando “Add” se añadirá la cuenta.

Aparecerá una ventana que solicita los datos de la cuenta de Ekiga definida en el archivo sip.conf de Asterisk, por lo tanto se rellenarán los datos en función de dicho archivo.

- Account Name (nombre de cuenta): Nombre de la cuenta sip usada, en el proyecto se ha usado un formato que indica la red a la que pertenece, el tipo de protocolo y el ordenador dónde está definida. Así una cuenta tipo sip de la red 1 perteneciente al ordenador 3 sería: 1003, dónde el 1 indica la red, el primer 0 el tipo de protocolo (sip), y el 03 indica el puesto del ordenador en el banco.
- Protocol (protocolo): posibilidad de elegir entre SIP y H.323, para el proyecto se hará uso de SIP
- Registrar (registro): Dirección IP del servidor dónde se ejecuta Asterisk. Será 192.168.X.2, siendo X el número de red
- Password (contraseña): contraseña definida en el archivo sip.conf para dicho cliente. Será pass para todos los clientes, tanto sip como iax.

Una vez aceptado el cuadro de diálogo, la cuenta aparecerá en el menú "Accounts", dónde se marcará el recuadro que aparece a la izquierda para que inicie el registro con Asterisk, si a la izquierda aparece como estado el mensaje "Registered" el proceso de registro habrá finalizado con éxito y la cuenta estará operativa y lista para realizar y recibir llamadas de otros terminales.

Figura 19: "Configuración de Ekiga"

3.4.3 Kiax

Kiax también viene instalado por defecto en la distribución OpenSUSE, por lo que no será necesario el paso previo de instalación.

Para arrancar el teléfono se puede hacer bien desde la consola:

```
#kiax &
```

O bien desde el menú del escritorio: aplicaciones, Internet, teléfono.

Una vez abierto se procederá a la configuración de la cuenta Kiax que previamente estará definida en el archivo iax.conf de Asterisk.

Para ello se seleccionará “File”, “Settings” y en la ventana que aparecerá se selecciona la pestaña “Accounts”. Pinchando sobre “New Account” se podrá definir la nueva cuenta.

Habrá que rellenar los siguiente parámetros:

- Account Name (nombre de la cuenta): Nombre de la cuenta iax usada, al igual que las cuentas sip, éstas tienen asociado un nombre que identifica el número de red, protocolo y posición del ordenador que tiene asociada la cuenta. Para una cuenta iax de la red2 y ordenador 3 el nombre sería: 2103, siendo 2 el número de red, 1 el tipo de protocolo (iax en este caso), y 03 la posición del ordenador en el banco.
- IAX Server (servidor IAX): servidor dónde está definido el cliente IAX, en éste caso será el de la centralita Asterisk, por lo tanto para la red 1 la dirección del servidor será 192.168.1.2
- Username (nombre de usuario): Nombre de la cuenta creada en Asterisk
- Password (contraseña): Contraseña definida en el archivo iax.conf asociada con dicha cuenta de usuario
- Caller ID y Caller Number (número e identificación del llamante): Datos asociados a la cuenta que se mostrarán a otros usuarios cuándo éste cliente realice una llamada.
- Preferred Codec (Códec preferido): Códecs de audio permitidos por Kiax. Deberemos seleccionar uno que esté disponible en la centralita Asterisk para que funcione correctamente, como puede ser GSM.

En caso de tener varias cuentas IAX definidas en Kiax, podremos seleccionar una para que sea la elegida por defecto seleccionando la opción “Make account default”. Una vez guardados los cambios tendremos la cuenta IAX activa y dispuesta a registrarse con el servidor de Asterisk.

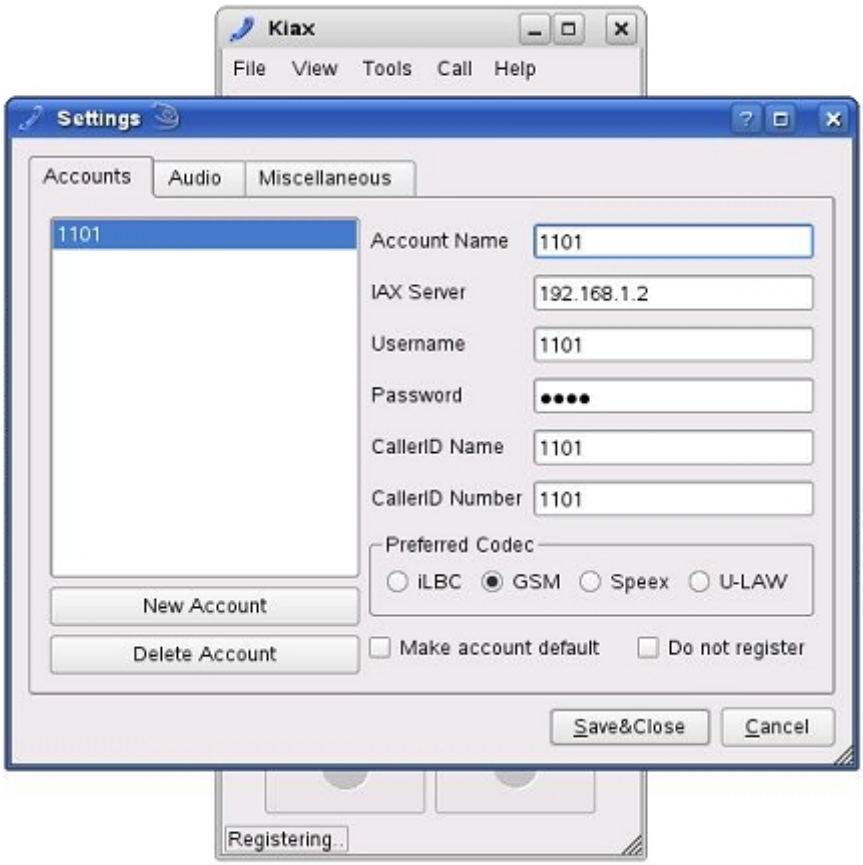


Figura 20: “Configuración de Kiax”

Capítulo 4: FUNCIONAMIENTO DE ASTERISK

4.1 Conceptos

Canal:

Un canal es una conexión que distribuye una llamada entrante o saliente dentro del sistema de Asterisk PBX.

Puede ser una conexión con la telefonía analógica, digital o VoIP.

No hay distinción entre líneas de teléfono o teléfonos, cada llamada se recibe en un canal distinto.

Entre los tipos de canales que hay se encuentran:

- IAX2, SIP y Skinny como protocolos VoIP
- Zap para la conexión con la línea de telefonía tradicional
- mISDN para las líneas RDSI.

Cada canal tiene su propio espacio de variables de canal asociadas que son borradas automáticamente cuándo finaliza la llamada correspondiente. Se modifican mediante el comando `Set` y se referencian mediante `${nombrevariable}`.

En caso de coincidir en nombre con las variables globales definidas por el usuario, el contenido de la variable de canal prevalecerá sobre la variable global

Dialplan:

Define el comportamiento de la centralita Asterisk PBX en cuánto a llamadas entrantes, salientes y forma de uso.

Se encuentra definido en el archivo *extensions.conf*, ubicado en */etc/asterisk/*.

El contenido está dividido en contextos y éstos a su vez en extensiones. Los contextos habituales son: `general`, `global` y los definidos por el usuario.

Contexto:

Engloba un determinado número de extensiones en su interior y define un comportamiento concreto dentro del dialplan.

Hay varios tipos de contextos como pueden ser:

- `[general]`: define las opciones de cada archivo de configuración de forma general.
- `[globals]`: define las variables globales que serán usadas como constantes posteriormente por las extensiones. Pueden ser llamadas por cualquier canal y en cualquier momento.
- `[definidos por el usuario]`: son un conjunto de contextos que definen el comportamiento completo de la centralita a través del manejo de extensiones y aplicaciones asociadas
- `[macro-nombre]`: es un tipo especial de contexto, su nombre empezará por macro- seguido del nombre correspondiente, y estará definida usando la extensión "`s`". Para saber qué extensión está haciendo uso de dicha macro se hará una llamada a `${MACRO_EXTEN}` que proporcionará la extensión actual.

Es posible incluir contextos dentro de otros y saltar entre ellos para ejecutar algún tipo de aplicación.

Extensión:

Es un número que tiene asociado una lista de aplicaciones a ejecutar.

Cada extensión se compone de: nombre, prioridad y aplicación

Normalmente el número de extensión corresponde con un cliente telefónico, otras veces es el destino de un salto desde una aplicación y gestiona un proceso a ejecutar. Hay tres tipos de extensiones:

- Literales: son una cadena numérica o incluso alfanumérica.
- Predefinidas: vienen establecidas por defecto y tienen un propósito específico, son las siguientes: `i,s,h,t,T,a,o`.
- Patrón: se utilizan para simplificar un conjunto de extensiones que tienen un patrón común. Comienzan por "`_`" y van seguidas por una letra (`X`, `Z` o `N`), por un conjunto de números entre corchetes e incluso un punto "`.`" para indicar que una vez que coincida el patrón, puede valer cualquier número que lo contenga.

Aplicación:

Es una instrucción asociada a una extensión.

Una vez se realiza una llamada o un salto a una extensión determinada, se ejecutan de forma secuencial distintos tipos de aplicaciones definidas en el dialplan para dicha extensión.

Hay varios tipos de aplicaciones: generales, de mantenimiento de llamada, de control de flujo, de manipulación de variables, de gestión de sonidos, de uso de buzón de voz y conferencias entre otras.

Las más utilizadas en la realización de los casos prácticos son: `Wait`, `Answer`, `Hangup`, `Dial`, `GotoIf`, `NoOp`, `Set`, `Background`, `Playback`, `Record`, `MeetMe`, `VoiceMail`, `VoiceMailMain`

4.2 Introducción a la configuración

Una vez se encuentra instalada la centralita y se se ha verificado que funciona correctamente se deberán configurar las conexiones telefónicas que la componen, para ello habrá que configurar los canales correspondientes, en éste caso se usarán dos, IAX y SIP.

El siguiente paso a realizar será configurar el plan de marcación (dialplan) que viene descrito en el archivo *extensions.conf*. El plan de marcación define cómo se enruta cada llamada en el sistema a través de varias aplicaciones hasta el destino final.

Esto compone una configuración básica, es posible añadir nuevas funcionalidades a *extensions.conf*, configurar distintos archivos y usar aplicaciones externas.

Entre los muchos archivos de configuración que forman parte de la centralita hay tres que destacan en importancia del resto y que son esenciales para una configuración básica de VoIP.

Sip.conf e *iax.conf* son archivos de configuración que definen el canal de comunicación para el protocolo correspondiente. Contienen las extensiones definidas y los contextos asociadas a ellas.

Extensions.conf contiene el plan de marcación de la centralita y maneja las llamadas entrantes y salientes en función de las extensiones asociadas a los contextos. Es el archivo más complejo de Asterisk, así que su uso y configuración se detallará minuciosamente para cada caso expuesto.

SIP.CONF

El archivo configura el canal para el protocolo SIP y añade nuevos usuarios o conecta con un proveedor SIP.

Al instalar Asterisk, el archivo se crea con los parámetros por defecto y con una guía para su configuración.

Tiene dos secciones distintas:

- `[general]`: contiene la configuración por defecto de todos los usuarios o proveedores. Se pueden sobrescribir los valores por defecto en la configuración de cada usuario o peer. Define valores como el puerto de comunicaciones, la dirección IP de conexión al servidor y los códecs de audio permitidos.
- `[extensión]`: contiene el tipo de conexión asociada a dicha extensión, que puede ser user, peer o friend dependiendo de si puede realizar o recibir llamadas. También define la contraseña de conexión al servidor, el host desde dónde se conecta, el identificador de llamadas y el contexto asociado a dicha extensión en el dialplan.

Una configuración básica del archivo *sip.conf* dónde se definen las extensiones 1001 y 1003 y se hace una referencia a un contexto determinado del plan de marcación definido en *extensions.conf*, podría ser la siguiente:

```
[general]
    context=default     ;contexto común para todas las extensiones si no se especifica otro distinto
    port=5060           ;puerto UDP de escucha por defecto de Asterisk para el protocolo SIP
    bindaddr=0.0.0.0    ;dirección IP para la conexión al servidor Asterisk (en éste caso todas las interfaces del servidor)
    disallow=all        ;deshabilita todos los códecs de audio para volver a definirlos más adelante
    allow=gsm           ;activa el códec gsm como prioritario para el protocolo SIP. Asterisk intentará usar éste códec en primer lugar y negociará su uso, si alguno de los dos extremos no lo puede usar pasará al siguiente códec definido.
    allow=alaw          ;el códec alaw se usará en caso de que no se pueda usar el gsm
    allow=ulaw          ;el códec ulaw se usará en caso de que no se puedan usar ni gsm ni alaw
[1001]      ;definición del usuario SIP y el contexto al que pertenece en extensions.conf
    type=friend         ;indica el tipo de usuario, hay tres tipos: "user" cuándo el usuario puede realizar llamadas, "peer" si las recibe y "friend" para ambas cosas. En éste caso se pueden realizar y recibir llamadas en la extensión 1001
    username=1001       ;el nombre de usuario puede ser cualquier cadena de caracteres, en éste caso se usará el nombre de la extensión
    secret=pass         ;contraseña para el registro de la extensión con el servidor
    host=dynamic        ;Dirección IP de la extensión. Puede tomar tres valores: una dirección IP, "dynamic" que indica que cualquier IP es válida pero necesita de una contraseña para el registro, o "static" dónde cualquier IP es válida y no es necesaria contraseña
    callerid=1001       ;identificador de llamada que se mostrará para esta extensión, puede ser cualquier cadena de caracteres
    context=prueba      ;indica el contexto asociado en el dialplan para dicha extensión
    qualify=yes         ;determina si la extensión es alcanzable en función del tiempo de respuesta de esta. Máximo 60 segundos
```

```
[1003]
    type=friend
    username=1003
    secret=pass
    host=dynamic
    callerid=1003
    context=prueba
    qualify=yes
```

Aunque no se han hecho uso de los parámetros `srvlookup` o `nat`, es interesante comentarlos.

El parámetro `srvlookup` se configura para conectarse con un servidor de VoIP de internet, para ello Asterisk realiza la resolución de nombres DNS para alcanzar la máquina del servidor.

En caso de hacer uso de NAT (Network Address Translation) para convertir direcciones IP dentro de una red, habrá que configurar el parámetro `nat`. SIP puede dar problemas con el uso de NAT, por lo tanto se desactivará ésta función.

Otra de las funcionalidades de SIP es el uso del registro de servidores o servicios externos para permitir la ubicación física de un usuario determinado. Cada usuario tiene una dirección lógica invariable respecto la ubicación física del usuario. La dirección lógica SIP es de la forma `usuario@dominio` y la dirección física depende del lugar dónde se conecte (de su dirección IP).

Cuándo un usuario inicia su terminal, el agente correspondiente envía una petición de registro al servidor de registro indicando la dirección física a la que se asociará la dirección lógica del usuario.

No se hará uso del registro porque todas los clientes tienen direcciones IP estáticas.

El registro se especificaría en la sección `[general]` con el siguiente formato:

```
register=>usuario:contraseña@dominio
```

Nos registraríamos con el proveedor con el nombre "`usuario`" y con la contraseña y dirección del servidor. En caso de que el servidor use un puerto distinto a `5060` se especificará al final de la línea de la siguiente forma:

```
register=>usuario:contraseña@dominio:puerto
```

Para una configuración más avanzada del archivo *sip.conf*, habrá que considerar las siguientes opciones para los tipos de usuarios "`user`" y "`peer`". Para el tipo "`friend`" valdrá cualquiera de las dos columnas:

User	**Peer**	**Definición y forma de uso**
context	context	Indica el contexto asociado en el dialplan para un usuario o peer
permit	permit	Permite una IP
deny	deny	No permite una IP
secret	secret	Contraseña para el registro
md5secret	md5secret	Contraseña encriptada con md5
dtmfmode	dtmfmode	Modo en el que se transmiten los tonos. Pueden ser "RFC2833" o "INFO"
canreinvite	canreinvite	Con "no" se fuerza a Asterisk a no permitir que los puntos finales intercambien mensajes RTP directamente
nat	nat	Con "yes" indica si el dispositivo está detrás de un NAT
callgroup	callgroup	Define un grupo de llamadas
language	language	Define las señales para un país. Debe estar presente en el archivo indications.conf
allow	allow	Permite habilitar un códec. Pueden ponerse varios en un mismo usuario. Posibles valores: "all", "alaw", "ulaw", "g723.1", "g729", "ilbc", "gsm"
disallow	disallow	Permite deshabilitar un códec. Puede tomar los mismos valores que allow
insecure	insecure	Define cómo manejar las conexiones con peers. Tiene los siguientes valores: very|yes|no|invite|port. Por defecto está configurado con "no" que indica que hay que autenticarse siempre
trustpid	trustpid	Si la cabecera Remote-Party-ID es de confianza. Por defecto "no"
progressinband	progressinband	Si se deben generar señales de banda siempre. Por defecto "never"
promiscredir	promiscredir	Permite soportar redirecciones 302. Por defecto "no"
callerid		Define el identificador cuándo no hay ninguna otra información disponible
accountcode		Los usuarios pueden estar asociados con un accountcode. Se usa para facturación
amaflags		Se usa para guardar en los CDR y temas de facturación. Puede ser "default", "omit", "billing", o "documentation"

incominglimit		Límite de llamadas simultáneas para un cliente
restrictcid		Se usa para esconder el ID del llamante. En desuso.
	mailbox	Extensión del contestador
	username	Si Asterisk actúa cómo cliente SIP éste es el nombre de usuario que presenta en el el servidor SIP al que llama
	fromdomain	Pone el campo From: de los mensajes SIP
	regexten	
	fromuser	Pone el nombre de usuario en el from por encima de lo que diga el callerID
	host	Dirección o host dónde se encuentra el dispositivo remoto. Puede tomar como valores: -una IP o host concreto -"dynamic", valdría cualquier IP pero necesita contraseña -"static", valdría cualquier IP pero no es necesario contraseña
	mask	
	port	Puerto UDP en el que responderá Asterisk
	qualify	Para determinar cuándo el dispositivo puede ser alcanzado
	defaultip	IP por defecto del cliente host= cuándo es especificado como "dynamic"
	rtptimeout	Termina la llamada cuándo llega a ese timeout si no ha habido tráfico rtp
	rtpholdtimeout	Termina la llamada cuándo llega a ese timeout si no ha habido tráfico rtp "on hold"

Cada vez que se cambie la configuración del archivo *sip.conf* habrá que volver a cargarla en el servidor, desde el CLI se hará uso del comando

```
CLI> sip reload
```

Para comprobar el estado de los "`peers`" o "`users`" definidos podremos usar los comandos

```
CLI> sip show peers
CLI> sip show users
```

Aparecerá información sobre nombre, host, dinamicidad, Nat, ACL, puerto y estado para cada uno de los usuarios mostrados

Si los clientes están definidos como "`friends`" aparecerán al ejecutar ambos comandos.

IAX.CONF

En dicho archivo se configura el canal IAX para el uso del protocolo IAX2 (Inter-Asterisk eXchange protocol) y se definen las extensiones que usarán dicho canal.

La estructura es similar a la del archivo *sip.conf*, contiene las siguientes secciones:

- `[general]`: dónde configurar los parámetros generales para todos los usuarios definidos. Define valores como el puerto de comunicaciones, la dirección IP de conexión al servidor y los códecs de audio permitidos, así como la gestión del ataque de contraseñas
- `[callnumberlimits]`: dónde se pueden limitar direcciones IP o rangos de direcciones IP. Los límites predominan sobre la entrada "`maxcallnumbers`"
- `[extensión]`: al igual que en *sip.conf* en ésta sección se define el comportamiento de cada extensión que usará el protocolo IAX2. Define el tipo de usuario, el host desde dónde se conecta, la contraseña, el identificador de llamadas, la calidad y el contexto asociado a dicha extensión en el plan de marcado

A modo de ejemplo se usará una configuración básica del archivo *iax.conf* para detallar la configuración. Se definen dos extensiones `1101` y `1103` que harán referencia al contexto prueba definido en el archivo *extensions.conf*.

```
[general]
    bindport=4569          ;puerto UDP de escucha por defecto de Asterisk para el
                           protocolo IAX
    bindaddr=0.0.0.0       ;dirección IP para la conexión al servidor Asterisk (en éste
                           caso todas la interfaces del servidor)
    delayreject=yes        ;mejora la seguridad contra ataques de fuerza bruta
                           retrasando el envío de los mensajes de autenticación
    disallow=all           ;deshabilita todos los códecs de audio para volver a definirlos
                           más adelante
    allow=gsm              ;activa el códec gsm como prioritario para el protocolo SIP.
                           Asterisk intentará usar éste códec en primer lugar y negociará
                           su uso, si alguno de los dos extremos no lo puede usar pasará
                           al siguiente códec definido.
    allow=alaw             ;el códec alaw se usará en caso de que no se pueda usar el
                           gsm
    allow=ulaw             ;el códec ulaw se usará en caso de que no se puedan usar ni
                           gsm ni alaw
    autokill=yes           ;se usa para comprobar el estado de las conexiones, si está
                           activo en "yes" y no se recibe un ACK en 2000ms se cancela
                           la conexión. Es útil para terminar llamadas que no tienen
```

```
                        respuesta
[1101]      ;definición del usuario IAX y el contexto al que pertenece en extensions.conf
    type=friend         ;indica el tipo de usuario, hay tres tipos: "user" cuándo el
                        usuario puede realizar llamadas, "peer" si las recibe y
                        "friend" para ambos casos. En éste caso se pueden realizar y
                        recibir llamadas en la extensión 1101
    host=dynamic        ;Dirección IP de la extensión. Puede tomar tres valores: una
                        dirección IP, "dynamic" que indica que cualquier IP es válida
                        pero necesita de una contraseña para el registro, o "static"
                        dónde cualquier IP es válida y no es necesaria contraseña
    secret=pass         ;contraseña para el registro de la extensión con el servidor
    context=prueba      ;indica el contexto asociado en el dialplan para dicha
                        extensión
    qualify=yes         ;determina si la extensión es alcanzable en función del tiempo
                        de respuesta de esta mediante un ping. Máximo 60 segundos
    callerid=1101       ;identificador de llamada que se mostrará para esta extensión,
                        puede ser cualquier cadena de caracteres

[1103]
    type=friend
    host=dynamic
    secret=pass
    context=prueba
    qualify=yes
    callerid=1103
```

Al igual que en el archivo *sip.conf*, el parámetro `srvlookup` sirve para resolver nombres DNS y alcanzar la máquina del servidor.

El registro de servidores se usa de la misma forma que en *sip.conf* y dentro de la sección `[general]`. No se hará uso de ésta funcionalidad ya que todos los clientes tienen direcciones IP estáticas.

```
register=>usuario:contraseña@dominio
```

Nos registraríamos con el proveedor con el nombre "`usuario`" y con la contraseña y dirección del servidor.

En caso de que el servidor use un puerto distinto a `4569` se especificará al final de la línea de la siguiente forma:

```
register=>usuario:contraseña@dominio:puerto
```

Los comandos que pueden usarse para definir el archivo *iax.conf* están definidos en la siguiente tabla:

Directiva	**General**	**Peer**	**User**	**Definición**
allow	Si	Si	Si	Códecs permitidos. Pueden ser: “gsm”, “ulaw”, “alaw”, “g729”, “speex” o “all”
disallow	Si	Si	Si	Desactiva los códecs definidos. Pueden ser: “gsm”, “ulaw”, “alaw”, “g729” o “all”
amaflags	Si			Usado para la grabación detallada de llamadas (CDR). Puede ser: “default”, “omit”, “billing”, “documentation”
srvlookup	Si			Resuelve los nombres DNS para las llamadas salientes. Valores “yes” o “no”
autokill	Si			Si no se recibe un ACK en 2000 ms (tiempo máximo estimado para una retransmisión) se considera que el host no está disponible
maxcallnumbers	Si			Limita la cantidad de llamadas permitidas para cada dirección IP remota. Una vez alcanzado el máximo de conexiones no se permite una nueva hasta que alguna de las anteriores termine
requirecalltoken		Si	Si	Validación de los tokens de las llamadas. Puede ser: “no” (opcional para esa extensión), “auto” (opcional hasta que el token soporte registro de peers) y “yes”
bandwidth	Si	Si	Si	Especifica el ancho de banda para controlar qué códecs serán utilizados en general. Puede ser: “low”, “medium” o “high”
bindaddr	Si			Dirección IP para la conexión al servidor. Se pueden unir varias, la primera se usa por defecto.
bindport	Si			Puerto UDP de conexión al servidor. Tiene que ser especificado antes que bindaddr. Por defecto 4569.
context		Si	Si	Contexto asociado en el dialplan para dicha extensión. Ver *extensions.conf*
jitterbuffer	Si			El buffer de jitter compensa la variación de retardo de una red. Funciona para el audio entrante. Puede ser: “yes” o “no”
maxjitterbuffer	Si			Define el tamaño máximo del búfer en función del tiempo (en milisegundos)
inkeys		Si	Si	Lista de claves públicas definidas en el sistema local que se pueden usar para autentificar al peer remoto. Se separan por “:”. Se definen en: */var/lib/asterisk/keys/<nombre>.pub*

outkey		Si	Si	Clave privada usada para autentificar el otro lado. Se definen en */var/lib/asterisk/keys/<nombre>.key*. Normalmente están encriptadas en 3DES
permit		Si	Si	Control del acceso de direcciones IP concretas. Se define de la forma: "dirección IP/máscara de red"
deny		Si	Si	Deniega el acceso a una dirección IP especifica. Se define como: "dirección IP/máscara de red"
qualify		Si	Si	Determina si la extensión es alcanzable en función del tiempo de respuesta de esta mediante un ping. Máximo 60 segundos
register	Si			Permite el registro con otro servidor IAX permitiéndole saber dónde estamos en caso de tener una dirección IP estática
secret		Si	Si	Contraseña para el registro en el servidor. Tipo de autenticación: md5 o texto plano
auth		Si	Si	Permite seleccionar el tipo de autenticación para registrarse con el servidor. Pueden ser: "md5", "plaintext" o "rsa". Si son md5 o plaintext, se corresponde con el comando secret. Si es rsa los nombres pueden ser definidos con el comando inkeys
encryption		Si	Si	Habilita la encriptación IAX2. Valores: "yes", "no"
mailbox		Si	Si	Número del buzón de voz
host				Dirección IP de la extensión. Puede tomar tres valores: <dirección IP>, "dynamic" que indica que cualquier IP es válida pero necesita de una contraseña para el registro, o "static" dónde cualquier IP es válida y no es necesaria contraseña
tos		Si	Si	IAX puede definir los bytes de TOS (Type Of Service) para mejorar el cálculo de la ruta. Pueden ser: "lowdelay", "throughtput", "reliability", "mincost" o "none"
trunkfreq	Si			Frecuencia de envío de los mensajes fragmentados (milisegundos)
trunk		Si	Si	En caso de estar activa la fragmentación deberá haber un temporizador hardware

Para comprobar el estado de los "`peers`" o "`users`" definidos podremos usar los comandos:

```
CLI> iax2 show peers
CLI> iax2 show users
```

Si los clientes están definidos como "`friends`" aparecerán al ejecutar ambos comandos.

EXTENSIONS.CONF

El plan de marcación (dialplan) es el corazón de la centralita, define el comportamiento y la lógica de funcionamiento. Está compuesto por distintas secciones llamadas contextos, las cuales comienzan por el nombre de dicha sección entre corchetes.

Se encuentra definido en el archivo *extensions.conf*.

Los contextos agrupan la lógica de funcionamiento del plan de marcación dependiendo de la gestión de las extensiones definidas en su interior.

La ejecución de dichos contextos es lineal, sin embargo pueden existir situaciones en las que no sea así, como un salto con una instrucción `Goto`, o un salto debido al resultado de una aplicación como puede ser un error de ejecución.

También es posible incluir contextos dentro de otros, dando lugar a una concatenación de contextos que se ejecutarán desde el más interno al más externo. En caso de tener varios incluidos dentro del mismo, el orden de ejecución será aquel en el que estén definidos.

El comando `include` permite la inclusión de contextos o de archivos de configuración. La forma de uso es la siguiente:

```
include => contexto
include “archivo.conf”
```

Es posible además ejecutar un programa o script, serán insertados dónde quede definido el comando `#exec`. Dicho comando funciona en todos los archivos de configuración, sin embargo es necesario activarlo en *asterisk.conf* con la opción “`execincludes`”

Un contexto define una forma particular de uso de una o varias extensiones.

Por ejemplo, puede enrutar una llamada en función de una extensión numérica, o puede contener la lógica para acceder a un buzón de voz.

El plan de marcación contiene los siguientes contextos o secciones:

- `[general]`: define los comandos generales del plan de marcación, como pueden ser: guardar el plan de marcación desde la línea de comandos, terminar la llamada en caso de que se quede inactiva o borrar las variables globales al reiniciar Asterisk entre otras
- `[globals]`: Define las variables globales como constantes para usarlas posteriormente en las extensiones. Pueden ser modificadas mediante el comando `SetGlobalVar` y pueden ser referenciadas mediante `${nombrevariable}`. Además, pueden ser llamadas por cualquier canal y en cualquier momento, aunque si el nombre coincide con una variable de canal, el contenido de la variable de canal prevalece sobre el de la variable global.
- `[contextos]`: Cualquier categoría distinta a "`general`" y "`globals`" representa un contexto con extensiones y aplicaciones definidas por el usuario que detallan el comportamiento de la centralita de forma precisa.

Los contextos contienen instrucciones. Éstas instrucciones son líneas del plan de marcación que tienen la siguiente sintaxis:

```
exten=>nombre_extensión, prioridad, aplicación
```

El nombre de extensión es el identificador de la extensión, puede ser una cadena numérica, un patrón o una extensión predefinida. La extensión será el destino de una llamada en Asterisk, y estará asociada a un número de teléfono o simplemente a una lógica de instrucciones para ese número.

La prioridad indica el orden de ejecución de la instrucción. Es necesario que la primera prioridad sea la `1`, la siguiente en ejecutarse será bien la que tenga una unidad más o "`n`" si no se desea especificar el orden exacto para facilitar la escritura del plan de marcación. Puede tener un alias dentro de un paréntesis después de la prioridad "`n`" para usarlas en situaciones de salto con la instrucción "`Goto`".

La aplicación define la forma de uso de dicha extensión. Las hay de muchos tipos, entre las más comunes se encuentran las aplicaciones de llamada, de reproducción de sonidos, de salto a otro contexto, etc. Cada aplicación tiene una estructura diferente y por lo general habrá que indicarle distintos parámetros para que ejecute dicha acción. En caso de que al ejecutar la aplicación de un error se saltará a la prioridad: prioridad actual+101 para resolver dicha situación, si no está definida esa prioridad finalizará la ejecución de las aplicaciones relacionadas con esa extensión.

La siguiente figura muestra un esquema de la organización de contextos dentro del dialplan:

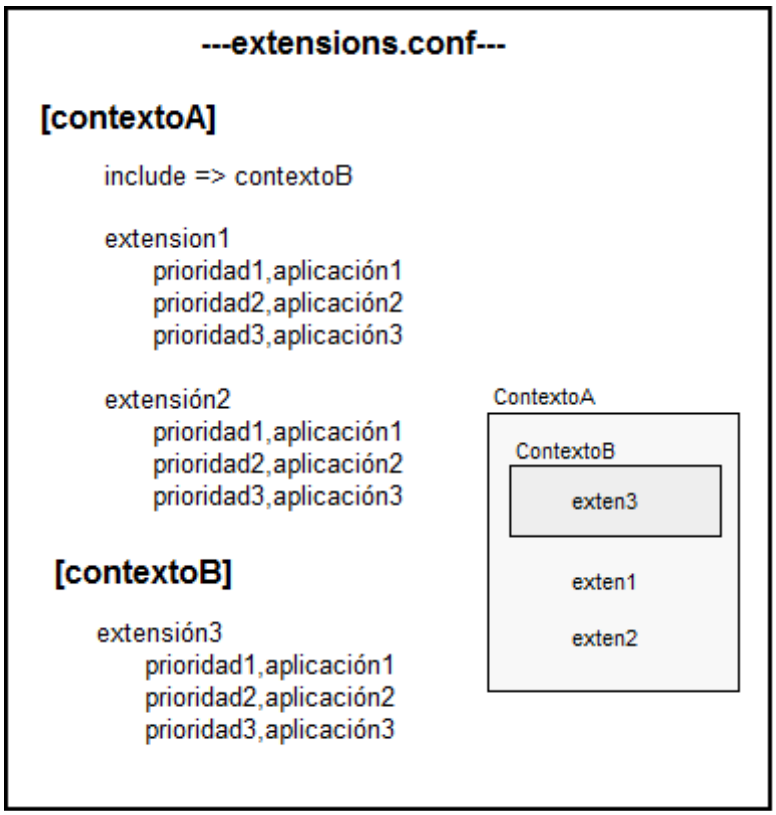


Figura 21: "Arquitectura del plan de marcación"

Un contexto puede tener una forma lineal de ejecución, o puede tener un salto a otro contexto si así está definido por la aplicación correspondiente o por el resultado de una instrucción.

El siguiente diagrama de flujo indica el funcionamiento del plan de marcación.

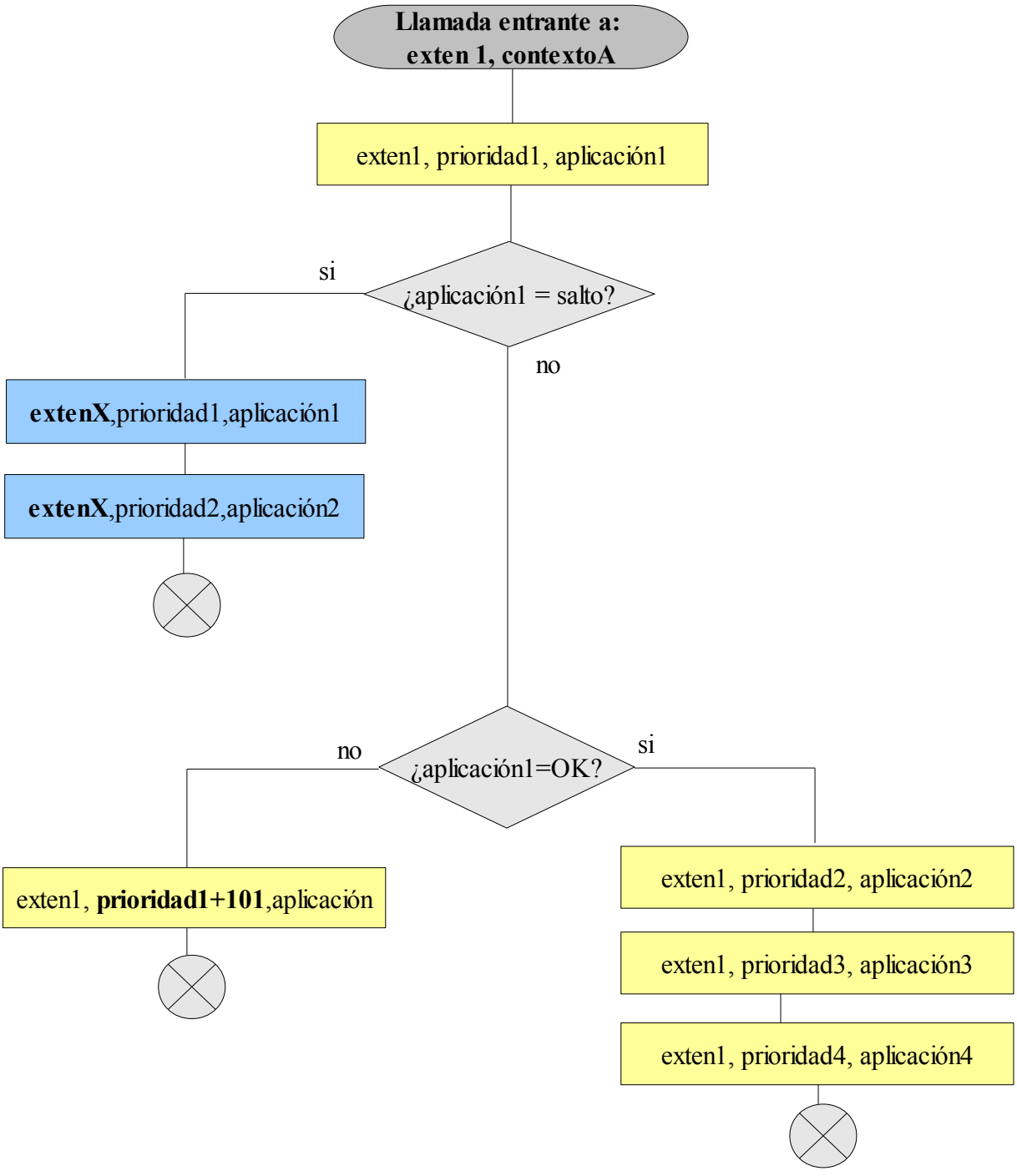


Figura 22: "Diagrama de flujo del plan de marcación"

Se usará una configuración básica de *extensions.conf* a modo de ejemplo para comunicar los canales sip e iax definidos en los archivos de configuración correspondientes. Se define un único contexto llamado `[prueba]` que define las llamadas a las extensiones `1001`, `1003`, `1101` y `1103` descritas en los archivos *sip.conf* e *iax.conf*.

```
[general]
     static=yes                ;si static=yes y writeprotect=no, para guardar los
                               cambios realizados desde la consola tendremos que
                               usar el comando "dialplan reload"
     writeprotect=no           ;en caso contrario se actualizará, pero perderemos los
                               comentarios escritos en el archivo
     autofallthrough=yes       ;Termina la llamada en curso con BUSY,
                               CONGESTION o HANGUP si se queda sin nada que
                               hacer.
     priorityjunming=yes       ;Si está activo permite el salto a la prioridad n=+101
                               en caso de error de la aplicación, dónde n es el número
                               de la línea que se está ejecutando
     clearglobalvars=no        ;Si está activo las variables globales son borradas con
                               un reinicio del plan de marcación o de Asterisk
[prueba]   ;contexto en el que están definidas las extensiones de los archivos sip.conf e
           iax.conf
     exten => 1001,1,Dial(SIP/1001,20,m) ;Al marcar la extensión 1001 se
                                         ejecutará la aplicación Dial, la cual
                                         realizará una llamada por el canal SIP a
                                         la extensión 1001 durante un tiempo
                                         máximo de 20 segundos y con la opción
                                         de música en espera activada
     exten => 1001,2,Hangup    ;cuelga la llamada y libera el canal utilizado

     exten => 1003,1,Dial(SIP/1003,20,m)
     exten => 1003,2,Hangup

     exten => 1101,1,Dial(IAX2/1101,20) ;Al marcar 1101 se realiza una llamada
                                        por el canal IAX2 a la extensión 1101
                                        (definida en el archivo extensions.conf)
                                        durante un tiempo máx. de 20 segundos
     exten => 1101,2,Hangup

     exten => 1103,1,Dial(IAX2/1103,20)
     exten => 1103,2,Hangup
```

En ésta configuración, el único contexto permite que los clientes SIP e IAX se puedan comunicar entre ellos, ya que todos están definidos en él, en sus archivos de configuración correspondientes.

Un cliente SIP como podría ser el `1001` comprobaría que su contexto asociado en el plan de marcación de *extensions.conf* es `[prueba]` y podría llamar a cualquier extensión que esté definida ahí, en éste caso podría llamar a `1003` (SIP) o a `1101` y `1103` (IAX).

Es muy importante que ninguna extensión se quede sin nada que hacer, por lo tanto se deberá cerrar cualquier conexión con el comando `Hangup`

Una forma de comprobar el funcionamiento del archivo *extensions.conf* es hacer un seguimiento paso a paso de las instrucciones ejecutadas, ésto se puede ver fácilmente desde el CLI de Asterisk, ya que en el servidor irán apareciendo los resultados de cada acción.

Un comando muy útil para la depuración de errores es `NoOp()`, ya que mostrará en la consola la línea de texto incluida o el resultado de una variable determinada. Es especialmente útil a la hora de comprobar hacia dónde va la línea de ejecución del plan de marcación cuándo ocurre un salto o un error en una aplicación.

Algunos de los comandos más útiles del CLI pueden ser el reinicio del plan de marcación o de la centralita:

```
CLI> dialplan reload
CLI> reload
```

También es posible mostrar una ayuda para las aplicaciones o funciones que deseamos buscar:

```
CLI> core show applications
CLI> core show application <command>
CLI> core show function <command>
```

4.3 Extensiones, variables y expresiones

Extensiones:

Una extensión es el destino de una llamada en Asterisk, puede ser una cadena alfanumérica, un patrón o una extensión predefinida.

Si se trata de una cadena alfanumérica podrá contener los números: `0-9`, los caracteres `#` y *, así como las letras: `ABCD`

Los patrones de marcado se usan para optimizar y agrupar extensiones con una lógica común. Comienzan con el símbolo "`_`" que identifica el comienzo del patrón.

- **`_`** comienzo del patrón
- **`X`** cualquier dígito entre 0 y 9
- **`Z`** cualquier dígito entre 1 y 9
- **`N`** cualquier dígito entre 2 y 9
- **`[14-7]`** cualquier dígito entre los corchetes, en este caso: 1, 4, 5, 6 y 7
- **`.`** cualquier carácter, 1 o más
- **`!`** el proceso termina tan pronto como se determine sin ambigüedad que no hay otras posibilidades

Asterisk también usa algunos nombres de extensiones para propósitos especiales, como son las extensiones predefinidas:

- **`s`**: Start. Se usa para entradas en un contexto que no tiene información de extensión. Ejemplo: IVR, macros...
- **`t`**: Timeout. Salta cuándo se cumple un tiempo establecido para una aplicación. Gracias a esta extensión la llamada no se pierde cuándo se cumple el plazo, sino que se lanza un mensaje y se reencamina.
- **`i`**: Invalid. Hace referencia al hecho de marcar una extensión que no está definida en el contexto. Puede manejar la extensión y reproducir un mensaje del tipo "la extensión marcada no existe". Se suele usar en IVR's
- **`T`**: Absolute Timeout. Fín de tiempo absoluto, asociada a la función TIMEOUT, salta cuándo el tiempo global de la llamada acaba independientemente del punto en el que se encuentre
- **`h`**: Hangup. Salta cuándo la otra extensión cuelga la llamada. Suele usarse para reproducir un mensaje tipo de despedida después de colgar. También se usa para liberar el canal después de terminar la llamada.

Variables:

Como en cualquier lenguaje de programación es fundamental que exista un mecanismo para almacenar variables. Generalmente se usan para simplificar el código y hacerlo más legible.

Podemos encontrar una lista completa de variables en: http://www.voip-info.org/wiki-Asterisk+variables

Una variable tiene la siguiente sintaxis:

```
${NOMBRE}
```

Es posible que en determinados casos nos interese únicamente una subcadena de una variable. Para solucionar este problema existen tres posibilidades:

- Recortar a partir de N posiciones: `${EXTEN:1}`
- Recortar las últimas N posiciones: `${EXTEN:-3}`
- Recortar posiciones específicas, considerando que la primera posición es el 0: `${EXTEN:1:3}`

Esto es aplicable a todo tipo de variables

Supongamos que tenemos una variable cuyo contenido es `${variable}=carmen`, la realización de las siguiente operaciones tendrá como resultado:

```
${variable:1}=armen
${variable:-3}=men
${variable:1:3}=ar
```

La variables pueden ser de cuatro tipos: globales, de canal, de entorno y compartidas

Variables globales:

Son las variables definidas en la sección `[globals]` del *extensions.conf* o bien son definidas usando el comando `SetGlobalVar`.

Una vez definidas pueden ser referenciadas por cualquier canal y en cualquier momento.

Si el nombre coincide con una variable de canal, el contenido de la variable de canal prevalecerá sobre el contenido de la variable global.

Un ejemplo de uso sería el siguiente, se podría definir una variable global que contenta una extensión y el protocolo necesario para efectuar una llamada y luego en *extensions.conf* hacer uso de la variable desde la aplicación `Dial()`

```
[globals]
EXT11=IAX2/ext11

[default]
exten => 111,1,Dial(${EXT11})
```

Variables de canal:

Son las variables que se pueden definir usando el comando `Set`.

Cada canal tiene su propio espacio de variables, por lo que no hay colisión entre las diferentes llamadas. Las variables son borradas automáticamente cuándo el canal termina la llamada.

Entre las muchas de las variables de canal existentes, las más usadas son las siguientes:

- `${DIALSTATUS}`: Sirve para saber el estado de la llamada, los posibles valores son: `ANSWER` (llamada respondida), `BUSY` (número ocupado), `NOANSWER` (rechazo de la llamada), `CANCEL` (cancelación de la llamada después de colgar), `CHANUNAVAIL` (canal no disponible), `CONGESTION` (estado no reconocido del número al que se llama) y `HANGUP` (finalizar la conexión)
- `${CONTEXT}`: Devuelve el nombre del contexto
- `${EXTEN}`: Devuelve el número de la extensión completa dentro del contexto en cuestión. Extensión destino de la llamada
- `${PRIORITY}`: Devuelve el número de la prioridad dentro de la extensión en cuestión
- `${CALLERID}`: Muestra el identificador del llamante, puede ser un número o cadena de texto.
- `${CHANNEL}`: Devuelve el nombre del canal en cuestión
- `${SECRET}`: Contiene la contraseña de Dundi
- `${NUMBER}`: Contiene la extensión que inicia la llamada.

Variables de entorno:

Son las relativas al manejo del entorno UNIX. Son muy poco usadas por el sistema Asterisk, ya que acceden a las variables de entorno del sistema UNIX.

La sintaxis es la siguiente:

```
${ENV(<nombre_variable>)}
```

Se podían usar por ejemplo para escribir un fichero en el directorio especificado por el nombre de la variable.

Variables compartidas:

Forman parte de una incorporación a una nueva versión de Asterisk, se pueden considerar como un subconjunto de variables locales o de canal que se comparten por dos o más canales.

La idea es extender el uso de una variable de canal a otros para cubrir necesidades muy concretas.

La sintaxis es la siguiente:

```
Set(SHARED(<nombre_variable>[,<canal>]))
```

Expresiones

Es posible realizar operaciones básicas en el manejo de variables o en llamadas a operaciones.

Los operadores que permiten las expresiones son:

- Aritméticos: + (suma), - (resta), * (multiplicación), / (división) y % (módulo)
- Comparación: = (igual), != (distinto), > (mayor que), >= (mayor o igual que), < (menor que), <= (menor o igual que)
- Booleanos & (y), | (o)

La sintaxis es la siguiente:

```
$[expresión1 operador expresión2]
```

Ejemplo de uso:

```
[general]
exten=>111,1,Set(contador=1)
exten=>111,n,Set(contador=$[${contador}+1])
```

4.4 Aplicaciones

Existen una serie de aplicaciones usadas de forma recurrente por Asterisk y que merece la pena que sean detalladas de forma específica, ya que forman parte de la construcción del plan de marcación.

Para obtener un listado de las aplicaciones disponibles basta con escribir en el CLI:

```
CLI> show applications
```

Y si lo que se desea es obtener una descripción detallada de una aplicación concreta habrá que escribir:

```
CLI> show application nombre_aplicación
```

Podemos encontrar un listado completo de aplicaciones en la página web:
http://www.voip-info.org/wiki/view/Asterisk+-+documentation+of+application+commands

Existen varias áreas de funcionamiento de las aplicaciones: aplicaciones generales, de gestión de llamadas, de control de flujo, de manipulación de variables, de reproducción de sonidos, de grabación y de conferencias y buzón de voz entre otros.

A continuación se detallan los más usados durante la realización de los casos prácticos del proyecto.

Aplicaciones generales:

- Authenticate

Autentifica a un usuario con la contraseña definida en la instrucción. Requiere que el usuario introduzca una contraseña durante la ejecución del plan de marcación.

Si el primer argumento de la aplicación `Authenticate` empieza con "`/`", se interpreta que a continuación irá la ruta de un archivo que contiene una lista de contraseñas aceptables.

```
Authenticate(password[,options[,maxdigits]])
Authenticate(/passwdfile|[,options[,maxdigits]])
Authenticate(/db-keyfamily,d[options[,maxdigits]])
```

Entre las opciones disponibles se encuentran:

`a`: Activa la variable `${ACCOUN TCODE}` con la contraseña introducida.

`d`: Interpreta la ruta como la clave de una base de datos

`j`: Salta a la prioridad n+101 si falla la autenticación y existe la prioridad

`m`: Interpreta la ruta como un archivo que contiene la contraseña con encriptación MD5

`r`: Elimina la clave de la base de datos después de la correcta autentificación (sólo aplicable con la opción "d")

`maxdigits`: número máximo de dígitos aceptables. 0 por defecto, que indica que no hay límite

- System

Ejecuta un comando del sistema de Linux. Si se necesita retornar un valor al plan de marcación habrá que hacer uso de las aplicaciones AGI o SHELL introducidas en la versión 1.6

```
System(command)
System(command arg1 arg2 etc)
System(command|args)
```

El resultado de la ejecución es devuelto en la variable de canal `SYSTEMSTATUS` y toma los siguientes valores

`FAILURE`: No se ha ejecutado el comando especificado

`SUCCESS`: El comando ha sido ejecutado satisfactoriamente

`APPERROR`: Accionado por ejemplo cuándo intenta eliminar un archivo, pero el archivo no estaba allí

Ejemplo de uso:

```
exten => s,1,system(echo "${DATETIME} - ${CALLERID} -
${CHANNEL}" >> /var/log/asterisk/calls)
```

- Wait

Espera durante un tiempo especificado. El argumento que se proporciona son el número de segundos a esperar.

Durante el tiempo de espera, todos los sonidos de entrada recibidos, incluidos tonos DTMF son ignorados.

Se usa normalmente antes de responder una llamada en un canal. La aplicación `Answer()` necesita unos milisegundos de espera para poder responder.

```
Wait(seconds)
```

El argumento no puede ser ni cero ni un número negativo, pero sí que sirven las fracciones como 0.3 ó 1.5

- WaitExten

Espera durante un tiempo máximo a que una extensión sea introducida por un usuario.

Normalmente se usan en IVR (Interactive Voice Response) dónde se pedirá que se introduzca una extensión determinada y de ahí se saltará a una parte concreta del plan de marcación.

Se puede definir como complementaria a la aplicación `Background()`, ya que ésta lanza una pista de audio para después capturar los tonos de marcación.

```
WaitExten(seconds)
WaitExten([seconds][|options])
```

La única opción disponible es:

`m[(x)]`: reproduce una pista de audio mientras espera que se introduzca una extensión. Opcionalmente se puede especificar la clase de música en espera entre paréntesis

Esta aplicación no funciona en Macros.

Habrá que considerar el ajuste "`autofallthrough`" definido en el archivo *extensions.conf*, ya que éste si está activo terminará la llamada en curso si se queda sin nada que hacer. En caso de que esté desactivada esta opción habrá que definir `WaitExten(n)` para forzar el cierre de una extensión.

En caso de que se cumpla el tiempo definido en la aplicación, el resultado dirigirá a la extensión estándar "`t`". Si la marcación no existe, redirige a la extensión estándar para extensiones inexistentes "`i`".

Gestión de llamadas:

- Answer:

Acepta la llamada entrante por el canal. En función de las aplicaciones que vengan a continuación puede que sea necesario que Asterisk controle éste canal.

El ciclo de facturación comenzaría al descolgar la llamada, por lo que si no es estrictamente necesario es mejor no utilizarla.

Si se indica un retraso, Asterisk esperará el número de milisegundos especificados después de responder la llamada. Si se quiere añadir un retraso anterior será necesario usar la aplicación `Wait(n)`.

```
Answer([delay])
```

La única opción que permite es:

`delay`: indica si debe esperar un número determinado de milisegundos después de contestar la llamada

- Dial

Sirve para realizar una llamada a un dispositivo concreto pasándole los argumentos propios del dispositivo destino

Establece una nueva conexión saliente en un canal y lo vincula con el canal de entrada existente. No importa la tecnología o protocolos usados, un usuario SIP puede llamar por ejemplo a un usuario IAX.

De la traducción y codificación de códecs de los distintos medios se encarga el sistema Asterisk.

```
Dial(type/identifier, timeout, options, URL)
```

Definición de parámetros:

- `type`: especifica el tipo de canal. Será uno de los siguientes tipos: Zap, SIP o IAX2
- `identifier`: especifica el número de teléfono a llamar en ese canal. El formato depende del canal y puede tener parámetros adicionales. Por ejemplo IAX2 permite una mayor precisión en la llamada
- `timeout`: este parámetro es opcional. Define el tiempo máximo de espera para que conteste un canal. Si no se especifica, esperará indefinidamente terminando sólo cuándo el canal cuelgue la llamada o retorne una señal de error u ocupado.
- `options`: este conjunto de parámetros también son opcionales, se detalla su uso a continuación.
- `URL`: opcional. Se envía al canal destino si la conexión sucede con éxito.

Opciones:

- `A(x)`: Reproduce un fichero antes de establecer la comunicación. El fichero será en este caso *x.gsm*
- `h`: permite al destinatario de la llamada colgar pulsando la tecla * (definida en *features.conf* → `featuremap` → `disconnect`)
- `H`: permite al llamante colgar pulsando * (definido igualmente en *features.conf*)
- `j`: Salta a la prioridad n+101 si todos los canales a los que ha llamado están ocupados.
- `k`: permite al destinatario parquear la llamada mediante la combinación de teclas definidas para el "call parking" definidas en el archivo *features.conf*
- `K`: igual pero para el llamante
- `m`: provee música en espera durante la llamada mientras espera que el destino conteste.
- `t`: permite al destinatario transferir la llamada pulsando la combinación de teclas definidas en *features.conf*. No afecta a las transferencias iniciadas a través de otros métodos
- `T`: igual pero para el llamante
- `w`: permite al destinatario iniciar una grabación al pulsar `*1` o aquello que esté definido en *features.conf*. Es necesario que `DYNAMIC_FEATURES=automon`, por lo que habrá que habilitarlo con la aplicación `Set()`.
- `W`: igual pero para el llamante
- `x`: permite al destinatario iniciar una grabación de ambos canales pulsando `*1` o aquello que esté definido en *features.conf*
- `X`: igual pero para el llamante

Binomio `type/identifier` según el protocolo:

- SIP: `SIP/[exten@]peer[:portno]`
  - `exten`: si está definida Asterisk la usará para conectar con la extensión.
  - `peer`: nombre del peer a conectar. Puede ser un `peer` o `friend` definido en *sip.conf*, una dirección IP, o un nombre de dominio
  - `portno`: puerto UDP definido, si se omite Asterisk usará el puerto por defecto para SIP, `5060`

- IAX2: `IAX2/[<user>[:<secret>]@]<peer>[:<portno>]`
`[/<exten>[@<context>][/<options>]]`

`user`:	Identificador de usurario del peer remoto o nombre del cliente configurado en *iax.conf* (opcional)
`secret`:	Contraseña (opcional). Opcionalmente puede ser el nombre de un archivo de clave RSA sin la extensión *.key* o *.pub*
`peer`:	Nombre del servidor al que se va a conectarse
`portno`:	Número de puerto para la conexión en el servidor (opcional)
`exten`:	Extensión en el servidor Asterisk remoto (opcional)
`context`:	Contexto a usar en el servidor Asterisk remoto (opcional)
`options`:	La única opción disponible es "`a`", que significa respuesta requerida.

Respuesta de dial, variable `DIALSTATUS`:

Esta variable contiene información sobre el último `Dial` realizado. Puede indicar:

- `CHANUNAVAIL`: canal no disponible
- `BUSY`: el destino está ocupado
- `NOANSWER`: sin respuesta
- `CANCEL`: llamada cancelada, el usuario cuelga antes de que se conecte la llamada
- `DONTCALL`: la política de privacidad impide la llamada
- `TORTURE`: la política de privacidad envía al menú de torture
- `CONGESTION`: indica congestión o cualquier cosa como un error al realizar la llamada.

Casos extra relacionados con la aplicación `Dial`:

- Llamada a varios a la vez:

```
Dial(type1/identifier1[&type2/identifier2[&type3/identifier3...]],
timeout, options, URL)
```

La aplicación `Dial()` permite llamar a un sólo dispositivo esperando a que éste conteste o llamar a varios, en tal caso sonarán todos a la vez, y una vez establecida la comunicación con el primero en descolgar se colgarán el resto de llamadas y por lo tanto dejarán de sonar.

- Rellamada:

```
RetryDial(announce|sleep|loops|Technology/resource
[&Technology2/resource2...][[timeout][|[options][|URL]]])
```

Otra de las aplicaciones relacionadas con esta aplicación es `RetryDial`, que permite realizar una rellamada en un tiempo determinado. Asterisk realiza la llamada, en caso de error reproduce el mensaje "`announce`", espera el tiempo definido en "`sleep`" y repite hasta el máximo de reintentos definido por "`loops`"

- Hangup

Cuelga incondicionalmente el canal de la llamada y devuelve `-1`. Si se especifica el comando "`causecode`" será el valor devuelto al finalizar la llamada..

Es importante hacer uso de esta aplicación al final del proceso relativo a una extensión, para que, si por ejemplo el destinatario de la llamada cuelga la llamada no se quede colgado el canal dedicado a dicha llamada en nuestro dispositivo.

```
Hangup(<causecode>)
```

El parámetro "`causecode`" será definido según el código ISDN

Control de flujo

- Goto

Salta la ejecución del plan de marcación a un contexto, extensión y prioridad definidos. Si sólo se pasa un parámetro se sobreentiende que se trata de una prioridad dentro del mismo contexto.

```
Goto([[context|]extension|]priority)
```

Se pueden realizar saltos en base a:

- prioridad (o etiqueta)
- extensión|prioridad (o etiqueta)
- contexto|extensión|prioridad (o etiqueta)

Si queremos saltar a un contexto, extensión y prioridad determinadas habrá que configurar los tres parámetros de la aplicación. Si por el contrario sólo queremos movernos dentro de nuestro contexto bastará con configurar los parámetros extensión y prioridad. Y si lo que se quiere realizar es un salto dentro de la misma extensión a otra línea distinta habrá que configurar la prioridad.

- GotoIf

Salta a una zona del plan de marcación en función del resultado de la expresión condicional.

Si es verdadera saltará a la primera etiqueta, y si es falsa continúa en la segunda etiqueta o en la siguiente prioridad de forma natural si no se especifica nada.

```
GotoIf(condition?label1[[:label2])
```

```
GotoIf(condition?[context1],[extension1],[priority1]:
[context2],[extension2],[priority2])
```

Los parámetros de los que se compone la aplicación son:

- `condition`: Condición a considerar para realizar el salto. En caso de no especificarse o ser `0` se considerará falsa y saltará a la segunda etiqueta
- `label1`: Etiqueta a la que salta cuándo la condición es verdadera
- `label2`: Etiqueta a la que salta cuándo la condición es falsa (opcional). Si no se especifica se continúa el curso natural del plan de marcación
- `[context1],[extension1],[priority1]`: Contexto, extensión y prioridad a la que salta cuándo la condición es verdadera. Son opcionales de forma individual y las posibles combinaciones son las definidas para la aplicación `Goto`
- `[context2],[extension2],[priority2]`: Contexto, extensión y prioridad a la que salta el código cuándo la condición es falsa. Son opcionales de forma individual y también lo es el conjunto entero

Esta aplicación permite realizar multitud de algoritmos de programación, uno de ellos podría ser por ejemplo un bucle for en el cual se va sumando un contador hasta que llega a 10. Para ello simplemente habrá que hacer una llamada a la extensión 111 y ésta se encargará de realizar todo el proceso.

Ejemplo FOR:

```
[bucle]
exten=>111,1,Set(i=1)
exten=>111,n(for),GotoIf($[${i}>10]?fin_for)
exten=>111,n,Set(i=$[${i}+1])
exten=>111,n,Goto(for)
exten=>111,n(fin_for),...
```

- GotoIfTime

Esta aplicación realiza un salto condicional en función de la hora definida.

Es muy útil en IVR's, para determinar si nos encontramos en horario de oficina o no, por ejemplo.

Sin embargo este uso obliga a modificar el plan de marcación continuamente para poder tenerlo actualizado. En la versión 1.8 se amplía la funcionalidad, en lugar de `GotoifTime` se usa `GotoIf` pero aplicando un recurso que sincronice los calendarios de diversos tipos como Exchange Server, iCalendar o Google Calendar a través una una URL.

```
GotoIfTime(<time range>,<days of week>,<days of month>,<months>?
[labeliftrue][:labeliffalse])
```

La definición de los parámetros es la siguiente:

- `time range`: define un rango horario de la forma: `<hora>':'<minuto>'-'<hora>':'<minuto>,` o define todo el rango horario de la forma: "*"
- `hora`: número entre 0 y 23, ambos incluidos
- `minuto`: número entre 0 y 59, ambos incluidos
- `days of week`: especifica un único día de la semana: `<diasem>`, usa un rango de días: `<diasem>'-'<diasem>`, o define todo el rango de días: "*".
- `diasem`: será uno de los siguientes días de la semana: `"sun"|"mon"|"tue"|"wed"|"thu"|"fri"|"sat"`
- `days of month`: especifica un único día del mes: `<diames>`, usa un rango de días: `<diames>'-'<diames>`, o define todo el rango del mes: "*"
- `diames`: número entre 1 y 31, ambos incluidos
- `months`: especifica un mes concreto: `<nombremes>`, un rango de meses: `<nombremes>'-'<nombremes>`, o el conjunto de todos los meses: "*"
- `nombremes`: será uno de los siguientes meses: `"jan"|"feb"|"mar"|"apr"|"may"|"jun"|"jul"|"aug"|"sep"|"oct"|"nov"|"dec"`
- `labeliftrue`: zona hacia dónde salta el programa cuándo se cumplen los requisitos de hora y fecha especificados.
- labeliffalse: zona hacia dónde salta el programa cuándo no se cumplen los requisitos.

Ejemplos de uso:

```
exten => 111,1,GotoifTime(10:00-14:00,*,12,oct?abierto:cerrado)
```

En este caso, la oficina estará abierta de 10 a 14 el día 12 de octubre, en otro horario estará cerrada

- NoOp

Esta aplicación no realiza ninguna operación.

Muestra los mensajes definidos durante la ejecución del código del plan de marcación en la línea de comandos de consola en el siguiente salto del código.

Los mensajes pueden ser una cadena de texto o el contenido de una variable.

Es muy útil a la hora de depurar el código, saber el estado de la variable o localizar dónde se encuentra la ejecución del programa en ese momento. Para mostrar el contenido de una variable, el nivel de `verbose` debe ser 3 o superior.

```
NoOp()
```

Ejemplos de uso:

```
exten => 111,1,NoOp(“prueba”)
exten => 222,1,NoOp(${EXTEN})
```

Manipulación de variables

-SetGlobalVar

Guarda un valor en una variable global especificada.

Si en el archivo extensions.conf tiene desactivada la opción `clearglobalvars`, entonces las variables globales permanecerán intactas a pesar de que se reinicie el sistema.

```
SetGlobalVar(variablename=value)
```

Definición de parámetros:

- `variablename`: nombre de la variable global a la que se le va a asignar un valor
- `value`: valor asociado a la variable

-Set

Guarda un valor en la variable. Se usa para definir los valores en las variables de canal o en las funciones del plan de marcación

```
Set(variablename=value|options])
```

Definición de parámetros:

- `variablename`: nombre de la variable
- `value`: valor asociado
- `options`: Sólo tiene una posible opción, "`g`", que hace la variable global.

Reproducción de sonidos

-Background

Reproduce un archivo o archivos de fondo mientras está a la espera de una extensión por parte del llamante. Devuelve el control a Asterisk, el cual puede ejecutar el plan de marcación mientras el audio continúa reproduciéndose.

Es habitual utilizarla junto a la aplicación `WaitExten`, dado que es importante establecer una restricción de tiempo.

Los dígitos marcados serán los que se utilicen para realizar una combinación a usar en el plan de marcación.

Por lo general se usa para reproducir el mensaje de bienvenida de un IVR.

```
Background(filename1[&filename2...][|options[|langoverride]
[|context]])
```

Parámetros:

- `filename`: archivo de audio a reproducir
- `options`: "`s`" si el canal aún no ha respondido con la aplicación `Answer` (o cualquier otra), entonces este `background` no se aplicaría,
  "`n`" trata de no responder al canal utilizando el sistema "Early Audio" hasta que no se detecta la primera marcación,
  "`m`" tan pronto detecta una asociación con una extensión para el mensaje de audio. Es muy útil cuándo queremos hacer un menú que consta de pulsaciones de un solo dígito.
- `langoverride`: especifica el lenguaje esperado en los archivos de audio (opcional)
- `context`: contexto en el plan de marcación dónde se buscará la extensión marcada (opcional)

Ejemplo:

```
exten => s,1,Answer
exten => s,2,Background(gracias)      //"gracias por llamar, presione 1 para ventas, 2 para soporte"
exten => s,3,WaitExten()
exten => 1,1,Goto(ventas,s,1)
exten => 2,1,Goto(soporte,s,1)
```

- Echo

Devuelve y reproduce el audio leído por la aplicación anterior.

```
Echo()
```

Ejemplo:

```
exten => 1,1,Playback(demo)
exten => 1,2,Echo
exten => 1,3,Playback(correcto)
```

- MusicOnHold

Reproduce la música en espera de forma indefinida.

Si se define la música en espera en el archivo *musiconhold.conf*, ésta se reproducirá automáticamente si la extensión queda en espera en cualquier momento.

```
MusicOnHold([class],[duration])
```

Parámetros:

- `class`: clase de música en espera. Si se omite, se utilizará la música en espera por defecto (opcional)
- `duration`: Duración en segundos de la música en espera, si se omite se reproducirá indefinidamente (opcional)

Generalmente se activa la música en espera en las siguientes situaciones:

- Cuándo llamamos a una extensión y se origina la llamada
- Cuándo se está conectando a un extensión particular
- Cuándo una llamada pasa por una red de pago
- Cuándo se está conectando a una conferencia

- Playback

Reproduce uno o más archivos de audio.

No es necesario especificar las extensión del archivo de audio. Los archivos son buscados en el directorio */var/lib/asterisk/sounds/* por defecto.

La diferencia con la aplicación `Background` es que `Playback` reproduce todo el archivo de audio hasta el final y no retorna el control hasta que termina la reproducción.

```
Playback(filename1[&filename2...][,options])
```

Parámetros:

- `filename`: nombre del archivo a reproducir (sin extensión)
- `options`:
  - `skip`: reproduce el archivo de sonido sólo si el canal ha contestado. Si no es así, el comando `Playback` termina sin reproducir nada.
  - `no answer`: reproduce el archivo de sonido, pero no es necesario que el canal haya contestado antes.
  - `j`: la aplicación salta a la prioridad n+101 si el archivo especificado no existe.
  - `say`: usa *say.conf* para interpretar la cadena

-SayDigits

Reproduce los dígitos de uno en uno en el lenguaje elegido (inglés por defecto).

```
SayDigits(digits)
```

- `digits`: puede ser bien unos dígitos definidos como una cadena numérica, o el contenido de una variable.

Grabación

- MixMonitor

Graba una llamada en el archivo especificado. Continúa la ejecución del plan de marcación normalmente.

Si no se especifica una ruta para guardar el archivo, se hará por defecto en */var/spool/asterisk/monitor/*.

```
MixMonitor(<file>.<ext>[|<options>[|<command>]])
```

Parámetros:

- `file.ext`:archivo y extensión dónde se guardará el archivo de audio de la grabación.
- `options`: `b`: sólo guarda el audio del archivo mientras la llamada está activa
  `a`: anexa el archivo en lugar de sobreescribirlo
  `v(x)`: ajusta el volumen de entrada según el factor de x: -4/4
  `V(x)`: ajusta el volumen de salida según el factor de x: -4/4
  `W(x)`: ajusta el volumen general según el factor de x: -4/4
- `command`: ejecuta el comando definido mientras dura la grabación.

- Record

Permite grabar un fichero de audio con el nombre y el formato definidos. Finaliza la grabación pulsando la tecla `#`.

Si no se especifica una ruta concreta dónde guardar el archivo será guardado en la ruta por defecto: */var/lib/asterisk/sounds/*.

Si existe un fichero con el mismo nombre y extensión, éste será reescrito. Es posible usar "`%d`" en el nombre del fichero, así se utilizará un índice que se incrementará automáticamente para no sobreescribir grabaciones. El nombre del fichero queda guardado en la variable de canal `${RECORDED_FILE}.`

```
Record(filename.format[|silence][|maxduration][|option])
```

Parámetros:

- `filename`: nombre del archivo de audio, puede contener los caracteres "`%d`" para incrementar en una unidad el nombre del archivo.
- `format`: formato usado para el archivo de audio, puede ser: `sln`, `g723`, `g729`, `gsm`, `h263`, `ulaw`, `alaw`, `vox`, `wav`, `WAV`
- `silence`: segundos de silencio permitidos antes de que la grabación pare. Si no se especifica o es `0`, la detección de silencio está deshabilitada (opcional)
- `maxduration`: duración máxima de la grabación en segundos. Si no se especifica o es `0`, no existe máximo. (opcional)
- `option`:
  - `a`: anexa el registro existente en lugar de reemplazarlo.
  - `n`: no contesta, pero graba cualquier cosa si la línea no ha sido descolgada
  - `q`: no reproduce el tono de "beep"
  - `s`: salta la grabación si la línea no ha sido descolgada
  - `t`: usa '*' en lugar de '#' para terminar la llamada
  - `x`: ignora todas las pulsaciones DTMF y continúa grabando hasta colgar
  - `k`: mantiene el archivo guardado después de colgar
  - `y`: termina la grabación si se recibe cualquier tono DTMF

El comando `Record` lanza un "beep" al canal cuándo empieza la grabación. Finaliza la grabación cuándo llega al máximo de silencio o duración máxima especificadas, cuándo se pulsa `#` o cuándo el canal se desactiva.

Conferencias y buzón de voz

- meetMe

Permite a los usuarios que marquen el número de extensión asociado con la aplicación `MeetMe`, entrar a una sala de conferencias.

Si se omite el número de conferencia, el usuario tendrá que indicar uno.
Es necesario un temporizador de Zaptel para que la conferencia funcione.

Permite a los usuarios subir o bajar el volumen de entrada y salida

```
MeetMe([confno][,[options][,pin]])
```

Parámetros:

- `confno`: número de conferencia. Está definida en el archivo *meetme.conf*. Si se omite, el usuario tendrá que especificar la conferencia a la cual quiere conectarse.
- `options`:
  - `1`: elimina el mensaje "eres el único en la sala" cuándo sólo hay un usuario
  - `c`: dice el número de usuarios conectados al entrar uno nuevo en la sala
  - `M`: habilita la música en espera cuándo en la conferencia hay un sólo usuario
  - `p`: permite a los usuarios abandonar la conversación pulsando la tecla `#`
  - `q`: no se lanzan pistas de audio a los participantes cuándo un usuario entra y sale de la sala
  - `d`: crea una sala de forma dinámica, si no está habilitado habrá que escoger una sala de la lista de salas en el fichero de configuración estático, no se puede crear una.
  - `v`: permite hacer videoconferencias
  - `r`: permite grabar la conferencia
  - `T`: permite la detección de la persona hablante en cada momento
  - `X`: permite abandonar la conferencia pulsando un dígito que ha de considerarse extensión, dentro de la lista de extensiones del mismo contexto en el que se ejecuta la aplicación `MeetMe`
  - `x`: cierra la conferencia cuándo el último usuario la abandona
  - `w`: no permite que se inicie la conferencia hasta que un usuario marcado acceda a la misma
  - `a`: modo administrador activado

-VoiceMail

Permite dejar un mensaje de voz en uno o varios buzones. Guarda el audio del canal en un archivo del buzón de voz configurado en *voicemail.conf*.

Los mensajes de voz serán guardados dentro del directorio de entrada para cada número del buzón de voz: */var/spool/asterisk/voicemail/context/boxnumber/INBOX/*

```
VoiceMail(boxnumber[@context][&boxnumber2[@context]][&boxnumber3],
[flags])
```

Parámetros:

- `boxnumber`: número del buzón de voz definido en el archivo *voicemail.conf*
- `context`: contexto asociado al buzón de voz (opcional)
- `flags`:
  - `s`: Omite las instrucciones: por favor deje su mensaje después del tono, cuándo haya terminado, cuelgue o presione la tecla `#`
  - `u`: Reproduce el mensaje de no disponible: La persona en la extensión "x" no está disponible.
  - `b`: Reproduce el mensaje de ocupado: La persona de la extensión "x" está ocupada
  - `g(#)`: Ajusta la ganancia de la grabación. El símbolo `#` es un entero que representa la cantidad de ganancia en decibelios. (sólo si se especifica como segundo argumento)

En la variable de canal `VMSTATUS` quedará registrado el estado de la ejecución de la aplicación `VoiceMail`. Los posibles valores son:

- `SUCCESS`: El llamante dejó el mensaje correctamente.
- `USEREXIT`: El llamante dejó el buzón de voz presionando las teclas correspondientes
- `FAILED`: El llamante no dejó el mensaje debido a un error.

- VoiceMailMain

Entra en el sistema del buzón de voz para comprobar los mensajes.

```
VoiceMailMain([[[s]mailbox][@context])
```

Parámetros:

- `s`: si aparece esta opción, se omite la comprobación de la contraseña.
- `mailbox`: número del buzón de voz
- `context`: si se especifica el contexto, el acceso se considera únicamente dentro del mismo contexto

MENU

- **1** Read voicemail messages
  - **3** Advanced options
    - **1** Reply
    - **2** Call back(1)
    - **3** Envelope
    - **4** Outgoing call(1)
    - **5** Send Message (only available if sendvoicemail=yes in voicemail.conf)
  - **4** Play previous message
  - **5** Repeat current message
  - **6** Play next message
  - **7** Delete current message
  - **8** Forward message to another mailbox
    - **1** Use Voicemailnumber (only available if usedirectory=yes in voicemail.conf)
    - **2** Use Voicemail Directory (only available if usedirectory=yes in voicemail.conf)
  - **9** Save message in a folder
    - **0** Save in new Messages
    - **1** Save in old Messages
    - **2** Save in Work Messages
    - **3** Save in Family Messages
    - **4** Save in Friends Messages
  - * Help; during msg playback: Rewind
  - # Exit; during msg playback: Skip forward
- **2** Change folders
  - **0** Switch to new Messages
  - **1** Switch to old Messages
  - **2** Switch to Work Messages
  - **3** Switch to Family Messages
  - **4** Switch to Friends Messages
- **3** Advanced Options
  - **5** Send Message (only available if sendvoicemail=yes in voicemail.conf)
    - **1** Use Voicemailnumber (only available if usedirectory=yes in voicemail.conf)
    - **2** Use Voicemail Directory (only available if usedirectory=yes in voicemail.conf)
- **0** Mailbox options
  - **1** Record your unavailable message
  - **2** Record your busy message
  - **3** Record your name
  - **4** Record your temporary message (new in Asterisk v1.2)
    - **1** Record your temporary message
    - **2** Erase your temporary message (going back to the standard message)

    - **5** Change your password
    - ***** Return to the main menu
- ***** Help
- # Exit

Después de grabar un mensaje (entrante, de ocupado o no disponible o un nombre)
-1: Acepta
-2: Vuelve a reproducir
-3: Vuelve a grabar
-0: Vuelve a pedir la operación

4.5 Funcionalidades como PBX

Aparte de las posibilidades que ofrece Asterisk dentro del plan de marcación, existen diversos ficheros que ayudarán a configurar otras funcionalidades propias de una centralita, como pueden ser: transferencias de llamadas, multiconferencia, gestión de la música en espera para una llamada, respuestas de voz interactivas mediante una grabación, registro de las llamadas realizadas y otras configuraciones extras.

Todas estas configuraciones estarán basadas en distintos archivos de Asterisk que se detallarán a continuación.

Transferencias

Entre las muchas opciones de mejora en las prestaciones que permite la centralita Asterisk se encuentra la transferencia de llamadas a otras extensiones

Para configurar este servicio se hará uso del archivo *features.conf* ubicado en el directorio */etc/asterisk/*, que definirá la opciones de transferencia, la ubicación de la llamada en espera o la recuperación de una llamada que está esperando a ser contestada. Así como la combinación de teclas a pulsar para realizar cualquier operación descrita.

Dicho archivo contiene los siguientes contextos:

- `[general]:` define las opciones generales de parqueo, contextos, almacenamiento, sonidos, quién puede transferir las llamadas, música en espera, etc.
- `[featuremap]:` define la combinación de teclas a pulsar para realizar cualquiera de las opciones permitidas como pueden ser: transferir una llamada, terminar, grabar, realizar una transferencia atendida, aparcar una llamada, etc.

  Estas funcionalidades sólo son posibles si Asterisk hace de puente entre las llamadas, es decir, el contexto usado tiene que estar incluido en *extensions.conf* y además la aplicación `Dial()` tiene que tener definidas las opciones correspondientes.
- `[applicationmap]:` en esta parte del archivo se pueden añadir funcionalidades particulares que luego se pueden usar durante una llamada, para ello habrá que definir una sintaxis específica con el nombre de la funcionalidad, la secuencia de tonos DTMF, quién tiene acceso, la aplicación a ejecutar, los argumentos, y la música en espera que se escuchará mientras se ejecuta la funcionalidad.

Una configuración básica del archivo *features.conf* podría ser la siguiente:

```
[general]
     parkext => 700              ;extensión a marcar para parquear una llamada
     parkpos => 701-710          ;extensiones reservadas para parquear una llamada
     context => parkedcalls      ;contexto usado para parquear las llamadas, tiene que
                                 estar incluido en extensions.conf para que sea válido
     parkinghints = yes          ;permite la monitorización de dónde están parqueadas
                                 las llamadas (comando "parkedcalls show" del CLI)
     parkingtime => 45           ;número de segundos en los que una llamada puede
                                 estar parqueada
     comebacktoorigin = yes      ;Si está activo la llamada parqueada después del
                                 tiempo especificado en parkingtime vuelve a la
                                 extensión que la parqueó.
     transferdigittimeout => 3       ;tiempo de espera entre dígitos a la hora de
                                     introducirlos para una transferencia
     courtesytone = beep         ;sonido que escucha la persona que responde a una
                                 llamada parqueada o cuándo empieza y termina la
                                 grabación de la llamada
     parkedplay = caller         ;a quién enviar el 'beep' cuándo se llama la extensión
                                 parqueada. Puede ser: callee (llamado), caller
                                 (llamante), both (ambos). Se deshabilita con "no"
     parkedcalltransfers = caller ;habilita o deshabilita la secuencia de tonos
                                  para transferir la llamada cuándo ya había sido
                                  transferida.
     parkedcallreparking = caller ;habilita o deshabilita la secuencia de tonos
                                  para parquear una llamada que ya estaba
                                  parqueada.
     parkedmusicclass = default   ;tipo de música en espera que escuchará la
                                  extensión que ha sido parqueada.
     xfersound = beep            ;sonido que indica que la transferencia se completó
                                 con éxito
     xferfailsound = beeperr     ;sonido que se escucha cuándo falla la transferencia
     findslot => next            ;mete la llamada en la siguiente posición libre del
                                 parking, por ejemplo si la 701 está ocupada la mete en
                                 la 702, así hasta completar las definidas en parkpos
     pickupexten = *8            ;es la extensión que hay que marcar para capturar una
                                 llamada del mismo pickupgroup, definido en sip.conf
     featuredigittimeout = 500       ;tiempo máximo de espera entre dígitos de
                                     activación
     atxfernoanswertimeout = 15      ;tiempo máx para contestar una llamada
                                     transferida con el método asistido.
```

```
    atxferdropcall = no          ;si quién transfiere una llamada con el método asistido
                                 cuelga antes de que la llamada sea transferida
                                 correctamente, Asterisk devuelve la llamada a quién la
                                 estaba transfiriendo. Si está activo la llamada no se
                                 devuelve y se considera terminada
    atxferloopdelay = 10         ;número de segundos de espera antes de devolver la
                                 llamada (si atxferdropcall=no)
    atxfercallbackretries = 2    ;número de veces que se intentará devolver la
                                 llamada
[featuremap]
    blindxfer => #1              ;secuencia de teclas a marcar para empezar la
                                 transferencia de una llamada. Transferencia a ciegas,
                                 por defecto #
    disconnect => *0             ;secuencia de teclas a marcar para terminar la
                                 llamada o para recuperar una llamada en una
                                 transferencia a ciegas
    automon => *1                ;secuencia de teclas a marcar para grabar una
                                 llamada (generará un archivo por interlocutor)
    atxfer => *2                 ;secuencia de teclas para una transferencia de llamada
                                 "asistida"
    parkcall => #7               ;secuencia de teclas para parquear la llamada.
                                 Podemos usar esta secuencia o transferir directamente
                                 la llamada a la extensión 700 con: #1 700
    automixmon => *3             ;secuencia de teclas para grabar la llamada en un
                                 único archivo mezclando las voces de los dos
                                 interlocutores.
```

Hay dos formas de transferir una llamada, a ciegas o de forma asistida.

La transferencia a ciegas se realiza con el comando `blindxfer`. Se usa para transferir una llamada a una extensión determinada mientras dura una conversación, una vez transferida la llamada quién la desvió sale de la conversación. Se pulsan la teclas `#1` y a continuación el número de extensión a la cual queremos desviar la llamada.

Un situación en la que se emplee sería en una llamada entre dos extensiones, y que una en un momento determinado decida pasar la llamada a otra extensión. Podrá desviar la llamada aquel que tenga los permisos de transferencia definidos en `parkedcalltransfers`, ya que se dan para el llamante, para el llamado, o para ambos.

La transferencia asistida se realiza con el comando `atxfer`. Se usa para transferir una llamada a una extensión anunciando al interlocutor que se le va a transferir una llamada. Una vez el intermediario ha anunciado al destinatario que se va a transferir, cuelga y la llamada sigue su curso. Se pulsan las teclas `*2` y a continuación el número de extensión a la cual se va a desviar la llamada.

Un ejemplo podría ser la llamada a una oficina en la cual se llama a recepción y desde ahí se llama al director para ver si desvía o no la llamada iniciada.

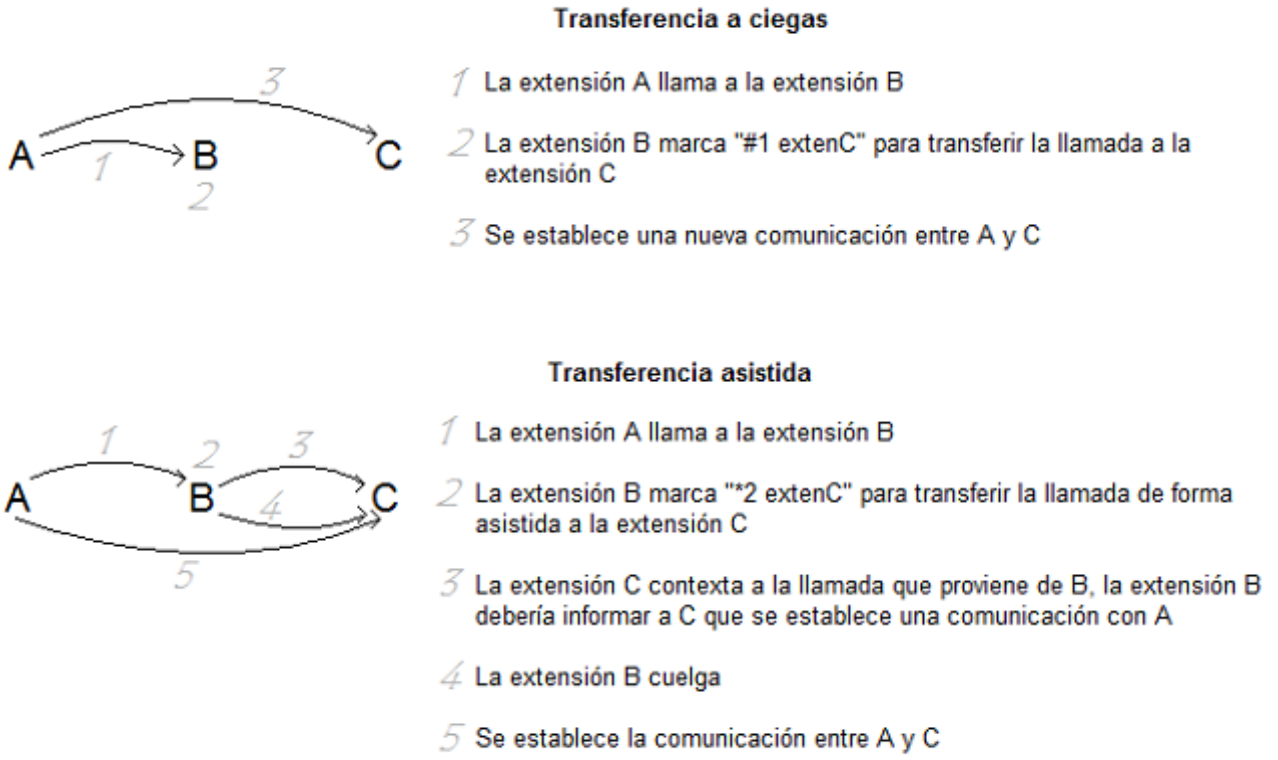


Figura 23: "Transferencia a ciegas y transferencia asistida"

Aparte de configurar dicho archivo, habrá que incluir el contexto `[parkedcalls]` en el archivo *extensions.conf* ubicado en */etc/asterisk/* para que sean posibles las transferencias y el parqueo.

La aplicación `Dial()` deberá recoger entre sus opciones las posibilidades asociadas a cada uno de los comandos asociados.

Opción	Valor de ejemplo	Opción de Dial()
blindxfer	#1	T, t
disconnect	*0	H, h
automon	*1	W, w
atxfer	*2	T, t
parkcall	#7	K, k
automixmon	*3	X, x

Para comprobar la ubicación de las llamadas parqueadas, desde el CLI se podrá ejecutar el comando: `parkedcalls show`

Conferencias

Otra de las opciones que hacen de Asterisk una de las mejores soluciones en el mercado es la posibilidad de crear conferencias de audio entre varias extensiones en tiempo real.

La aplicación `MeetMe()` permite que varios conferenciantes conversen juntos como si estuvieran en la misma localización física.

Las principales características son:

- La posibilidad de crear conferencias protegidas con contraseña de acceso
- Administración de la conferencia (silencios, bloqueos de participantes...)
- La opción de crear una conferencia estática o dinámica

Las opciones de configuración para el sistema de conferencias de `MeetMe` se encuentran definidas en el archivo *meetme.conf* ubicado *en /etc/asterisk/.*

Este archivo de configuración está dividido en los siguientes contextos:

`-[general]`: define los parámetros generales de las conferencias y su uso, así como la información de conexión y desconexión de usuarios y el control de buffer del audio para evitar distorsiones

`-[rooms]`: define las múltiples salas de conferencias y las contraseñas opcionales.

Una configuración básica de *meetme.conf* podría ser la siguiente:

```
[general]
     audiobuffers = 32          ;número de paquetes de audio de 20ms que serán
                                guardados en un buffer de memoria. Permite
                                sincronizar el audio de los distintos participantes.
     schedule = yes             ;posibilita las conferencias programadas mediante
                                MySQL
     logmembercount = yes       ;actualiza el tiempo real cuándo un usuario entra o
                                sale de la conferencia
     fuzzystart = 300           ;segundos de margen para permitir que un usuario
                                entre a una conferencia programada en el tiempo que
                                ya ha empezado
     earlyalert = 3600          ;segundos de margen para anunciar a un usuario que
                                una conferencia programada aún no ha empezado
     endalert = 120             ;segundos en los que se anuncia el fin de una
                                conferencia programada antes de que finalice
[rooms]
     conf => 1234, 111, 222     ;se define la sala de conferencias 1234, con la
                                contraseña de acceso opcional 111 y la contraseña de
                                administrador opcional 222
```

La sintaxis de la configuración de la sala de conferencias es la siguiente:

```
conf => numconferencia[,pin][,pinadmin]
```

Una vez la configuración de *meeteme.conf* está completa, será necesario definir en el archivo *extensions.conf* el uso de la aplicación `MeetMe()`, para ello se asociará a una extensión determinada a la cual llamar para poder conectarse a una de las conferencias especificadas.

Es posible limitar el tamaño de la sala de conferencias usando una macro para `meetme` en *extensions.conf* de la forma:

```
[conferencia]
      exten => 1234,1,Set(confmax=10)
      exten => 1234,2,Macro(meetme)
[macro-meetme]
      exten => s,1,MeetMeCount(${MACRO_EXTEN},count)
      exten => s,2,GotoIf($[${count}>${confmax}]?103)
      exten => s,3,Meetme(${MACRO_EXTEN})
      exten => s,4,Hangup
      exten => s,103,Playback(conf-invalid)
      exten => s,104,Hangup
```

Para ello se define un tope de usuarios en la sala de conferencias (`confmax=10`) que servirá de referencia para comparar con el número de usuarios que están ya dentro de la sala.

En la variable "`count`" se almacenará el número de participantes en la conferencia 1234 que viene referenciada por la variable `${MACRO_EXTEN}`. En caso de que el número de usuarios dentro de la conferencia sea mayor que 10, entonces se saltará a la línea 103, que lanzará una aplicación de audio que argumentará que la conferencia no es válida. De no ser así, el nuevo participante entrará en la sala de conferencias 1234 (`${MACRO_EXTEN}`).

Por lo tanto, cuándo un usuario quiere acceder a dicha sala, lo primero que se realiza es una comprobación para ver si ha llegado al límite de usuarios la sala o no.

Es posible usar la sala de conferencias en tiempo real, pera ello habrá que definir una base de datos en MySQL que contenga sus parámetros. Para que dicha base de datos funcione el comando schedule de *extensions.conf* tiene que estar definido como yes.

Para poder hacer uso de la aplicación `MeetMe()` es necesario tener un temporizador, para ello se usará una tarjeta Zaptel de Digium o bien un simulador de Zaptel (ztdummy) que aportará un temporizador sin canales. Éste último viene como módulo por defecto en el Kernel a partir de la versión 2.6

Buzón de voz

Otra de las prestaciones que ofrece Asterisk es el uso del buzón de voz.

Puede actuar como contestador en caso de que una llamada no fuera descolgada o que la línea se encuentre ocupada y permitiendo dejar un mensaje de audio. Además ofrece la posibilidad de enviar el mensaje por correo electrónico, o configurar tus propios mensajes de bienvenida al contestador.

Muchas de las características del sistema de buzón de voz de Asterisk incluyen:

- Protección ilimitada de contraseña para los buzones de voz
- Distinción entre los estados “no disponible” y “ocupado”
- Saludos personalizados
- Posibilidad de asociar teléfonos con más de un buzón y buzones con más de un teléfono
- Notificación de audio mediante correo electrónico si la opción está activa
- Almacenamiento de los mensajes del buzón de voz

El archivo que define todas las prestaciones del buzón de voz es archivo *voicemail.conf* ubicado en */etc/asterisk/*

Contiene los siguientes contextos:

- `[general]:` configura las opciones globales del buzón de voz, como cantidad de mensajes del buzón, tiempo para cada grabación, número de reintentos, etc
- [`zonemessages`]: define las zonas horarias y su formato. La hora para distintos usuarios no es la misma y para poder informar sobre la hora en que recibió el mensaje es necesario fijar distintas zonas horarias. Definidas en */usr/share/zoneinfo/*
- [`default`]: resto de contextos usados para definir los buzones de los usuarios. Se pueden tener todos los usuarios en un solo contexto por ejemplo `[default]` o tener más de un contexto.

Una posible configuración del archivo *voicemail.conf* podría ser la siguiente:

```
[general]
     format = wav          ;indica el formato en que se guardarán los mensajes de voz.
                           Opciones: "wav49", "gsm", "wav"
     serveremail = usuario@dominio.com     ;indica el origen de los mensajes
                                           de notificación por correo
                                           electrónico
     attach = yes          ;Indica si se envía un archivo en las notificaciones de email
     maxmsg = 100          ;indica el número máximo de mensajes en un buzón
     maxsecs = 180         ;indica la duración máxima en segundos de cada mensaje de
                           voz
     minsecs = 3           ;número mínimo de segundos para que un mensaje de voz sea
                           reconocido como tal y enviado a la casilla del destinatario
     maxgreet = 60         ;duración máxima del mensaje de bienvenida personalizado
     skipms = 3000         ;intervalo (ms) usado para saltar hacia delante o hacia atrás
                           en el mensaje del buzón que está siendo reproducido
     maxsilence = 10       ;indica los segundos de silencio que debe detectar el servidor
                           para cortar la llamada al buzón. Por defecto es 0, que
                           equivale a un tiempo infinito
     silencethreshold = 128 ;representa el nivel de audio y sirve para definir qué se
                            considera silencio. Un número menor indica mayor
                            sensibilidad al ruido
     maxlogins = 3         ;número máximo de veces que nos podemos equivocar al
                           introducir la contraseña de acceso al buzón de voz
     moveheard = yes       ;mueve los mensajes escuchados a la carpeta OLD de forma
                           automática
     userscontext = default          ;contexto predefinido para los usuarios
     directoryintro = dir-intro      ;mensaje de introducción de la aplicación
                                     directory
     charset = ISO-8859-1   ;estándar ISO de los mensajes de texto que se enviarán
                            para notificar la llegada de un mensaje de voz
     pbxskip = yes          ;quita la abreviación [PBX] en el remitente del
                            mensaje
     fromstring = buzon_voz ;nombre que aparecerá en el correo electrónico como
                            remitente
     usedirectory = yes     ;permite buscar en el directorio a la persona a la que
                            queremos dejar o reenviar el correo de voz
     sendvoicemail = yes    ;permite enviar un mensaje de voz a otra extensión
                            (opción 5 del menú avanzado)
     tz = central           ;huso horario predefinido para indicar la fecha y hora
                            del correo recibido
```

```
saycid = yes                ;anuncia el número que llamó antes de reproducir el mensaje
listen-control-forward-key = #      ;tecla para adelantar el mensaje que se está escuchando
listen-control-reverse-key = *      ;tecla para ir atrás en el mensaje que se está escuchando
listen-control-pause-key = 0        ;tecla para poner en pausa el mensaje
listen-control-restart-key = 2      ;tecla para volver a escuchar el mensaje desde el inicio
listen-control-stop-key = 13456789  ;teclas para parar el mensaje y volver al menú del contestador
backupdeleted = 100             ;número máximo de mensajes en la capeta DELETED
[zonemessages]
colombia=America/Bogota|'vm-received' aebY 'digits/at' HM
eastern=America/New_York|'vm-received' Q 'digits/at' Imp
central=America/Chicago|'vm-received' Q 'digits/at' Imp
central24=America/Chicago|'vm-received' q 'digits/at' H N 'hours'
military=Zulu|'vm-received' q 'digits/at' H N 'hours' 'phonetic/z_p'
european=Europe/Copenhagen|'vm-received' a d b 'digits/at' HM
```

;zonas horarias definidas con un formato específico para usar en el contestador. Si existen zonas horarias distintas será útil definirlas y configurar el buzón según la hora exacta. Se detallarán a continuación

`[default]` *;definición para las extensiones pertenecientes al contexto [default]*

`1001 => 123,exten1001,usuario@dominio.com` *;definición de la contraseña, nombre de usuario y correo electrónico de la extensión 1001*

El contexto `[zonemessages]` se rige por el siguiente formato a la hora de definir una zona horaria:

```
zona=País/Ciudad|Opciones
```

- `País/Ciudad`: Corresponden con el país de la zona horaria definida y la ciudad a la que se hace referencia. Los pares País/Ciudad tienen que estar definidos en el directorio /usr/share/zoneinfo para que sean válidos.
- `Opciones`:

`'fichero'`	Nombre del fichero de audio a reproducir
`${VAR}`	Variable de sustitución
`A, a`	Día de la semana (sábado, domingo,...)
`B,b,h`	Mes (enero, febrero,...)
`d,e`	Día del mes numérico (primero, segundo,...)
`Y`	Año

`I, i`	Hora, en formato de 12 horas
`H, k`	Hora, en formato de 24 horas
`M`	Minutos
`P, p`	AM o PM
`Q`	"hoy", "ayer"
`R`	Tiempo 24 horas, incluidos minutos

El resto de contextos que se detallan después del contexto `[zonemessages]` son usados para definir los buzones de voz de los usuarios.

Por defecto se usa `[default]` para contener a todas las extensiones, pero pueden haber muchos y muy variados tipos

A continuación se detalla el formato del contexto:

```
extensión => contraseña, nombre de usuario, correo de usuario,
correo de notificación, opciones
```

- `extensión`: extensión de acceso al buzón de voz
- `contraseña`: contraseña que necesitará la extensión para acceder al buzón de voz
- `nombre de usuario`: nombre del cliente de la extensión
- `correo de usuario`: correo al que se enviarán los mensajes
- `correo de notificación`: correo alternativo dónde pueden ser enviadas las notificaciones para administración o control
- `opciones`: sirven para redefinir las opciones del contexto `[general]` o para especificar una zona horaria para el usuario. Tipos: `attach, serveremail, tz, saycid, review, operator, callback, dialout, exitcontext`

Para hacer uso del buzón de voz habrá que modificar los archivos de los protocolos dónde están definidos las extensiones y el plan de marcación.

Tanto en *sip.conf* como en *iax.conf* habrá que especificar el número de buzón de voz correspondiente para cada extensión con la opción:

```
mailbox=extensión@contexto
```

- `extensión`: extensión definida dentro del archivo *voicemail.conf*, en el ejemplo anterior sería 1001. Puede ser la misma que la extensión principal u otra distinta, para evitar crear demasiadas extensiones asociadas a una se ha preferido usar la misma para ambos casos
- `contexto`: contexto dónde se encuentra definida la extensión de acceso al buzón de voz, en el ejemplo anterior sería default

En el archivo *extensions.conf* se puede hacer uso a partir de ahora de las aplicaciones `Voicemail()` y `VoiceMailMain()` especificando la extensión de acceso al buzón solicitado y el contexto asociado a dicha extensión.

En el caso de `VoiceMailMain()` que regula el acceso del cliente a su buzón de voz se solicitará la contraseña definida en *voicemail.conf* para cada cliente

Respuesta de voz interactiva (IVR)

Asterisk ofrece además servicios de respuesta de voz interactiva o IVR (Interactive Voice Response). El propósito de éste servicio es tomar de una llamada entrante los datos suficientes que harán que el sistema realice una acción u otra y devuelva el resultado al llamante.

Tradicionalmente, los sistemas IVR eran complejos, caros y difíciles de implementar, sin embargo, Asterisk lo simplifica substancialmente.

Un ejemplo de sistema de este tipo lo tenemos en los contestadores automáticos que responden a nuestra llamada y que nos indican qué tecla pulsar para realizar cualquier opción. Dentro del sistema del IVR la llamada se encaminará por un lugar u otro en función de nuestra respuesta y dependiendo del camino obtendremos un resultado distinto.

Otro ejemplo lo podemos encontrar en un contestador que salta si la llamada está fuera de horario laboral y por lo tanto no se puede contactar con ningún cliente, para ello lo primero que se validará es si se cumple o no la condición temporal y en función de eso se hará una cosa u otra.

Sin embargo es mucho más que un simple contestador automático, ya que requiere de:

- una grabación que le indique al llamante lo que se espera de él
- un método para recibir la entrada del llamante
- una lógica para verificar que la respuesta es válida
- una lógica para determinar el siguiente paso en el IVR
- un mecanismo de almacenamiento en caso de ser necesario.

Desde el punto de vista del llamante, cada IVR comienza con una grabación que indica las posibles opciones del IVR y solicita al llamante que pulse una tecla.

El segundo componente es el método para recibir la entrada desde el llamante, generalmente se usan las aplicaciones `Background()` y `WaitExten()` para capturar la respuesta. La aplicación `Background()` reproduce la grabación y a la vez se mantiene a la espera de una entrada por parte del cliente.

Una vez recibida la entrada hay que determinar su validez pudiendo hacer uso de un recurso externo para procesarla, como puede ser una base de datos o un programa. La forma de comprobar la validez de forma interna se hará por descarte, ya que quedarán definidas las opciones válidas y aquellas que no queden registradas se tomarán como un error.

A menudo un IVR tiene múltiples pasos y es necesario solicitar de nuevo información para seguir con el proceso hasta el final, sin embargo habrá que tener especial cuidado para no hacerlo demasiado extenso.

Es especialmente útil usar herramientas de grabación de audio para personalizar de antemano los mensajes que serán reproducidos por el IVR, por lo tanto se debe contar con la aplicación `Record()` para ello.

Para utilizar el IVR dentro del plan de marcación habrá que asociarlo a una extensión a la cual llamar para que se reproduzca todo el proceso.

Un ejemplo de uso y configuración básico podría ser el siguiente:

```
[IVR]
exten => 123,1,Answer()
exten => 123,2,Goto(principal,s,1)

[principal]
exten => s,1,Wait(1)
exten => s,n,BackGround(saludo)
exten => s,n,WaitExten()

exten => 1,1,Dial(SIP/111,20,m)
exten => 1,n,Hangup

exten => 2,1,Dial(SIP/222,20,m)
exten => 2,n,Hangup

exten => i,1,Playback(invalid)
exten => i,n,Hangup
exten => t,1,Goto(principal,s,1)
exten => h,1,Hangup
```

En éste ejemplo, cualquier cliente que marque la extensión `123` será redirigida al contexto `[principal]` dónde está contenido el IVR.

En primer lugar se reproducirá el archivo de audio "`saludo`" mientras queda a la espera de la respuesta por parte del cliente. El archivo podría reproducir la siguiente información: "`bienvenido, si desea hablar con la extensión 111 marque 1, si desea hablar con la extensión 222, marque 2, gracias`"

En tal caso el cliente puede:

- marcar `1` ó `2`, por lo tanto llamará a la extensión correspondiente y una vez finalizada la llamara se cerrará el canal
- marcar cualquier otro número, por lo tanto se detectará una entrada inválida y se redirigirá la llamada a la extensión "`i`", la cuál reproduce el mensaje "`invalid`" y cuelga
- si el cliente tarda más tiempo del necesario irá a la extensión "`t`", que desvía la llamada al contexto `[principal]` dónde se reproducirá el mensaje de saludo y se volverá a esperar una entrada por parte del cliente
- si por el contrario el cliente cuelga, salta a la extensión "`h`" que cierra el canal de audio correspondiente.

Para grabar nuestro propio archivo de saludo se puede hacer de la siguiente forma:

```
[grabar]
exten => 456,1,Answer()
exten => 456,n,Record(/var/lib/asterisk/sounds/es/saludo.wav)
exten => 456,n,Wait(2)
exten => 456,n,Playback(saludo)
exten => 456,n,Wait(2)
exten => 456,n,Hangup
```

Por lo tanto, cualquier cliente que marque la extensión `456` iniciará una grabación que se guardará en el archivo `saludo.wav`.

Una vez grabado el archivo se finaliza con `#`, se reproduce para comprobar que es correcto y se termina la llamada.

Para sobrescribir el archivo basta con volver a llamar a la extensión `456` de nuevo e iniciar una grabación.

Monitorización y registro de llamadas

Cuándo aparecen problemas en el sistema Asterisk, es útil tener unos registros de las actividades acontecidas. Los parámetros que almacenan esta información están definidos en */etc/asterisk/logger.conf*.

Un almacenamiento excesivo de datos podría hacer que el sistema se colapsara, así que es necesario tener un equilibrio entre los detalles y las capacidades de almacenamiento.

Éste archivo permite definir diferentes niveles de registro para cada salida. Tiene el siguiente formato:

```
[general]
    appendhostname = yes      ;el archivo de registro terminará con el nombre del dominio
    queue_log = yes           ;usado para mantener un registro de las colas de espera
    queue_log_name = queue_log ;nombre del archivo dónde guardamos los registros de las colas de espera
    rotatestrategy = rotate   ;los archivos de registros se pueden volver de un tamaño muy grande haciendo poco cómoda la consulta. Para evitar esto usamos el programa logrotate. Con este sistema el archivo más viejo tendrá el número secuencial más alto
    event_log = yes           ;de manera predefinida Asterisk registra en un archivo los eventos genéricos
    dateformat = %F%T         ;personaliza el mensaje de fecha que se muestra en el debugador. Tiene el formato yyyy-mm-dd HH:MM:SS
[logfiles]      ;especifica qué registros habrá en cada archivo.
                Eventos a registrar: debug, notice, warning, error, verbose, dtmf
    console => notice,warning,error,dtmf  ;en la consola aparecerán las noticias, avisos, errores y los tonos dtmf tecleados, se omite debug por contener demasiada información para mostrarla en pantalla
    messages => notice,warning,debug      ;en el archivo messages aparecen las noticias, avisos y debug
    full => error     ;en el archivo full sólo aparecerán los errores del servidor
```

Es posible especificar dentro del contexto `[logfiles]` tantos archivos como se deseen.

Todos los archivos generados se almacenarán en el directorio */var/log/asterisk/*, a excepción del archivo "`console`" que es distinto al resto y únicamente se mostrará como salida en la línea de comandos de Asterisk o CLI, pero no como un archivo.

Tipos de eventos:

- `notice`: Es un simple evento del que Asterisk quiere hacer constancia. Es posible ver muchos durante un reinicio, o durante una llamada
- `warning`: Representa un problema con importancia suficiente para afectar a una llamada (se puede incluso desconectar la llamada porque el flujo no puede continuar). Es necesario dirigirlos hacia el CLI
- `error`: Representan problemas significantes en el sistema que deben ser dirigidos con urgencia hacia el CLI
- `debug`: La debugación es útil sólo cuándo existe un problema de código de Asterisk. No se debe usar para buscar errores en el plan de marcación, aunque es bueno para obtener los informes de errores que genera
- `verbose`: Es uno de los eventos más útiles, pero a la vez uno de los menos utilizados debido a que puede generar un fallo en el disco duro
- `dtmf`: Registrar los tonos DTMF puede ser eficaz para averiguar porqué una llamada no se envía correctamente en un sistema de IVR por ejemplo
- `fax`: Proporciona mensajes del tipo fax-related
- `*` Este evento recoge toda la información posible. No usar a menos que se sepa si se puede manejar la cantidad de datos a almacenar.

Es posible ver las características definidas en *logger.conf* en la línea de comandos de Asterisk mediante:

```
CLI> logger show channels
```

El sistema de detalle de llamadas o CDR (Call Detail Records) de Asterisk se usa para llevar un control de todas las llamadas que se han realizado o recibido en el sistema.

Se usa generalmente en el control de la facturación, como prevención de fraudes y evaluación de la calidad de servicio.

La configuración del archivo *cdr.conf* dará como resultado un archivo que contendrá toda la información referente a las llamadas y que se ubicará en el directorio */cygroot/asterisk/log/cdr-csv/Master.csv*

Un ejemplo de configuración del archivo *cdr.conf* podría ser el siguiente:

```
[general]
     enable = yes              ;registra cada intento de llamada a una extensión. Uso del
                               logging de CDR
     unanswered = no           ;controla el reporte de llamadas que no han sido atendidas
                               con un solo participante
     endbeforehexten = no      ;normalmente los CDR no se cierran hasta que las
                               extensiones terminan su ejecución. Habilitando esta
                               opción, el CDR se cierra antes de ejecutar la extensión
                               "h"
     initiatedseconds = no     ;redondea el campo "billsec" en base a los
                               microsegundos de la llamada para pasarlos a segundos
     batch = no                ;uso de un buffer de datos en la escritura del CDR
     size = 100                ;número de CDRs acumulados en el búffer antes de pasarlos a
                               la base de datos. Para que se tenga en cuenta, batch tiene que
                               estar en yes
     time = 300                ;número de segundos antes de que Asterisk provoque la
                               escritura de CDRs en la base de datos, batch tiene que estar
                               en yes
     scheduleronly = yes       ;uso en la generación de un gran volumen de CDRs
                               que se pondrán en una base de datos
     safeshutdown = yes        ;previene el cierre de Asterisk hasta que el búffer se
                               haya escrito completamente en el CDR
[csv]
     usegmtime = no
     loguniqueid = no
     loguserfield = no
     accountlogs = yes
```

Para que los CDRs se logueen en *Master.csv* hay que definir la categoría `[csv]`. No es necesaria una base de datos

Existen extensiones al CDR que permiten almacenar los registros en una base de datos MySQL usando el archivo *cdr_mysql.conf*

Además existen aplicaciones open source que permiten gestionar el CDR para facilitar la lectura y búsqueda de la información solicitada, como pueden ser: astbill, AreskiStat, A2billing y muchas otras.

Para configurar el sistema de detalle de llamadas, aparte del archivo *cdr.conf*, habrá que configurar los archivos *cdr_manager-conf* y *cdr_custom.conf*

Para obtener los eventos del manager hay que asegurase de que el archivo *cdr_manager.conf* existe y que la sección `[general]` contiene "`enabled=yes`"

Esta configuración se usa para habilitar o deshabilitar el envío de CDR a través de la interfaz manager de Asterisk, por lo tanto tendrá que estar activa para que se establezca comunicación.

Cuándo estamos conectados a la interfaz manager, y una llamada finaliza, un evento será enviado a todo lo que haya conectado a dicha interfaz

Por último hay que asegurarse que el archivo *cdr_custom.conf* esté presente y contenta la sección `[mappings]`, la cual define los campos de salida y el orden de éstos para generar el archivo *Master.csv.*

Asterisk generará una entrada dentro del archivo *Master.csv* para cada llamada, cuyos principales campos son:

- `clid`: identificador del llamante (nombre)
- `src`: número identificador del llamante
- `dst`: extensión destino
- `dcontext`: contexto destino
- `channel`: canal origen
- `dstchannel`: canal destino
- `lastapp`: última aplicación usada en el plan de marcación. Para llamadas salientes será `Dial()`
- `lastdata`: parámetros establecidos para la última aplicación usada en el dialplan
- `start`: fecha y hora de inicio de la llamada
- `answer`: fecha y hora en el que la llamada ha sido contestada
- `end`: fecha y hora en el que la llamada ha sido terminada
- `duration`: duración total de la llamada, desde que se generó hasta el corte
- `billsec`: duración de la llamada desde que fue atendida hasta el corte
- `disposition`: estado de la llamada (atendida, no atendida, ocupada, fallida)
- `amaflags`: flags según el tipo de CDR (`default/omit/billing/documentation`)
- `accountcode`: código de la cuenta. Activado con `SetAccountcode` en el plan de marcación o en el archivo de configuración de canal (ej: *sip.conf*)
- `uniqueid`: identificador único para esa llamada
- `userfield`: activado en el dialplan con `SetCDRUserfield`

Para confirmar el estado del CDR desde el CLI se puede ejecutar:

```
CLI> cdr status
```

Asterisk Gateway Interface (AGI)

El plan de marcación de Asterisk tiende a ser más simple cada vez, por lo tanto, a veces es necesario poder incluir partes del plan de marcación en diferentes lenguajes de programación para adaptarse a los requisitos de los usuarios.

El AGI de Asterisk permite que programas externos en forma de scripts se comuniquen con el plan de marcación.

Dichos scripts pueden construirse en múltiples lenguajes de programación como pueden ser: PHP, Python, Java, C y Perl, aunque se puede utilizar cualquier otro lenguaje soportado por Unix.

Asterisk tiene por lo tanto una potente interfaz que permite controlar las llamadas en el lenguaje de programación que se elija. Esto permite una gran variedad de enfoques a la hora de implementar una aplicación AGI y hace que sea más fácil la integración de Asterisk con otros sistemas.

El intercambio de información del script AGI con Asterisk se realiza a través de los canales: `STDIN`, `STDOUT` y `STDERR`.

- `STDIN:` es usado para obtener la información de Asterisk
- `STDOUT:` para enviar información a Asterisk
- `STDERR:` para enviar información de debugging

La forma de comunicación es la siguiente: el script envía comandos a Asterisk escribiendo en el canal `STDOUT`, Asterisk los lee, actúa en consecuencia y genera una respuesta por cada comando que es leída por el script en el canal `STDIN`

Un ejemplo de uso podría ser el siguiente:

En el archivo *extensions.conf* se añadiría una línea con una llamada al script AGI de la siguiente forma:

```
exten => 333,1,AGI(programa.php)
```

A continuación habría que crear el script *programa.php* en el directorio */var/lib/asterisk/agi-bin/*

Una vez hecho todo ésto bastará con llamar a la extensión 333 para que el programa AGI se ejecute.

Hay varios tipos de AGI:

- AGI basada en procesos: Es la versión más simple pero menos eficiente de todas, usa la aplicación `AGI()` y los canales *stdin* y *stdout* para comunicarse con Asterisk.
- AGI mejorada (EAGI): Usa la aplicación `EAGI()` y además de los canales *stdin* y *stdout*, Asterisk provee además un canal de audio unidireccional.
- AGI rápida (FastAGI): La conexión se hace mediante TCP, se realiza por un servidor FastAGI y no necesita que se abra un nuevo proceso de control para cada llamada. Usa la aplicación `AGI()` pero incluyendo como parámetros `agi://URL`. Es más eficiente a pesar de que la implementación es más compleja.
- AGI asíncrona (Async AGI): Usa la interfaz manager de Asterisk para que los comandos asíncronos de encolamiento de una cola AGI sean ejecutados. Se utiliza mediante la aplicación `AGI()` añadiendo como parámetros `agi:async`. Permite que una aplicación AMI pueda controlar las llamadas usando comandos externos de AGI, sin embargo es un método bastante complejo.

Una aplicación AGI contendrá una serie de variables que serán la primera información que se envíe desde Asterisk, como pueden ser: `agi_request, agi_channel, agi_language, agi_type, agi_uniqueid, agi_version, agi_callerid, agi_calleridname, agi_callingpress, agi_callingani2, agi_callington, agi_calingtns, agi_dnid, agi_rdnis, agi_context, agi_extension, agi_priority, agi_enhanced, agi_accountcode, agi_threadid, agi_arg_<argument number>`

Una vez una sesión AGI se ha puesto en marcha, Asterisk comienza con el procesamiento de las respuestas para enviar a la aplicación AGI. Una vez que los comandos han sido enviados a Asterisk, no se procesan más hasta que el comando enviado sea procesado y Asterisk devuelva una respuesta con el resultado.

Comandos AGI:

- `ANSWER`
- `ASYNCAGIBREAK`
- `CHANNELSTATUS`
- `CONTROL STREAM FILE`
- `DATABASEDEL`
- `DATABASEDELTREE`
- `DATABASEPUT`
- `EXEC`
- `GET DATA`

- GET FULLVARIABLE
- GET OPTION
- GET VARIABLE
- GOSUB
- HANGUP
- NOOP
- RECEIVE CHAR
- RECEIVE TEXT
- RECORDFILE
- SAY ALPHA
- SAY DIGITS
- SAY NUMBER
- SAY PHONETIC
- SAY DATE
- SAY TIME
- SAY DATETIME
- SEND IMAGE
- SEND TEXT
- SEND AUTOHANGUP
- SET CALLERID
- SET CONTEXT
- SET EXTENSION
- SET MUSIC
- SET PRIORITY
- SET VARIABLE
- SPEECH ACTIVATEGRAMMAR
- SPEECH CREATE
- SPEECH DEACTIVATEGRAMMAR
- SPEECH DESTROY
- SPEECH LOADGRAMMAR
- SPEECH RECOGNIZE
- SPEECH SET
- SPEECH UNLOADGRAMMAR
- STREAM FILE
- TDD MODE
- VERBOSE
- WAIT FOR DIGIT

Una sesión AGI termina cuándo la aplicación AGI está preparada para ello. La forma dependerá de si se usa una AGI basada en procesos, una FastAGI o una Async AGI.

-AGI basada en procesos o FastAGI: la aplicación AGI puede cerrar la conexión en cualquier momento y el plan de marcación sigue su ejecución de forma normal. Si el canal se cierra durante la ejecución de la AGI se producirá un error y se enviará el código -1.

-Async AGI: cuándo se usa este tipo de aplicación, la interfaz AMI provee el mecanismo para notificar el cierre del canal. En este punto, el canal vuelve al plan de marcación.

Asterisk Manager Interface (AMI)

La interfaz Asterisk Manager es un sistema de monitorización y mantenimiento que permite comprobar los eventos que ocurren en el sistema en tiempo real, y establece control sobre las acciones de Asterisk. Permite hacer todo lo que haría la consola de Asterisk.

Entre las acciones posibles se encuentran la información de estado y la creación de llamadas.

El administrador de programas de la capacidad de ejecutar comandos y solicitar información desde el servidor Asterisk. Sin embargo, no es un mecanismo de autentificación muy seguro, porque usa contraseñas de texto sin formato, y todos los terminales reciben todos los eventos.

Por lo tanto, el administrador de Asterisk se debe usar sólo en una red local de confianza.

Las construcciones de permiso y denegación le permiten restringir el acceso a determinadas extensiones o subredes.

El uso de la interfaz de manager de Asterisk presenta los siguiente problemas:

-La documentación sobre el protocolo y la funcionalidad del manager está incompleta

-Tiene problemas con el manejo de varias conexiones a la vez, sobre todo cuándo el número de conexiones supera a cinco.

-Es recomendado el uso de un proxy para sistemas que hagan un uso intensivo del manager, como pueden ser sistemas de monitorización y campaña telefónica.

El archivo *manager.conf* define la forma en que los programas se autentifican con el administrador.

El administrador de comandos tiene diferentes grados de privilegio. Puede controlar la lectura y escritura de estos comandos con el uso de `read` y `write` en dicho archivo.

Un ejemplo de configuración de *manager.conf* podría ser el siguiente:

```
[general]
      enabled = yes            ;habilita el administrador AMI
      port = 5038              ;puerto TCP usado por defecto
      bindaddr = 127.0.0.1     ;dirección IP de escucha para los conexiones al
                               manager, en éste caso sólo acepta conexiones
                               del pc local
      webenabled = yes         ;permite la conexión mediante una interfaz web
      displayconnects = yes    ;muestra las conexiones a la AMI como
                               mensajes de verbose impresos en la consola de
                               Asterisk
[hello]           ;creación de la cuenta para el usuario "hello"
      secret = world           ;contraseña de la cuenta de administrador
      deny = 0.0.0.0/0.0.0.0   ;impide el acceso a todas la direcciones IP que
                               se quieran conectar (ACL)
      permit = 127.0.0.1/255.255.255.0    ;permite el acceso a las
                                           direcciones especificadas, en este
                                           caso es el pc
      read = system,call,log,verbose,command,agent,user
      write = system,call,log,verbose,command,agent,user
            ;proporciona permisos de lectura y escritura para los siguientes
            eventos. Los privilegios posibles son: system, call, log, verbose,
            command, agent, user, config, command, dtmf, reporting, cdr,
            dialplan, originate, agi, cc, aoc y all, el cual los abarca a todos
      eventfilter = !Channel: DAHDI*;usa un filtrado de eventos, una especie
                                     de lista negra, todos van precedidos por
                                     un signo de exclamación
```

Una vez la configuración de la AMI está activa, es posible activar el servidor HTTP mediante el archivo *http.conf*. Un ejemplo sería el siguiente:

```
[general]
      enabled = yes            ;habilita la interfaz HTTP
      bindaddr = 127.0.0.1     ;dirección de acceso al servidor, en éste caso
                               sólo permite conexiones desde el pc local
      bindport = 8080          ;puerto de acceso por defecto
```

Hay múltiples formas de conectar con la interfaz manager de Asterisk, pero un socket TCP es la forma más común, para ello hará uso de `telnet`.

Conexión mediante comandos de consola vía TCP:

El conjunto de acciones que determinan una instrucción se llama paquete y está formado por líneas del tipo "`clave:valor`". Los tipos de claves que conforman un paquete son:

- `Action`: paquete originado en el cliente, requiere que se lleve a cabo una acción concreta. Contiene el nombre de la acción y los parámetros de la misma.
- `Response`: respuesta de Asterisk a la acción requerida por el cliente.
- `Event`: datos correspondientes a un evento generado dentro del núcleo de Asterisk.

Se pueden proveer parámetros adicionales, como por ejemplo un número a llamar o un canal a desconectar. Y también se pueden añadir variables. El formato será el siguiente:

```
Action:<action type><CRLF>
<Key 1>: <Value 1><CRLF>
<Key 2>: <Value 2><CRLF>
…
Variable: <Variable 1>=<Value 1><CRLF>
Variable: <Variable 2>=<Value 2><CRLF>
…
<CRLF>
```

Ejemplos de paquetes Action:

`-Command`:	ejecuta un comando específico (privilegios: command, all)
`-Events`:	controla el flujo de los eventos
`-Hangup`:	cuelga el canal (privilegios: call, all)
`-IAXpeers`:	lista los peers IAX (privilegios: system, all)
`-ListCommands`:	lista los comandos disponibles del manager
`-Logoff`:	desconexión del manager
`-MailboxCount`:	verifica la cantidad de mensajes en el buzón de voz (privilegios: call, all)
`-MailboxStatus`:	verifica el estado del buzón de voz (privilegios: call, all)
`-Originate`:	origina una llamada (privilegios: call, all)
`-ParkedCalls`:	lista las llamadas parqueadas
`-QueueAdd`:	agrega un miembro a la cola (privilegios: agent, all)
`-QueueRemove`:	borra un miembro de la cola (privilegios: agent, all)
`-SIPpeers`:	lista los peers SIP (privilegios: system, all)
`-Status`:	estado (privilegios: call, all)

Pasos a seguir para comprobar que la conexión con el servidor es posible:

-Conectar a la interfaz de manager de Asterisk usando un socket TCP en el puerto 5038
-Identificar usando la acción `Login`
-Ejecutar la acción `Ping`
-Salir usando la acción `Logoff`

```
$ telnet localhost 5038
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Asterisk Call Manager/1,1
Action: Login
Username: hello
Secret: world

Response: Success
Message: Authentication accepted

Action: Ping

Response: Success
Ping: Pong
Timestamp: 1235679879.54768

Action: Logoff

Response: Goodbye
Message: Thanks for all the fish.

Connection closed by foreign host.
```

Una vez hecho todo esto queda verificado que la AMI acepta conexiones vía TCP

Conexión mediante HTTP:

También es posible usar la AMI sobre HTTP, las acciones utilizadas serán las mismas que para la AMI sobre TCP y las respuestas tendrán el mismo formato aunque pueden ir codificadas en distintos formatos como XML.

Las cuentas usadas para conectar la AMI por HTTP son las mismas que las configuradas en el archivo *manager.conf* ubicado en */etc/asterisk/*.

La interfaz HTTP permite integrar el control de llamadas de Asterisk dentro de un servicio web.

Para establecer el acceso a la AMI a través de HTTP es necesario que *manager.conf* y *http.conf* estén configurados y activados.

En el archivo *manager.conf* la opción `enabled` tiene que contener el valor `yes` y webenabled también tiene que ser `yes` para permitir el acceso a través de HTTP.

Y en el archivo *http.conf*, la opción `enabled` tiene que estar activada con yes

Entre las muchas aplicaciones creadas para el desarrollo de la AMI hay una que destaca especialmente, la AsteriskGUI o interfaz gráfica de Asterisk.

La AsteriskGUI es una interfaz gráfica de administración de PBX bajo código open source y desarrollada por Digium.

Está escrita en HTML y JavaScript y usa la AMI sobre HTTP para toda interacción con Asterisk.

Es usada habitualmente en sistemas de Asterisk comprimidos, ya que no requiere un software adicional para funcionar con la centralita.

Se puede descargar de:

```
http://svn.digium.com/svn/asterisk-gui/branches/2.0
```

4.6 Uso de callfiles

Hay múltiples formas de iniciar una llamada:

- Mediante archivos *.call*: es un archivo de texto que ubicado concreto hace que Asterisk genere una llamada saliente.
- Mediante la interfaz AMI
- Usando el comando `originate` en el CLI.

Los archivos de llamadas o callfiles no son otra cosa que unos archivos de texto estructurados que especifican a Asterisk cómo gestionar las llamadas salientes y qué hacer una vez conectadas.

Para que sean ejecutados, habrá que moverlos al directorio */var/spool/asterisk/outgoing/*.

Asterisk controlará a cada segundo si hay algún archivo nuevo en la carpeta y lo procesará enseguida. Cuándo aparece un nuevo archivo, Asterisk inicia una llamada en función del contenido del archivo.

Sin embargo, es posible crear un archivo con un tiempo futuro de modificación, entonces Asterisk esperará para ejecutarlo hasta que llegue el momento adecuado para crear la llamada justo a tiempo.

Es importante crear el callfile en un directorio distinto y una vez creado, moverlo con el comando "`mv`" al directorio final */var/spool/asterisk/outgoing/*, o sino la centralita leerá sólo una parte del archivo, no al completo.

Una vez ejecutado el archivo de llamadas será borrado del directorio, a no ser que se especifique lo contrario. En caso de que no se haya ejecutado completamente por problemas y exista la posibilidad de una rellamada, éste seguirá existiendo hasta que pueda ser ejecutado correctamente. Si se sobrepasa el límite de rellamadas, es posible dejar reflejado el error en dicho archivo ubicado en el directorio de salida para su posterior visualización.

Si en *modules.conf*, el comando `autoload` está definido de forma negativa, habrá que asegurarse de que *pbx_spool.so* se cargue, sino los archivos de llamadas no funcionarán

Sintaxis

El archivo de llamada se define mediante pares de clave y valor, definiendo cada línea una instrucción distinta dentro del archivo

Las directivas para la configuración de una llamada son:

`-Channel: <channel>`	Canal usado para la nueva llamada saliente, el formato es tecnología/recurso, de la misma forma que se usa en la aplicación `Dial()`. Obligatoria
`-Callerid: <callerid>`	Identificador de llamada a usar.
`-WaitTime: <number>`	Tiempo en segundos en los que se espera que la llamada sea contestada antes de darla por fallada y que se inicie el ciclo de llamadas. Por defecto, 45 segundos.
`-MaxRetries: <number>`	Número de reintentos después de que una llamada haya fallado, no incluye la llamada inicial. Por defecto es 0, no permite reintentos
`-RetryTime: <number>`	Tiempo en segundos de espera antes de reintentar una llamada. Por defecto, 300 segundos
`-Account: <account>`	Código de la llamada, se usa para asignar el CDR.

Una vez definida la llamada de destino, hay dos opciones:

-Ejecutar una aplicación concreta

-Definir un salto al plan de marcación según el contexto, extensión y prioridad deseadas

Ejecutar una aplicación:

`-Appication: <appname>`	Aplicación a ejecutar en la extensión destino de la llamada
`-Data: <args>`	Argumentos de la aplicación

Saltar al plan de marcación:

`-Context: <context>`	Contexto del plan de marcación que buscará
`-Extension: <exten>`	Extensión del contexto especificado
`-Priority: <priority>`	Prioridad de la extensión especificada, puede ser numérica o una marca
`-Setvar: <var=value>`	Es posible asignar valores a las variables disponibles en el canal como si fueran definidas en el plan de marcación de la forma `Set(var=value)`

El procesamiento del archivo de llamada termina cuándo la llamada es respondida y ha terminado, si dicha llama no ha sido contestada la primera vez que se ejecuta el archivo y se agotan los intentos disponibles, o también si el archivo no ha podido ser leído y analizado correctamente.

Para especificar qué hacer con el archivo una vez procesado se usa la siguiente directiva:

`-Archive: <yes|no>` Si se selecciona "`no`", el archivo de llamada será borrado al finalizar. Si por el contrario se usa "`yes`", el archivo es movido al directorio */var/spool/asterisk/outgoing_done/*

Si el archivo finalmente es guardado es posible añadir la siguiente línea al archivo *.call*

`-Status: <exitstatus>` Puede tomar los valores "`Expired`", "`Completed`" o "`Failed`"

Ejemplos:

ejemplo1.call

```
Channel: SIP/101
Callerid: callfile
MaxRetries: 3
RetryTime: 60
WaitTime: 30
Appication: Playback
Data: hello-world
```

Para ejecutar el archivo *ejemplo1.call* habrá que moverlo al directorio */var/spool/asterisk/outgoing/*, allí se procederá a la lectura del archivo, en primer lugar se realizará una llamada a la extensión 101 usando el canal SIP para tal efecto, una vez la llamada sea contestada se reproduce el archivo *hello-world*, que está ubicado en el directorio */var/lib/asterisk/sounds/hello-world*, y a continuación finaliza la llamada.

ejemplo2.call

```
Channel: SIP/300
Callerid: callfile2
MaxRetries: 10
RetryTime: 60
WaitTime: 30
Context: prueba
Extension: 100
Priority: 1
```

La ejecución del archivo *ejemplo2.call* es similar al del *ejemplo1.call*, sin embargo, en éste, en lugar de ejecutar una orden explícita, se realiza un salto a contexto definido en el archivo *extensions.conf*. Ese contexto, extensión y prioridad determinadas puede realizar cualquier acción o conjunto de acciones, por lo tanto la variedad de opciones es mayor.

Por ejemplo, en el archivo *extensions.conf* podría estar definido el contexto `[prueba]` de la siguiente forma:

```
[prueba]
      exten => 100,1,Answer()
      exten => 100,n,Wait(5)
      exten => 100,n,Playback(hello-world)
      exten => 100,n,Wait(5)
      exten => 100,n,Hangup()
```

4.7 DUNDi

DUNDi (Distributed Universal Number Discovey) es un protocolo punto a punto que permite buscar y compartir planes de marcación entre servidores IP-PBX.

Destaca por su completa distribución sin necesidad de una autoridad centralizada que gestione todo lo que se envía. Publica rutas que a su vez son accesibles mediante los protocolos IAX, SIP y H.323, aunque no define unas pautas de señalización o control de protocolo para VoIP.

DUNDi se usa dentro de una empresa como un PBX sin punto de fallo centralizado, y añade la posibilidad de incluir nuevas extensiones y recursos de red para los servidores sin necesidad de una configuración adicional, sino que son actualizados automáticamente por la red conformada por varios servidores.

Además, permite que los proveedores de puedan usar números de teléfono reales desde internet. Todo ello sin cargos de suscripción, publicación o información.

Al eliminar la necesidad de un monopolio también elimina cualquier coste por compartir servicios de telefonía a través de la red.

El documento que regula el contexto E.164 es Digium GPA (Digium General Peering Agreement), que define la forma de comunicarse.

Dicha información se encuentra en el siguiente enlace: http://dundi.com/PEERING.pdf

En Asterisk existe una implementación del protocolo DUNDi ya definida, puesto que Marc Spencer fue el creador de ambos sistemas de comunicación.

Los servidores que hacen uso del protocolo DUNDi se pueden definir como nodos intercomunicados gracias al intercambio información que se efectúa entre ellos. Cada nodo decide qué extensiones y qué contextos anuncia al resto de los nodos, así como la prioridad de las extensiones para seleccionar el destino de la llamada.

Configuración

Para hacer uso del protocolo DUNDi dentro de una red es necesario definir un plan de distribución de la información, así como modificar los archivos de configuración pertinentes, que en este caso serán *iax.conf*, *extensions.conf* y *dundi.conf*, también es conveniente crear unas llaves de autenticación que permitan codificar la información enviada entre servidores.

- **Esquema:** representará la distribución de los servidores, la forma de conexión entre ellos y los recursos compartidos.

- **Plan de marcación:** define el plan de marcación de cada servidor, tanto las extensiones internas, como aquellas que querrán ser compartidas con el resto de servidores de la red.

- **Iax.conf:** se creará un enlace troncal que permitirá la comunicación con los otros servidores. En él se definirá una contraseña para enviar la información a otro servidor y también se definirá el contexto dónde serán redirigidas las llamadas entrantes de otros servidores.

- **Creación de llaves de autenticación:** permiten el acceso a los servidores por medio del enlace troncal definido en *iax.conf*. Se generan en los servidores y se comparten con los servidores vecinos. Se generan en el directorio */var/lib/asterisk/keys* con el comando `#astgenkey -n` "nombre_llave".

- **extensions.conf:** se definirán 5 contextos

- `[general]:` contendrán las características comunes para todos los contextos
- [`buscardundi`]: en el caso de no ser encontradas las extensiones en el plan de marcación local, este contexto las redirigirá hacia el recurso DUNDi para que realice la búsqueda en otros servidores.
- [`compartir`]: en este contexto se publican las extensiones que se desean compartir con otros servidores. En caso de usar prefijos en el plan de marcación, se compartirán de forma completa.
- [`entrantesdundi`]: este contexto redirige las llamadas entrantes al contexto correspondiente del plan de marcación interno para ese servidor. En caso de usar prefijos, estos serán eliminados.
- [`contextoslocales`]: uno o varios contextos que incluirán a todas las extensiones del plan de marcación propio de ese servidor.

– **dundi.conf:** define el establecimiento, las relaciones entre servidores DUNDi y las respuestas. Cabe destacar los siguientes contextos dentro del archivo de configuración:

- `[general]`: entre varias cosas define el nombre del servidor, el puerto UDP usado para este tipo de comunicaciones, el 4520, el identificador de identidad, que normalmente es el identificador de red o la dirección MAC del propio servidor, el tiempo de búsqueda de respuesta y el número de saltos entre nodos.

- `[mappings]`: contiene una línea por cada recurso DUNDi que se va a compartir con otro servidor. Dicha definición de recurso contendrá información del contexto de las extensiones que se publican, el peso de la ruta compartida, el protocolo usado, el enlace troncal definido en *iax.conf*, una variable que contiene la clave del recurso DUNDi, la dirección IP de dicho servidor, así como la variable que contendrá el número solicitado en la búsqueda.

-[IdServidorCompartir]: dentro del contexto que identifica al servidor con el que se comparten los recursos DUNDi (normalmente corresponde con la dirección MAC) se define la dirección IP del servidor de destino, el nombre de las llaves para recibir y realizar la consulta, el nombre de los recursos DUNDi que queremos compartir con el servidor, y el orden de búsqueda de la información.

El contexto [mappings] definirá los recursos DUNDi a compartir de la siguiente forma:

```
recursoDUNDi=>contextoCompartir,peso,tecnología,destino[,opciones]
```

Entre las distintas opciones podemos encontrar:

`nounsolicited`
`nocomunsolicit`
`residential`
`commercial`
`mobile`
`nopartial`
`${NUMBER}` *;contiene la respuesta del número solicitado*
`${IPADDR}` *;contiene la IP del sistema local.*
`${SECRET}` *;contiene el valor de la clave rotatoria definida en la localización* `secretpath`

– **sip.conf:** contiene la definición de extensiones locales del servidor, dónde se indica el número de extensión, la contraseña, el puerto y el contexto en que están definidas.

Entre las distintas comprobaciones del protocolo DUNDi que se pueden realizar en la línea de comandos, nos encontramos con:

```
CLI > dundi show peers
```

La cuál mostrará la dirección MAC y la IP del servidor dónde se ejecuta, y el estado actual.

También es posible realizar una búsqueda de extensiones desde los distintos servidores, averiguando así en qué servidor están definidas, mediante el comando:

```
CLI > dundi lookup exten@recursodundi
```

Por ejemplo, desde el servidor A es posible hacer una búsqueda de una extensión que esté definida en el servidor B o C, el resultado de dicha búsqueda dirá el protocolo usado, el enlace troncal que ha sido utilizado para la conexión, las direcciones IP y MAC del servidor dónde está alojada dicha extensión, y si existe o no.

Ejemplo

El objetivo es implementar, a modo de ejemplo, el protocolo DUNDi para interconectar tres servidores Asterisk A, B y C, logrando establecer llamadas entre ellos a través de las extensiones locales descritas en sus planes de marcación correspondientes.

Cada servidor gestiona las comunicaciones de una red local formada por tres terminales definidos en cada uno de ellos, y con una numeración distinta en función del servidor al que pertenezcan.

Será necesario crear un plan de marcación para cada uno y configurar los archivos *iax.conf*, *extensions.conf*, *dundi.conf*, e *iax.conf* así como crear unas llaves para la comunicación codificada entre ellos.

Esquema:

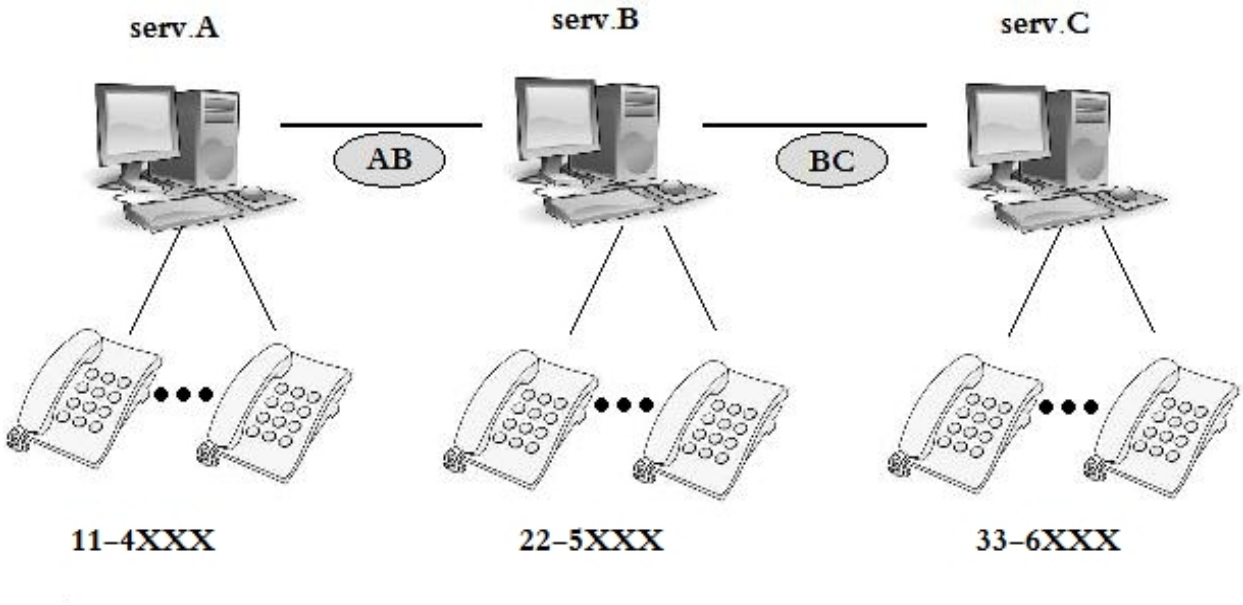


Figura 24: "Interconexión de tres servidores Asterisk mediante DUNDi"

Plan de marcación:

	Serv. A	**Serv. B**	**Serv. C**
Prefijo	11	22	33
Nº locales	4XXX	5XXX	6XXX

El servidor A tiene dentro de su red local tres extensiones, 4001, 4002 y 4003, las cuales se comunican internamente sin necesidad de un prefijo, sin embargo, para poder contactar con ellas desde otro servidor habrá que marcar 11 + nº local.

Los servidores B y C funcionan de la misma forma, tienen tres extensiones cada uno, desde la 5001 a la 5003 en el servidor B y desde la 6001 a la 6003 en el servidor C.

Para llamar a ellas desde otro servidor ser hará mediante el prefijo correspondiente, 22 para las extensiones que estén definidas en el servidor B, y 33 pasa las que lo estén en el servidor C.

La comunicación entre el servidor A y el servidor B se llevará a cabo gracias al recurso DUNDi AB que se definirá posteriormente en el archivo *dundi.conf*

Así mismo, la comunicación entre los servidores B y C será gracias al recurso DUNDi BC definido de igual forma en el archivo *dundi.conf*

iax.conf:

Se crea un enlace troncal IAX en el archivo */etc/asterisk/iax.conf* para tener comunicación con los otros dos servidores. Este archivo, en cada servidor, contendrá los siguientes parámetros.

```
[troncal]
    type=friend              ;especifica el tipo de comunicación, en este caso bidireccional
    dbsecret=dundi/secret    ;contraseña que se usará para enviar información a otro servidor
    context=entrantesdundi   ;contexto dónde serán redirigidas las llamadas entrantes de otros servidores
```

llaves de autenticación:

Creación de llaves de autenticación que permiten tener acceso a los demás servidores por medio de un enlace troncal iax.

Para crearlas, en el directorio */var/lib/asterisk/keys* se tecleará el siguiente comando:

```
#astgenkey -n "nombre_llave"
```

Donde se crearán dos llaves con el nombre elegido y con las extensiones *pub* y *key*, las cuales se compartirán con el servidor más próximo.

En el servidor A se crearán unas llaves llamadas *llaveAB.pub* y *llaveAB.key*, las cuáles se copiarán al servidor B en el directorio */var/lib/asterisk/keys.*

En el servidor C se realizará la misma operación, pero en este caso, las llaves que se copiarán al servidor B se llamarán *llaveBC.pub* y *llaveBC.key*

Por lo tanto, el servidor A contendrá las llaves "`llaveAB`", el servidor B las llaves "`llaveAB`" y "`llaveBC`" y el servidor C las "`llaveBC`". Estando todos comunicados entre sí gracias al servidor B.

extensions.conf:

El archivo de configuración *extensions.conf* contendrá los siguientes contextos:

- `[general]`: define las características comunes para todos los contextos
- `[buscardundi]`: contexto último para encontrar la extensión solicitada en otros servidores por medio del recurso DUNDi definido en *dundi.conf*
- `[compartir]`: contexto dónde se publican las extensiones que se comparten con otros servidores, con el formato: prefijo + nº local
- `[entrantesdundi]`: contexto al que llegan las llamadas entrantes de otros servidores, dichas llamadas será redirigidas al contexto local correspondiente
- `[local]`: este contexto (o contextos) incluye a todas las extensiones del plan de marcación

servidor A:

```
[general]
      static=yes
      writeprotect=no
      autofallthrough=no
      clearglobalvars=yes
      priorityjumping=no

[buscardundi]
      switch => DUNDi/AB      ;búsqueda a través del recurso DUNDi AB
      switch => DUNDi/BC      ;búsqueda a través del recurso DUNDi BC

[compartir]
      exten => _114XXX,1,NoOp ;extensiones del servidor A a publicar

[entrantesdundi]
      exten => _114XXX,1,Goto(local-a,${EXTEN:2},1)
      ;redirecciona al contexto [local-a], aquellas llamadas para las extensiones
      114XXX, eliminando el prefijo usado en dicho servidor.
```

```
[local-a]
      include => buscardundi  ;en caso de que la extensión destino no se
                              encuentre en este contexto se buscará en
                              [buscardundi], que hará uso del recurso
                              correspondiente para realizar una búsqueda en
                              los servidores B o C
      exten => 4001,1,Dial(SIP/4001,10,tT)
      exten => 4001,2,Hangup

      exten => 4002,1,Dial(SIP/4002,10,tT)
      exten => 4002,2,Hangup

      exten => 4003,1,Dial(SIP/4003,10,tT)
      exten => 4003,2,Hangup
```

servidor B:

```
[general]
      static=yes
      writeprotect=no
      autofallthrough=no
      clearglobalvars=yes
      priorityjumping=no

[buscardundi]
      switch => DUNDi/AB
      switch => DUNDi/BC

[compartir]
      exten => _225XXX,1,NoOp

[entrantesdundi]
      exten => _225XXX,1,Goto(local-b,${EXTEN:2},1)

[local-b]
      include => buscardundi

      exten => 5001,1,Dial(SIP/5001,10,tT)
      exten => 5001,2,Hangup

      exten => 5002,1,Dial(SIP/5002,10,tT)
```

```
exten => 5002,2,Hangup

exten => 5003,1,Dial(SIP/5003,10,tT)
exten => 5003,2,Hangup
```

servidor C:

```
[general]
      static=yes
      writeprotect=no
      autofallthrough=no
      clearglobalvars=yes
      priorityjumping=no

[buscardundi]
      switch => DUNDi/AB
      switch => DUNDi/BC

[compartir]
      exten => _336XXX,1,NoOp

[entrantesdundi]
      exten => _336XXX,1,Goto(local-c,${EXTEN:2},1)

[local-c]
      include => buscardundi

      exten => 6001,1,Dial(SIP/6001,10,tT)
      exten => 6001,2,Hangup

      exten => 6002,1,Dial(SIP/6002,10,tT)
      exten => 6002,2,Hangup

      exten => 6003,1,Dial(SIP/6003,10,tT)
      exten => 6003,2,Hangup
```

dundi.conf:

Este archivo de configuración definirá los parámetros y recursos necesarios para que la comunicación se lleve a cabo usando el protocolo DUNDi

Se usarán como direcciones MAC de los servidores A, B y C las siguientes combinaciones: `AA:AA:AA:AA:AA:AA, BB:BB:BB:BB:BB:BB y CC:CC:CC:CC:CC:CC`

Y como direcciones IP de dichos servidores: `aaa.aaa.aaa.aaa, bbb.bbb.bbb.bbb` y `ccc.ccc.ccc.ccc`

servidor A:

```
[general]
      department=IT
      organization=UPCT
      locality=CT
      stateprov=MU
      country=SP
      email=prueba@voip.com
      phone=666666666
      bindaddr=0.0.0.0 ;Dirección IP de conexión con el servidor. En este caso
                       escucha en cualquier interfaz
      port=4520        ;puerto DUNDi por defecto
      entityid=AA:AA:AA:AA:AA:AA ;identificador de red o dirección MAC
      cachetime=5      ;Tiempo máximo de búsqueda de respuesta., en
                       segundos Por defecto 3600
      ttl=3            ;time to live, tiempo de vida de la consulta o número de
                       saltos
      autokill=yes     ;en caso de no obtener respuesta de una extensión,
                       Asterisk envía CANCEL a los peers

[mappings]
AB=>compartir,0,IAX2,troncal:${SECRET}@aaa.aaa.aaa.aaa/
${NUMBER},nopartial
```

`AB:` *recurso DUNDi que se va a compartir con los demás nodos de la red*

`compartir`: *contexto definido en el plan de llamadas que contendrá todas las rutas que se van a compartir*

`0`: *peso de la ruta compartida. Si son extensiones locales se usará 0, en caso de que sean rutas que no tengan conexión directa se usará un número mayor. Si existen varias rutas para un mismo número se usará la de menor peso.*

`IAX2`: *protocolo usado para la comunicación DUNDi*

`troncal`: *enlace definido en el archivo iax.conf para la comunicación de los servidores*

`${SECRET}`: *variable que contiene la clave del recurso DUNDi y que se usa para autenticar el enlace troncal*

`aaa.aaa.aaa.aaa`:*dirección IP del servidor A*

`${NUMBER}`:*variable que contendrá la extensión solicitada*

`nopartial`: *opción usada para prevenir que se entreguen resultados que contengan sólo una parte de la extensión*

```
[BB:BB:BB:BB:BB:BB]   ;identificador de red con el que se comparte el recurso DUNDi (puede ser la dirección MAC)
model=symmetric       ;permite hacer y recibir consultas
host=bbb.bbb.bbb.bbb  ;dirección IP del servidor B
inkey=llaveAB         ;nombre de la llave usada para recibir la consulta
outkey=llaveAB        ;nombre de la llave usada para realizar la consulta
include=all           ;nombre de aquello que queramos compartir con el servidor. Es posible usar all para permitir todo el mapeo
permit=all            ;contextos DUNDi a los que tendrá acceso este nodo
qualify=yes           ;se controla periódicamente que la conexión con el nodo esté activa
order=primary         ;orden de búsqueda para cada servidor. Puede ser primary,secondary, tertiary y quartiary
```

servidor B:

```
[general]
        department=IT
        organization=UPCT
        locality=CT
        stateprov=MU
        country=SP
        email=prueba@voip.com
        phone=666666666
        bindaddr=0.0.0.0
        port=4520
        entityid=BB:BB:BB:BB:BB:BB
        cachetime=5
        ttl=3
        autokill=yes
```

```
            [mappings]
            AB=>compartir,0,IAX2,troncal:${SECRET}@bbb.bbb.bbb.bbb/$
{NUMBER},nopartial
            BC=>compartir,0,IAX2,troncal:${SECRET}@bbb.bbb.bbb.bbb/$
{NUMBER},nopartial
            [AA:AA:AA:AA:AA:AA]
                  model=symmetric
                  host=aaa.aaa.aaa.aaa
                  inkey=llaveAB
                  outkey=llaveAB
                  include=all
                  permit=all
                  qualify=yes
                  order=primary

            [CC:CC:CC:CC:CC:CC]
                  model=symmetric
                  host=ccc.ccc.ccc.ccc
                  inkey=llaveBC
                  outkey=llaveBC
                  include=all
                  permit=all
                  qualify=yes
                  order=primary
```

servidor C:

```
            [general]
                  department=IT
                  organization=UPCT
                  locality=CT
                  stateprov=MU
                  country=SP
                  email=prueba@voip.com
                  phone=666666666
                  bindaddr=0.0.0.0
                  port=4520
                  entityid=CC:CC:CC:CC:CC:CC
                  cachetime=5
                  ttl=3
```

```
            autokill=yes

      [mappings]
      BC=>compartir,0,IAX2,troncal:${SECRET}@bbb.bbb.bbb.bbb/$
{NUMBER},nopartial
      [BB:BB:BB:BB:BB:BB]
            model=symmetric
            host=bbb.bbb.bbb.bbb
            inkey=llaveBC
            outkey=llaveBC
            include=all
            permit=all
            qualify=yes
            order=primary
```

sip.conf

En este archivo se definirán las extensiones locales para cada servidor y su contexto correspondiente.

Servidor A:

```
[general]
      bindport=4569
      bindaddr=0.0.0.0
      delayreject=yes
      disallow=all
      allow=gsm
      allow=ulaw
      allow=alaw
      autokill=yes

[4001]
      type=friend
      username=4001
      secret=pass
      host=dynamic
      callerid=4001
      context=local-a
      qualify=yes
```

```
[4002]
      type=friend
      username=4002
      secret=pass
      host=dynamic
      callerid=4002
      context=local-a
      qualify=yes

[4003]
      type=friend
      username=4003
      secret=pass
      host=dynamic
      callerid=4003
      context=local-a
      qualify=yes
```

Servidor B:

```
[general]
      bindport=4569
      bindaddr=0.0.0.0
      delayreject=yes
      disallow=all
      allow=gsm
      allow=ulaw
      allow=alaw
      autokill=yes

[5001]
      type=friend
      username=5001
      secret=pass
      host=dynamic
      callerid=5001
      context=local-b
      qualify=yes
```

```
[5002]
      type=friend
      username=5002
      secret=pass
      host=dynamic
      callerid=5002
      context=local-b
      qualify=yes

[5003]
      type=friend
      username=5003
      secret=pass
      host=dynamic
      callerid=5003
      context=local-b
      qualify=yes
```

Servidor C:

```
[general]
      bindport=4569
      bindaddr=0.0.0.0
      delayreject=yes
      disallow=all
      allow=gsm
      allow=ulaw
      allow=alaw
      autokill=yes

[6001]
      type=friend
      username=6001
      secret=pass
      host=dynamic
      callerid=6001
      context=local-c
      qualify=yes
```

```
[6002]
      type=friend
      username=6002
      secret=pass
      host=dynamic
      callerid=6002
      context=local-c
      qualify=yes

[6003]
      type=friend
      username=6003
      secret=pass
      host=dynamic
      callerid=6003
      context=local-c
      qualify=yes
```

Capítulo 5:
CASOS PRÁCTICOS

En este apartado se detallarán las implementaciones realizadas en el laboratorio con el fin de el funcionamiento de las distintas posibilidades que ofrece la centralita Asterisk en cuánto a configuración y uso.

Uso y distribución del laboratorio:

El laboratorio IT-2 está compuesto por 15 ordenadores con una distribución SUSE de Unix, que usaremos hasta el final de los casos de uso.

La disposición es de 3 ordenadores por banco de trabajo. Se usará esto para definir una red por cada uno de los bancos, dando lugar a 5 redes distintas, con las direcciones de red desde la 192,198,1,0 a la 192,168,5,0

Cada red está compuesta por tres ordenadores, dos de ellos tienen configurados los clientes de VoIP: Kiax y Ekiga, y el tercero tendrá la centralita Asterisk v.1.8.3.2 instalada y corriendo continuamente en el sistema.

El esquema del laboratorio sería el siguiente:

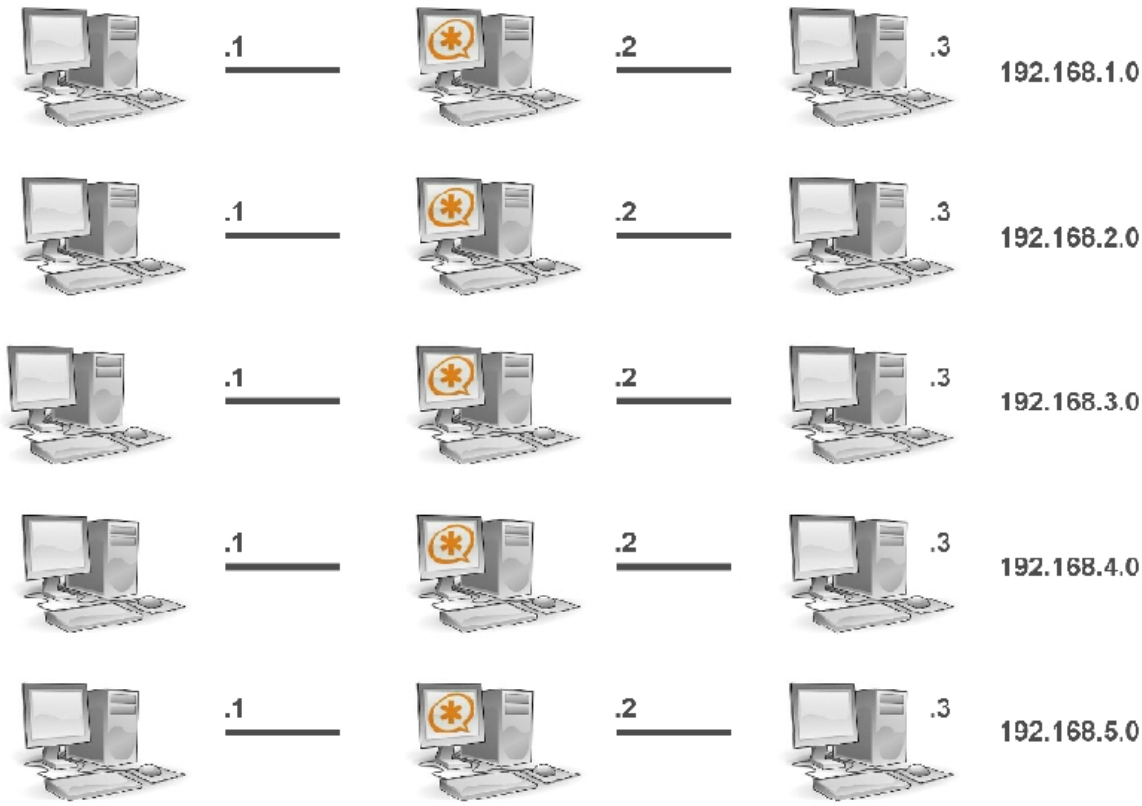


Figura 25: “Esquema del laboratorio”

5.1 CONFIGURACIÓN Y PUESTA A PUNTO

Objetivos:

- Configurar las direcciones IP de la red
- Definición de extensiones
- Definir los clientes SIP e IAX de la centralita
- Configurar los softphones Kiax y Ekiga
- Crear un plan de marcación básico que permita llamar a todos los terminales

Metodología:

Figura 26: "Esquema de conexión de la red 1"

El primer caso práctico consistirá en el uso de una red de telefonía gestionada por la centralita Asterisk y que permita la comunicación sin restricciones entre todos sus componentes.

Se usará una única red para la configuración de prueba, creando a su vez una comunicación básica con la centralita Asterisk.

Para ello, dos ordenadores actuarán como clientes usando los protocolos SIP e IAX y un ordenador actuará como servidor.

Los clientes serán los ordenadores primero y tercero de cada banco de trabajo e implementarán los softphones Kiax y Ekiga, que trabajarán con los protocolos IAX y SIP respectivamente.

El servidor tendrá instalada la centralita de Asterisk v.1.8.3.2, la cuál se encontrará corriendo en el servidor para dar funcionamiento al plan de marcación de llamadas y para identificar los distintos clientes definidos ahí.

Configuración de las direcciones IP de la red:

Se usará a modo de ejemplo la red 1, situada en el primer banco de trabajo, con dirección 192.168.1.0.

Los ordenadores que la conforman tendrán las direcciones de red 192.198.1.1, 192.168.1.2 y 192.168.1.3, definidas de izquierda a derecha.

Para ello se usará el comando:

```
#ifconfig eth0 192.168.1.X netmask 255.255.255.0
```

Dónde X tomará los valores 1, 2 ó 3 según el ordenador y la posición en el banco; empezando a numerar de izquierda a derecha.

Definición de extensiones:

Cada extensión tendrá un formato determinado por el número de la red a la que pertenece, el protocolo y la ubicación del cliente.

El formato será el siguiente:

```
extensión = nºred+protocolo+ubicación
```

- `nºred`: dígito que tomará los valores de 1 a 5, dependiendo de la red a la que pertenezca y al banco de trabajo dónde esté definida.
- `Protocolo`: está definido por una dígito que contendrá los valores: 0 en caso de que el protocolo usado sea SIP, ó 1 en caso de que sea IAX.
- `Ubicación`: usará dos dígitos, de forma que si en un momento determinado la red crece en número de clientes aún sea posible implementarlos sin modificar en exceso el código. De esta forma admitiría 100 clientes conectados a la centralita por cada protocolo. En este caso las posibles combinaciones son 01 y 03, que identifican a su vez al ordenador en el que están definidas.

Por lo tanto, en la red de este ejemplo, para las extensiones que usen el protocolo SIP, el rango de extensiones será de la forma 100X, siendo X el último número de la dirección de red del ordenador, es decir, la ubicación física en el banco de trabajo. Para el protocolo IAX las extensiones tendrán el formato 110X.

Esta forma de numeración de las extensiones y de las direcciones de red de los ordenadores, permitirá identificar rápidamente a qué extensión nos referimos y dónde está situada físicamente.

La configuración de cada ordenador se compondrá por la definición de la dirección de red, la extensión SIP y la extensión IAX que están definidas en los clientes Ekiga y Kiax respectivamente.

Por ejemplo, el ordenador situado a la izquierda del banco de trabajo tendrá como dirección IP: 192.168.1.1, como extensión SIP definida en el cliente Ekiga: 1001 y como extensión IAX definida en Kiax: 1101.

Definir los clientes SIP e IAX de la centralita

Los ordenadores 1 y 3 de cada banco de trabajo funcionarán como clientes del servidor que está situado en el puesto central.

Dichos clientes tendrán definidas extensiones que habrán de ser configuradas en el servidor para que se pueda establecer la comunicación.

Los archivos que contienen la configuración de las extensiones y la forma de uso son *sip.conf* e *iax.conf*

El archivo *sip.conf* tendrá los siguientes contextos:

- `[general]:` especifica los parámetros generales de la configuración, como puerto e interfaz de conexión con el servidor, así como los códecs permitidos.
- `[1001]:` definición de la extensión 1001, la cuál especifica el tipo de usuario, el nombre, la contraseña, el host y el contexto en el cuál está definido su uso.
- `[1003]:` define de la misma forma la configuración de la extensión 1003.

sip.conf

```
[general]
      context=default
      port=5060
      bindaddr=0.0.0.0
      disallow=all
      allow=gsm
      allow=alaw
      allow=ulaw

[1001]
      type=friend
      username=1001
      secret=pass
      host=dynamic
      callerid=1001
      context=prueba1
      qualify=yes
```

```
[1003]
      type=friend
      username=1003
      secret=pass
      host=dynamic
      callerid=1003
      context=prueba
      qualify=yes
```

Se usa el puerto UDP por defecto de Asterisk para el protocolo SIP, el `5060`.

La interfaz de conexión con el servidor Asterisk será cualquiera que esté activa, por lo tanto se hará uso de la dirección de `bindaddr 0.0.0.0`.

Se desactivan todos los códecs de forma general para especificar a continuación el orden de uso en caso de que alguno no esté activo.

Los usuarios SIP 1001 y 1003 se definirán con el tipo `friend`, indicando que podrán realizar y recibir las llamadas pertinentes.

Tanto para el nombre de usuario como para el identificador de llamada se ha usado el nombre de la extensión, sin embargo, es posible usar cualquier cadena de caracteres para definirlos. El hecho de usar el nombre de extensión es simplemente por una rápida localización.

La contraseña usada para el registro de la extensión con el servidor será `pass`, tanto para la extensión 1001 como para la 1003. Dicha contraseña deberá coincidir con la que se define en la configuración de los clientes Ekiga y Kiax.

Se ha definido el host como dinámico a pesar que la distribución del laboratorio indica que las direcciones correspondientes serían la 192.168.1.1 y 192.168.1.3. Sin embargo, el hecho de definir el host de forma dinámica permite poder cambiar la ubicación de la extensión en un momento determinado a otro host sin necesidad de cambiar el código.

El contexto asociado a ambas extensiones en el plan de marcación es el contexto prueba1, que se definirá a continuación el el archivo *extensions.conf*

La herramienta `qualify` permite determinar si la extensión es alcanzable en el tiempo de respuesta estimado, 60 segundos por defecto.

El archivo *iax.conf* definirá los siguientes contextos:

- `[general]:` especifica puerto de comunicación con Asterisk, dirección IP para la conexión con el servidor, y los posibles códecs a usar y su orden.
- `[1101]:` define el tipo de usuario, el host dónde está situada la extensión, la contraseña de registro, el identificador de llamada y el contexto
- `[1103]:` define los parámetros del usuario IAX 1103

iax.conf

```
[general]
      bindport=4569
      bindaddr=0.0.0.0
      delayreject=yes
      disallow=all
      allow=gsm
      allow=ulaw
      allow=alaw
      autokill=yes
[1101]
      type=friend
      host=dynamic
      secret=pass
      context=prueba1
      qualify=yes
      callerid=1101
      requirecalltoken=no
[1103]
      type=friend
      host=dynamic
      secret=pass
      context=prueba1
      qualify=yes
      callerid=1103
      requirecalltoken=no
```

Se usa el puerto UDP de escucha por defecto de Asterisk para el protocolo IAX, el `4569`.

Se ha configurado la dirección IP de conexión con el servidor Asterisk como `0.0.0.0`, que indica que todas las interfaces pueden comunicarse con el servidor. Esto permite en un momento determinado poder cambiar una interfaz por otra en el cliente en caso de que alguna falle.

De la misma forma que para el protocolo SIP, se deshabilitan los códecs de audio para volver a definirlos a continuación según el orden deseado. En este caso se han usado los más extendidos y los que mejores prestaciones ofrecen, `gsm`, `alaw` y `ulaw`.

A modo de finalización de la llamada en caso de que no se obtenga respuesta del cliente se ha configurado `autokill`, éste permite cancelar una llamada después de un tiempo de espera de un ACK de 2000 ms.

Se definen los dos usuarios IAX correspondientes a los clientes del banco de trabajo 1.

El tipo asignado para ambos es `friend`, permitiendo la realización y la recepción de llamadas en dichas extensiones.

Como dirección IP de la extensión, de la misma forma que para SIP, correspondería la del puesto en el que estén definidos los clientes, sin embargo, se ha optado por elegir una dirección IP dinámica la cuál permite cualquier dirección IP válida.

Se ha definido como contraseña la cadena de caracteres `pass`, la cuál se usará para el registro del cliente con el servidor.

El contexto dónde están definidas ambas extensiones será prueba1, al igual que para las extensiones SIP.

Como identificador de llamada se ha usado el mismo nombre de la extensión, 1101 ó 1103, según corresponda.

Como forma de ver si la extensión es alcanzable o no durante un tiempo máximo de 60 segundos, se ha usado la opción `qualify`.

Configurar los softphones Kiax y Ekiga

Como se ha expuesto anteriormente, cada ordenador usará los clientes Kiax y Ekiga y por lo tanto tendrán definidas las extensiones correspondientes.

Para el ordenador con dirección IP 192.168.1.1, se definirá el cliente SIP con la extensión 1001 y el cliente IAX con la 1101.

Para el ordenador con dirección IP 192.168.1.3, se definirá el cliente SIP con la extensión 1003 y el cliente IAX con la 1103.

Para configurar una extensión en el cliente Ekiga se seguirá el siguiente procedimiento:

Se creará una cuenta de Ekiga, en el menú "Edit" se seleccionará la opción "Accounts" y posteriormente, seleccionando "Add" se añadirá la cuenta correspondiente a la extensión definida en el servidor.

Los valores a configurar en el cliente serán:

- Account Name: nombre de la cuenta SIP, en este caso serán 1001 y 1003 en función de la ubicación del ordenador.
- Protocol: SIP
- Registrar: 192.168.1.2 (dirección IP del servidor Asterisk)
- Password: contraseña definida en el archivo *sip.conf* para dicho cliente. En este caso será `pass`.

Para configurar una extensión en el cliente Ekiga se hará de la siguiente forma:

Una vez abierto el cliente, se seleccionará "File", "Settings" y en la ventana que aparecerá se selecciona la pestaña "Accounts". Pinchando sobre "New Account" se podrá definir la nueva cuenta.

Los parámetros a configurar serán:

- Account Name: nombre de la cuenta IAX usada, en este caso será 1101 ó 1103, en función del ordenador en el que se configure.
- IAX Server: 192.168.1.2 (dirección IP del servidor Asterisk)
- Username: nombre de usuario de la cuenta creada en Asterisk, se usará el mismo que el nombre de la cuenta, 1101 ó 1103.
- Password: `pass`, definida en el archivo *iax.conf*
- CallerID Name: 1101 ó 1103
- CallerID Number: 1101 ó 1103
- Preferred Codec: GSM, códec permitido por Kiax y disponible en la centralita Asterisk.

Crear un plan de marcación básico que permita llamar a todos los terminales

El archivo *extensions.conf* contendrá el plan de marcación de la centralita. En este caso práctico se ha optado por configurar uno lo más simple posible y que abarque todas las extensiones. Por tanto, se ha creado un único contexto en el cuál estarán definidas todas y cada una de las extensiones que conforman la red del banco de trabajo 1.

Dicho archivo se compone de dos contextos:

- `[general]:` define los comandos generales del plan de marcación, como guardar el plan desde la línea de comandos, terminar la llamada si queda inactiva o borrar las variables globales al reiniciar la centralita.
- `[prueba1]:` define las extensiones y las aplicaciones de las que hacen uso.

extensions.conf

```
[general]
      static = yes
      writeprotect = no
      autofallthrough = yes
      clearglobalvars = no
[prueba1]
      exten => 1001,1,Dial(SIP/1001,20,m)
      exten => 1001,2,Hangup

      exten => 1003,1,Dial(SIP/1003,20,m)
      exten => 1003,2,Hangup

      exten => 1101,1,Dial(IAX2/1101,20)
      exten => 1101,2,Hangup

      exten => 1103,1,Dial(IAX2/1103,20)
      exten => 1101,2,Hangup
```

Se ha configurado la opción de guardar los cambios realizados desde la línea de comandos usando la opción `dialplan reload`, esta opción es posible gracias a la configuración de `static` y `writeprotect`.

En caso de que la llamada se quede a la espera de una instrucción que no llega, se finaliza según corresponda, con `BUSY`, `CONGESTION` o `HANGUP`. Se ha optado por esta medida de seguridad

para permitir liberar el canal correspondiente a la llamada en un tiempo razonable, no pudiendo quedar la llamada establecida de forma indefinida sin respuesta.

Se ha especificado que tras un reinicio de Asterisk o del plan de marcación no se borren las variables globales, a fin de que sigan manteniendo los valores con las que han sido configuradas, a no ser que una instrucción concreta obligue a su borrado.

Se ha creado un único contexto para todas las extensiones permitiendo así que todas se puedan comunicar con todas. Cada extensión ejecuta la aplicación `Dial()`, la cuál usa el protocolo definido para tal caso SIP o IAX2 y la extensión a la que va dirigida la llamada. El tiempo máximo de espera de respuesta es de 20 segundos.

También se ha configurado la opción `m` para las extensiones SIP 1001 y 1003, la cuál pone una música de espera de fondo mientras la llamada se está ejecutando. De esta forma se diferencia cuándo se llama a una extensión SIP y cuándo a una IAX con sólo escuchar el sonido.

5.2 COMUNICACIÓN RESTRICTIVA

Objetivos:

- Modificar el plan de marcación según el tipo de cliente
- Modificar la definición de los clientes SIP e IAX

Metodología:

Figura 27: "Comunicación restrictiva"

El objetivo de este caso práctico es comprobar el funcionamiento del plan de marcación cuándo hay restricción de llamadas del tipo que sea.

En este caso se ha optado por hacer una distinción entre protocolos, por lo tanto las extensiones del protocolo SIP tendrán una definición distinta que las del protocolo IAX. Para ello será necesario hacer una división de contextos en el plan de marcación según el tipo de extensión y protocolo usado, también será necesario reconfigurar la definición de los clientes puesto que el contexto del plan de marcación variará.

Se parte desde la base de que tanto softphones como las direcciones IP están configurados para tener comunicación con la centralita.

Modificar el plan de marcación según el tipo de cliente

Para comprobar de primera mano cómo funciona la restricción de llamadas según el tipo de extensión, se ha creado un plan de marcación que separa los contextos en función del protocolo usado por el cliente.

En éste caso, el archivo *extensions.conf* constará de los siguientes contextos:

- `[general]:` define los parámetros generales de la configuración
- `[prueba2-sip]:` define las extensiones que usan el protocolo SIP, en este caso las 1001 y 1003, y además incluye el contexto [prueba2-iax] para que las extensiones sip puedan realizar llamadas a las iax
- `[prueba2-iax]:` define las extensiones que usan IAX, 1101 y 1103

extensions.conf

```
[general]
      static = yes
      writeprotect = no
      autofallthrough = yes
      clearglobalvars = no
[prueba2-sip]
      exten => 1001,1,Dial(SIP/1001,20,m)
      exten => 1001,2,Hangup

      exten => 1003,1,Dial(SIP/1003,20,m)
      exten => 1003,2,Hangup

      include => prueba2-iax

[prueba2-iax]
      exten => 1101,1,Dial(IAX2/1101,20)
      exten => 1101,2,Hangup

      exten => 1103,1,Dial(IAX2/1103,20)
      exten => 1103,2,Hangup
```

El contexto `[prueba2-sip]` contiene al contexto `[prueba2-iax]`, por lo que los clientes SIP podrán realizar llamadas a clientes SIP e IAX, mientras que los clientes iax sólo pueden comunicarse entre ellos mismos.

Modificar la definición de los clientes SIP e IAX

Se ha optado por separar el plan de marcación según el protocolo que usen las extensiones, por lo tanto es necesario modificar los archivos *sip.conf* e *iax.conf* para que el contexto se corresponda con el del plan de marcación.

El resto de definición de ambos archivos quedará tal y cómo estaba con la excepción siguiente:

sip.conf

```
[general]
…
[1001]
…
context=prueba2-sip
…
[1003]
…
context=prueba2-sip
…
```

iax.conf

```
[general]
…
[1101]
…
context=prueba2-iax
…
[1103]
…
context=prueba2-iax
…
```

5.3 INTERCONEXIÓN DE DOS REDES

Objetivos:

- Definir la forma de conexión de dos redes
- Realizar la configuración IP necesaria para la interconexión
- Creación de un nuevo cliente SIP e IAX para tal efecto
- Modificación del plan de marcación en contextos internos y externos

Metodología:

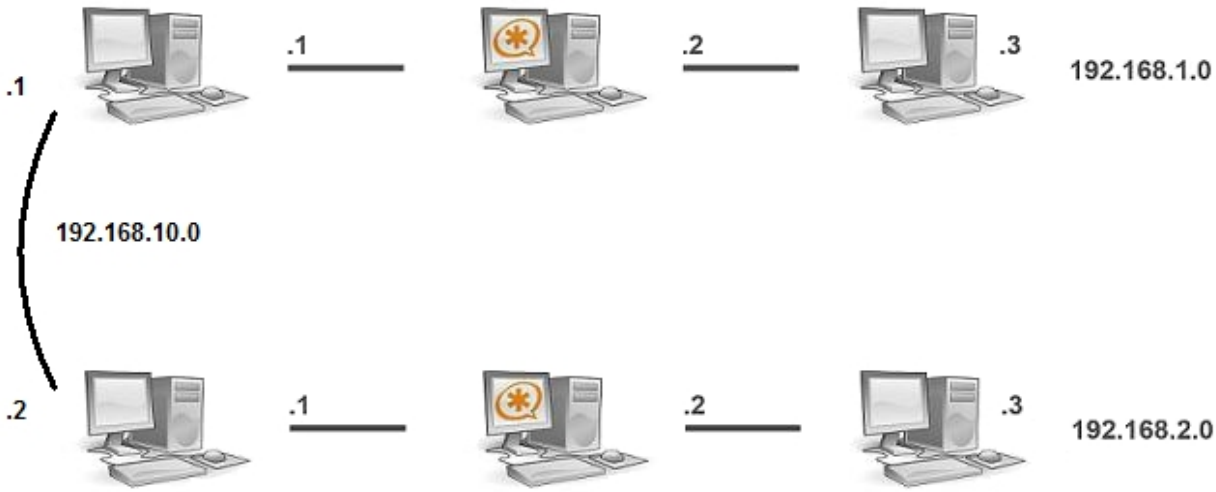


Figura 28: "Esquema de conexión de las redes 1 y 2"

El objetivo principal de este caso práctico es la interconexión de dos redes, cada una con su propio servidor Asterisk, para permitir la realización de llamadas entre las distintas extensiones de cada red sin ningún tipo de problema.

Se partirá del caso dos, por lo tanto se distinguirá en el plan de marcación entre los dos tipos de protocolos usados SIP e IAX y además tendrán características y opciones distintas.

Definir la forma de conexión de dos redes

Haciendo uso de las características físicas del laboratorio, la interconexión de dos redes se hará gracias a una segunda interfaz Ethernet de la que disponen los ordenadores marcados con el número uno de cada banco.

Para ello bastará con configurar dichos ordenadores de ambas redes en modo router y realizar las configuraciones necesarias para permitir la comunicación.

La conexión lógica se hará gracias a la creación de un nuevo cliente tanto en *sip.conf* como en *iax.conf*, que apuntará al servidor de la red contraria y permitirá la comunicación entre ambas centralitas.

Las llamadas que lleguen a través de este cliente se considerarán internas de dicha red, ya que llegan a una extensión definida en ella.

Realizar la configuración IP necesaria para la interconexión:

Los ordenadores del banco 1 tendrán la siguiente configuración:

- PC1: Direcciones IP: 192.168.1.1 y 192.168.10.1, está última corresponde con la red punto a punto conectada con el PC1 del banco 2
  Clientes: 1001 SIP, 1101 IAX
  Usado como router

- PC2: Dirección IP: 192.168.1.2
  Servidor Asterisk

- PC3: Dirección IP: 192.168.1.3
  Clientes: 1003 SIP, 1103 IAX

Los ordenadores del banco 2 tendrán la siguiente configuración:

- PC1: Direcciones IP: 192.168.2.1 y 192.168.10.2, siendo la última la correspondiente a la red punto a punto que se conecta con el PC1 del banco de trabajo 1.
  Clientes: 2001 SIP, 2101 IAX
  Usado como router

- PC2: Dirección IP: 192.168.2.2
  Servidor Asterisk

- PC3: Dirección IP: 192.168.2.3
  Clientes: 2003 SIP, 2103 IAX

Los comandos necesarios para configurar una dirección IP son:

```
#ifconfig ethX 192.168.X.X netmask 255.255.255.0
```

Modificando el valor de X según sea necesario para cada caso. Por ejemplo, para los ordenadores numerados con 1 habrá que configurar dos interfaces Ethernet, puesto que se conectan con dos redes. El resto de ordenadores sólo tendrán una disponible.

Será necesario configurar el gateway por defecto por el que se desviarán las búsquedas que no correspondan a esa red.

Se tomará como gateway de cada red el PC1, puesto que es el que se usa como router de conexión con la red contraria.

Por lo tanto los PCs 2 y 3 de cada red se configurarán de la siguiente forma:

```
#route add default gw 192.169.X.1
```

Para el PC1 de ambas redes, el gateway será distinto, se usará la dirección IP de la interfaz que se conecta a la red punto a punto.

Los comandos a usar serán:

```
#route add default gw 192.168.10.1    para el PC1 de la red 1
#route add default gw 192.168.10.2    para el PC1 de la red 2
```

Además, ambos ordenadores deberán estar configurados en modo router, permitiendo así la conexión de los dos bancos de trabajo.

Para ello se hará uso de las siguientes instrucciones:

```
#echo “1”/proc/sys/net/ipv4/ip-forward
```

Creación de un nuevo cliente SIP e IAX para tal efecto

Será necesaria la creación de un nuevo cliente, tanto para el protocolo SIP como para el IAX, que apuntará al servidor de la red contraria y servirá para establecer una comunicación entre ambas.

Para ello se ha creado un nuevo cliente, en el banco de trabajo que conforma la red 1, llamado `[red2]`, el cuál especificará el ordenador dónde está ubicado y el contexto al cuál pertenece.

El ordenador dónde está ubicado será el servidor de la red contraria y el contexto será el de la propia red, usando un contexto como si de una llamada interna se tratase.

No está permitido el uso de contraseña para el nuevo cliente puesto que genera un error de conexión.

Para la red 1 la configuración será la siguiente:

sip.conf

```
[general]
      …
[1001]
      type = friend
      …
      context = prueba3-sip
      …
[1003]
      type = friend
      …
      context = prueba3-sip
      …
[red2]
      type = friend
      username = red2
      host = 192.168.2.2
      callerid = red2
      context = internas-sip1
      qualify = yes
```

El contexto usado para el cliente `[red2]` es un contexto propio de la red 1 que permite que las llamadas que provengan de la red 2 puedan comunicarse con las extensiones de la red 1.

Al usar un contexto interno se evita que se forme un bucle al llamar de una red a otra. Es decir, una llamada de la red 1 hacia la red 1 nunca pasará a través de la red 2, ya que esto se evita al definir el uso como interno. O una llamada proveniente de la red 2 no podrá tener como destino la red 2 y pasar a través del cliente que comunica ambas redes, sería redundante.

El archivo *iax.conf* contendrá el cliente `red2` definido de la misma forma que en *sip.conf*, con la excepción de que el contexto usado será `internas-iax1`

Modificación del plan de marcación en contextos internos y externos:

El archivo *extensions.conf* se dividirá por protocolos de comunicación que a su vez estarán divididos de forma interna y externa según la extensión destino.

Las extensiones propias de la red estarán definidas en el contexto `[internas-sip1]` o [internas-iax1]. Y las extensiones que corresponden con la otra red en los contextos `[externas-sip]` o `[externas-iax]`.

Así mismo, las extensiones definidas para usar el protocolo SIP podrán llamar tanto a las que usan SIP como a las que usan IAX

Los contextos serán los siguientes:

- `[prueba3-sip]`: `[internas-sip1]`: define las extensiones que usan SIP en la red
  `[externas-sip]`: define la forma de llamar a las extensiones de la red 2 que usan SIP a través del cliente `[red2]` incluye el contexto `[prueba3-iax]`
- `[prueba3-iax]`: `[internas-iax1]`: define las extensiones que usan IAX en la red
  `[externas-iax]`: define la forma de llamar a las extensiones de la red 2 que usan IAX a través del cliente `[red2]`

extensions.conf

```
[general]
      static = yes
      writeprotect = no
      autofallthrough = yes
      clearglobalvars = no

;--------------------------------------------
;SIP
;--------------------------------------------

[prueba3-sip]
      include => internas-sip1
      include => externas-sip
      include => prueba3-iax

[internas-sip1]
      exten => 1001,1,Dial(SIP/1001,20,m)
      exten => 1001,2,Hangup

      exten => 1003,1,Dial(SIP/1003,20,m)
      exten => 1003,2,Hangup

[externas-sip]
      exten => _200X,1,Dial(SIP/red2/${EXTEN},20,m)
      exten => _200X,2,Hangup

;--------------------------------------------
;IAX
;--------------------------------------------

[prueba3-iax]
      include => internas-iax1
      include => externas-iax
```

```
[internas-iax1]
      exten => 1101,1,Dial(IAX2/1101,20)
      exten => 1101,2,Hangup

      exten => 1103,1,Dial(IAX2/1103,20)
      exten => 1103,2,Hangup

[externas-iax]
      exten => _210X,1,Dial(IAX2/red2/${EXTEN},10)
      exten => _210X,2,Hangup
```

Un ejemplo de comunicación entre un cliente de la red definida en el banco 2, con un cliente de la red del banco 1 sería de la siguiente forma:

Un cliente cualquiera, por ejemplo SIP, quiere realizar una llamada a otro cliente de la red contraria.

Para ello buscará en el archivo *sip.conf* de la red 2 sus permisos, dicho cliente se puede comunicar con cualquier extensión, ya que `[prueba3-sip]` le da permiso para llamara a cualquiera, sea interna o externa.

Dicho cliente puede llamar a una extensión de la red 1, como puede ser `100X` o `110X`, lo que le enrutará a la red correspondiente gracias al cliente `[red1]` que permitirá la comunicación con dicha red.

Suponiendo que llama a un cliente SIP, como pueden ser `1001` ó `1003`, el servidor de la red contraria detectará que un cliente SIP llamado `[red2]` quiere establecer comunicación con una de sus extensiones internas en la red 1.

Buscará en el archivo *sip.conf* para ver los permisos del cliente `[red2]` y comprobará que incluye el contexto `[internas-sip1]`. Dicho contexto, en el archivo *extensions.conf* tendrá definidas las extensiones `1001` y `1003`, por lo que la llamada podrá cursarse con éxito y el cliente SIP de la red 2 habrá mantenido una comunicación con el cliente SIP de la red 1 que haya elegido.

NOTA: La configuración de la red correspondiente al banco 2 será igual, salvo algunas pequeñas modificaciones. Las extensiones comenzarán por 2, en lugar de 1. El cliente definido para interconectar ambas redes se llamará `[red1]` puesto que es a la red 1 a la que apunta. Los contextos internos y externos se adaptarán a la red 2, serán llamadas internas las del tipo 2XXX y externas aquellas que empiecen con 1, ya que corresponden a la red contraria. Para llamar a la red 1 será necesario el cliente `[red1]` en lugar de `[red2]`

5.4 USO Y CONFIGURACIÓN DEL BUZÓN DE VOZ.

Objetivos:

- Definición de uso del buzón de voz
- Configuración de *voicemail.conf*
- Modificación de los clientes SIP e IAX
- Modificación del plan de marcación incluyendo las opciones propias del buzón

Metodología:

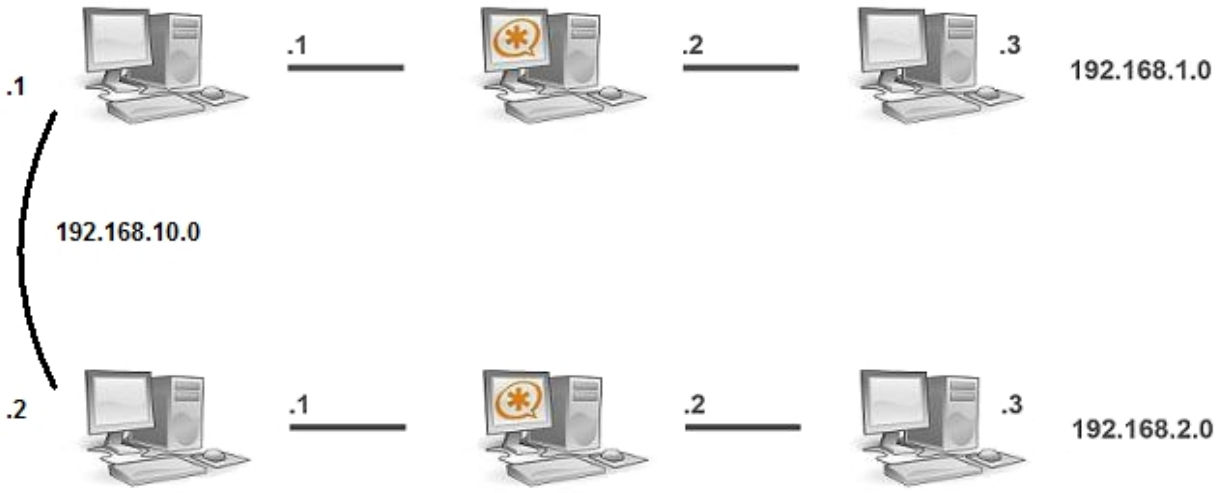


Figura 29: "Configuración del buzón de voz"

El objetivo principal de este caso práctico es configurar el buzón de voz para los clientes de forma que en caso de que la extensión destino esté ocupada o no disponible, se pueda dejar un mensaje privado para dicha extensión.

Para llevar a cabo el uso del buzón de voz habrá que configurar el archivo *voicemail.conf*, dónde se definen las extensiones y los contextos asociados, así como la zona horaria a la que pertenece.

A su vez, será necesario modificar las extensiones definidas en los archivos *sip.conf* e *iax.conf* para indicar la extensión correspondiente del buzón de voz previamente definida en el archivo *voicemail.conf* y el contexto asociado.

La configuración del plan de marcación se mantendrá, incluyendo para cada extensión interna una posibilidad distinta según la llamada sea respondida, la extensión esté ocupada o no esté disponible. En estas últimas dos opciones se ha hecho uso del uso del buzón de voz para dejar un mensaje de audio.

Para acceder al buzón de voz, cada extensión deberá marcar la extensión asociada a él, en este caso se ha usado la `2999`, y a continuación deberá hacer uso de la contraseña que se ha especificado en el archivo *voicemail.conf*.

Para las extensiones externas no será necesario usar las distintas opciones para la llamada, ya que ésta pasará a la red contraria y será ahí dónde se gestionará si la llamada es atendida o por el contrario la extensión está ocupada o no disponible.

Definición de uso del buzón de voz:

El objetivo del uso del buzón de voz es asegurar que cada extensión pueda tener un sistema de captación de audio para los mensajes que serán dejados en caso de que dicha extensión esté ocupada o no disponible.

Cada extensión tendrá asociado un buzón de voz propio dónde se guardarán dichos mensajes. El número de buzón, para evitar confusiones, será el mismo que dicha extensión.

Haciendo las diferenciaciones necesarias en el archivo *extensions.conf*, se podrá separar entre llamadas ocupadas y no disponibles, saltando en cada caso el buzón correspondiente para cada extensión.

Los mensajes del buzón quedarán guardados en el directorio */var/spool/asterisk/voicemail/context/boxnumber/INBOX/*.

Cada una de las extensiones podrá acceder a su buzón de voz propio con sólo marcar una extensión definida para tal efecto, en este caso la `2999`, ahí saltará un menú dónde se podrá elegir el mensaje que se desea, que se guarde en un sitio determinado, que se repita, o que salte al siguiente.

También será posible grabar un mensaje de menú propio, como el de extensión no válida, ocupada, o incluso el nombre asociado a dicha extensión.

Cada archivo de clientes SIP e IAX, tendrá que definir para cada extensión el buzón de voz correspondiente.

La configuración y el uso de los buzones vendrá especificado en el archivo *voicemail.conf*, dónde se definirá la zona horaria, las opciones generales y los propios buzones.

Configuración de *voicemail.conf*

El archivo *voicemail.conf* contiene la definición de todas las prestaciones del buzón de voz.

Está dividido en los siguientes contextos:

- `[general]`: configura las opciones generales, tales como la cantidad de mensajes, el tiempo máximo y el mínimo, el número de reintentos de acceso al buzón o las teclas de control
- `[zonemessages]`: define las zonas horarios y el formato para cada una
- `[buzon-sip]`: definición del buzón de voz para las extensiones SIP, en este caso 1001 y 1003
- `[buzon-iax]`: definición del buzón para las extensiones IAX, 1101 y 1103

En este caso se ha optado por dejar la configuración general establecida de forma básica, puesto que corresponde con lo buscado en este caso de estudio.

En cuánto al contexto `[zonemessages]` se ha añadido una nueva zona horaria con el siguiente formato:

```
madrid = Europe/Madrid|'vm-received' Q 'digits/at' R
```

Se ha optado por diferenciar los buzones según el tipo de protocolo que usan las extensiones asociadas para seguir con el planteamiento inicial de hacer una diferencia entre SIP e IAX. Así pues, se han creado dos buzones de voz `[buzon-sip]` y `[buzon-iax]` los cuales definen la extensiones correspondientes.

La contraseña usada para el acceso es para todas las extensiones la misma, 123

```
[buzon-sip]
1001 => 123,ext1001,correo@upct.es
1003 => 123,ext1003,correo@upct.es
[buzon-iax]
1101 => 123,ext1101,correo@upct.es
1103 => 123,ext1103,correo@upct.es
```

Modificación de los clientes SIP e IAX

Tanto los clientes SIP como los IAX tendrán que definir la opción correspondiente al buzón de voz.

Para ello, en el archivo *sip.conf* se añadirán las siguientes líneas:

sip.conf

```
[general]
…
[1001]
…
mailbox=1001@buzon-sip
…
[1003]
…
mailbox=1003@buzon-sip
…
[red2]
…
```

`Mailbox` contiene la extensión correspondiente del buzón de voz SIP definida para el cliente SIP, así cómo el contexto dónde está ubicada.

La modificación del archivo *iax.conf* será de la misma forma:

iax.conf

```
[general]
…
[1101]
…
mailbox=1101@buzon-iax
…
[1103]
…
mailbox=1103@buzon-iax
…
[red2]
…
```

Se añade la definición de `mailbox`, que contendrá el buzón de voz correspondiente a la extensión y el contexto.

Modificación del plan de marcación incluyendo las opciones propias del buzón

Para hacer uso del buzón de voz en las distintas extensiones internas será necesario modificarlas para que gestionen distintas posibilidades, que la llamada sea atendida, que la extensión destino esté ocupada o que esté no disponible.

En los casos en los cuáles la extensión destino no contesta la llamada se procederá a grabar un mensaje, que irá a su buzón de voz correspondiente.

Para que cada extensión pueda entrar a la aplicación buzón de voz se ha escogido la extensión 2999.

Al realizar una llamada a dicha extensión automáticamente entra en el menú del buzón de voz de la extensión que ha realizado la llamada.

No es necesario definir una gestión de buzón de voz para el caso de las extensiones externas, puesto que están definidas en la red contraria con las distintas posibilidades de gestión del buzón.

En caso de que se definieran las distintas opciones, los mensajes de buzón de voz correspondientes a las extensiones que corresponden a la otra red quedarían en la red desde dónde se ha realizado la llamada, con la consiguiente imposibilidad de acceder después a dichos mensajes desde la otra red.

Pongamos por caso que una extensión de la red 1 quiere llamar a una extensión de la red 2, la llamada se realizaría conforme a lo definido en *extensions.conf* de la red 1.

Si en la definición de cómo llamar a una extensión de la red 2 se incluye la posibilidad de dejar un mensaje en caso de que la llamada no haya sido atendida, dicho mensaje quedaría grabado en el servidor 1 (en el directorio */var/spool/asterisk/voicemail/context/boxnumber/INBOX/*). Si la extensión de la red 2 quiere acceder al mensaje no podrá, puesto que éste se encuentra en otra red distinta.

La solución a este problema es definir la forma de uso del buzón de voz para las llamadas internas de cada red.

Por lo tanto, una vez definido de ésta forma, si una extensión de la red 1 quiere realizar una llamada a una extensión de la red 2, en las llamadas internas del archivo *extensions.conf* de la red 2 estarán definidas las distintas posibilidades.

Si finalmente la llamada no se cursa y se deja un mensaje por parte de la extensión de la red 1 éste quedará almacenado en el servidor 2, pudiendo acceder en cualquier momento la extensión destino.

Con ésta solución sólo es necesario crear extensiones de buzón de voz para las extensiones propias de la red, no para todas las extensiones que conforman el caso de estudio.

extensions.conf

```
[general]
      …

;---------------------------
;SIP
;---------------------------

[prueba4-sip]
      include => internas-sip1
      include => externas-sip
      include => prueba4-iax

[internas-sip1]
      exten => 1001,1,Dial(SIP/1001,20,m)
      exten => 1001,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
      exten => 1001,n(unavail),Voicemail(1001@buzon-sip,u)
      exten => 1001,n,Hangup
      exten => 1001,n(busy),Voicemail(1001@buzon-sip,b)
      exten => 1001,n,Hangup

      exten => 1003,1,Dial(SIP/1003,20,m)
      exten => 1003,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
      exten => 1003,n(unavail),Voicemail(1003@buzon-sip,u)
      exten => 1003,n,Hangup
      exten => 1003,n(busy),Voicemail(1003@buzon-sip,b)
      exten => 1003,n,Hangup

      exten => 2999,1,Answer()
      exten => 2999,n,VoiceMailMain(${CALLERID(num)}@buzon-sip)
      exten => 2999,n,Hangup

[externas-sip]
      exten => _200X,1,Dial(SIP/red2/${EXTEN},20,m)
      exten => _200X,2,Hangup
```

```
;----------------------------------------
;IAX
:----------------------------------------

[prueba4-iax]
      include => internas-iax1
      include => externas-iax

[internas-iax1]
      exten => 1101,1,Dial(IAX2/1101,20,m)
      exten => 1101,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
      exten => 1101,n(unavail),Voicemail(1101@buzon-iax,u)
      exten => 1101,n,Hangup
      exten => 1101,n(busy),Voicemail(1101@buzon-iax,b)
      exten => 1101,n,Hangup

      exten => 1103,1,Dial(IAX2/1103,20,m)
      exten => 1103,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
      exten => 1103,n(unavail),Voicemail(1103@buzon-iax,u)
      exten => 1103,n,Hangup
      exten => 1103,n(busy),Voicemail(1103@buzon-iax,b)
      exten => 1103,n,Hangup

      exten => 2999,1,Answer()
      exten => 2999,n,VoiceMailMain(${CALLERID(num)}@buzon-iax)

[externas-iax]
      exten => _210X,1,Dial(IAX2/red2/${EXTEN},10)
      exten => _210X,2,Hangup
```

5.5 SALA DE CONFERENCIAS

Objetivos:

- Configuración de las distintas salas de conferencias
- Diferencias entre la configuración de la red 1 y la configuración de la red 2.
- Implementación en el archivo extensions.conf

Metodología:

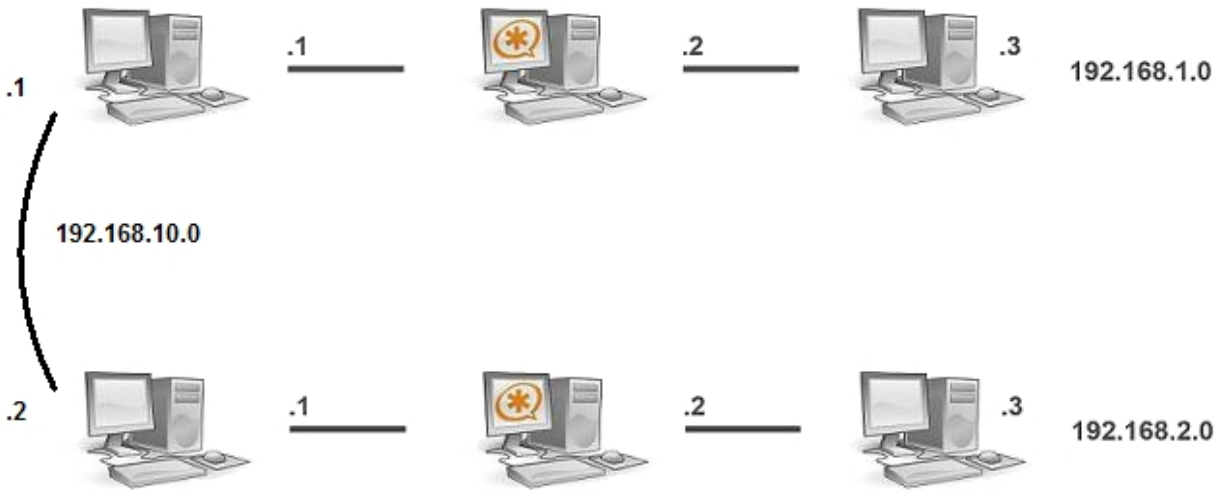


Figura 30: "Sala de conferencias"

El objetivo de este caso de estudio es realizar una configuración que permita que aquellas extensiones que lo deseen puedan unirse a una sala de conferencias común.

Para ello se hará uso de un único servidor que contenga la información de conexión a la sala de conferencias. En éste caso se ha optado por usar el servidor 1.

La configuración de la sala de conferencias está definida en el archivo *meetme.conf*, el cuál contendrá las distintas salas a usar.

En *extensions.conf*, haciendo uso de la aplicación `MeetMe()` se podrá acceder a cualquiera de las salas definidas.

Configuración de las distintas salas de conferencias

El archivo *meetme.conf* definirá las salas de conferencias que podrán albergar a todos los usuarios que marquen el número correspondiente de sala.

Dicho archivo se compone de dos contextos:

- `[general]`: contiene la definición general para varias parámetros, como pueden ser los buffers de audio, el uso de MySQL, las alertas y la conexión o desconexión de los distintos usuarios.
- `[rooms]`: contiene la definición de las distintas salas de conferencias, así como la contraseña de acceso a cada una de ellas.

En éste caso se han usado las opciones definidas por defecto en el archivo *meetme.conf* para el contexto `[general]`

Éstas han sido:

```
[general]
      audiobuffers = 32
      schedule = yes
      logmemebercount = no
      fuzzystart = 300
      earyalert = 3600
      endalert = 120
```

Dichas opciones definen por orden de aparición, el número de paquetes de audio de 20 segundos guardados en el buffer de memoria, permite la programación mediante MySQL, actualiza el tiempo real cuándo un usuario entra o sale de la conferencia, define los segundos de margen permitidos para que un usuario entre a una conferencia programada que ya ha empezado, define los segundos de margen para anunciar a un usuario que la conferencia programada aún no ha empezado, y por último, especifica los segundos anteriores al aviso del fin de la conferencia programada.

En cuánto al contexto `[rooms]` se ha optado por definir dos salas de conferencias sin contraseña de acceso para facilitar la rápida conexión por parte de las distintas extensiones.

```
[rooms]
      conf => 1234
      conf => 4321
```

Las salas de conferencias estarán definidas con las extensiones 1234 y 4321, que deberán marcar los clientes que quieran acceder a una u otra.

Para que la sala de conferencias pueda funcionar es necesario hacer uso de un temporizador, para ello se usará una tarjeta Zaptel de Digium o bien un simulador de Zaptel (ztdummy), que nos aporta temporizador sin canales. Éste último viene como módulo por defecto en el Kernel a partir de la versión 2.6

En caso de que el número de clientes sea muy elevado se puede limitar el tamaño de la sala de conferencias para que restrinja el acceso a más clientes de los deseados. En éste caso no ha sido necesario puesto que el máximo de participantes en una conferencia pueden ser cuatro, aquellos ordenadores que disponen de los clientes SIP e IAX.

También es posible usar la sala de conferencias en tiempo real, definiendo una base de datos en MySQL que contenga todos sus parámetros. En éste caso se ha descartado el uso para simplificar la configuración.

Diferencias entre la configuración de la red 1 y la configuración de la red 2:

Es imprescindible usar un único servidor que contenga la información para que no existan duplicidades de salas y para que todas las extensiones, tanto de la red 1 como de la red 2 estén en una sala común si así lo desean. Pudiendo por tanto comunicarse en grupo.

En el caso de usar salas de conferencias definidas en cada servidor, sólo estaría permitido el acceso para las extensiones propias de esa red, pero no estaría permitida la comunicación entre redes.

En éste caso se ha optado por usar el servidor 1 para gestionar las salas de conferencias.

Por lo tanto, las configuraciones serán distintas en función del servidor.

El servidor 1 contendrá la definición del archivo *meetme.conf* y también tendrá la definición del uso de la aplicación `MeetMe()`, la cuál tendrá asociada la extensión `1234`.

Por el contrario, el servidor 2 únicamente tendrá definida en dicha extensión una llamada a la extensión `1234` de la red 1, que iniciará la conexión a la sala de conferencias seleccionada.

Implementación en el archivo extensions.conf

El archivo *extensions.conf* contiene la información relativa a las distintas extensiones que conforman el plan de marcación.

Para ambos servidores la configuración será similar. Se creará un contexto `[meetme]` con distintas opciones dependiendo del servidor.

En el servidor 1, `[meetme]` contendrá la aplicación `MeetMe()` que permite el acceso a las salas de conferencias. Para ello se hará uso de la extensión `1234`.

En el servidor 2, el mismo contexto `[meetme]` será una llamada mediante IAX a la extensión `1234` de la red 1 para que permita el acceso a la aplicación.

El contexto `[meetme]` estará incluido dentro de los contextos `[prueba5-sip]` y `[prueba5-iax]`, permitiendo así que las extensiones de dicha red puedan acceder a él.

Además, en el caso del servidor 1, será necesario incluir el contexto dentro de `[internas-iax1]`, puesto que éste es el contexto definido para el cliente iax `red2`, que será a través del cuál lleguen las llamadas cursadas desde la red 2 con destino la sala de conferencias.

Extensions.conf (servidor red1)

```
[general]
      …

;---------------------------
;SIP
;---------------------------

[prueba5-sip]
      include => internas-sip1
      include => externas-sip
      include => prueba5-iax
      include => meetme

[internas-sip1]
      …

[externas-sip]
      …

;-----------------------------------------
;IAX
:-----------------------------------------

[prueba5-iax]
      include => internas-iax1
      include => externas-iax
      include => meetme

[internas-iax1]
      …
      include => meetme

[externas-iax]
      …
```

```
;---------------------------------------
;MEETME
:---------------------------------------

[meetme]

      exten => 1234,1,Answer()
      exten => 1234,n,MeetMe(,pcs)
```

Un cliente de la red 1, al realizar una llamada a la extensión `1234` entra a la aplicación `Meetme()`, la cuál solicita el número de la sala de conferencias a la que se desea entrar. Esto se debe a que no se ha especificado en la aplicación una única sala, así pues, cualquiera de las salas definidas en *meetme.conf* serán válidas.

Entre las distintas opciones disponibles en la aplicación, se ha optado por elegir:

- `p`: permite salir de la conferencia al pulsar #
- `c`: indica el número de usuarios en la conferencia
- `s`: reproduce el menú al marcar *

La configuración del servidor de la red 2 es la siguiente:

Extensions.conf (servidor red2)

```
[general]
      …

;---------------------------
;SIP
;---------------------------

[prueba5-sip]
      include => internas-sip2
      include => externas-sip
      include => prueba5-iax
      include => meetme
```

```
[internas-sip2]
      …

[externas-sip]
      …

;----------------------------------------
;IAX
:----------------------------------------

[prueba5-iax]
      include => internas-iax2
      include => externas-iax
      include => meetme

[internas-iax2]
      …

[externas-iax]
      …

;----------------------------------------
;MEETME
:----------------------------------------

[meetme]

      exten => 1234,1,Dial(IAX2/red1/1234,20)
      exten => 1234,n,Hangup
```

En éste caso el uso del contexto `[meetme]` por parte de los clientes internos será mediante una llamada a la red 1, que contiene la definición de los buzones de voz y el uso.

Se ha optado por una llamada usando el cliente `red1` que apunta al servidor de la red 1. Como protocolo de transporte se ha elegido `IAX2` por su mayor rapidez en el envío de información.

5.6 PARQUEAMIENTO DE LLAMADAS

Objetivos:

- Definición y uso del archivo *features.conf* para la transferencia de llamadas
- Modificación del plan de marcación para gestionar la transferencia.
- Pruebas de configuración y puesta a punto

Metodología:

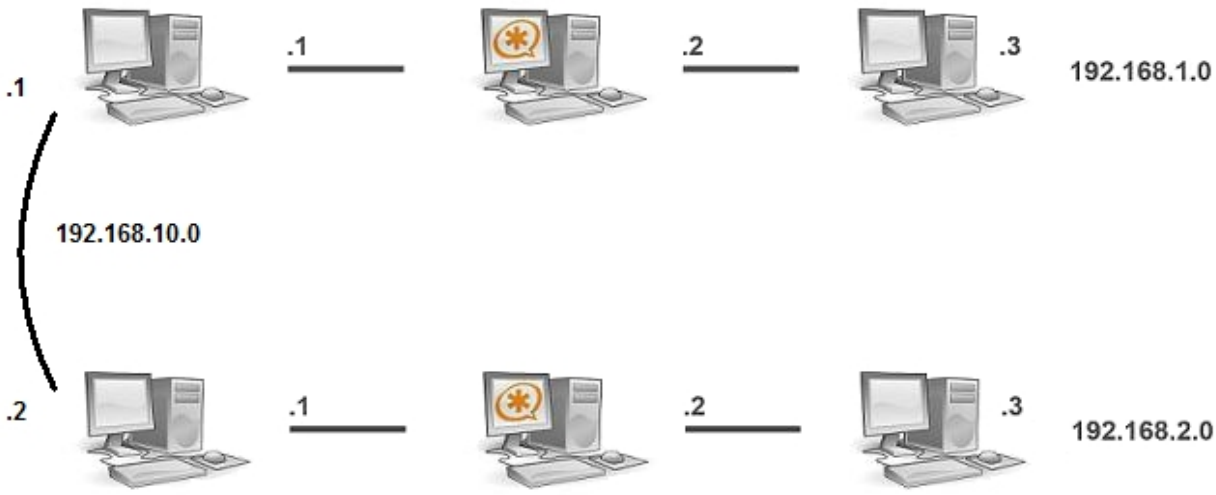


Figura 31: "Configuración del parqueamiento de llamadas"

El objetivo de este caso de uso es definir y verificar el uso de la transferencia y el parqueo de llamadas entre las distintas extensiones.

Para ello será necesario configurar el archivo *features.conf*, que será el encargado de definir el contexto necesario a incluir en el plan de marcación, así como las combinaciones de teclas a marcar para realizar las distintas funciones.

También se definirán las posibilidades de transferencia de llamadas y estacionamiento dentro de las opciones de la aplicación `Dial()` definida en cada extensión de *extensions.conf*

Definición y uso del archivo *features.conf* para la transferencia de llamadas

El archivo *features.conf* define la forma de uso y la combinación de teclas necesarias para realizar las operaciones de transferencia y parqueo.

Para ello se han usado dos contextos:

- `[general]`: define las opciones generales de parqueo, contexto asociado, almacenamiento, música en espera, etc.
- `[featuremap]`: define la combinación de teclas a usar para realizar cualquiera de las opciones permitidas, transferencia de una llamada, finalización de una llamada, grabación, poner una llamada en espera...

Haciendo uso de una configuración estándar se ha definido el archivo *features.conf* de la siguiente manera:

features.conf:

```
parkext => 700
parkpos => 701-710
context => parkedcalls
parkinghints = yes
parkingtime => 45
courtesytone = beep
parkedplay = caller
parkedcalltransfers = caller
parkedcallreparking = caller
parkedcallhangup = caller
parkedcallrecording = caller
perkeddynamic = no
adsipark = no
findslot = next
parkedmusicclass = default
transferdigittimeout => 3
xfersound = beep
xferfailsound = beeperr
pickupexten = *8
pickupsound = beep
pickupfailsound = beeperr
featuredigittimeout = 1000
```

```
        atxfernoanswertimeout = 15
        atxferdropcall = no
        atxferloopdelay = 10
        atxfercallbackretries = 2

    [featuremap]
        blindxfer => #1
        disconnect => *0
        automon => *1
        atxfer => *2
        parkcall => #7
        automixmon => *3
```

La extensión usada para parquear una llamada será `700`, dicha llamada quedará almacenada de forma temporal en una extensión entre la `701` y la `710`, que están reservadas para dicho almacenamiento.

El comando `findslot` indica que de haber una llamada que sea parqueada ocupará la siguiente posición libre del parking hasta que estén todas ocupadas.

El contexto usado para el parqueo de llamadas es `[parkedcalls]`, que deberá estar incluido en *extensions.conf* para que sea válido.

`Parkingtime` define el número de segundos en los que la llamada puede estar parqueada, en éste caso se ha optado por 45 segundos de espera. En caso de ser superado el tiempo la llamada volverá a la extensión que la puso en espera (comando `comebacktoorigin=yes`)

Los segundos de espera entre introducción de dígitos para realizar una transferencia vienen especificados por el comando `transferdigittimeout.`

En cuánto a los distintos sonidos que pueden emitirse, tales como la respuesta a una llamada parqueada, que la transferencia se ha cursado con éxito o que la grabación ha terminado, se ha optado por usar el sonido `beep`, así como para los sonidos que implican fallo, `beeperr.`

Los comandos `parked` en general especifican quién maneja las distintas opciones como son la recepción de beep al llamar a una extensión parqueada y los tonos para llamadas transferidas o parqueadas. Es posible elegir entre `callee` (llamado), `caller` (llamante) o `both` (ambos), en éste caso se ha optado por `caller`.

Para el caso de las transferencias de llamadas con el método asistido se ha optado por configurar el comando `atxferdropcall = no`, puesto que si quién transfiere la llamada mediante este método cuelga antes de que sea transferida por completo Asterisk devuelve la llamada a quién realizaba la transferencia. En caso de estar activo, la llamada no se devolvería y se consideraría finalizada.

El tiempo de espera antes de devolver la llamada al origen son 10 segundos (en caso de que `atxferdropcall` sea configurado de forma negativa), el comando que especifica el tiempo es `atxferloopdelay`.

El número de veces que se intentará devolver la llamada será 2, especificado en `atxfercallbackretries`.

En cuánto a `[featuremap]`, dicho contexto contiene las secuencias de teclas a marcar para realizar cualquier acción especificada en el contexto.

Para realizar una transferencia a ciegas (`blindxfer`) se usará la secuencia `#1`

Para la transferencia asistida (`atxfer`), `*2`

Para parquear una llamada (`parkcall`), se marcará `#7`, aunque también es posible transferir directamente la llamada a la extensión 700 de la forma: `#1 700` (dónde la llamada quedará guardada en la primera extensión libre de las que están reservadas para tal fin, `701-710`)

Para terminar una llamada o para recuperar una llamada en una transferencia a ciegas (`disconnect`), se usará la combinación `*0`

Para grabar una llamada dónde los interlocutores quedarán grabados por separado (`automon`), se pulsará `*1`

Para que la grabación contenga las voces de ambos interlocutores (`automixmon`), se usará `*3`.

Cada uno de estos comandos tiene asociada una opción en la aplicación `Dial()`, que definirá para cada extensión los permisos correspondientes ya sea para la extensión destino, la extensión llamante o para ambas.

Modificación del plan de marcación para gestionar la transferencia.

Para que ésto sea posible es necesario que esté especificado el contexto `[parkedcalls]` dentro del plan de marcación para que las combinaciones de teclas definidas en `[featuremap]` se puedan llevar a cabo.

Y además, que la aplicación `Dial()` tenga definidas las correspondientes posibilidades.

Para proveer la funcionalidad de parqueamiento de llamadas en el receptor, la aplicación `Dial()` tendrá que especificar las siguientes opciones:

- `h`: permite que el receptor termine la llamada o recupere una llamada en una transferencia a ciegas
- `k`: permite que el receptor de la llamada al parque, para ello marcará #7
- `t`: permite una transferencia de llamada asistida o a ciegas por parte del receptor
- `w`: permite realizar una grabación de la llamada (generando un archivo por interlocutor)
- `x`: permite grabar una llamada en un único archivo, mezclando ambos interlocutores

Se ha elegido al receptor para que gestione las llamadas teniendo en cuenta que la gran mayoría de este tipo de llamadas las realiza una persona externa a una centralita y es la centralita la que se encarga de transferirlas o ponerlas en espera, e incluso de grabar una conversación, no permitiendo de este modo que el llamante pueda usar dichas opciones.

Por supuesto es una forma de configuración restringida que se puede ampliar permitiendo el control por parte del llamante también, en cuyo caso además de las opciones `h, k, t, w y x`, habrá que añadir `H, K, T, W y X` para dar los permisos necesarios.

El archivo *extensions.conf* ha sido configurado de la siguiente forma:

extensions.conf

```
[general]
      …

;---------------------------
;SIP
;---------------------------

[prueba6-sip]
      include => internas-sip1
      include => externas-sip
      include => prueba6-iax
      include => meetme
      include => parkedcalls

[internas-sip1]
      exten => 1001,1,Dial(SIP/1001,20,mhktwx)
      exten => 1001,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
      exten => 1001,n(unavail),Voicemail(1001@buzon-sip,u)
      exten => 1001,n,Hangup
      exten => 1001,n(busy),Voicemail(1001@buzon-sip,b)
      exten => 1001,n,Hangup

      exten => 1003,1,Dial(SIP/1003,20,mhktwx)
      …
      exten => 1003,n,Hangup

      …

[externas-sip]
      exten => _200X,1,Dial(SIP/red2/${EXTEN},20,mhktwx)
      exten => _200X,n,Hangup

;---------------------------------------
;IAX
:---------------------------------------

[prueba6-iax]
      include => internas-iax1
```

```
      include => externas-iax
      include => meetme
      include => parkedcalls

[internas-iax1]
      exten => 1101,1,Dial(IAX2/1101,10,hktwx)
      exten => 1101,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
      exten => 1101,n(unavail),Voicemail(1101@buzon-iax,u)
      exten => 1101,n,Hangup
      exten => 1101,n(busy),Voicemail(1101@buzon-iax,b)
      exten => 1101,n,Hangup

      exten => 1103,1,Dial(IAX2/1103,10,hktwx)
      …
      exten => 1103,n,Hangup

      …

[externas-iax]
      exten => _210X,1,Dial(IAX2/red2/${EXTEN},10,hktwx)
      exten => _210X,n,Hangup

;----------------------------------------
;MEETME
:----------------------------------------

[meetme]

      …
```

Como se puede observar la implementación del uso de las transferencias de llamadas y la puesta en espera es muy sencilla.

En éste caso, se ha hecho una distinción de opciones en función de los protocolos, que no ha sido otra que usar la música de fondo para las llamadas que se realizan a través del protocolo `SIP` para poder diferenciales rápidamente de aquellas que usan el protocolo `IAX`. Esto es una forma cómoda de identificar las llamadas a la hora de hacer distintas pruebas en el laboratorio.

Se ha considerado la posibilidad de crear un contexto llamado `[parking]` que, haciendo uso de la aplicación `ParkedCall()` recogiese las llamadas parqueadas en las extensiones `701-710`, sin embargo, esto no es necesario, ya que mediante la llamada directa a una extensión dónde está parqueada la llamada es suficiente para recogerla, como veremos en los siguientes ejemplos.

Pruebas de configuración y puesta a punto:

Las posibilidades de uso son muchas y muy variadas, sin embargo todas han sido configuradas para que sea la extensión receptora de la llamada quién las gestione.

En cuánto a las transferencias de llamadas podemos dividir las pruebas en dos estilos, transferencias a ciegas y atendidas.

Para realizar una transferencia a ciegas bastará con marcar en la extensión receptora el patrón de números definido para tal efecto, que en éste caso será `#1` y el número hacia dónde queramos transferir la llamada.

Se han realizado una serie de pruebas para comprobar el funcionamiento:

- `#1 + nº ext.`: Transfiere la llamada a la extensión marcada. El teléfono sonará en dicha extensión
- `#1 + 700`: Parquea la llamada en una de las extensiones reservadas para tal efecto, en éste caso la `701`. Para recoger la llamada basta con marcar `701` desde cualquier extensión, haciendo que el llamante inicial y quién recoge la llamada se comuniquen.

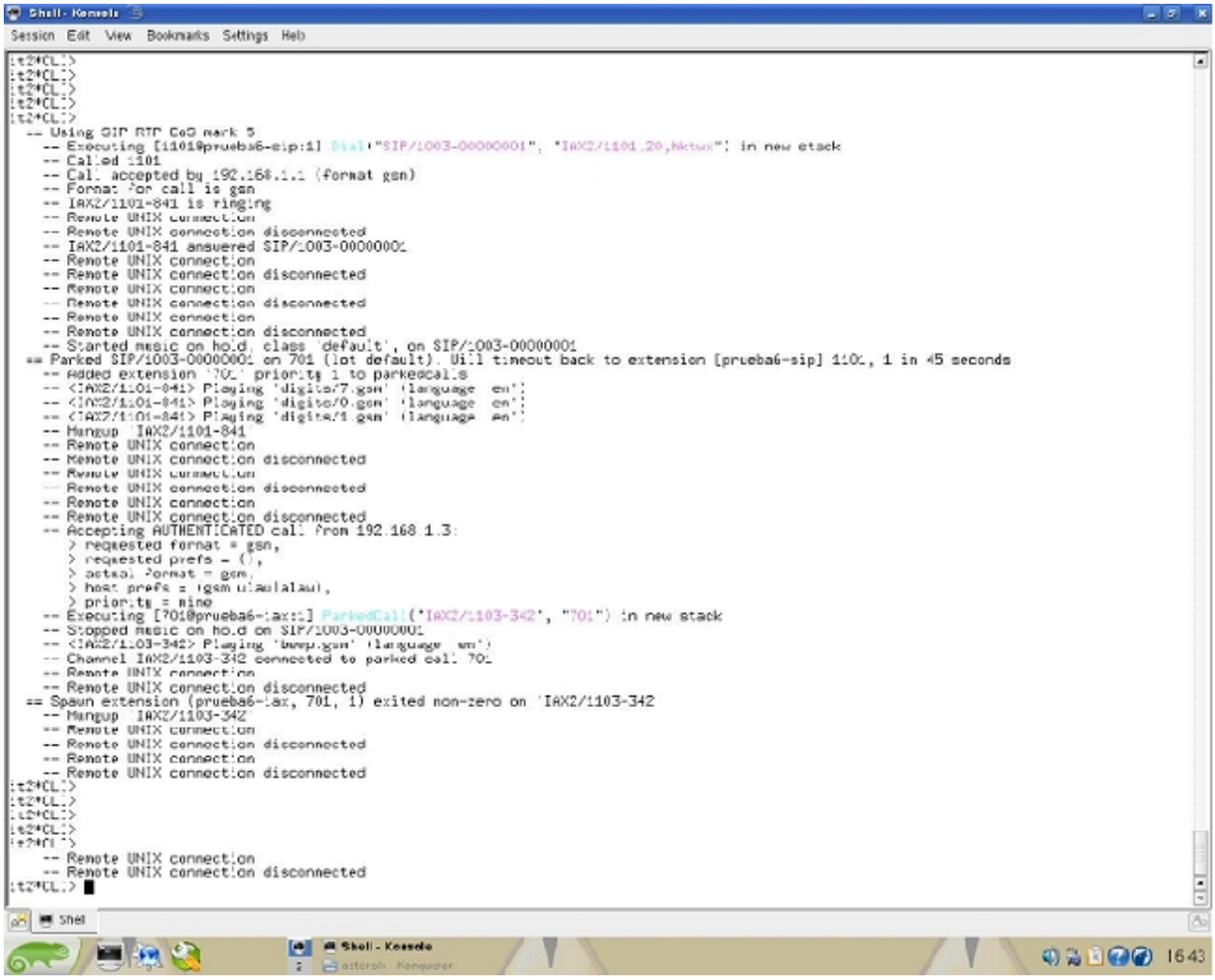


Figura 32: “Llamada parqueada y recogida por otra extensión”

En éste ejemplo, la extensión `1003` realiza una llamada a la extensión `1101`, que tiene la potestad de decidir el curso de la llamada según lo especificado en la aplicación `Dial()` y según los parámetros definidos en *features.conf*. Dicha extensión, la `1101`, realiza un parqueo marcando `#7`, por lo cuál la llamada que inició la extensión `1003` quedará a la espera ubicada en una extensión para tal caso, en el ejemplo se ve claramente que la extensión será la `701` puesto que es la primera que está libre. La extensión `1101` cuelga y deja en espera a la extensión `1003` hasta que la extensión `1103` marca la extensión dónde se ubica y recupera la llamada, estableciendo una comunicación entre `1003` y `1103`.

Para el caso de las transferencias atendidas se pulsarán las teclas `*2`:

- `*2 + nº ext.` : Transferencia atendida a la extensión marcada. El teléfono sonará en dicha extensión y se comunicará con quién transfiere la llamada que le indica que le va a pasar con otra extensión, cuándo la extensión que hace de intermediaria cuelga, se establece la comunicación entre el llamante inicial y la extensión hacia dónde se ha desviado la llamada.

Pongamos por caso que la extensión `1001` llama a la extensión `1003` y ésta estima por conveniente transferir la llamada a otra extensión, por ejemplo la `1103`. En dicho caso, la extensión `1003` marcará `*21103` en el teclado del teléfono, y en `1103` sonará una llamada; después de comunicar que se va a transferir la llamada, el trabajo de la extensión `1003` ha finalizado y por tanto cuelga, estableciéndose así una comunicación entre `1001` y `1103` de forma asistida.

Para parquear una llamada directamente y dejarla en espera bastará con marcar `#7`:

- `#7`: Parquea la llamada directamente en la primera extensión reservada que se encuentre libre sin necesidad de indicar que va a ser parqueada. Para contestar a dicha llamada será necesario llamar a la extensión `701-710` que corresponda. Es posible hacer uso del comando `parkedcalls show` en el CLI para que muestre qué llamadas están parqueadas y dónde se ubican.

Para desconectar una extensión de una llamada:

- `*0`: Desconecta la extensión

En caso de querer realizar una grabación de audio podemos optar por separar los canales de comunicación de los interlocutores o mezclarlos en uno sólo:

- `*3`: Se inicia una grabación de audio que une ambos canales de comunicación. Finaliza marcando `*3` de nuevo. El archivo contendrá un nombre similar a: `auto-1306842957-2102-2003`.
- `*1`: Separa ambos canales de audio en la grabación.

Es posible transferir llamadas entre redes sin problemas, sin embargo, si una llamada está parqueada sólo pueden recogerla las extensiones que pertenezcan a la misma red que el receptor de la llamada, que será quién la parquee.

Una extensión de la red 1 en caso de que se haya llamado a una extensión de la red 1, o una extensión de la red 2 en caso de que se haya llamado a una extensión de la red 2. Da igual desde dónde se inicie la llamada original que será puesta en espera, es quién parquea quién delimita la red de la persona que puede contestar.

Esto es debido a que el parqueamiento de una llamada implica guardar la llamada temporalmente en una extensión de un servidor concreto para que luego otra extensión pueda recoger esa llamada.

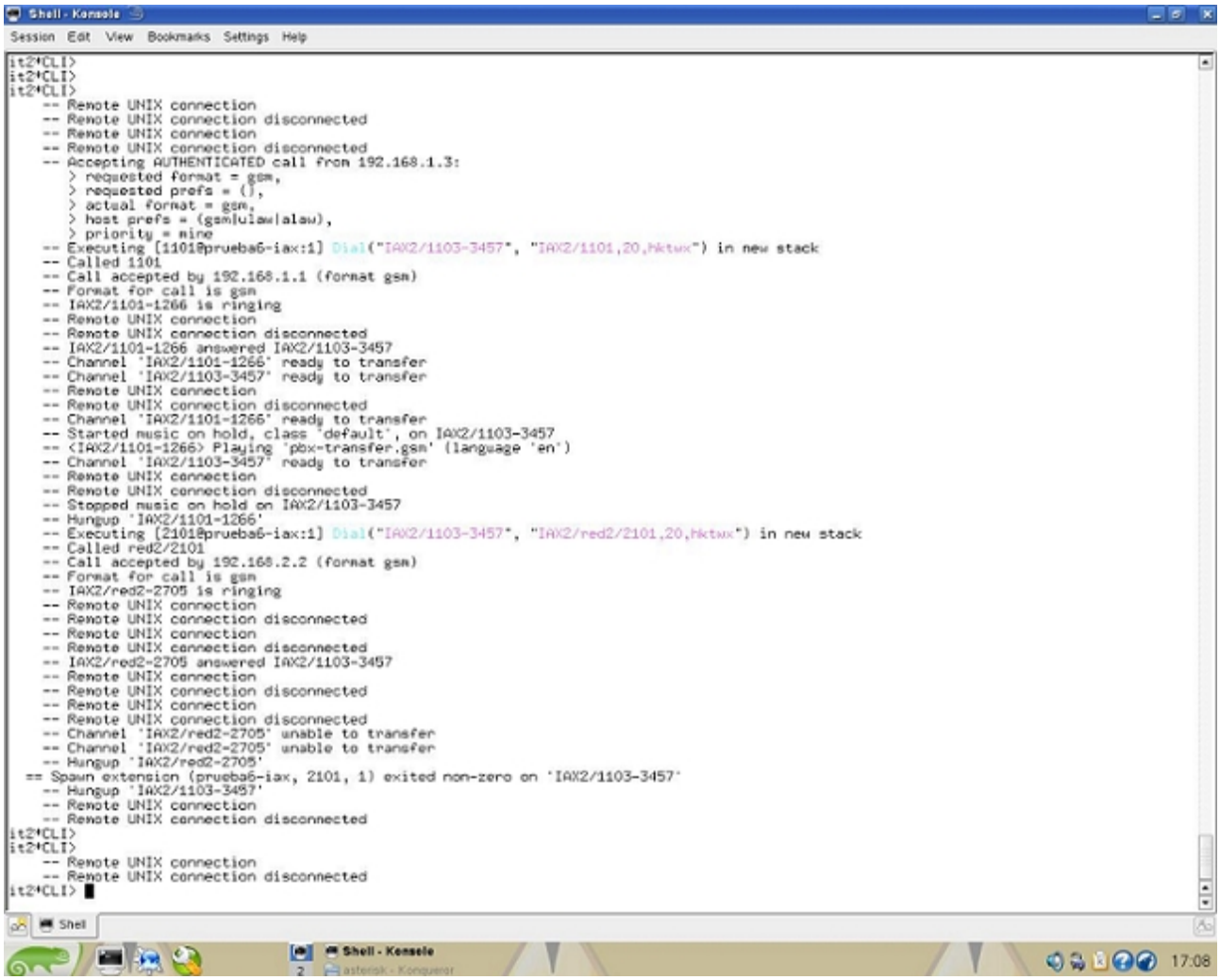


Figura 33: “Llamada transferida a otra red”

En éste ejemplo se realiza una llamada desde la extensión `1103` a la extensión `1101`, que tiene permisos para ejecutar cualquier transferencia o parqueo. Realiza una transferencia atendida a la extensión `2101` perteneciente a la red2, en cuánto la extensión que ha servido de enlace cuelga, se establece una comunicación entre la extensión `1103` y la `2101`.

5.7 RESPUESTA DE VOZ INTERACTIVA (IVR)

Objetivos:

- Establecer un diagrama de uso de la respuesta de voz interactiva
- Definir el funcionamiento dentro del plan de marcación

Metodología:

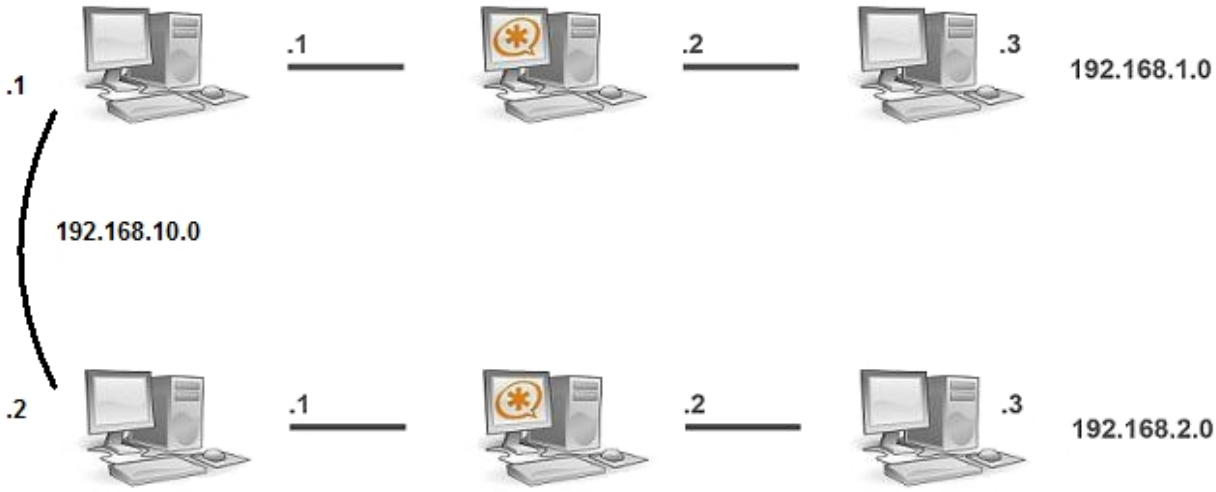


Figura 34: "Configuración de la respuesta de voz interactiva - IVR"

Es un sistema automatizado de respuesta interactiva, orientado a entregar y/o capturar información a través del teléfono, permitiendo el acceso a servicios de información u otras operaciones.

El objetivo de este caso de estudio es implementar una solución de respuesta de voz interactiva de forma que el llamante, por medio de un menú pre-grabado, pueda elegir una de entre todas las opciones que le son dadas.

Será necesario crear un archivo de audio con las instrucciones que formará parte de la respuesta de voz interactiva. Para ello se definirá un contexto para tal fin.

Además, se hará uso de un contexto que define el funcionamiento del IVR dentro del plan de marcación.

Establecer un diagrama de uso de la respuesta de voz interactiva:

La respuesta de voz interactiva (IVR) consistirá en varios menús de forma que el cliente que realice la llamada pueda elegir con qué extensión puede hablar.

Para simplificar el código se ha optado porque sean las extensiones SIP las receptoras de las llamadas a través de los diferentes menús. Por lo tanto, cualquier extensión que llame a la definida para el acceso al IVR podrá contactar con las extensiones `1001`, `1003`, `2001` y `2003`.

El IVR consta de un menú principal y dos submenús que serán distintos en función de la opción elegida en el menú principal.

Menú principal: contendrá una grabación de audio llamada `elegirRed` con los siguientes datos:

"marque 1 para comunicarse con la red1, marque 2 para comunicarse con la red2"

Como posibles opciones a marcar por el cliente existen:

- Marca 1 ó 2: el sistema salta al menú correspondiente según haya marcado 1 ó 2
- Marca cualquier otro número: el sistema lo identifica como una entrada no válida y finalizará.
- Sobrepasa el tiempo estimado: vuelve a reproducir el mensaje inicial del menú principal y vuelve a la espera de que el cliente marque una ruta.
- Cuelga: si el llamante colgara se daría por finalizada la conexión y se cerraría el canal de audio correspondiente.

Submenú 1: contendrá una grabación de audio llamada `elegirExtenRed1` con los siguientes datos:

"marque 1 para hablar con la extensión 1001, marque 2 para hablar con la extensión 1003 o marque 3 para volver al menú principal"

Entre las distintas posibilidades existen:

- Marca 1: Se inicia una llamada a la extensión `SIP 1001`
- Marca 2: Se inicia una llamada a la extensión `SIP 1003`
- Marca 3: Vuelve al menú principal y ejecuta de nuevo el mensaje de bienvenida
- Marca cualquier otro número: se reproduce un mensaje de entrada no válida y se cuelga la llamada
- Sobrepasa el tiempo estimado: vuelve a reproducir el submenú 1 a la espera de que el llamante marque un número.
- Cuelga: finaliza la conexión y se cierra el canal de audio.

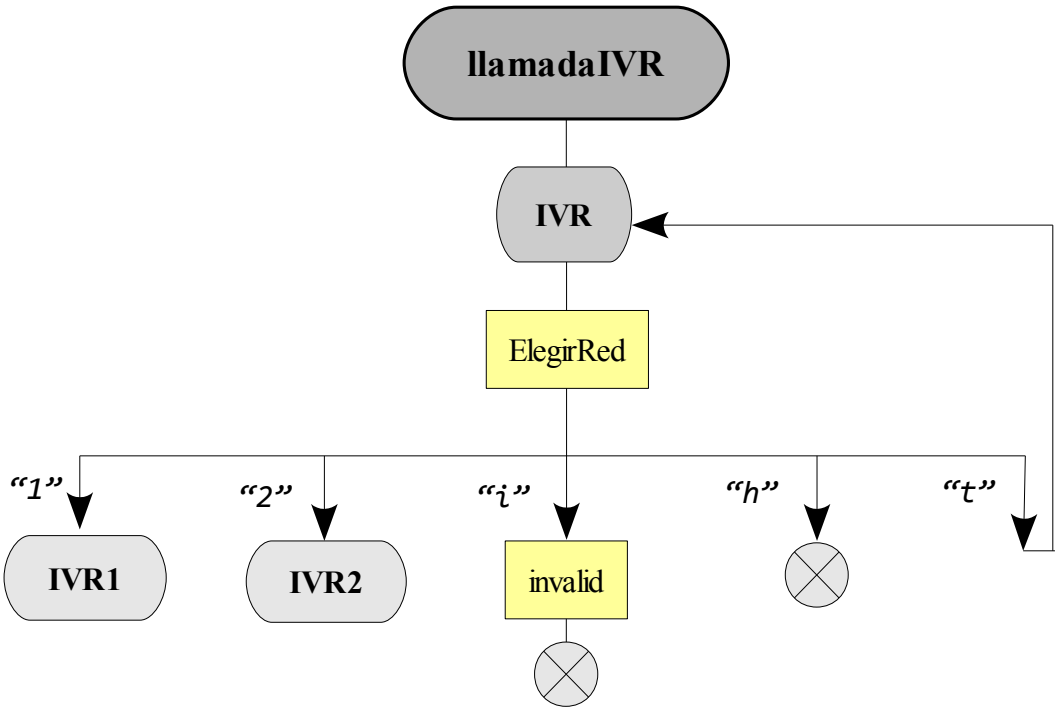
Submenú 2: la grabación de audio llamada `elegirExtenRed2` será:

"marque 1 para comunicarse con la extensión 2001, marque 2 para comunicarse con la extensión 2003, o marque 3 para volver al menú principal"

Entre las muchas opciones que puede marcar el cliente se encuentran:

- Marca 1: Se inicia una llamada a la extensión `SIP 2001` de la red contraria
- Marca 2: Se inicia una llamada a la extensión `SIP 2003` de la red contraria
- Marca 3: Vuelve al menú principal y ejecuta de nuevo el mensaje de bienvenida
- Marca cualquier otro número: se reproduce un mensaje de entrada no válida y se cuelga la llamada
- Sobrepasa el tiempo estimado: vuelve a reproducir el submenú 2 a la espera de que el llamante marque un número.
- Cuelga: finaliza la conexión y se cierra el canal de audio.

Diagrama de flujo de funcionamiento del contexto `[llamadaIVR]`:



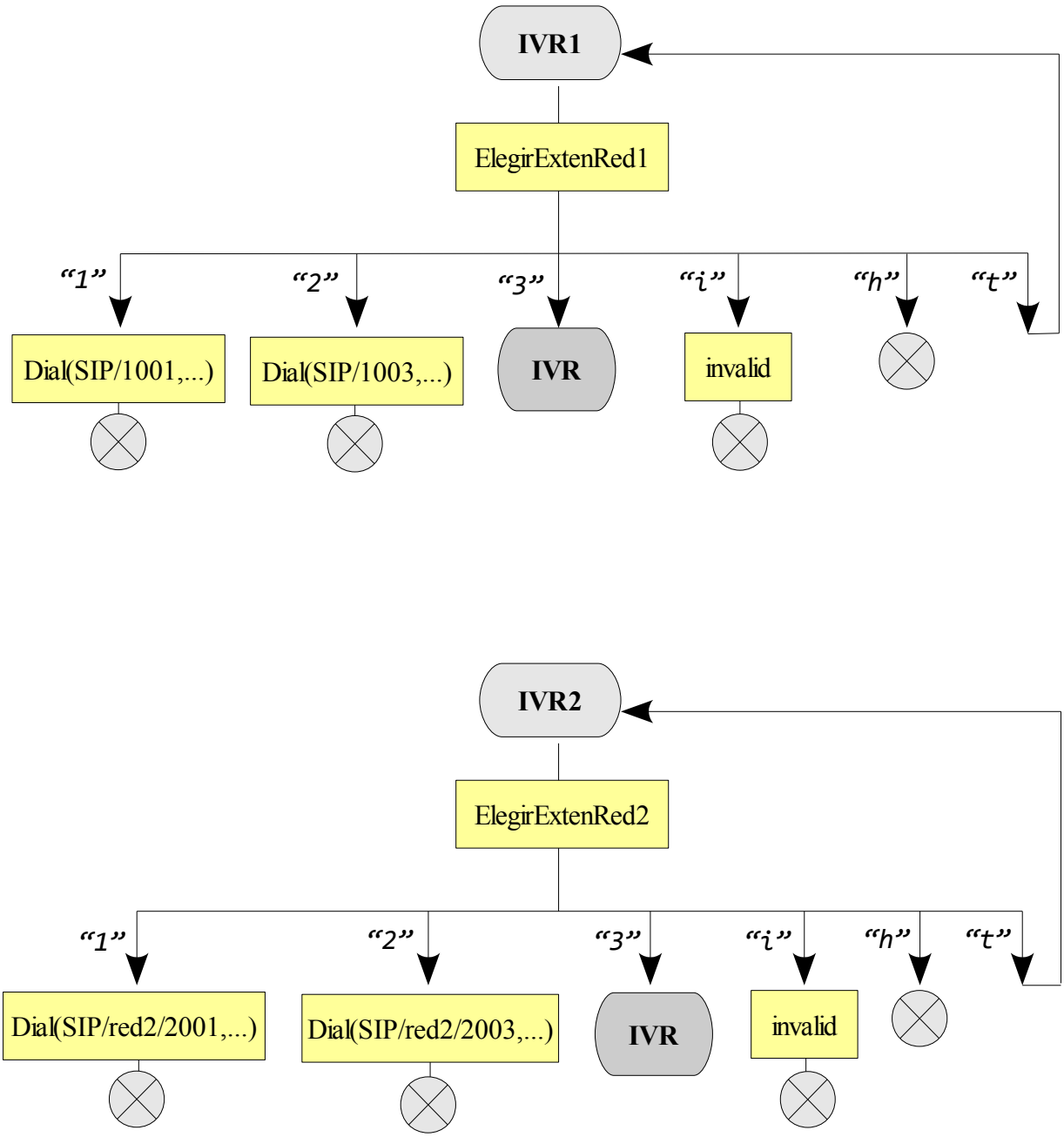


Figura 35: "Diagrama de flujo del contexto [llamadaIVR]"

También será necesario hacer uso de un contexto que gestione las grabaciones que después formarán parte de los distintos menús del IVR.

Para ello se necesitarán una serie de archivos que guarden las grabaciones iniciales que luego serán renombradas y reubicadas al lugar definitivo desde dónde se enlazarán para que se reproduzcan en el lugar conveniente del menú del IVR.

Para la grabación de los mensajes que componen los diferentes menús se ha optado por crear un contexto para tal fin llamado `[grabacionAudio]`, al cuál se accederá al marcar cualquier extensión entre el rango `660` y `669`.

El objetivo es grabar distintos archivos de audio sin que se superpongan, pudiendo tener hasta 10 archivos distintos para usarlos según interese.

Los archivos se guardarán en el directorio */var/lib/asterisk/sounds/en/temporal/* con formato wav.

El nombre con el que serán guardados hará referencia a la extensión marcada para la grabación, quedando con el nombre: `audioX.wav`, siendo X un valor entre 0 y 9.

Por ejemplo, para el caso de llamar a la extensión `665`, el archivo de audio guardado tendrá el nombre `audio5.wav`.

Una vez grabados los archivos de audio deseados, se renombrarán y se ubicarán en la carpeta correspondiente para su ejecución.

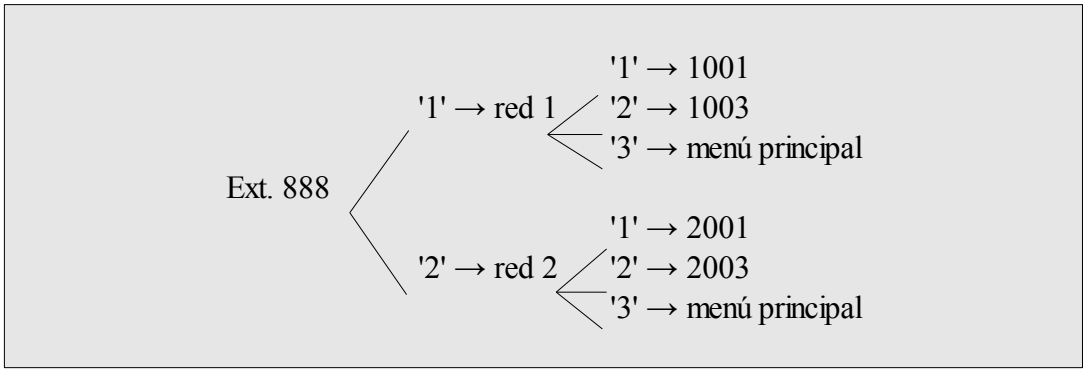
En éste caso, los archivos serán renombrados con `elegirRed`, `elegirExtenRed1` y `elegirExtenRed2` y se guardarán en el directorio */var/lib/asterisk/sounds/en/IVR_prueba/* creado para tal efecto.

Definir el funcionamiento dentro del plan de marcación:

La respuesta de voz interactiva se definirá en un contexto del plan de marcación llamado `[llamadaIVR]`, el cuál será incluido en los contextos correspondientes para que pueda ser ejecutado tanto por las extensiones de la red 1, como por las extensiones de la red 2 que harán uso del plan de marcación definido en la red 1 y accederán a él mediante una llamada a la extensión correspondiente.

La extensión elegida para el acceso al plan de marcación es la `888`.

El esquema de funcionamiento del contexto `[llamadaIVR]` es el siguiente:



El contexto `[llamadaIVR]` se encuentra definido en la red1, por lo tanto cualquier extensión de la red2 que quiera tener acceso a él, tendrá que realizar una llamada a la extensión `888` de dicha red.

El protocolo usado para la comunicación entre la red 2 y la red 1 para hacer uso del IVR será `IAX2` debido a su robustez frente a `SIP`.

Para la red 1, la configuración de *extensions.conf* será la siguiente:

extensions.conf de red 1:

```
[general]
          …

;---------------------------
;SIP
;---------------------------

[prueba7-sip]
     include => internas-sip1
     include => externas-sip
     include => prueba7-iax
     include => meetme
     include => parkedcalls
     include => llamadaIVR
     include => grabacionAudio

[internas-sip1]
     …

[externas-sip]
     …

;---------------------------------------
;IAX
:---------------------------------------

[prueba7-iax]
     include => internas-iax1
     include => externas-iax
     include => meetme
     include => parkedcalls
     include => llamadaIVR
     include => grabacionAudio
```

```
[internas-iax1]
    …
    include => meetme
    include => llamadaIVR

[externas-iax]
    …

;---------------------------------------
;MEETME
;---------------------------------------

[meetme]

    …

;---------------------------------------
;respuesta de voz interactiva IVR
;---------------------------------------

[llamadaIVR]
    exten => 888,1,Answer()
    exten => 888,n,Wait(1)
    exten => 888,n,Goto(IVR,s,1)

[IVR]
    exten => s,1,Wait(1)
    exten =>
s,n,BackGround(/var/lib/asterisk/sounds/en/IVR_prueba/elegirRed)
    exten => s,n,WaitExten()

    exten => 1,1,Goto(IVR1,s,1)
    exten => 2,1,Goto(IVR2,s,1)

    exten => i,1,Playback(invalid)
    exten => i,2,Hangup
    exten => t,1,Goto(IVR,s,1)
    exten => h,1,Hangup
```

```
       [IVR1]
           exten => s,1,Wait(1)
           exten =>
s,n,BackGround(/var/lib/asterisk/sounds/en/IVR_prueba/elegirExtenRed1)
           exten => s,n,WaitExten()

           exten => 1,1,Dial(SIP/1001,20,mhktwx)
           exten => 1,2,Hangup
           exten => 2,1,Dial(SIP/1003,20,mhktwx)
           exten => 2,2,Hangup
           exten => 3,1,Goto(IVR,s,1)

           exten => i,1,Playback(invalid)
           exten => i,2,Hangup
           exten => t,1,Goto(IVR1,s,1)
           exten => h,1,Hangup

       [IVR2]
           exten => s,1,Wait(i)
           exten =>
s,n,BackGround(/var/lib/asterisk/sounds/en/IVR_prueba/elegirExtenRed2)
           exten => s,n,WaitExten()

           exten => 1,1,Dial(SIP/red2/2001,20,hktwx)
           exten => 1,2,Hangup
           exten => 2,1,Dial(SIP/red2/2003,20,hktwx)
           exten => 2,2,Hangup
           exten => 3,1,Goto(IVR,s,1)

           exten => i,1,Playback(invalid)
           exten => i,2,Hangup
           exten => t,1,Goto(IVR2,s,1)
           exten => h,1,Hangup
```

```
;--------------------------------------
;grabación de archivos de audio
;--------------------------------------

[grabacionAudio]
    exten => _66X,1,Answer()
    exten => _66x,n,Wait(2)
    exten =>
_66X,n,Record(/var/lib/asterisk/sounds/en/temporal/audio${EXTEN:2:3})
    exten => _66X,n,Wait(2)
    exten =>
_66X,n,Playback(/var/lib/asterisk/sounds/en/temporal/audio${EXTEN:2:3})
    exten => _66X,n,Wait(2)
    exten => _66X,n,Hangup
```

Las novedades respecto a esta nueva configuración son la creación de dos nuevos contextos, `[llamadaIVR]` y `[grabacionAudio]`.

Ambos podrán ser usados por cada extensión que compone el plan de marcación, por lo tanto se incluirán en el contexto principal que abarca tanto las llamadas entrantes como salientes.

Además, para permitir el acceso desde la red contraria, que será a través de una conexión `IAX2`, se añade el contexto `[llamadaIVR]` dentro del contexto `[internas-iax1]`, puesto que la comunicación desde la otra red se considerará como una llamada interna una vez cursada al servidor correspondiente, por lo tanto deberá tener los permisos necesarios para acceder al contexto requerido.

Para hacer uso de la respuesta de voz interactiva, cualquier extensión puede llamar a la extensión correspondiente, en éste caso se ha hecho uso de la `888`.

Una vez marcada e iniciada la comunicación, el programa saltará al menú `[IVR]` que reproducirá le mensaje `elegirRed` y en función de lo que haya marcado el cliente saltará a una zona del programa u otra.

Quedan reflejadas todas las opciones posibles al abarcar tanto las opciones que saltan a los submenús como cualquier otra entrada distinta.

Para interpretar las acciones marcadas por el cliente se ha hecho uso de la aplicación `WaitExten()` que espera durante un tiempo indeterminado hasta que el cliente marque una tecla y una vez lo ha hecho la compara con las múltiples opciones definidas.

En el primer menú tenemos como opciones a elegir hablar con la red 1 o con la red 2, para ello bastará con marcar `1` ó `2` para saltar al contexto correspondiente.

En caso de querer acceder a la red 1, de la misma forma que antes, saltará una grabación de audio con las indicaciones a marcar para hablar con una extensión u otra o bien para volver al menú principal.

En éste caso quedan reflejadas las posibilidades de llamar tanto a la extensión `SIP 1001` como a la `1003`, o por el contrario volver al menú principal.

Para llamar a dichas extensiones bastará con realizar una simple llamada mediante la aplicación `Dial()` con el protocolo `SIP` y la extensión correspondiente.

Para acceder a las extensiones `SIP` de la red 2 habrá que hacerlo previo marcado de la opción `2` en el menú principal.

A continuación se accederá al contexto `[IVR2]` que contendrá en primer lugar una reproducción de un mensaje pre-grabado con los pasos a seguir para comunicarse con las extensiones o bien para volver al menú principal.

En caso de marcar `1` ó `2` se hará una llamada a dichas extensiones gracias a la aplicación `Dial()` y al cliente `red2` que permite la comunicación con la red contraria. El protocolo usado para tal fin será `SIP`, ya que las extensiones a llamar están definidas en él.

Al marcar `3` se accederá de nuevo al menú principal.

Todas las excepciones quedan reflejadas, ya sea una entrada no válida, un tiempo de espera excesivo o que el cliente cuelgue y termine con la comunicación.

En caso de que la entrada no sea válida se procederá a reproducir el mensaje `invalid` y a finalizar la llamada.

Si por el contrario la excepción ha sido un tiempo excesivo de respuesta se volverá a reproducir el menú correspondiente al contexto.

Finalmente, si es el cliente quién cuelga, se finalizará la llamada y se liberará el canal de audio.

El contexto `[grabacionAudio]` permite grabar distintos archivos de audio que se podrán reutilizar en el IVR, sólo tendremos que marcar la extensión asociada y comenzar a grabar.

Para grabar los archivos se hace uso de la aplicación `Record()` que los nombrará de la forma `audioX.wav`, siendo X un número entre 0 y 9 según la extensión marcada.

Para finalizar una grabación de audio se pulsará `#`.

En caso de querer sobrescribirla bastará con volver a marcar la extensión a borrar.

Una vez grabados los archivos será necesario cambiarles el nombre y ubicarlos en el directorio correspondiente.

Para la red 2, la configuración del plan de marcación es algo distinta, en éste caso se ha elegido llamar a la red 1 para tener acceso al IVR.

El objetivo de ésta solución ha sido minimizar al máximo el código y hacer uso del que ya está configurado. En configuraciones más amplias y con mayor distinción entre una red y otra merece la pena una configuración por separado.

En ésta no ha sido necesario hacerlo así por la sencillez, sin embargo, cabe comentarlo como una posible mejora a la hora de ampliar las distintas redes.

Sin embargo sí que se ha considerado necesario definir el contexto `[grabacionAudio]` por una razón principalmente, puede ser útil tener un contexto mediante el cuál se puedan grabar archivos de audio para su posterior utilización.

Además, quedaría descartado su uso a través de otra red, ya que sería la red destino la que tuviera almacenado dicho archivo, no quedaría reflejado en la red que ha iniciado la llamada para efectuar dicha grabación.

La configuración del plan de marcación de la red 2 es la siguiente:

extensions.conf de red 2:

```
[general]
          …

;---------------------------
;SIP
;---------------------------

[prueba7-sip]
     include => internas-sip2
     include => externas-sip
     include => prueba7-iax
     include => meetme
     include => parkedcalls
     include => llamadaIVR
     include => grabacionAudio

[internas-sip2]
     …
[externas-sip]
     …
;---------------------------------------
;IAX
:---------------------------------------

[prueba7-iax]
     include => internas-iax2
     include => externas-iax
     include => meetme
     include => parkedcalls
     include => llamadaIVR
     include => grabacionAudio
```

```
[internas-iax1]
    …

[externas-iax]
    …

;---------------------------------------
;MEETME
;---------------------------------------

[meetme]

    …

;---------------------------------------
;respuesta de voz interactiva IVR
;---------------------------------------

[llamadaIVR]
    exten => 888,1,Dial(IAX2/red1/888,20)
    exten => 888,n,Hangup

;---------------------------------------
;grabación de archivos de audio
;---------------------------------------

[grabacionAudio]
    exten => _66X,1,Answer()
    exten => _66x,n,Wait(2)
    exten =>
_66X,n,Record(/var/lib/asterisk/sounds/en/temporal/audio${EXTEN:2:3})
    exten => _66X,n,Wait(2)
    exten =>
_66X,n,Playback(/var/lib/asterisk/sounds/en/temporal/audio${EXTEN:2:3})
    exten => _66X,n,Wait(2)
    exten => _66X,n,Hangup
```

Esta configuración del funcionamiento básico de un sistema de respuesta interactiva tiene muchas y muy distintas modificaciones.

Por ejemplo, podría programarse un sistema de respuesta que dependiendo de la hora o del día de la semana reprodujera un tipo de grabación u otra.

Esto es especialmente válido para anunciar que una empresa está cerrada en ese momento, ya sea porque la llamada se ha realizado fuera del horario de oficina o por vacaciones.

Para usar condiciones temporales es necesario valerse de la aplicación GotoIfTime(), que en función de que se cumplan o no sus parámetros temporales saltará a una zona de código u otra.

Otras posibilidades de uso serían definir un IVR que gestionara colas de espera diferenciadas por departamento (ventas, compras, asistencia técnica...). O bien un IVR que permitiera dirigirse directamente a una extensión o en caso contrario que se envíe la llamada a una operadora.

5.8 USO DE CALLFILES

Objetivos:

- Definir la estructura de un archivo callfile.
- Especificar el funcionamiento del contexto cita
- Uso de mensajes de broadcast mediante los contextos emergencia y subscripción.
- Configuración del plan de marcación.

Metodología:

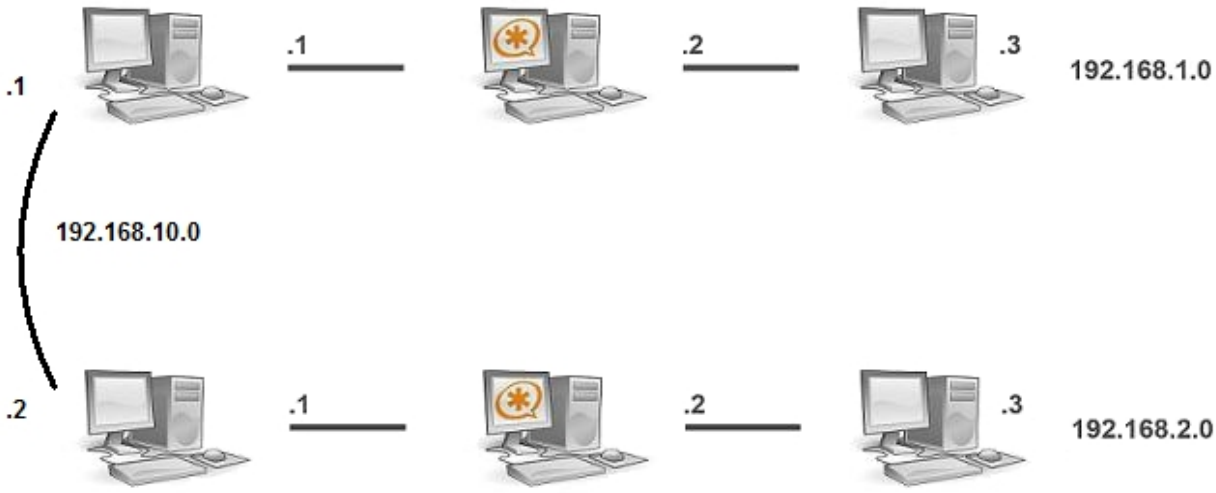


Figura 36: "Configuración de callfiles"

El principal objetivo es usar la variabilidad de opciones que nos permiten los archivos callfiles para crear distintos contextos y funcionalidades.

Para ello se hará uso de un sistema de citas que recoge los datos de la extensión destino y la hora para que se produzca la llamada correspondiente.

Así como también se ha ideado un sistema de envío de mensajes predefinidos para un grupo subscrito a dicho servicio.

Definir la estructura de un archivo callfile:

Los archivos de llamadas o callfiles son archivos de texto estructurados que especifican el uso y la gestión de las llamadas salientes.

Para ejecutarlos bastará con moverlos al directorio */var/spool/asterisk/outgoing/*.

Asterisk llevará un control exhaustivo del contenido de dicho directorio para procesar los archivos situados en él de una forma inmediata o bien esperará a que el tiempo de modificación del archivo coincida con el instante de lectura del contenido del directorio.

Es imprescindible que los archivos se creen en un directorio distinto y luego se muevan al directorio final, de lo contrario no serán reconocidos al quedar incompletos.

Una vez ejecutado el archivo callfile correctamente es borrado, a no ser que se especifique lo contrario en la definición.

Un archivo callfile tiene una configuración definida mediante pares de clave y valor que conforman las distintas líneas e instrucciones dentro del archivo.

La estructura general es la siguiente:

`Channel: <channel>`	Canal usado para la llamada saliente
`Callerid: <callerid>`	Identificador
`WaitTime: <number>`	Tiempo de espera antes de darla por no válida
`MaxRetries: <number>`	Número máximo de reintentos
`RetryTime: <number>`	Tiempo de espera entre reintentos
`Account: <account>`	Código de llamada a usar en CDR

A partir de ahí existen dos posibilidades, ejecutar una aplicación concreta o saltar a una zona del plan de marcación para que ejecute todo aquello que esté definido a continuación.

Por lo tanto, las posibles configuraciones son:

Ejecutar una aplicación:

`Application: <appname>`	Aplicación a ejecutar
`Data: <args>`	Argumentos de dicha aplicación

Saltar a un contexto del plan de marcación:

`Context: <context>`	Contexto del plan de marcación
`Extension: <exten>`	Extensión del contexto especificado
`Priority: <priority>`	Prioridad del contexto
`Setvar: <var=value>`	Asignación de valores a variables

Es posible además archivar el archivo y no borrarlo al finalizar la ejecución, para ello habrá que hacer uso de la instrucción:

`Archive: <yes|no>`

Incluso se puede añadir una línea complementaria en caso de que sea guardado con el estado de la ejecución:

`Status: <exitstatus>` Puede ser: `Expired`, `Completed` o `Failed`

Por definición, el canal usado tendrá al forma: protocolo/extensión_destino, el resto de variables habrá que definirlas de una forma útil, por ejemplo, el tiempo de espera para tomar la llamada como no contestada podrían ser unos 30 segundos, el número de reintentos 3 y el tiempo para realizar una rellamada 300 segundos.

Especificar el funcionamiento del contexto cita:

El contexto `[cita]` que posteriormente se definirá en *extensions.conf* tendrá como finalidad ser una especie de buzón de voz que permite concertar una futura cita telefónica con una extensión determinada.

El objetivo del contexto es permitir que al llamar a la extensión 4567 podamos indicar un número de extensión con la cual concertar una futura cita, además de un día y una hora concretas. Para ello se hará uso de los callfiles, que definirán la extensión destino de una llamada (en este caso nuestra propia extensión) y la aplicación llevada a cabo, que será una llamada a la extensión con la cual se desea concertar la cita.

Dicho contexto será definido en ambos servidores haciendo uso de la descentralización, lo que permite una mayor tolerancia a fallos. En caso de que el servidor de la red contraria fallara no afectaría a las citas de dicha red.

Por lo tanto el tráfico entre redes en un sistema que se supone que puede tener una gran carga de volumen queda restringido únicamente al relativo al cambio de red para una llamada realizada por un callfile.

Para definir el comportamiento del contexto `[cita]` será necesario hacer uso de múltiples variables para guardar los datos necesarios que se usarán posteriormente, así como también será necesario usar archivos de audio que contengan la información necesaria para que el llamante pueda realizar cualquier acción que sea solicitada.

Los distintos archivos de audio contendrán la información siguiente:

- `tfno:` *"Introduzca la extensión con la que quiere concertar una cita. Para terminar pulse #"*
- `añomesdia:` *"Introduzca año, mes y día de la cita"*
- `horamin:` *"Introduzca hora y minuto de la cita"*
- `confirmacioncita:` *"Marque 1 si está de acuerdo con la cita. Marque 2 para modificarla"*

En cuánto a las variables usadas nos encontramos:

- `${numero}`: contiene la extensión con la cuál se desea concertar la cita
- `${fecha}`: contiene el año, mes y día de la cita que se quiere concertar
- `${hora}`: contiene la hora y minutos de la cita
- `${sino}`: contiene el resultado de la confirmación o no confirmación de la cita

El funcionamiento detallado del contexto comienza con una llamada a la extensión 4567, la cuál reproduce en primer lugar el mensaje `tfno`, indicando a continuación por parte del llamante el número de extensión con la que concertar la cita, quedando guardada en la variable `numero`.

También se solicitará la fecha de la forma año, mes y día de la cita, para ello saltará de forma automática el mensaje `añomesdia`, quedando registrada la fecha en la variable `fecha` creada para tal fin.

De la misma forma, será necesario saber la hora y el minuto exactos a la que concertar la cita. La variable `hora` contendrá dichos datos que el cliente marcará una vez se haya reproducido el mensaje `horamin`.

Una vez introducidos extensión y fecha de la cita se solicita una confirmación para comprobar que los datos dados son los deseados. Para ello se reproduce el mensaje `confirmacioncita` que contendrá el resultado de la acción por parte del cliente. La variable que contendrá una de esas opciones será `sino`, que valdrá 1 ó 2 en función de que se confirmen los datos o que las entradas no sean válidas respectivamente.

Si los datos introducidos no son correctos se inicia de nuevo el procedimiento de solicitud de extensión y fecha para concretar una cita.

Si por el contrario son correctos comienza la creación del archivo callfile que gestione la cita programada.

El destino de la llamada generada por el archivo callfile será la extensión que ha solicitado la cita y a continuación se llamará de forma automática mediante la ejecución de dicho archivo a la extensión con la que se ha querido reservar la cita.

Por tanto, el llamante siempre será propio de la red en la cuál se genera el callfile y la cita puede ser cualquier extensión, ya sea de la red 1 o de la red 2.

Lo siguiente será averiguar qué tipo de protocolo usa la extensión llamante. Puesto que a la hora de la numeración hemos diferenciado las extensiones entre SIP (10XX) e IAX(11XX) para la red 1, bastará con ver si el segundo dígito es 0 y por lo tanto se trata de una extensión SIP, o en cambio es IAX.

En caso de que sea SIP, la primera línea del archivo callfile será: `Channel: SIP/{ID}`, dónde `ID` contendrá la extensión que ha efectuado la llamada para concertar la cita y por lo tanto será la receptora de la posterior llamada realizada mediante el callfile.

Si es IAX el protocolo usado por la extensión llamante, la primera línea será: `Channel: IAX2/{ID}`.

El resto del archivo será el mismo para ambas opciones, salvo los parámetros de la aplicación `Dial()`, la cuál contendrá la ruta para la comunicación con la extensión de la cita, ya sea de la red 1, de la red 2, SIP o IAX.

La configuración común de todos los archivos será: `Callerid:Cita`, `WaitTime:30`, `Maxretries:3`, `RetryTime:300`, `Account:{ID}`, `Application:Dial`

Los argumentos de la aplicación `Dial()` variarán dependiendo de la extensión con la que concertar la cita. Pudiendo ésta ser de la red 1 o la red 2, así cómo del protocolo, SIP o IAX.

En primer lugar será necesario saber a qué red pertenece. La variable `numero` contiene la extensión con la que concertar la cita y ésta a su vez tendrá el formato usado hasta ahora, 1XXX para la red 1, o 2XXX para la red2.

Una vez comprobado el primer dígito de la variable `numero` podremos saber de qué red se trata para usar un tipo de marcación u otra.

En caso de que la red sea la 1, habrá que saber además qué tipo de protocolo es el que usa dicha extensión, por lo tanto, según el segundo dígito sabremos si se trata de SIP (10XX) o de IAX (11XX).

Si se trata de una extensión SIP de la red 1, los argumentos de la variable Dial() se escribirán dentro del callfile con el formato: `Data: SIP/{numero}`

Si el protocolo es IAX será: `Data: IAX2/{numero}`

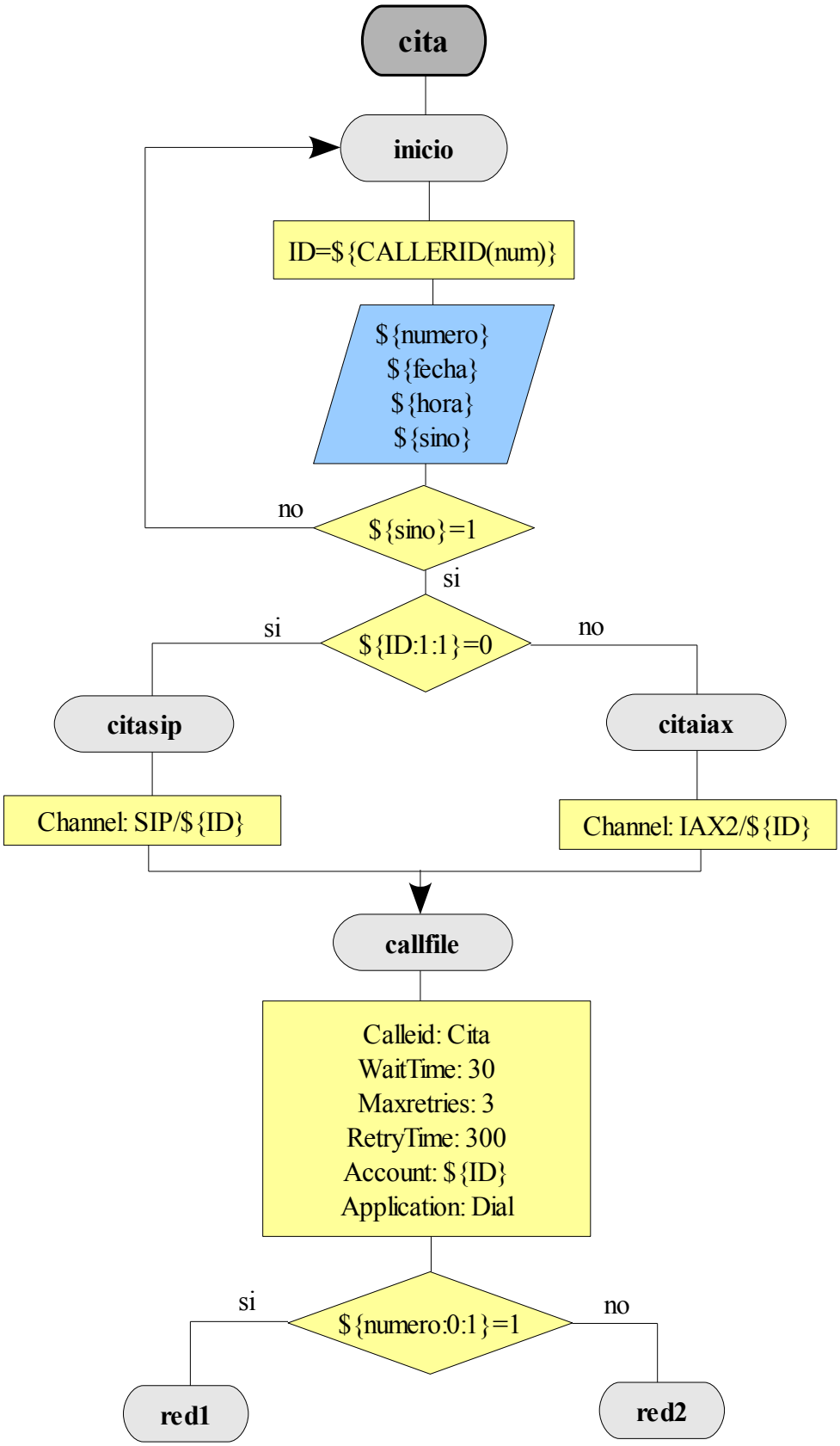
Para la red 2 el procedimiento es igual, habrá que saber el protocolo de la extensión para usarlo en la llamada.

Si la extensión es SIP (20XX), la línea a añadir al callfile será `Data: SIP/red2/{numero}`, si se trata de IAX (21XX) será `Data: IAX2/red2/{numero}`

Una vez definidos los argumentos de `Dial()` habrá que cambiar la hora de la última modificación del archivo a la indicada por el cliente. Para ello se hará uso del comando `touch -t`. Con ésto se consigue que el archivo se ejecute a la hora definida, no antes.

Como última acción habrá que mover el archivo callfile al directorio */var/spool/asterisk/outgoing/* para que Asterisk lo ejecute cuándo la hora de modificación coincida con la hora real del sistema.

Cabría la posibilidad de que el contexto `[cita]` sólo estuviera reflejado en un servidor y se accediera a él desde otro con una llamada. La configuración del contexto `[cita]` sería muy distinto puesto que primero habría que averiguar desde qué red se realiza la llamada y hacia qué red va dirigida y luego ver el protocolo que usa dicha extensión, por lo tanto la realización sería mucho más intrincada. De todas formas, lo ideal es que cada servidor tenga su propio manejo del contexto `[cita]` y que el destino de la cita pueda variar. Posibilitando con ello una mayor tolerancia a fallos en caso de que un servidor deje de funcionar.


cita
inicio
ID=${CALLERID(num)}
${numero}
${fecha}
${hora}
${sino}
no
${sino}=1
si
si
${ID:1:1}=0
no
citasip
citaiax
Channel: SIP/${ID}
Channel: IAX2/${ID}
callfile
Calleid: Cita
WaitTime: 30
Maxretries: 3
RetryTime: 300
Account: ${ID}
Application: Dial
si
${numero:0:1}=1
no
red1
red2

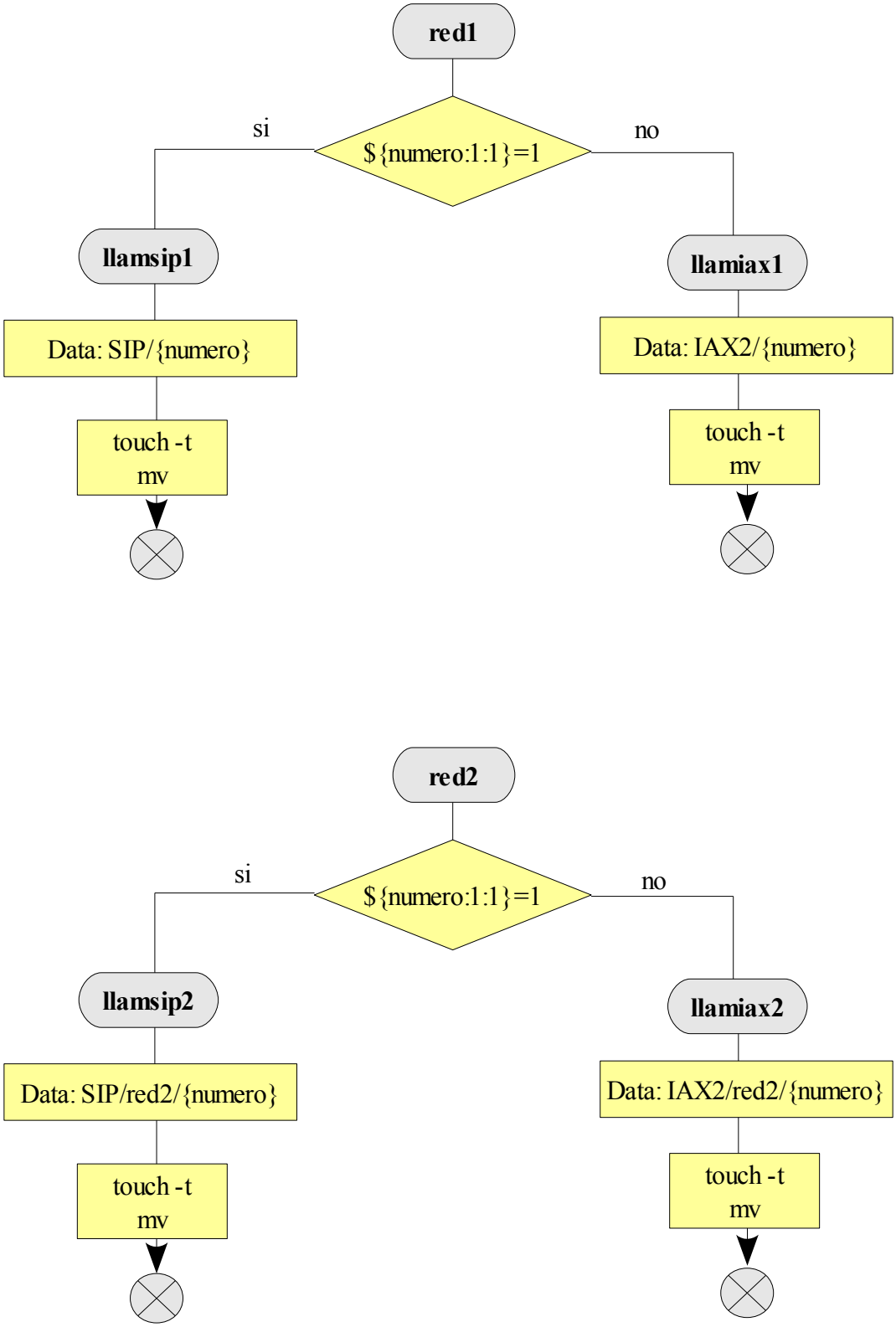


Figura 37: “Diagrama de flujo del contexto [cita]”

Uso de mensajes de broadcast mediante los contextos emergencia y subscripción:

Una de las múltiples posibilidades de los archivos callfiles es enviar mensajes de grupo para advertir de un estado de emergencia o simplemente para enviar una información común.

Para ello se ha optado por crear un servicio de emergencia, el cuál se activa al llamar a la extensión `112`, dónde se grabará un mensaje y éste será enviado a todas las extensiones que se hayan subscrito a dicho servicio.

Serán necesarios dos contextos, `[emergencia]` y `[subscripcion]`, para poder abarcar tanto la llamada en sí y la grabación del mensaje, como la subscripción de las extensiones a dicho servicio.

- `[emergencia]`

El contexto `[emergencia]` permitirá grabar un archivo de audio al llamar a la extensión `112`, que quedará almacenado en el servidor en la ruta */var/lib/asterisk/sounds/en/emergencia/mensaje.wav*, usando la misma ruta tanto en el servidor de la red 1 como en el de la red 2.

Una vez guardado el archivo de audio, se ejecutarán los callfiles de las extensiones que se hayan subscrito al servicio. Para ello bastará con mover los archivos, previamente creados, al directorio */var/spool/asterisk/outgoing/*, dónde Asterisk los ejecutará de forma inmediata

Los archivos de audio usados son los siguientes:

- `bienvenida:` *"Introduzca el mensaje de emergencia. Para terminar pulse #"*
- `mensaje`: mensaje introducido por el usuario con extensión wav.
- `confirmacion:` *"Pulse 1 para enviar su mensaje a todas las extensiones. Pulse 0 para cancelar"*

Las variables usadas dentro del contexto son:

- `${conf}`: contiene el resultado de la confirmación del mensaje de emergencia grabado por el usuario.

El funcionamiento exacto de dicho contexto comienza con una llamada a la extensión `112` con la finalidad de grabar un mensaje de emergencia que sea distribuido a todas aquellas extensiones subscritas al servicio.

Comienza con una grabación que da la bienvenida al llamante y que insta a que grabe el mensaje deseado que quiera compartir, quedando guardado en el archivo `mensaje.wav` gracias a la aplicación `Record()`.

Una vez reproducido dicho mensaje para verificar la grabación, el usuario podrá enviarlo a todas las extensiones subscritas o bien salir del contexto si no ha quedado satisfecho con el resultado.

El procedimiento para enviar el mensaje no es otra cosa que mover los callfiles al directorio */var/spool/asterisk/outgoing/* para que Asterisk los ejecute. Dichos archivos habrán sido creados previamente y contendrán tanto la extensión a la que llamar como la aplicación a ejecutar, que no será otra que una reproducción del mensaje de emergencia.

Para salvaguardar una copia de la subscripción de las extensiones en forma de archivos callfiles será necesario hacer uno de un directorio dónde queden almacenados y que sea una copia de éstos la que pase al directorio final, ya que tras la ejecución de los archivos éstos serán eliminados por completo.

Se creará un directorio llamado *callfiles* que contendrá uno llamado *copia* en su interior, en */callfiles/* quedarán guardados los callfiles originales fruto de la subscripción, y en */callfiles/copia/* habrá una copia temporal de ellos que se moverá al directorio destino para que puedan ser ejecutados por Asterisk.

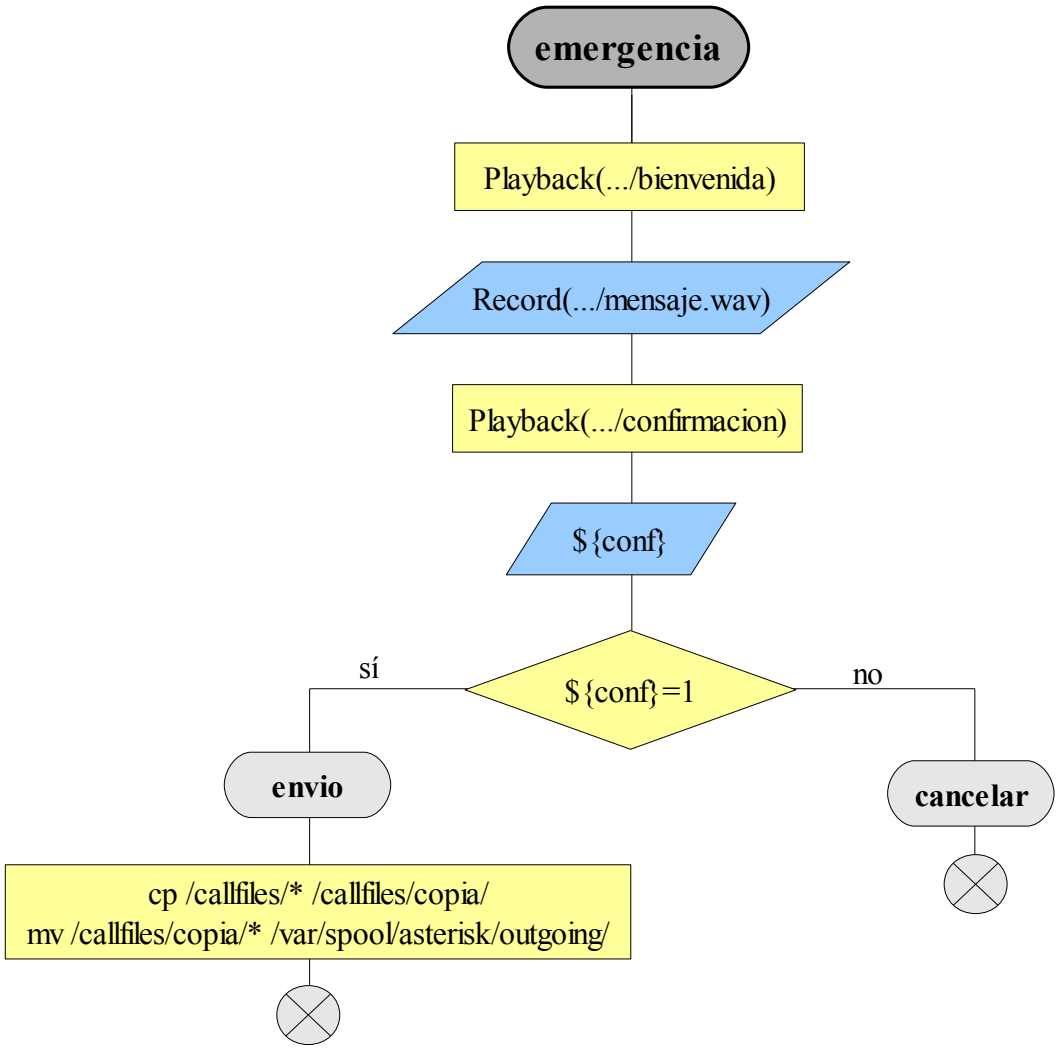


Figura 38: "Diagrama de flujo del contexto [emergencia]"

- `[subscripcion]`

El propósito de este contexto es realizar una suscripción a un servicio determinado como es el de emergencia. También es posible cancelar la suscripción previa.

La forma de subscribirse será realizando una llamada a la extensión `555`, la cuál creará un archivo callfile con el nombre de la extensión y dónde se especificará la extensión destino de la llamada (la propia extensión llamante) así como la aplicación que realiza, en cuyo caso será la reproducción del archivo `mensaje.wav` que contendrá la grabación de emergencia a distribuir.

Para eliminar la subscripción bastará con eliminar el callfile correspondiente a la extensión que no desea mantener el servicio.

Además, como medida preventiva en caso de que uno de los servidores pueda dejar de funcionar, y asegurando que ambas redes puedan contener la misma información en cuánto a subscripciones se ha optado por realizar la misma operación en el servidor de la red contraria.

Se realizará una llamada a una extensión determinada valiéndose del cliente `red2`. En la red 2 deberán existir dichas extensiones y además generarán un archivo callfile con la ruta a cada una de las extensiones suscritas, que serán distintas para cada servidor.

De la misma forma, la red 1 deberá contemplar aquellas extensiones de las llamadas que puedan provenir de la red 2.

Para distinguir el objetivo de la llamada se ha usado un código numérico compuesto de dos dígitos, `00` seguido de la extensión correspondiente servirá para cancelar una suscripción, y `01` para realizarla.

En la red1 habrá que incorporar las extensiones:

- `0120XX`: suscripción al servicio de una extensión SIP de la red 2
- `0121XX`: suscripción al servicio de una extensión IAX de la red 2
- `0020XX`: cancelación del servicio por parte de una extensión SIP de la red 2
- `0021XX`: cancelación del servicio por parte de una extensión IAX de la red 2

Para la red 2, las extensiones que habrán de incorporarse dentro del contexto [subscripcion] serán:

- `0110XX`: suscripción al servicio de una extensión SIP de la red 1
- `0111XX`: suscripción al servicio de una extensión IAX de la red 1
- `0010XX`: cancelación del servicio por parte de una extensión SIP de la red 1
- `0011XX`: cancelación del servicio por parte de una extensión IAX de la red 1

El resultado será que tanto el servidor de la red 1 como el de la red 2 tendrá subscritas las mismas extensiones, aunque la ruta del callfile sea distinta.

Los archivos de audio usados en dicho contexto son:

- `intro:` *"Bienvenido al servicio de subscripción para mensajes broadcast. Pulse 1 para subscribirse. Pulse 0 para cancelar la subscripción"*
- `fin1:` *"Subscripción realizada con éxito"*
- `fin0:` *"Subscripción anulada"*

En cuánto a las variables manejadas tenemos:

- `${extenred1}`: Contiene la extensión llamante (`CALLERID(num)`)
- `${confirmar}`: Contiene la respuesta del usuario a la subscripción
- `${dialred2}`: Contiene la ruta para llamar a la extensión de la red 2 que generará el callfile que permita la subscripción al servicio. La forma de la ruta será: `protocolo/red2/código de subscripción+${extenred1}`, por lo tanto podrá contener tanto la subscripción como la cancelación de la extensión llamante en la red contraria.
- `${extenred2}`: Contendrá la extensión destino de la llamada (efectuada desde la red contraria), salvo los dos primeros dígitos que son los propios del código de subscripción.

El comportamiento del contexto `[subscripción]` comienza con una llamada a la extensión `555` con la finalidad de subscribirse al servicio de compartición de mensajes de emergencia en grupo.

Una vez guardada en una variable la extensión llamante con el fin de manejar los dígitos de una forma más cómoda posteriormente, se reproducirá un mensaje de audio que instará al cliente a subscribirse al servicio o por el contrario a borrarse de él.

Teniendo en cuenta que la subscripción consiste en la creación de un archivo callfile con el nombre de la extensión, la cancelación del servicio no es otra cosa que un borrado de dicho archivo, tanto en el servidor desde dónde se efectúa la llamada como en el contrario.

Ahora bien, si lo que desea el cliente es contar con ese servicio en su haber, habrá que seguir una serie de pasos para averiguar si se trata de una extensión SIP o IAX, para ello, como en casos anteriores es necesario ver si el segundo dígito es `0` y por lo tanto la extensión es SIP o es `1` y por lo tanto es IAX.

Una vez resuelto el tipo de protocolo que usa la extensión se escribirá la primera línea del archivo callfile de la siguiente forma:

`Channel: SIP/${extenred1}` o `Channel: IAX2/${extenred1}`

Además, la variable `dialred2` contendrá la ruta de llamada para la extensión espejo en la red contraria.

En caso de que la extensión llamante sea SIP, la variable `dialred2` contendrá el valor: `SIP/red2/01${extenred1}`. Éstos serán los argumentos que usará la aplicación `Dial()` cuándo llame a la red 2 para crear el archivo que subscriba a la extensión de la red 1.

Si la extensión llamante fuera por el contrario IAX, `dialred2` contendría el valor: `IAX2/red2/01{extenred1}`. Así mismo sería usado más adelante para llamar a la extensión correspondiente de la red 2.

El resto del cuerpo del callfile será el mismo independientemente de si se trata de una extensión que use uno u otro protocolo.

```
Callerid: Emergencia
WaitTime: 30
Maxretries: 3
RetryTime: 300
Account: ${extenred1}
Application: Playback
Data: /var/lib/asterisk/sounds/en/emergencia/mensaje
```

Una vez terminado el archivo que tendrá como nombre el de la extensión llamante y estará ubicado en un directorio en el cuál Asterisk no supervise constantemente, en éste caso /callfiles/, se reproducirá el mensaje de audio que confirmará que la subscripción se ha realizado de forma correcta.

Por último se hará una llamada a la extensión correspondiente de la red 2 para que actualice sus subscripciones y añada ésta extensión.

Ahora bien, también será necesario especificar las extensiones a las que se puede llamar desde la red 2. De la misma forma, para subscribirse al servicio se hará uso del código `01`, y para borrarse de él se usará `00`.

Por lo tanto, las extensiones que recojan las llamadas desde la red 2 serán del tipo `0120XX` en caso de tratarse de una extensión SIP, `0121XX` si fuera una extensión IAX quién quisiera añadir su subscripción también en la red 1, o por el contrario aquellas que empezaran por `00`, como `0020XX` que cancelaría el servicio de una extensión SIP o `0021XX` que lo cancelaría para una extensión IAX.

Una vez claros los cuatro tipos de extensiones que pueden ser llamados desde la red 2 es necesario definirlas por completo.

En el caso de la subscripción al servicio la única variación entre los callfiles de las extensiones que usan SIP o IAX es la primera línea, que identificará el protocolo usado. Dicha línea además deberá mostrar una ruta de llamada a la extensión de la red 2, por lo tanto habrá que hacer uso del cliente `red2` para que la gestione.

El resto del callfile será el mismo que para la subscripción de las extensiones de la propia red, a diferencia de `Account`, que en éste caso tendrá el valor `${extenred2}`

Para eliminar una subscripción de una extensión que pertenezca a la red contraria bastará con eliminar el archivo callfile cuyo nombre coincida con dicha extensión.

La duplicación de archivos de subscripción en ambos servidores tiene por objetivo que la tolerancia a fallos del sistema sea mayor.

Después de muchas pruebas se ha escogido éste sistema por la simplicidad de uso y modificación posterior, sin embargo, antes de haber sido escogido se han valorado otros sistemas, así como los aspectos negativos de ellos.

Teniendo constancia de que una duplicación de archivos callfile en los distintos servidores conllevaría distintos archivos se barajaron posibilidades como:

Crear los archivos y las copias en un servidor y copiarlos a otro:

- Crear en el mismo servidor los archivos necesarios, tanto para ese servidor como la copia que habría de pasarse al servidor contrario. El principal problema era que había que tener un conocimiento extenso de la configuración del otro servidor y la forma que tiene de comunicarse con el primero, por lo tanto no sería válido en sistemas dónde el número de redes y de administradores fuera mediano o grande. Otro de los problemas surgía por la necesidad de copiar el archivo mediante `rcp`, el cuál no proporciona seguridad y es un posible punto de fallo.

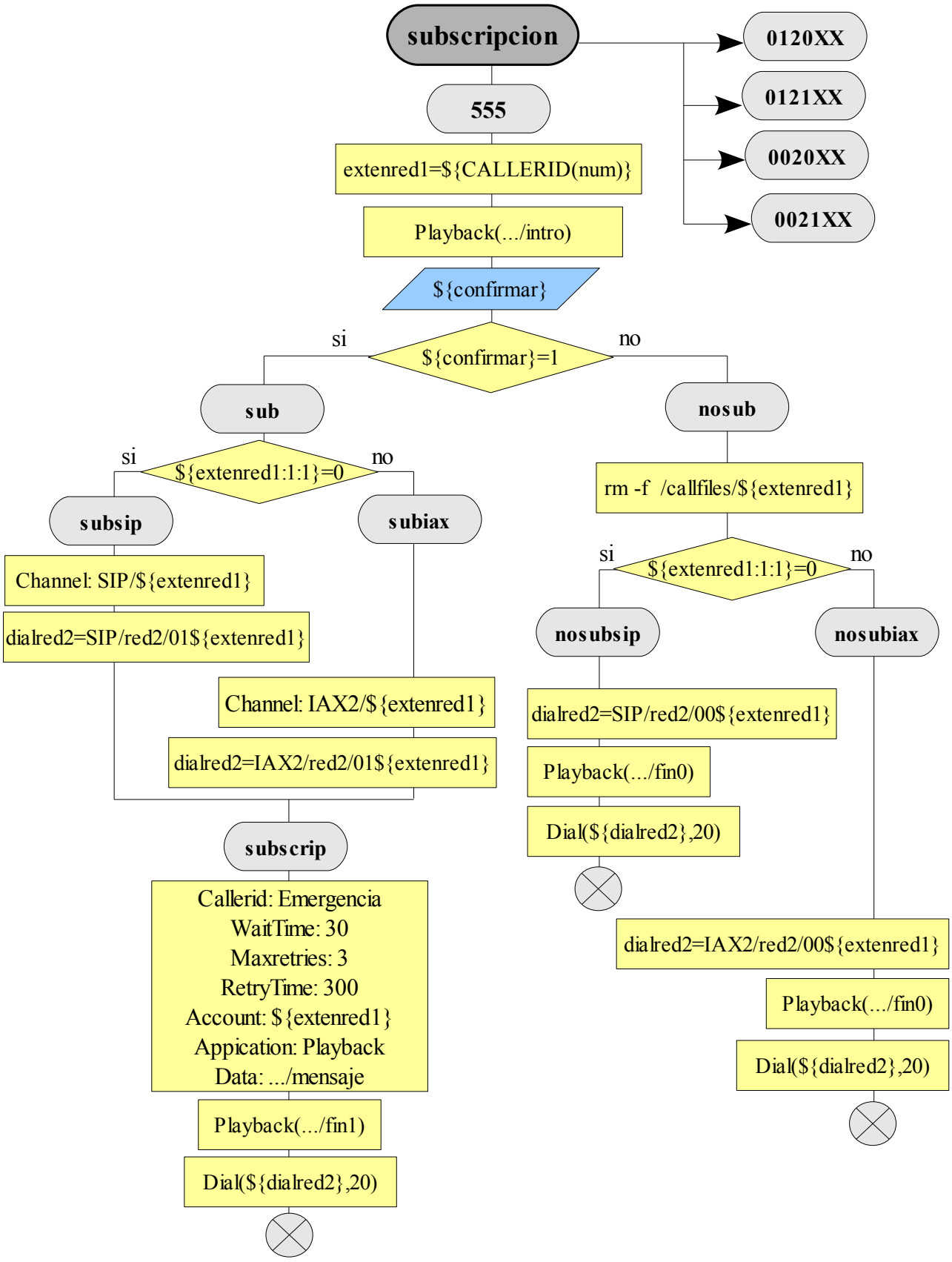
Llamar al servidor de la red contraria para que genere los archivos de subscripción necesarios.

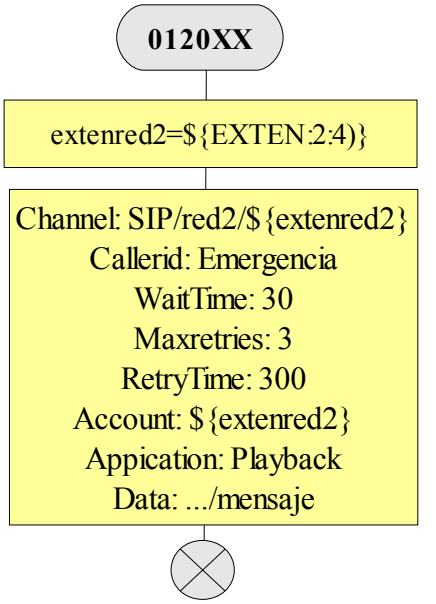
- Haciendo uso de la configuración inicial del contexto `[subscripcion]` cabría la posibilidad de que al llamar a la extensión `555` ésta recogiera la llamada y en función de si pertenecía a la propia red o a la red contraria, realizara un proceso u otro. Sin embargo, el principal problema aparece al comprobar el contenido de la variable `${CALLERID(num)}`, la cuál contiene la extensión llamante en caso de que pertenezca a esa misma red, o por el contrario contiene el cliente `red1` o `red2`, pero no la extensión que quiere iniciar el proceso de subscripción, por lo tanto, a falta de cliente para generar el archivo callfile se descarta esta opción.
- Otra de las opciones fue recoger una llamada en una extensión concreta de la red contraria y que se genere un archivo para cada uno de los clientes. Con ésto se soluciona la necesidad de conocer cómo funcionan el resto de servidores. Aparece el problema del requerimiento de código para abarcar cada extensión y cada posibilidad, sin embargo se pueden aunar las extensiones que tengan el mismo comportamiento.

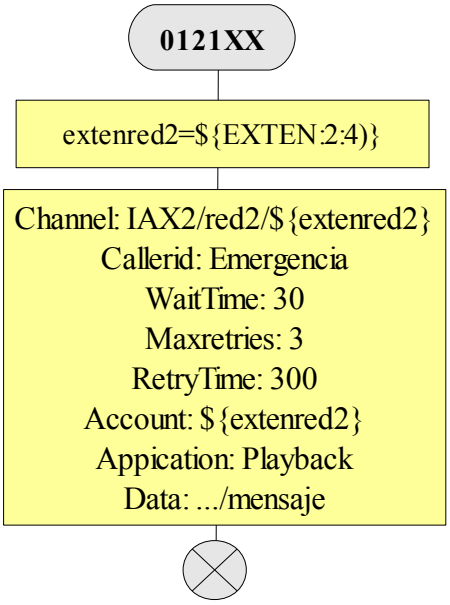
La última de las posibilidades fue la escogida, diferenciando en primer lugar si se trata de una subscripción o de una cancelación del servicio y después el tipo de protocolo usado para así poder crear el correspondiente archivo que indique la forma de llamar a dicha extensión en la red contraria.

Ha sido necesario añadir dentro de la extensión de recepción de la llamada, la propia extensión que inicia el proceso, puesto que no es posible traspasar variables de una red a otra. La única forma de averiguar la extensión que llamó para subscribirse al sistema era incluirla dentro de la extensión de recepción en el servidor contrario.

Para ello se ha usado un código que preceda a la extensión y que a su vez indique lo que desea hacer el cliente.


subscripcion
0120XX
0121XX
0020XX
0021XX
555
extenred1=${CALLERID(num)}
Playback(.../intro)
${confirmar}
si
${confirmar}=1
no
sub
nosub
si
${extenred1:1:1}=0
no
subsip
subiax
rm -f /callfiles/${extenred1}
Channel: SIP/${extenred1}
dialred2=SIP/red2/01${extenred1}
si
${extenred1:1:1}=0
no
nosubsip
nosubiax
Channel: IAX2/${extenred1}
dialred2=IAX2/red2/01${extenred1}
dialred2=SIP/red2/00${extenred1}
Playback(.../fin0)
Dial(${dialred2},20)
subscrip
Callerid: Emergencia
WaitTime: 30
Maxretries: 3
RetryTime: 300
Account: ${extenred1}
Appication: Playback
Data: .../mensaje
dialred2=IAX2/red2/00${extenred1}
Playback(.../fin0)
Dial(${dialred2},20)
Playback(.../fin1)
Dial(${dialred2},20)


0120XX
extenred2=${EXTEN:2:4)}
Channel: SIP/red2/${extenred2}
Callerid: Emergencia
WaitTime: 30
Maxretries: 3
RetryTime: 300
Account: ${extenred2}
Appication: Playback
Data: .../mensaje


0121XX
extenred2=${EXTEN:2:4)}
Channel: IAX2/red2/${extenred2}
Callerid: Emergencia
WaitTime: 30
Maxretries: 3
RetryTime: 300
Account: ${extenred2}
Appication: Playback
Data: .../mensaje

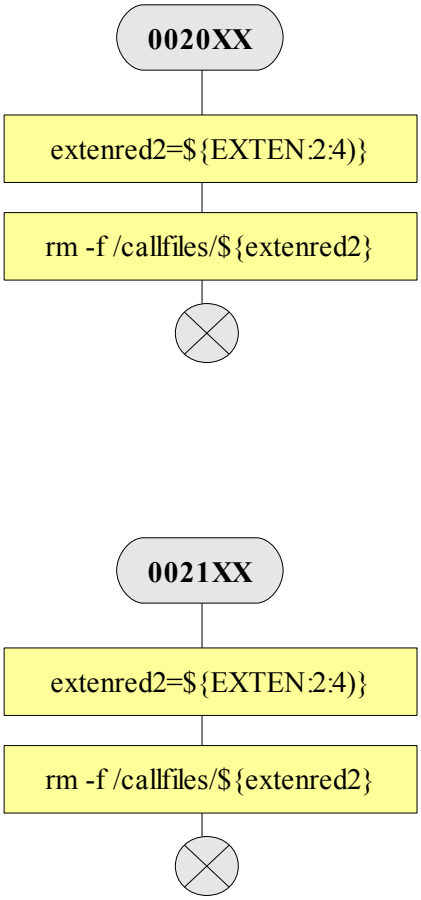


Figura 39: "Diagrama de flujo del contexto [subscripcion]"

Configuración del plan de marcación:

Al plan de marcación del caso de uso anterior se han añadido tres nuevos contextos independientes: `[cita]`, `[emergencia]`, `[subscripcion]`

Cada extensión del plan de marcación tendrá acceso a cada uno de ellos, ya sea para concertar una cita, enviar un mensaje de emergencia o bien subscribirse al servicio de envío de mensajes grupales.

Los contextos `[cita]` y `[emergencia]` serán iguales tanto en el servidor de la red 1 como en el de la red 2.

Sin embargo se ha querido añadir robustez a las subscripciones y se ha usado la creación de archivos callfiles de una extensión en ambos servidores para tal fin.

Para ello se ha optado por usar la extensión `555` para realizar la subscripción o cancelación de un usuario de dicha red y distintas extensiones para recoger las llamadas que generarán los archivos de los clientes de la red contraria.

Dichos archivos variarán la ruta de llamada en función de si se llama a la red a la que pertenece el cliente o si se hace a la red contraria, haciendo uso de los clientes `red1` y `red2` según corresponda.

A modo representativo se añade el archivo de configuración del plan de marcación.

extensions.conf de la red 1:

```
[general]
      …

;--------------------------
;SIP
;--------------------------
[prueba8-sip]
      include => internas-sip1
      include => externas-sip
      include => prueba8-iax
      include => meetme
      include => parkedcalls
      include => llamadaIVR
      include => grabacionAudio
      include => cita
      include => emergencia
      include => subscripcion

[internas-sip1]
      …
      include => subscripcion

[externas-sip]
      …

;-------------------------------------
;IAX
;-------------------------------------
[prueba7-iax]
      include => internas-iax1
      include => externas-iax
      include => meetme
      include => parkedcalls
      include => llamadaIVR
      include => grabacionAudio
      include => cita
      include => emergencia
      include => subscripcion
```

```
[internas-iax1]
      …
      include => meetme
      include => llamadaIVR
      include => subscripcion

[externas-iax]
      …

;---------------------------------------
;MEETME
;---------------------------------------
[meetme]

      …

;--------------------------------------
;respuesta de voz interactiva IVR
;--------------------------------------
[llamadaIVR]

      …

;--------------------------------------
;grabación de archivos de audio
;--------------------------------------
[grabacionAudio]

      …

;--------------------------------------
;cita telefónica
;--------------------------------------
[cita]
      exten => 4567,1,Answer()
      exten => 4567,n,Wait(1)

      ;inicio
      exten => 4567,n(inicio),Set(ID=${CALLERID(num)})
      exten => 4567,n,Playback(/var/lib/asterisk/sounds/en/cita/tfno)
      exten => 4567,n,Read(numero,,15,,2,10)
      exten => 4567,n,SayDigits(${numero})
```

```
      exten => 4567,n,Wait(1)
      exten => 4567,n,Playback(/var/lib/asterisk/sounds/en/cita/horamin)
      exten => 4567,n,Read(hora,,4,,2,10)
      exten => 4567,n,SayDigits(${hora})

      exten =>
4567,n,Playback(/var/lib/asterisk/sounds/en/cita/confirmacioncita)
      exten => 4567,n,Read(sino,,1,,,5)

      exten => 4567,n,GotoIf($["${sino}"="1"]?cita:inicio)

      ;cita
      exten => 4567,n(cita),GotoIf($["${ID:1:1}"="0"]citasip:citaiax)

      ;citasip
      exten => 4567,n(citasip),System(echo Channel:SIP/${ID}>>/tmp/callback)
      exten => 4567,n,Goto(callfile)

      ;citaiax
      exten => 4567,n(citaiax),System(echo Channel:IAX2/${ID}>>/tmp/callback)

      ;callfile
      exten => 4567,n(callfile),System(echo Callerid:Cita>>/tmp/callback)
      exten => 4567,n,System(echo WaitTime:30>>/tmp/callback)
      exten => 4567,n,System(echo Maxretries:3>>/tmp/callback)
      exten => 4567,n,System(echo RetryTime:300>>/tmp/callback)
      exten => 4567,n,System(echo Account:${ID}>>/tmp/callback)
      exten => 4567,n,System(echo Application:Dial>>/tmp/callback)
      exten => 4567,n,GotoIf($["${numero:0:1}"="1"]?red1:red2)

      ;red1
      exten => 4567,n(red1),GotoIf($["${numero:1:1}"="0"]?llamsip1:llamiax1)

      ;llamsip1
      exten => 4567,n(llamsip1),System(echo Data:SIP/${numero}>>/tmp/callback)
      exten => 4567,n,System(touch -t ${fecha}${hora} /tmp/callback)
      exten => 4567,n,System(mv /tmp/callback /var/spool/asterisk/outgoing/
${fecha}${hora})
      exten => 4567,n,Hangup

      ;llamiax1
      exten => 4567,n(llamiax1),System(echo Data:IAX2/${numero}>>/tmp/callback)
      exten => 4567,n,System(touch -t ${fecha}${hora} /tmp/callback)
```

```
      exten => 4567,n,System(mv /tmp/callback /var/spool/asterisk/outgoing/$
{fecha}${hora})
      exten => 4567,n,Hangup

      ;red2
      exten => 4567,n(red2),GotoIf($["${numero:1:1}"="0"]?llamsip2:llamiax2)

      ;llamsip2
      exten => 4567,n(llamsip2),System(echo Data:SIP/red2/$
{numero}>>/tmp/callback)
      exten => 4567,n,System(touch -t ${fecha}${hora} /tmp/callback)
      exten => 4567,n,System(mv /tmp/callback /var/spool/asterisk/outgoing/$
{fecha}${hora})
      exten => 4567,n,Hangup

      ;llamiax2
      exten => 4567,n(llamiax2),System(echo Data:IAX2/red2/$
{numero}>>/tmp/callback)
      exten => 4567,n,System(touch -t ${fecha}${hora} /tmp/callback)
      exten => 4567,n,System(mv /tmp/callback /var/spool/asterisk/outgoing/
{fecha}${hora})
      exten => 4567,n,Hangup

;--------------------------------------
llamadas broadcast de emergencia
;--------------------------------------

[emergencia]
      exten => 112,n,Answer()
      exten => 112,n,Wait(2)
      exten =>
112,n,Playback(/var/lib/asterisk/sounds/en/emergencia/bienvenida)
      exten => 112,n,Wait(1)
      exten => 112,n,Record(/var/lib/asterisk/sounds/en/emergencia/mensaje.wav)
      exten => 112,n,Wait(1)
      exten => 112,n,Playback(/var/lib/asterisk/sounds/en/emergencia/mensaje)
      exten => 112,n,Wait(1)
      exten =>
112,n,Playback(/var/lib/asterisk/sounds/en/emegencia/confirmacion)
      exten => 112,n,Read(conf,,1,,1,5)
      exten => 112,n,GotoIf($["${conf}"="1"]?envio:cancelar)

      exten => 112,n(envio),System(cp /callfiles/* /callfiles/copia/)
      exten => 112,n,System(mv /callfiles/copia/*
```

```
/var/spool/asterisk/outgoing/)
      exten => 112,n,Hangup

      exten => 112,n(cancelar),Hangup

;-------------------------------------
servicio de subscripción para mensajes de grupo o broadcast
;-------------------------------------

[subscripcion]
      exten => 555,1,Answer()
      exten => 555,n,Set(extenred1=${CALLERID(num)})
      exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/intro)
      exten => 555,n,Read(confirmar,,1,,,5)
      exten => 555,n,GotoIf($["${confirmar}"="1"]?sub:nosub)

      ;sub
      exten => 555,n,(sub),GotoIf($["${extenred1:1:1}"="0"]?subsip:subiax)

      ;subsip
      exten => 555,n(subsip),System(echo Channel:SIP/${extenred1}>>/callfiles/
${extenred1})
      exten => 555,n,Set(dialred2=SIP/red2/01${extenred1})
      exten => 555,n,Goto(subscrip)

      ;subiax
      exten => 555,n(subiax),System(echo Channel:IAX2/${extenred1}>>/callfiles/
${extenred1})
      exten => 555,n,Set(dialred2=IAX2/red2/01${extenred1})

      ;subscrip
      exten => 555,n(subscrip),System(echo Callerid:Emergencia>>/callfiles/
${extenred1})
      exten => 555,n,System(echo WaitTime:30>>/callfiles/${extenred1})
      exten => 555,n,System(echo Maxretries:3>>/callfiles/${extenred1})
      exten => 555,n,System(echo RetryTime:300>>/callfiles/${extenred1})
      exten => 555,n,System(echo Account:${extenred1}>>/callfiles/${extenred1})
      exten => 555,n,System(echo Application:Playback>>/callfiles/${extenred1})
      exten => 555,n,System(echo
Data:/var/lib/asterisk/sounds/en/emergencia/mensaje>>/callfiles/${extenred1})
      exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/fin1)
      exten => 555,n,Dial(${dialred2},20)
      exten => 555,n,Hangup
```

```
      ;nosub
      exten => 555,n(nosub),System(rm -f /callfiles/${extenred1})
      exten => 555,n,GotoIf($["${extenred1:1:1}"="0"]?nosubsip:nosubiax)

      ;nosubsip
      exten => 555,n(nosubsip),Set(dialred2=SIP/red2/00${extenred1})
      exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/fin0)
      exten => 555,n,Dial(${dialred2},20)
      exten => 555,n,Hangup

      ;nosubiax
      exten => 555,n(nosubiax),Set(dialred2=IAX2/red2/00${extenred1})
      exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/fin0)
      exten => 555,n,Dial(${dialred2},20)
      exten => 555,n,Hangup

      ;extensiones para aceptar las subscripciones de la red2

      exten => _0120XX,n,Answer()
      exten => _0120XX,n,Set(extenred2=${EXTEN:2:4})
      exten => _0120XX,n,System(echo Channel:SIP/red2/${extenred2}>>/callfiles/
${extenred2})
      exten => _0120XX,n,System(echo Callerid:Emergencia>>/callfiles/
${extenred2})
      exten => _0120XX,n,System(echo WaitTime:30>>/callfiles/${extenred2})
      exten => _0120XX,n,System(echo Maxretries:3>>/callfiles/${extenred2})
      exten => _0120XX,n,System(echo RetryTime:300>>/callfiles/${extenred2})
      exten => _0120XX,n,System(echo Account:${extenred2}>>/callfiles/
${extenred2})
      exten => _0120XX,n,System(echo Application:Playback>>/callfiles/
${extenred2})
      exten => _0120XX,n,System(echo
Data:/var/lib/asterisk/sounds/en/emergencia/mensaje>>/callfiles/${extenred2})
      exten => _0120XX,n,Hangup

      exten => _0121XX,n,Answer()
      exten => _0120XX,n,Set(extenred2=${EXTEN:2:4})
      exten => _0121XX,n,System(echo Channel:IAX2/red2/
${extenred2}>>/callfiles/${extenred2})
      exten => _0121XX,n,System(echo Callerid:Emergencia>>/callfiles/
${extenred2})
      exten => _0121XX,n,System(echo WaitTime:30>>/callfiles/${extenred2})
```

```
      exten => _0121XX,n,System(echo Maxretries:3>>/callfiles/${extenred2})
      exten => _0121XX,n,System(echo RetryTime:300>>/callfiles/${extenred2})
      exten => _0121XX,n,System(echo Account:${extenred2}>>/callfiles/
${extenred2})
      exten => _0121XX,n,System(echo Application:Playback>>/callfiles/
${extenred2})
      exten => _0121XX,n,System(echo
Data:/var/lib/asterisk/sounds/en/emergencia/mensaje>>/callfiles/${extenred2})
      exten => _0121XX,n,Hangup

      exten => _0020XX,n,Answer()
      exten => _0020XX,n,Set(extenred2=${EXTEN:2:4})
      exten => _0020XX,n,System(rm -f /callfiles/${extenred2})
      exten => _0020XX,n,Hangup

      exten => _0021XX,n,Answer()
      exten => _0021XX,n,Set(extenred2=${EXTEN:2:4})
      exten => _0021XX,n,System(rm -f /callfiles/${extenred2})
      exten => _0021XX,n,Hangup
```

El plan de marcación de la red 2 es muy similar a éste, con las diferencias lógicas que suponen cambiar las extensiones y la forma de relacionarse con el servidor de la red 1.

El contexto `[cita]` apenas sufre modificaciones. De la misma forma que en la red 1 al llamar a la extensión `4567` se solicita el número de extensión con el que concertar la cita y fecha.

Una vez confirmado habrá que generar el archivo callfile. Para ello habrá qué saber qué protocolo usa la extensión llamante para así dejarlo reflejado en `Channel` dentro del archivo.

Otra cosa a tener en cuenta a la hora de completar el archivo *callfile* de la cita será saber con qué extensión se desea contactar y dónde se ubica, para ello, de la misma forma habrá que saber si pertenece a la propia red o a la contraria.

En caso de que la extensión con la que se quiere concertar la cita pertenece a la red contraria habrá que hacer uso del cliente `red1` que permitirá la comunicación entre ambas redes.

Una vez finalizado el archivo callfile, sea cual sea su contenido, se modificarán los parámetros temporales para que se ejecute a la hora y fecha seleccionada por el cliente.

Por último se enviará al directorio de Asterisk para que sea ejecutado.

El contexto `[emergencia]` es idéntico que el de la red 1. El mensaje grabado por el usuario quedará guardado en la misma ruta con el fin de usarla dentro de los archivos callfiles independientemente de donde provengan.

Será este contexto el que vuelque todos los archivos de subscripción al directorio correspondiente de Asterisk para que sean ejecutados a la vez.

El contexto `[subscripcion]` será el que más cambios sufra. El procedimiento será el mismo, llamar a la extensión `555` para crear un archivo callfile que dé fe de que se ha subscrito al servicio de mensajería de voz.

En función del tipo de protocolo de la extensión que quiere subscribirse se usa una línea u otra en la configuración de `Channel` en el archivo callfile.

Se usa además una variable llamada `dialred1` que contendrá parte del contenido que necesita la aplicación `Dial()` para comunicarse con la red 1. Contendrá el protocolo, el cliente asociado para conectar con la red 1 llamado `red1`, el código de subscripción (`01` para subscribirse, `00` para borrarse) y la extensión que se ha subscrito.

A continuación se creará el resto del archivo callfile, cuyo nombre coincide con la extensión que se desea subscribir, especificando por tanto la aplicación `Playback()` y los argumentos asociados a ella, como es que reproduzca el archivo de audio `mensaje` que será el que grabe el usuario al llamar a la extensión `112`.

Para borrarse del servicio bastará con eliminar los archivos callfile con el nombre de la extensión.

Además, deberán existir una serie de extensiones que permitan recibir las llamadas realizadas desde la red1 para subscribirse o borrarse del sistema de envío de mensajes. Por lo tanto se crearán unas extensiones con la finalidad de borrar o de crear un archivo callfile de una extensión que desea subscribirse desde la red contraria.

Así pues, las extensiones usadas para tal fin serán:

- `0110XX`: subscripción de una extensión SIP de la red 1
- `0111XX`: subscripción de una extensión IAX de la red 1
- `0010XX`: cancelación de subscripción de una extensión SIP de la red 1
- `0011XX`: cancelación de subscripción de una extensión IAX de la red 1

5.9 INTERFAZ GRÁFICA

Objetivos:

- Interfaz de Asterisk, usos y posibilidades
- Usos de comandos de consola mediante Telnet
- Uso de AJAM (Asynchronous Javascript Asterisk Manager)
- Uso de la interfaz gráfica Asterisk GUI 2.0

Metodología:

Entre muchas otras cosas, la interfaz de Asterisk Manager (Asterisk Manager Interface – AMI), permite la conexión, configuración y modificación de la centralita. Además, permite el acceso a otros usuarios para que puedan realizar los cambios oportunos.

Es especialmente útil cuándo no hay un acceso directo a la centralita o cuándo por seguridad no se quiere conectar un cliente de forma directa y por precaución prefiere hacerlo mediante el `manager`.

Interfaz de Asterisk, usos y posibilidades:

La AMI es una aplicación que los programas externos pueden usar para la comunicación y control de Asterisk, es decir, permite realizar todo aquello que haría la propia consola de la centralita.

El administrador de programas da la capacidad de ejecutar comandos y solicitar información desde el servidor Asterisk. Sin embargo, no es un mecanismo de autenticación muy seguro, porque usa contraseñas de texto sin formato, y todos los terminales acaban recibiendo todos los eventos. El administrador por lo tanto se debe usar sólo en una red de área local de confianza. Sin embargo existe la posibilidad de cifrarla usando TSL/SSL

Los distintos permisos y denegaciones pueden restringir el acceso a determinadas extensiones o a subredes, dando prioridad de acceso a otras en cuándo a modificación y gestión de la centralita.

El archivo *manager.conf* define la forma en que los distintos usuarios o aplicaciones se autentican con él y los diferentes grados de privilegios asociados. Especificando en cada caso los permisos de lectura y escritura para las distintas opciones de configuración y control de Asterisk.

Una vez definido el archivo de configuración *manager.conf*, es posible realizar y monitorizar llamadas, monitorizar los distintos canales y colas, y ejecutar comandos.

La configuración elegida para el archivo *manager.conf* es la siguiente:

manager.conf

```
[general]
     enabled = yes
     webenabled = yes
     port = 5038
     bindaddr = 0.0.0.0
     allowmultiplelogin = yes
     httptimeout = 60
[manager]
     secret = pass
     deny = 0.0.0.0/0.0.0.0
     permit = 192.168.1.1/255.255.255.0
     permit = 127.0.0.1
     writetimeout = 100
     read =
system,call,log,verbose,agent,user,config,dtmf,reporting,cdr,dialplan
     write = system,call,agent,user,config,command,reporting,originate
```

Lo primero que hará que hacer será habilitar la funcionalidad AMI, la cuál viene definida por defecto con un `no`. Así como también se ha activado la conexión vía web para usar posteriormente las distintas implementaciones gráficas que se detallarán en los próximos apartados.

Se ha usado el puerto de escucha que viene definido por defecto, `5038`. No es recomendable cambiar el puerto usado por defecto. Para conectar con AMI desde un cliente será necesario especificarlo.

También se ha permitido que todas las direcciones del servidor se puedan conectar. Esto permite poder modificar una dirección IP o añadir otras sin ningún tipo de problema.

Se ha definido también el cliente `manager` que, usando la contraseña `pass` podrá conectarse con el manager de Asterisk.

Las posibles direcciones que permiten conectarse han sido restringidas a la del PC1 del banco 1, o a la propia del servidor.

En cuánto a los permisos para enviar y recibir eventos y contactos se ha optado por usar un amplio abanico que permita las máximas posibilidades con la idea de poder manejar la centralita sin problema alguno. Sin embargo, es posible restringir los permisos en caso de que sea necesario.

Los permisos son los siguientes:

- `system`: Información general sobre el sistema y ejecutar comandos básicos como `reload`, `shutdown`, `restart`...
- `call`: Información sobre canales ya existentes
- `verbose`: Información de lo que se ve en la consola (solo eventos)
- `agent`: Información sobre agentes y colas y cómo manipular los estados de estos.
- `user`: Permite enviar y recibir eventos personalizados
- `config`: Permite leer y escribir archivos de configuración de Asterisk
- `command`: Permiten ejecutar comandos de la consola CLI
- `dtmf`: Recepción de tonos DTMF
- `reporting`: Permite obtener información sobre llamadas y sistema
- `cdr`: Información sobre el contenido del CDR tras una llamada
- `dialplan`: Permite recibir las líneas del dialplan que se ejecuten
- `originate`: Permite realizar llamadas

Entre las distintas posibilidades de uso y conexión con la interfaz de Asterisk existen:

- Conexión por comandos de consola usando Telnet
- AJAM (Asynchronous Javascript Asterisk Manager)
- Interfaz gráfica usando Asterisk GUI 2.0

Es posible ayudarse de otros lenguajes como PHP para configurar la conexión con el manager y así poder usar distintas aplicaciones.

Es posible crear una interfaz web propia que solucione las necesidades del cliente. Para ello deberá contar con: gestión de extensiones, aprovisionamiento de teléfonos, gestión de rutas de salida, así como permisos, seguridad...

Usos de comandos de consola mediante Telnet:

Una de las formas más sencillas de conexión con el manager es gracias al uso de Telnet.

Para ello será necesario acceder al puerto definido para el manager `5038` (TCP) y acceder gracias al usuario definido y la contraseña asociada. Según el archivo *manager.conf* se ha definido como cliente `manager` y como contraseña `pass`.

Una vez conectados es posible ejecutar los comandos a los cuales tengamos acceso mediante las opciones `read` y `write` de dicho archivo de configuración.

En este caso tenemos todos los accesos permitidos, así que podemos incluso reiniciar la centralita desde el PC1 del banco 1.

La forma de comunicación dentro del manager es por líneas del tipo clave:valor. Las distintas claves son: `action`, `response` y `event`. En `action` se detallará aquello que desea realizar el cliente; `response` y `event` serán las respuestas dadas por la centralita, ya sea en forma de respuesta concreta al evento o bien en forma de evento general.

Un esquema básico de la conexión mediante telnet sería el siguiente:

```
#telnet 192.168.1.2 5038
>action:login
>username:manager
>secret:pass
>events:on

>response:success
>message:authentication accepted

>...
```

Hay distintas formas de conectarse al manager: mediante PHP programando el acceso, mediante Telnet o simplemente desde la consola de Linux sin usar ninguna aplicación para ello. El único requisito es que se haga mediante TCP y que comience con `login` para acceder.

Entre las muchas acciones que se pueden realizar a modo de ejemplo y comprobación de la configuración se ha optado por las siguientes:

- mostrar una lista de todos los comandos
- mostrar los clientes SIP y desconexión
- reiniciar la centralita

A continuación y de forma gráfica se muestran las acciones realizadas:

Mostrar una lista de comandos gracias a la acción `listcommands`: muestra todos los comandos posibles, la definición y los permisos asociados a cada uno de ellos

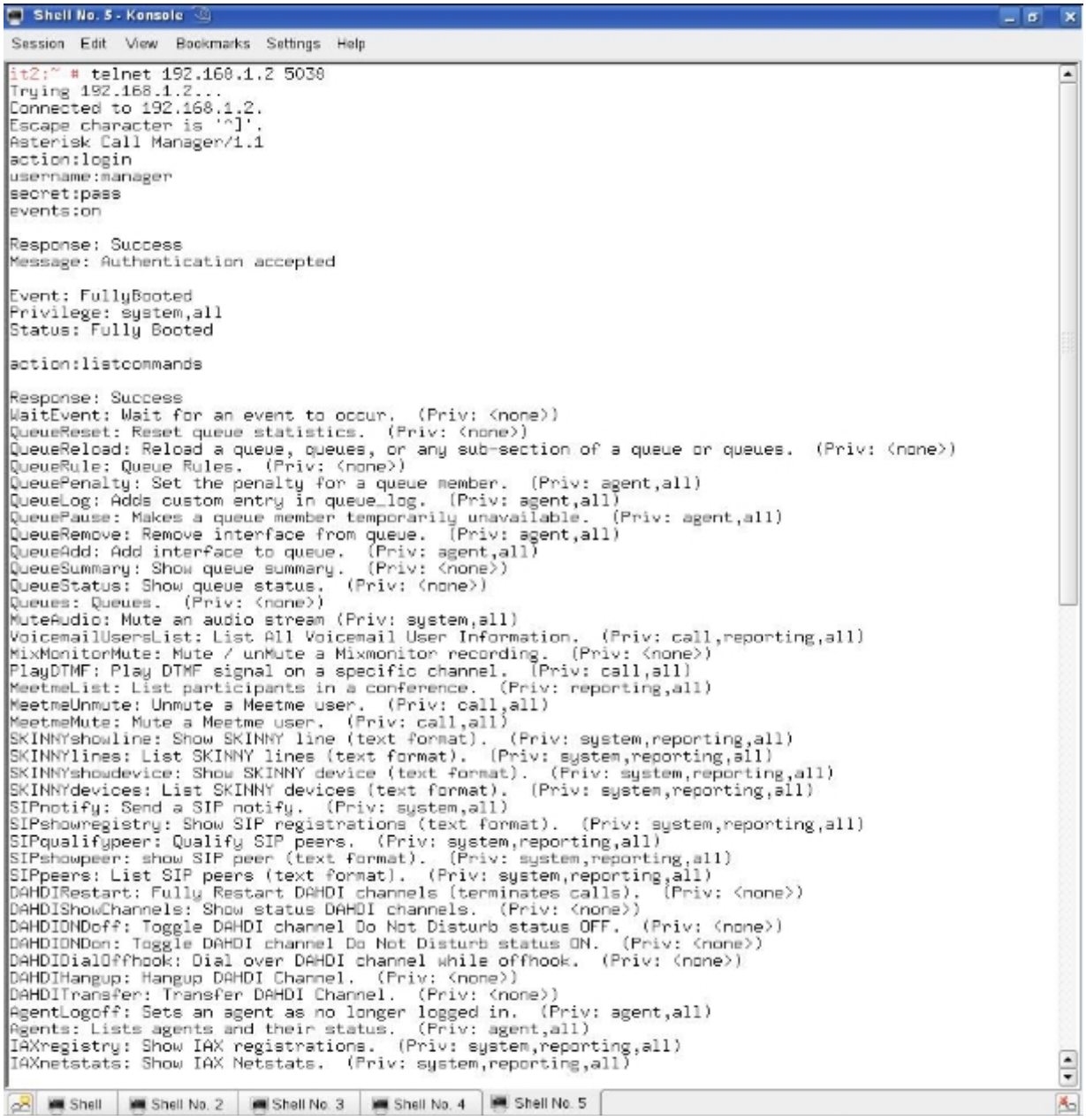


Figura 40: “AMI, listcommands”

Otra de las posibilidades es mostrar por ejemplo los clientes SIP que están definidos en la centralita. Esta acción nos proporcionará el nombre de los clientes, la dirección IP y el estado en el que se encuentran entre varias cosas más. Para ello se usará el comando `sip show peers` como se puede ver en el siguiente ejemplo:

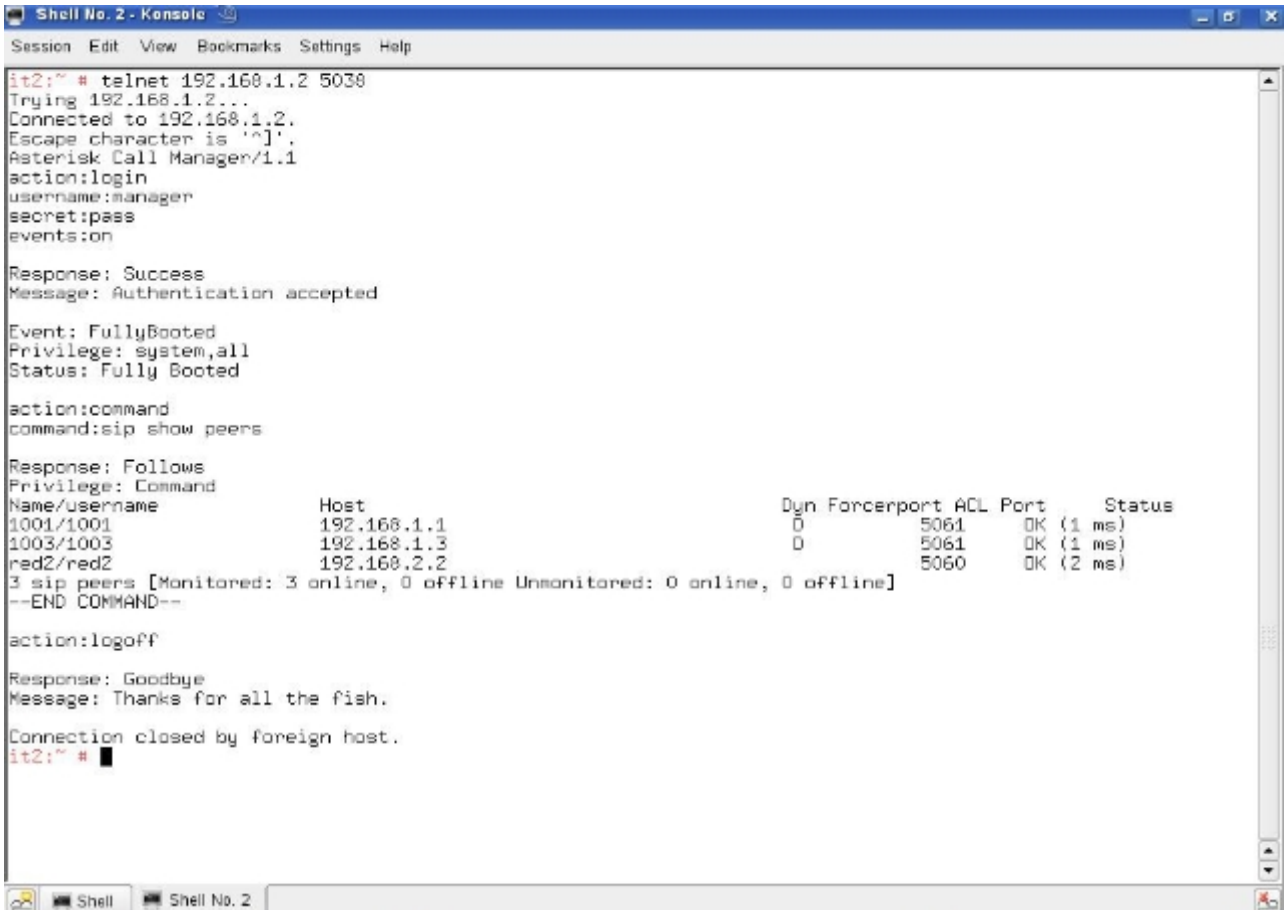


Figura 41: “AMI, comando sip show peers”

El último de los ejemplos por parte del cliente contiene el reinicio de la centralita Asterisk mediante el comando `core restart now`.

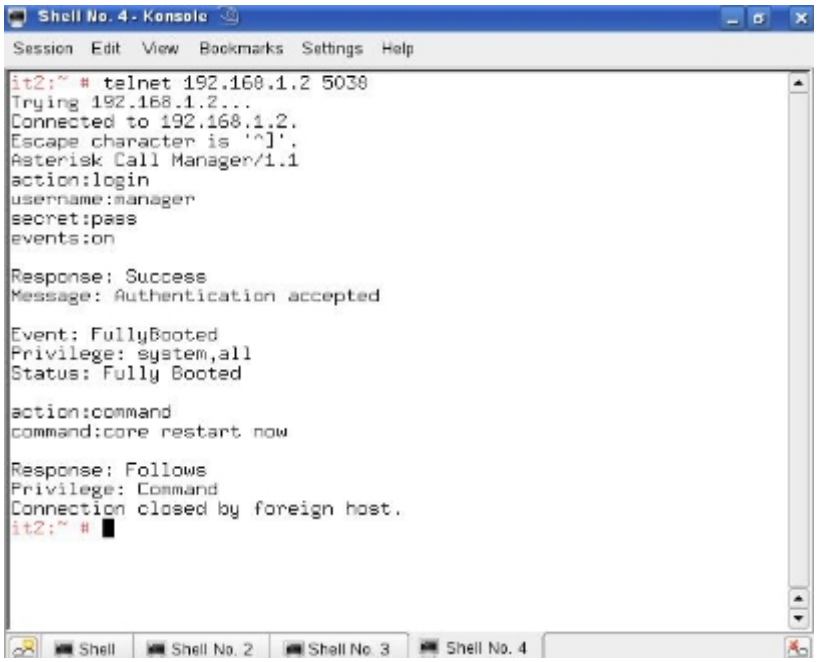


Figura 42: "AMI, comando core restat now"

Mientras tanto, en la línea de comandos del servidor nos encontramos con el siguiente resultado tras ejecutar el reinicio de la centralita:

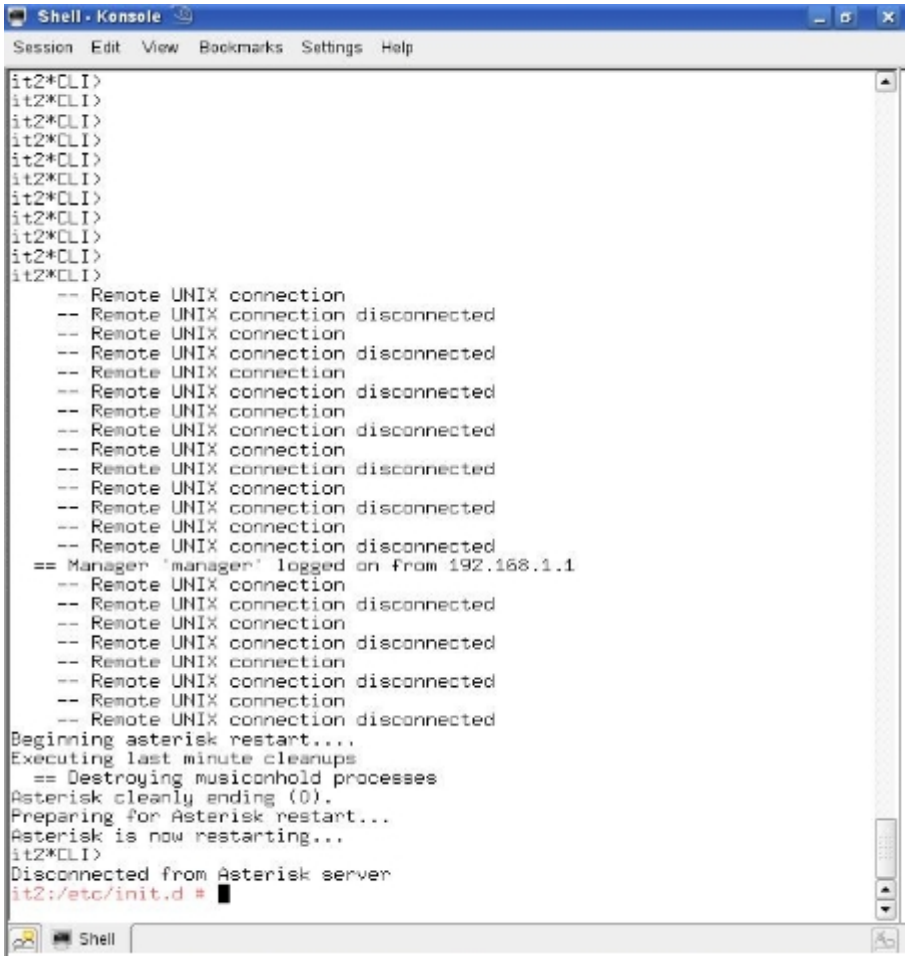


Figura 43: "AMI, resultado en servidor del comando core restart now"

Uso de AJAM (Asynchronous Javascript Asterisk Manager):

AJAM permite usar una interfaz web para manejar el manager, creando y modificando los archivos de configuración.

El objetivo es aumentar la seguridad, evitando tener acceso local al sistema. Utilizando para ello la encriptación TLS/SSL. Por lo tanto se consigue modificar la información directamente desde el manager sin necesidad de acceso directo a los archivos.

Para poder usar AJAM es necesario configurar el servidor HTTP de Asterisk que se ubica en el directorio */etc/asterisk/http.conf*.

La configuración será la siguiente:

http.conf

```
[general]
     enabled = yes
     bindaddr = 0.0.0.0
     bindport = 8088
     prefix = asterisk
     enablestatic = yes
     redirect = / /static/config/index.html
```

Es necesario activar el servidor para permitir el acceso a AJAM.

Se ha optado por permitir el acceso a todas las direcciones IP que quieran conectarse al servidor y se ha configurado el puerto `8088` para tal fin.

Se ha usado el prefijo por defecto, sin embargo es posible cambiarlo a otro que sea más apropiado teniendo en cuenta que éste será el inicio de la dirección que necesita el servidor para encontrar el documento HTML.

Además, habrá que asegurarse que estén activas `enabled` y `webenabled` en el archivo *manager.conf*. Así como introducir un tiempo de finalización de la conexión HTTP en `httptimeout`.

Es imprescindible también que exista el cliente y la contraseña que permiten el acceso al manager.

Una vez finalizada la configuración habrá que comprobar el funcionamiento. Para ello se abrirá un navegador desde el PC1 del banco 1 (192.168.1.1) ya que es el PC que tiene permitido el acceso a la interfaz de Asterisk y escribiremos:

`http://192.168.1.2:8088/asterisk/manager?action=login&username=manager&secret=pass`

o bien:

`http://192.168.1.2:8088/asterisk/rawman?action=status`

Aparecerá una interfaz gráfica simple que permitirá el acceso a distintas acciones, como mostrar una lista de comandos disponibles o ejecutar un comando cualquiera.

Una vez conectados a la interfaz web, se ha mostrado una lista de comandos para saber las posibles acciones que se pueden realizar.

A continuación se muestra el resultado del comando `list commands`:

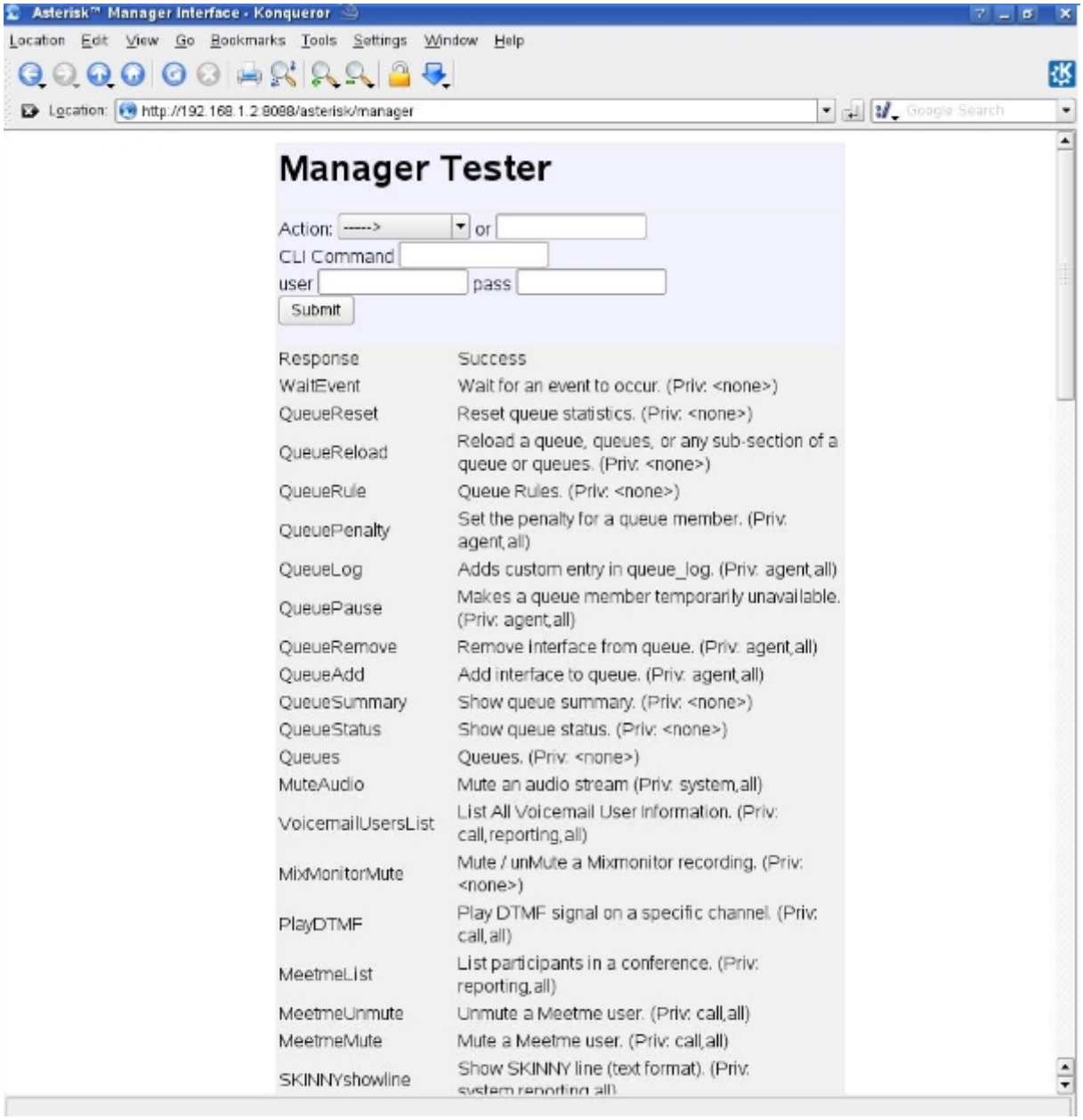


Figura 44: "AJAM, comando listcommands"

También se ha usado el comando `sip show peers` en el CLI dando como resultado los clientes SIP que están definidos en la configuración de Asterisk, así como sus direcciones IP asociadas y el estado en el que se encuentran.

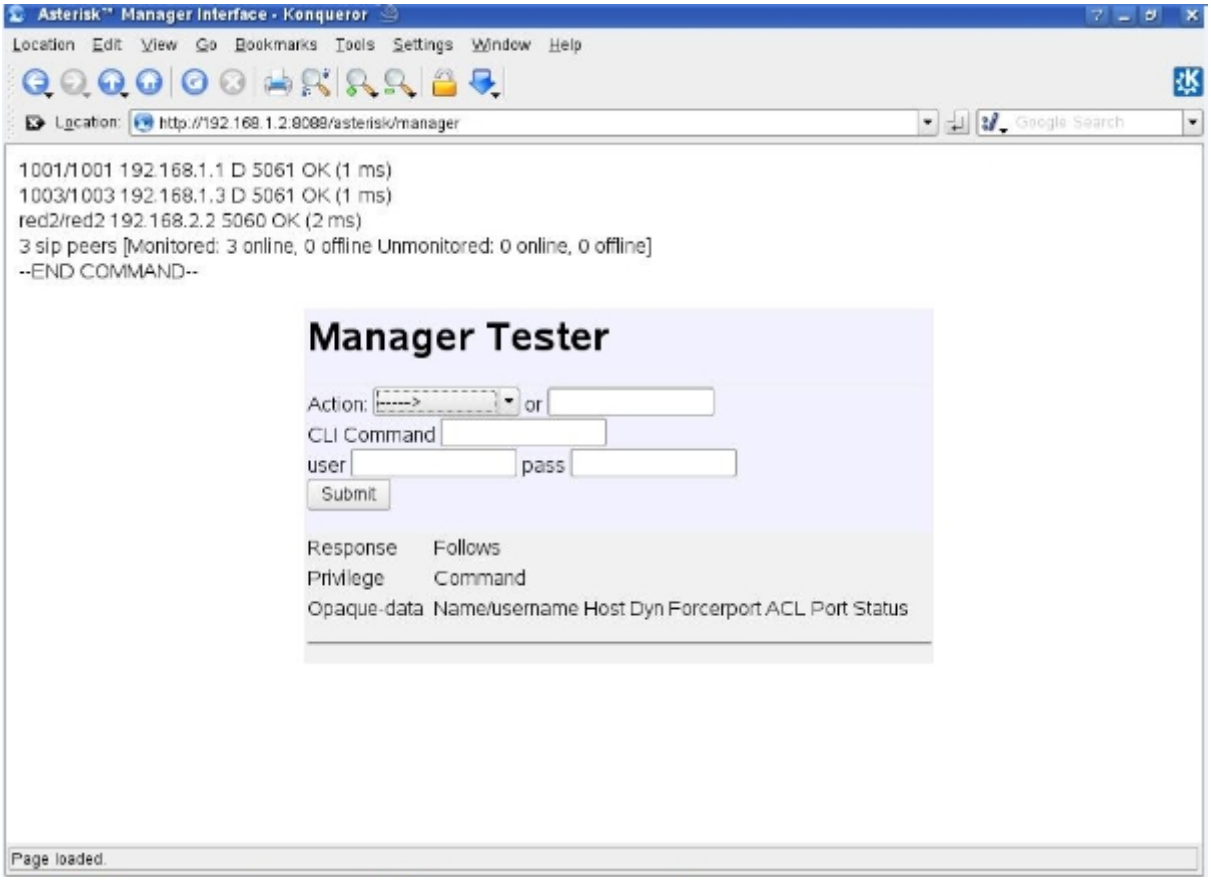


Figura 45: "AJAM, comando sip show peers"

Uso de la interfaz gráfica Asterisk GUI 2.0:

Otra de las posibles opciones a la hora de conectar con el manager es usar la interfaz gráfica Asterisk GUI 2.0.

El principal problema de la interfaz es que no permite el acceso a la configuración previa de Asterisk, así que descartamos su uso por ser tan restringido.

Aún así se ha estimado oportuno hacer una prueba para ver la interfaz y el funcionamiento.

Para descargar e instalar la interfaz seguiremos el siguiente procedimiento:

```
#svn co http://svn.asterisk.org/svn/asterisk-gui/branches/2.0
#cd 2.0/
#./configure
#make
#make install
#cd /usr/src/2.0
#make checkconfig
```

Una vez realizado el procedimiento queda abrir el navegador y escribir la siguiente dirección:

`http://192.168.1.2:8088/asterisk/static/config/cfgbasic.html`

En caso de escribir `http://192.168.1.2:8088` nos redireccionará de forma automática a `http://192.168.1.2:8088/asterisk/static/config/index.html`

A continuación se muestra la interfaz de inicio al acceder al servidor de una forma gráfica a través de Asterisk GUI 2.0

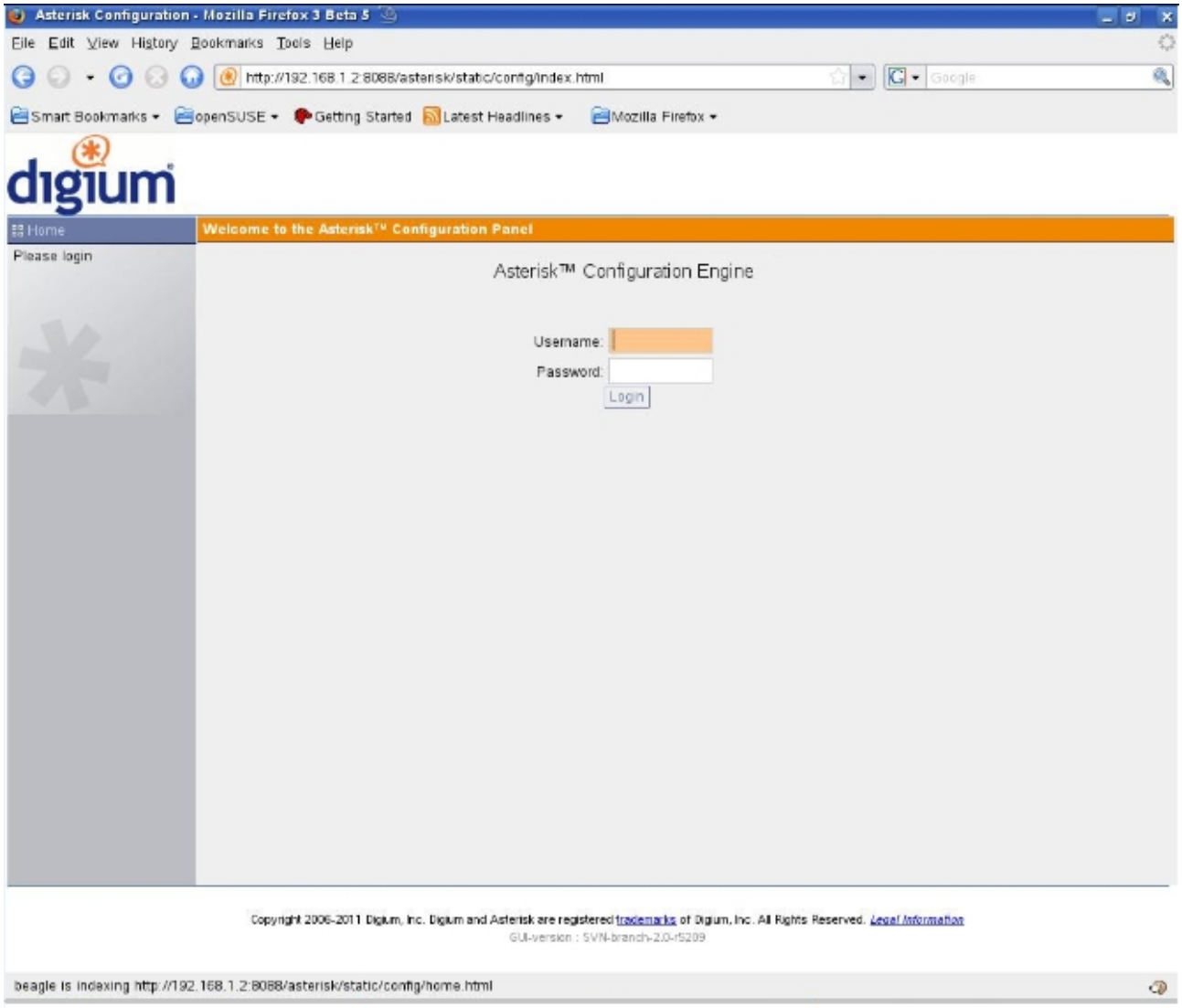


Figura 46: “GUI, inicio”

También es posible ver el estado del sistema y realizar configuraciones básicas, sin embargo es imposible acceder a una configuración previamente definida.

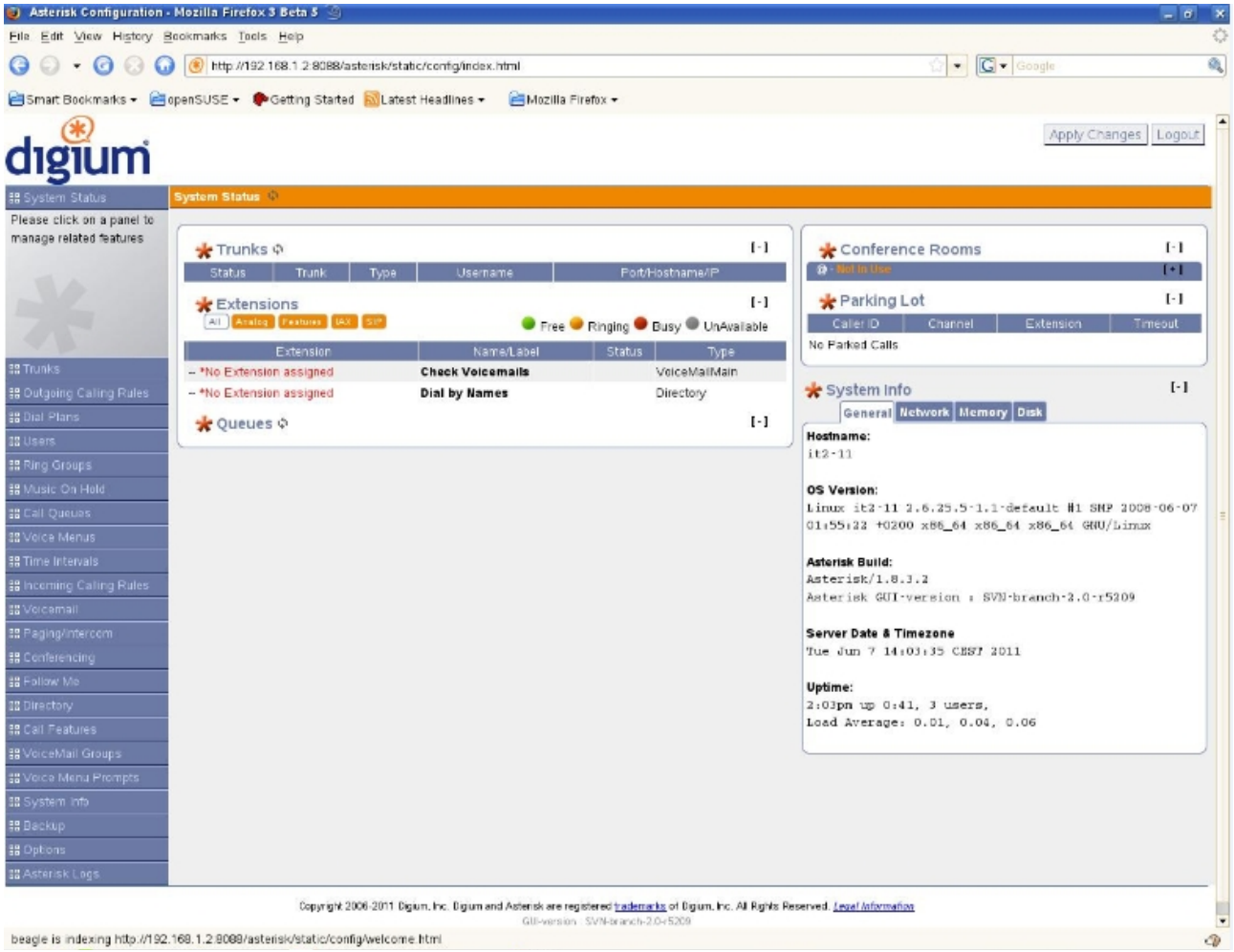


Figura 47: “GUI, estado del sistema”

5.10 CALL DETAIL RECORD (CDR)

Objetivos:

– Uso y configuración de CDR

Metodología:

El sistema de detalle de llamadas (Call Detail Records – CDR) controla todas las llamadas realizadas o recibidas en el sistema.

Cabe destacar su gran labor en cuanto a control de la facturación y evaluación de la calidad del servicio, por esto se ha decidido hacer uso del sistema.

Aunque existen herramientas en el mercado que de una forma gráfica expresan el control de las llamadas es imprescindible saber cómo funciona realmente a nivel de archivo de configuración. Así pues se ha optado por un desarrollo básico con el objetivo de instalar y comprobar la funcionalidad del sistema.

La configuración necesitará de tres archivos para funcionar correctamente: *cdr.conf*, *cdr_manager.conf* y *cdr_custom.conf*. Actuando los tres en conjunto darán como resultado un archivo con toda la información referente a las llamadas del sistema, llamado *Master.csv* y ubicado en */cygroot/asterisk/log/cdr-csv/.*

Existen ampliaciones al CDR básico que se ha usado en este caso de estudio y que permiten almacenar los registros en una base de datos MySQL, configurando para ello el archivo *cdr_mysql.conf.*

La configuración del archivo *cdr.conf* es la siguiente:

cdr.conf:

```
[general]
     enable = yes
     unanswered = no
     endbeforehexten = no
     initiateseconds = no
     batch = no
     size = 100
     time = 300
     scheduleronly = no
     safeshutdown = yes
```

```
[csv]
     usegmtime = no
     loguniqueid = no
     loguserfield = no
     accountlogs = yes
```

Una vez habilitado el registro de llamadas se procede a configurar el resto de las opciones del archivo, como que se cierre el CDR después de que las extensiones hayan terminado su ejecución por completo (`endbeforehexten`).

Además se ha optado por no usar un buffer de almacenamiento temporal en la escritura del CDR, por tanto se evitan problemas de que se almacene un registro y se pueda perder por un fallo del sistema. De esta forma se realiza instantáneamente.

También es importante definir que Asterisk no se cierre hasta que no se hayan volcado por completo los datos en el CDR

Para que los detalles de las llamadas queden registrados en el archivo *Master.csv* es necesario definir la categoría `[csv]` y el apartado `accountlogs`.

Para que los eventos del manager del Asterisk queden reflejados en el CDR es necesario activar el archivo *cdr_manager.conf* de la siguiente forma:

cdr_manager.conf:

```
[general]
     enabled = yes
```

Cuándo estamos conectados a la interfaz manager, y una llamada finaliza, un evento será enviado a todo lo que haya conectado a la interfaz.

También es necesario que el archivo *cdr_custom.conf* esté presente y contenga en la sección `[mappings]` los campos de salida y el orden de éstos para que aparezcan reflejados en *Master.csv.*

cdr_custom.conf:

```
[mappings]
     Master.csv => ${CSV_QUOTE(${CDR(did)})}, ${CSV_QUOTE(${CDR(src)})},
${CSV_QUOTE(${CDR(dst)})}, ${CSV_QUOTE(${CDR(dcontext)})},
${CSV_QUOTE(${CDR(channel)})}, ${CSV_QUOTE(${CDR(dstchannel)})},
${CSV_QUOTE(${CDR(lastapp)})}, ${CSV_QUOTE(${CDR(lastdata)})},
${CSV_QUOTE(${CDR(start)})}, ${CSV_QUOTE(${CDR(answer)})},
${CSV_QUOTE(${CDR(end)})}, ${CSV_QUOTE(${CDR(duration)})},
${CSV_QUOTE(${CDR(billsec)})}, ${CSV_QUOTE(${CDR(disposition)})},
${CSV_QUOTE(${CDR(amaflags)})}, ${CSV_QUOTE(${CDR(accountcode)})},
```

```
${CSV_QUOTE(${CDR(uniqueid)})}, ${CSV_QUOTE(${CDR(userfield)})},
${CDR(sequence)}
```

Esta configuración contiene todos los campos posibles en el control de llamadas.

El orden de aparición en *Master.csv* será: ID del llamante, extensión llamante, extensión destino, contexto destino, canal origen, canal destino, última aplicación usada, parámetros de la última aplicación, fecha y hora de inicio de la llamada, fecha y hora en que ha sido contestada, fecha y hora en que ha terminado, duración total, duración total atendida, estado de la llamada, banderas de control de CDR, código de la cuenta, identificador único de llamada, y activado en el plan de marcación.

A continuación se muestra una captura de pantalla del archivo *Master.csv* perteneciente a la red 1:

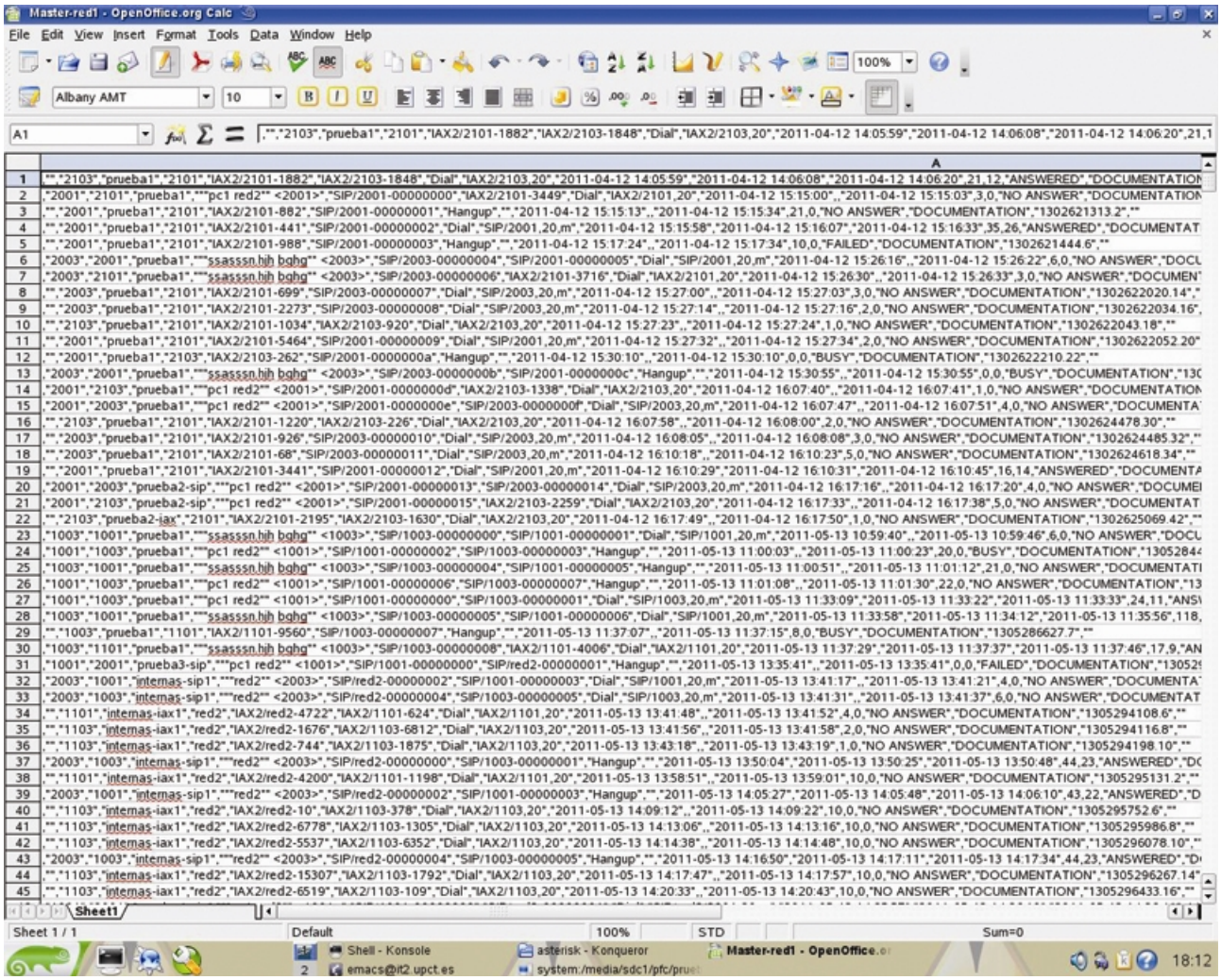


Figura 48: "Archivo Master.csv"

5.11 DUNDI SIMPLE

Objetivos:

- Definir y configurar una intercomunicación básica entre redes usando DUNDi.

Metodología:

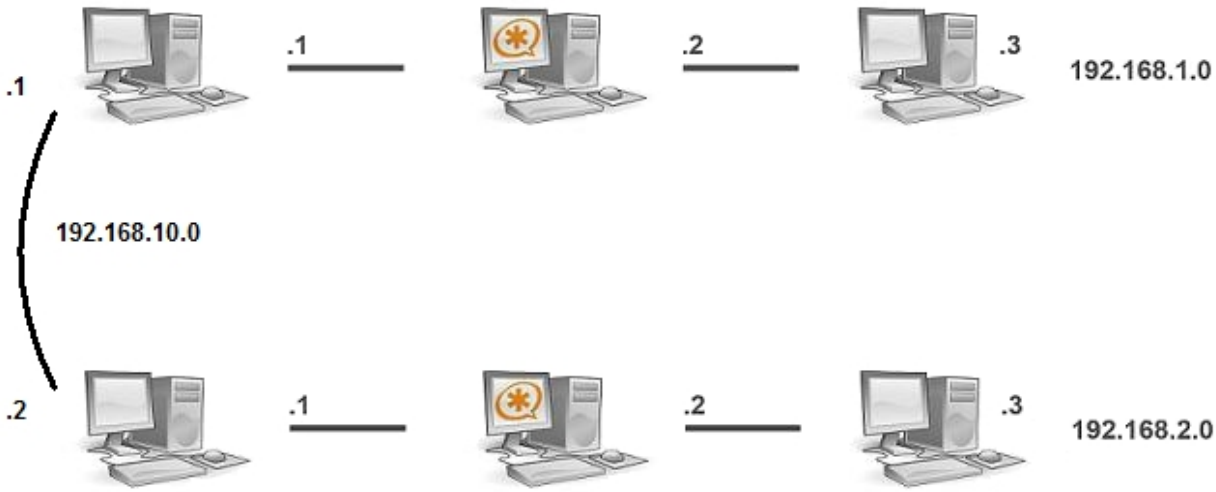


Figura 49: "Configuración de DUNDi simple"

Como se ha explicado anteriormente, el protocolo DUNDi permite buscar y compartir planes de marcación entre distintos servidores. Para ello hace uso de la publicación de las rutas deseadas por parte de cada servidor.

Éste sistema de compartición de información hace de DUNDi una implementación sin puntos de fallo centralizados, y permite además la inclusión de nuevas extensiones sólo con modificar una línea, aquella que indica lo que se desea compartir.

El principal objetivo de éste apartado es realizar una interconexión entre dos redes de una forma básica utilizando el protocolo DUNDi.

Para ello se parte de la base de la configuración de red original, que consta de dos redes gestionadas por dos servidores con Asterisk instalado y funcionando.

Cada red contendrá cuatro extensiones, dos SIP y dos IAX. Las cuáles deberán ser compartidas con el servidor de la red contraria gracias a DUNDi.

Para intercomunicar ambos servidores se ha descartado el uso del cliente `[red2]` o `[red1]` que hasta ahora se ha estado utilizando. En su lugar se ha preferido usar un enlace y un recurso que permitan el intercambio de información sin necesidad de apuntar a la red contraria y especificar cómo acceder a cada una de las extensiones requeridas.

La diferencia principal es la simplicidad y la disminución de código, que ahora permitirá publicar las extensiones que se deseen compartir y por otra parte permitirá una búsqueda de una extensión solicitada entre las publicadas por los distintos servidores.

Es necesario realizar un esquema con la distribución lógica de las redes y la forma de compartir la información, así como también modificar los archivos de configuración: *iax.conf*, *extensions.conf* y *dundi.conf*, además es recomendable usar unas llaves de autenticación que cifren la información que se envía entre los servidores.

Esquema lógico de distribución:

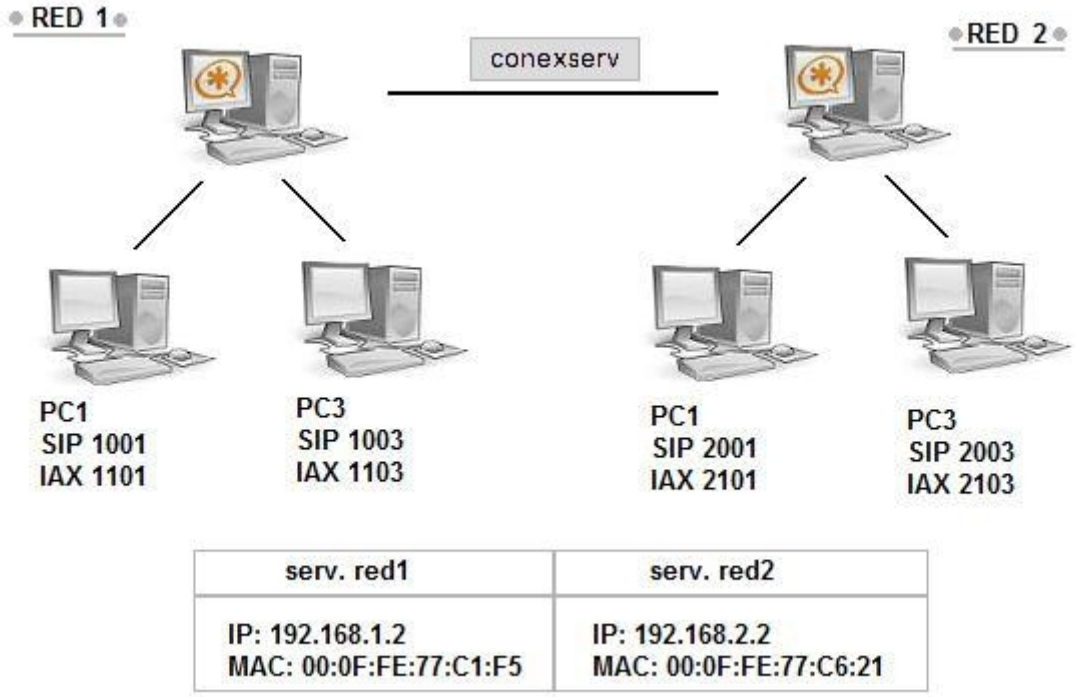


Figura 50: "Configuración lógica de las redes 1 y 2 con DUNDi"

Plan de marcación:

Para este caso, en el que se comparten las extensiones SIP e IAX, se ha usado el recurso DUNDi `conexserv`, el cuál permite la publicación y la búsqueda de extensiones entre la red 1 y la red 2. Dicho recurso estará definido en el archivo *dundi.conf*.

sip.conf:

Este contexto contiene los clientes SIP pertenecientes a la red 1, 1001 y 1003.

sip.conf:

```
[general]
     context = default
     port = 5060
     bindaddr = 0.0.0.0
     disallow = all
     allow = gsm
     allow = alaw
     allow = ulaw
[1001]
     type = friend
     username = 1001
     secret = pass
     host = dynamic
     callerid = 1001
     context = prueba11
     qualify = yes
[1003]
     type = friend
     username = 1003
     secret = pass
     host = dynamic
     callerid = 1003
     context = prueba11
     qualify = yes
```

iax.conf:

En el archivo *iax.conf,* aparte de las extensiones correspondientes, se creará un enlace que permitirá la comunicación entre ambos servidores.

Especificará, entre otras cosas el contexto dónde serán redirigidas las llamadas que entren desde el servidor de la red contraria.

iax.conf:

```
[general]
    …
[1101]
    …
[1103]
    …
[enlace]
    type = friend
    dbsecret = dundi/secret
    context = entrantesdundi
```

Se ha definido el cliente `enlace`, usado para el intercambio de información de forma tal que cualquier petición desde la red 2 entrará al contexto `[entrantesdundi]` definido en el archivo *extensions.conf*.

A su vez se ha optado por utilizar una contraseña que cifre la información entre servidores.

Para la red 2, la configuración del archivo *iax.conf* será igual, contendrá la definición de las extensiones y el cliente `enlace` que permitirá que las extensiones de la red contraria puedan acceder al servidor. El contexto asociado será también `[entrantesdundi]`.

Llaves de autenticación:

Será necesario crear unas llaves de autenticación que permitan el acceso a los demás servidores por medio del enlace troncal IAX.

Se crearán en el directorio */var/lib/asterisk/keys/* con el siguiente comando:

```
#astgenkey -n llave
```

Dando lugar a los archivo *llave.pub* y *llave.key* que se copiarán al servidor de la red contraria en la misma ubicación.

El archivo *llave.pub* podría ser el siguiente:

llave.pub

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDd/HnB5Zn31XEx0HsypSXPCEOT
RnIGqAe7t8bqp0pX6k3nxrysz0cbTB23yJB3kN2DH8TWTIEROM0FMnMfY+WCUHss
gNMxP5DxWtD5dAVUdqnoKpJ5ETIjb8l5zpzaK/GTEi0tNvNzYXePCe8bevmnCQ4O
jCVxMdj2SD/ygodIiQIDAQAB
-----END PUBLIC KEY-----
```

extensions.conf:

El archivo de configuración del plan de marcación es el que más cambios sufre, puesto que habrá que reajustar los contextos para plantear las distintas necesidades que requiere DUNDi en cuánto a búsqueda y compartición de extensiones.

Se dividirá en los siguientes contextos:

- `[general]`: define las características comunes del plan de marcación
- `[busquedadundi]`: contexto que inicia la búsqueda de extensiones en el otro servidor.
- `[compartidas]`: especifica las extensiones que se desean compartir con el otro servidor
- `[entrantesdundi]`: contexto al que llegan las llamadas provenientes de otros servidores.
- `[prueba11]`: contexto que define todas las extensiones propias del servidor.

extensions.conf

```
[general]
     static = yes
     writeprotect = no
     autofallthrough = yes
     clearglobalvars = no

[busquedadundi]
     switch => DUNDi/conexserv

[compartidas]
     exten => 1001,1,NoOp
     exten => 1003,1,NoOp
     exten => 1101,1,NoOp
     exten => 1103,1,NoOp

[entrantesdundi]
     exten => _1XXX,1,Goto(prueba11,${EXTEN},1)

[prueba11]
     include => busquedadundi
```

```
exten => 1001,1,Dial(SIP/1001,20,m)
exten => 1001,2,Hangup

exten => 1003,1,Dial(SIP/1003,20,m)
exten => 1003,2,Hangup

exten => 1101,1,Dial(IAX2/1101,20)
exten => 1101,2,Hangup

exten => 1103,1,Dial(IAX2/1103,20)
exten => 1103,2,Hangup
```

Sólo serán compartidas las extensiones de las que se quiere que la otra red tenga conocimiento.

Por lo tanto existirá una restricción en cuánto a las extensiones que se van a publicar y no se limita el acceso a la búsqueda de extensiones por parte de las otras redes en el contexto `[entrantesdundi]`.

Si se compartieran las extensiones de la forma: `exten => _1XXX,1,NoOp`, todas las llamadas en la red 2 que comiencen por `1XXX` serán enrutadas a la red 1 y en ésta se verá que no están definidas, dando lugar a un error, por lo tanto es más conveniente compartir las extensiones concretas.

A su vez, el servidor de la red 2 publicará en el contexto `[compartidas]` las extensiones `2001`, `2003`, `2101` y `2103` para que sean accesibles desde cualquier extensión de la red 1.

El recurso DUNDi utilizado para la comunicación será `conexserv`.

dundi.conf

Este archivo describirá la forma de comunicación entre ambos servidores, en función del recurso DUNDi asociado a cada cual, dónde se describirán los contextos en los que se publican las extensiones, el protocolo y el cliente que hace de nexo de unión para realizar las búsquedas necesarias en la otra red.

dundi.conf:

```
[general]
     department = proyectoVoIP
     organization = UPCT
     locality = Cartagena
     stateprov = Murcia
     country = España
     email = your@email.com
     phone = +0000000

     bindaddr = 0.0.0.0
     port = 4520
     entityid = 00:0F:FE:77:C1:F5
     cachetime = 30
     ttl = 5
     autokill = yes
     secretpath = dundi

[mappings]
     conexserv    =>    compartidas,0,IAX2,enlace:${SECRET}@192.168.1.2/${NUMBER},nopartial

[00:0F:FE:77:C6:21]
     model = symmetric
     host = 192.168.2.2
     inkey = llave
     outkey = llave
     include = all
     permit = all
     order = primary
```

En éste caso se ha permitido que cualquier dirección IP se pueda conectar con el servicio DUNDi. El puerto definido usado es el puerto por defecto, el `4520`.

Se definirá también un identificador del servidor al que pertenece, en éste caso se ha usado la dirección MAC del servidor de la red 1.

En el contexto `[mappings]` describe el comportamiento del recurso DUNDi `conexserv` que se va a compartir con los demás nodos de la red.

El contexto del plan de marcación que contiene las rutas a compartir es `compartidas`.

El peso de la ruta `0`, por lo tanto se consideran las extensiones como locales.

El protocolo usado para la comunicación es `IAX2`.

El enlace usado para la comunicación entre los servidores y definido en *iax.conf* es `enlace`.

`${SECRET}` contendrá la clave del recurso DUNDi que se usa para utenticar el enlace troncal.

Será necesario además incluir la dirección IP del servidor de la red 1, así como opcionalmente la variable que contiene la extensión solicitada, `${NUMBER}`, y la prevención de resultados que contengan una parte de la extensión buscada.

El último contexto comienza con la dirección MAC como identificador del servidor de la red 2, indicando la dirección IP de dicho servidor y los nombres de las llaves usadas para recibir y realizar la consulta.

También se define qué se quiere compartir con el servidor, en éste caso se ha optado por no restringir las opciones. Además se permiten todos los contextos a los que tendrá acceso este nodo.

Se especifica que el nodo puede realizar y recibir consultas, y se controla que la conexión esté activa.

En el caso de la red 2, el archivo de configuración será prácticamente igual, en `[general]` se especificará la dirección MAC del servidor de la red 2, y en `[mappings]` el recurso y la forma de comunicarse con el servidor de la red 1, para ello se hará uso del contexto `compartidas` definido en *extensions.conf*, se usará el cliente `enlace` definido en *iax.conf*, y además se usará la dirección IP de dicho servidor, la `192.168.2.2`

Por último, el contexto al que hace referencia la dirección MAC del servidor de la red 1 contendrá la dirección IP del servidor de esa red y el nombre de las llaves usadas para el envío de información y los permisos correspondientes.

Una de las pruebas realizadas ha sido una llamada desde una extensión de la red 2 a la extensión `1103` de la red 1.

La siguiente captura de pantalla muestra el intercambio de información desde el servidor de la red 1:

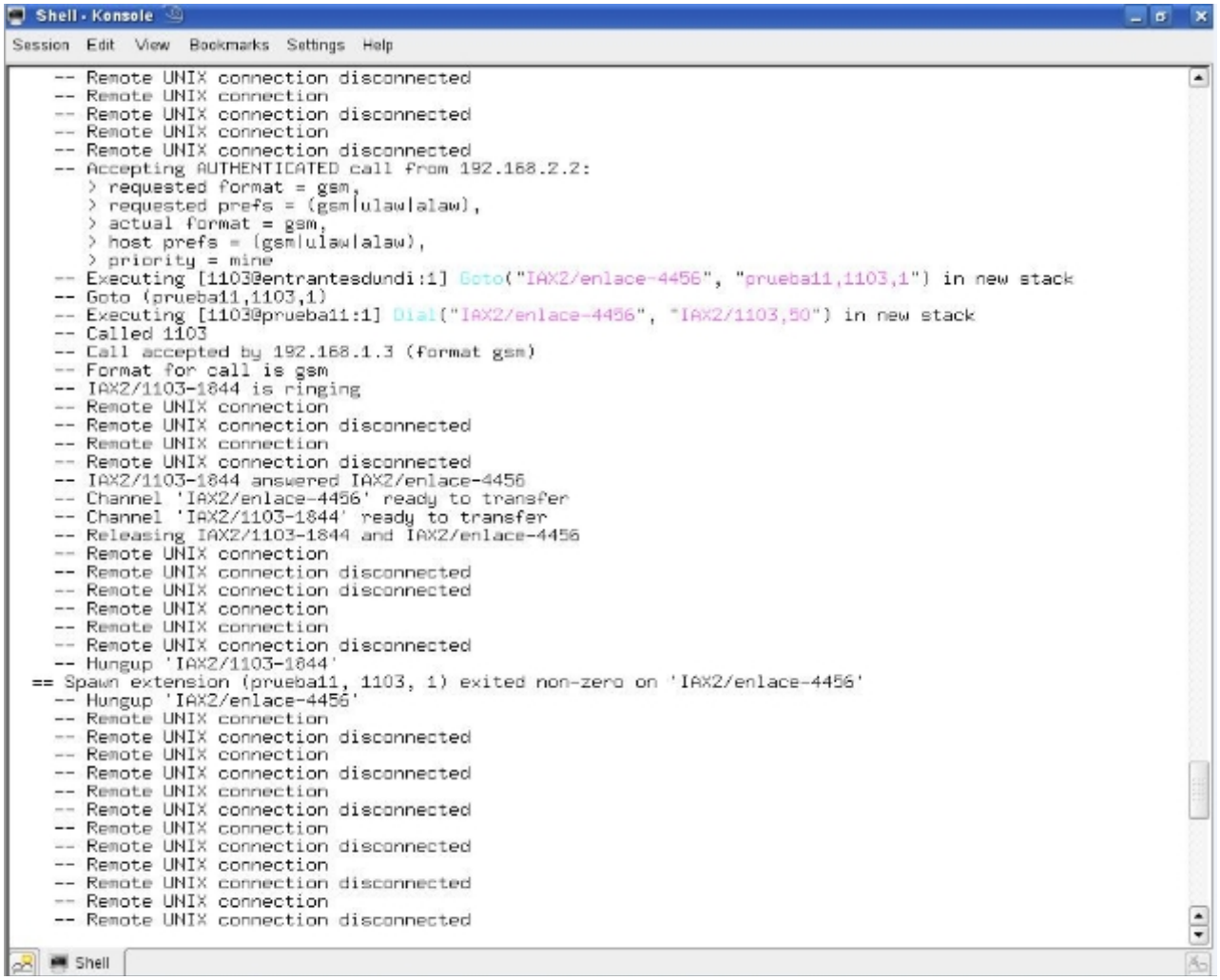


Figura 51: “Llamada cursada de la red 2 a la red 1, vista en servidor 1”

5.12 USO DE DUNDI

Objetivos:

- Hacer uso de los callfiles dentro de una configuración basada en DUNDi

Metodología:

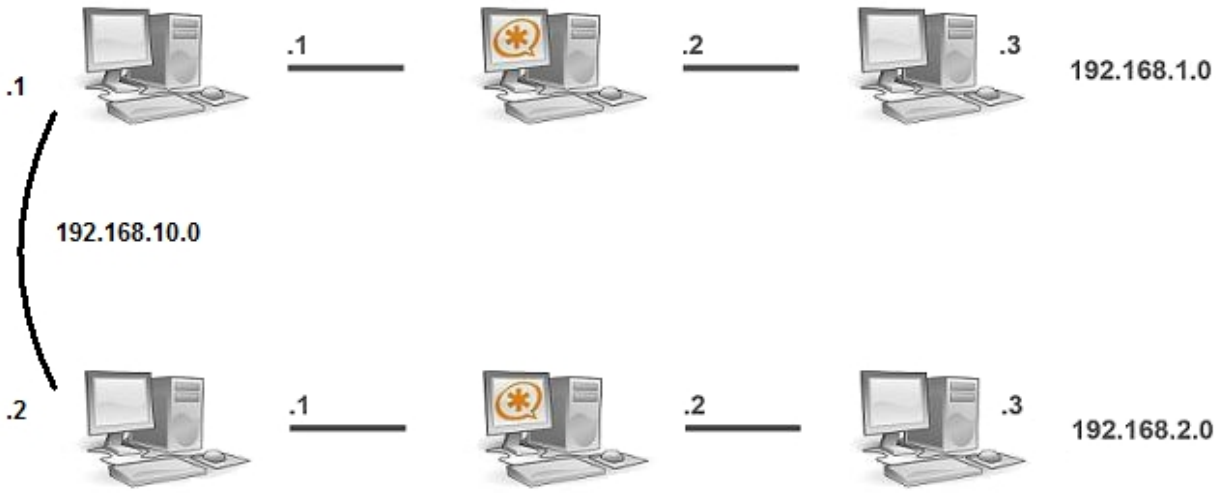


Figura 52: "Configuración final con DUNDi"

El último de los casos de estudio se basa en la configuración completa del servidor y todas sus posibilidades, usando DUNDi, para con ello permitir la posible expansión de las redes, mejorando a su vez el código y la configuración.

Se definirá por tanto un servidor con las implementaciones definidas en los contextos `[meetme]`, `[llamadaIVR]`, `[grabaciónAudio]`, `[cita]`, `[emergencia]`, y `[subscripcion]`.

Incluyendo además la configuración propia usada para implementar DUNDi, dividiendo los contextos iniciales del plan de marcación en: `[general]`, `[busquedadundi]`, `[compartidas]`, `[entrantesdundi]` y `[prueba12]`.

El resto de la configuración relativa a DUNDi, como la creación de un enlace troncal en *iax.conf*, la creación de llaves y la configuración de *dundi.conf*, quedará como en el caso del estudio anterior.

Esquema lógico de distribución:

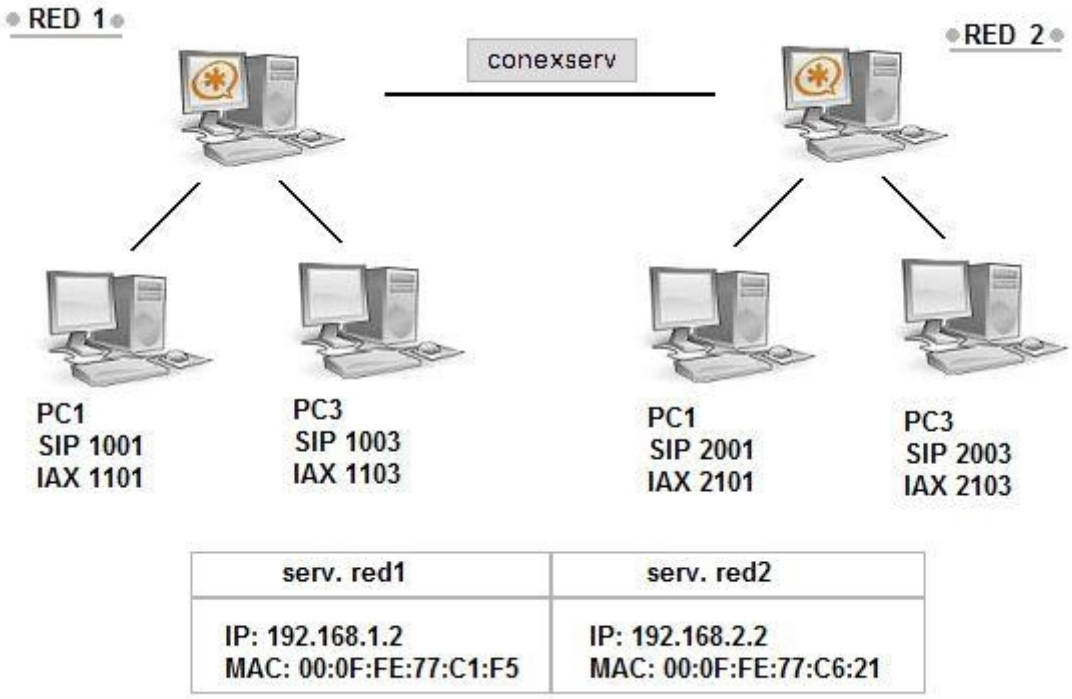


Figura 53: "Configuración lógica con DUNDi, final"

En éste esquema se muestra la conexión lógica de los servidores de la red 1 y la red 2, así como la forma de comunicación gracias al recurso DUNDi `conexserv`, y dónde se especifican las extensiones existentes en cada red.

Además es imprescindible dejar constancia de las direcciones IP y MAC de cada servidor, para posteriormente hacer uso de ellas en el archivo *dundi.conf*.

Plan de marcación:

Aparte de las extensiones SIP e IAX existen otras que dan acceso a los distintos servicios que conforman el plan de marcación del servidor correspondiente, como pueden ser el acceso a `meetme`, o al `IVR`.

Por lo tanto, es imprescindible compartir aquellas extensiones a las que se quiera dar acceso al otro servidor.

En el caso del servidor de la red 1, el número de extensiones a compartir será mayor, puesto que en éste servidor estará definido el código que permita el acceso tanto a `meetme` como al `IVR`. Por lo tanto, el cualquier extensión de la red 2 que quiera hacer uso de estos servicios tendrá por fuerza que dirigirse a la red 1 para llevarlas a cabo. Así pues, estas extensiones han de ser compartidas.

Además, como solución a la problemática de la gestión de archivos callfile dentro de DUNDi se ha optado por no hacer una copia de seguridad en otro servidor de cada una de las extensiones que se subscriban a las llamadas de emergencia. La solución a ésto ha sido realizar una copia del mensaje de emergencia al otro servidor una vez haya sido grabado y el volcado de los callfiles correspondientes en cada servidor.

Para ello se ha hecho uso de una extensión extra que permita mover los archivos callfile al directorio correspondiente para que se ejecuten. La extensión usada para tal fin es, `1122` para llamar a la red 2, y `1121` para permitir que la red 2 llame a la red 1.

Extensiones usadas en los servidores y acción asociada:

- Servidor de la red 1:

1001 y ***1003***	Extensiones SIP ***(compartir)***
1101 y ***1103***	Extensiones IAX ***(compartir)***
`2999` y `3999`	Acceso al buzón de voz, SIP e IAX respectivamente
1234	Aplicación Meetme ***(compartir)***
888	Respuesta de voz interactiva IVR ***(compartir)***
`66X`	Grabación de audio
`4567`	Cita telefónica
`112`	Emergencia
1121	Volcado de callfiles en caso de emergencia ***(compartir)***
`555`	Subscripción al servicio de emergencia

- Servidor de la red 2:

2001 y ***2003***	Extensiones SIP ***(compartir)***
2101 y ***2103***	Extensiones IAX ***(compartir)***
`2999` y `3999`	Acceso al buzón de voz, SIP e IAX respectivamente
`1234`	Salto a la red 1 para: aplicación Meetme
`888`	Salto a la red 1 para: Respuesta de voz interactiva IVR
`66X`	Grabación de audio
`4567`	Cita telefónica
`112`	Emergencia
1122	Volcado de callfiles en caso de emergencia ***(compartir)***
`555`	Subscripción al servicio de emergencia

sip.conf:

Éste archivo de configuración que define las extensiones SIP, no sufrirá modificación alguna, quedando por tanto definidas únicamente las extensiones `1001` y `1003` en la red 1. Para la red 2, la configuración del archivo contendrá a las extensiones `2001` y `2003`, cuyo contexto será `[prueba12]`, y la contraseña asociada a cada extensión: `pass`

iax.conf:

En éste archivo de configuración, además de las extensiones correspondientes, `1101` y `1103` para la red 1, ó `2101` y `2103` para la red 2, se definirá un enlace troncal que permita la comunicación de los servidores gracias a DUNDi.

iax.conf de red 1:

```
[general]
    …
[1101]
    …
[1103]
    …
[enlace]
    type = friend
    dbsecret = dundi/secret
    context = entrantesdundi
```

Dicho cliente tendrá como contexto asociado `[entrantesdundi]` definido en el archivo *extensions.conf*, para permitir la entrada de cualquier solicitud de información de la otra red.

Llaves de autenticación:

Serán necesarias para permitir el acceso al otro servidor a través del enlace troncal IAX definido previamente.

Se crearán en */var/lib/asterisk/keys/* con el comando: `#astgenkey -n llave`, dando lugar a los archivos *llave.pub* y *llave.key*, que se copiarán en el servidor de la red contraria para que comparta las mismas claves de autenticación.

extensions.conf:

El archivo de configuración *extensions.conf* contiene el plan de marcación del servidor, define el uso de las extensiones y configura las distintas posibilidades que ofrece.

Está dividido en los siguiente contextos:

- `[general]:` define las características comunes para todo el plan de marcación
- `[busquedadundi]:` permite buscar las extensiones requeridas en el otro servidor.
- `[compartidas]:` publica las extensiones que se desean compartir con el servidor de la red contraria.
- `[entrantesdundi]:` contexto de entrada para las llamadas que llegan desde la otra redirigidas.
- `[prueba12]:` contexto que define las extensiones propias de la red.
- `[meetme]:` define la forma de uso de las salas de conferencias. Definido en la red 1.
- `[llamadaIVR]:` contexto que gestiona la respuesta de voz interactiva. Definido en la red 1.
- `[grabacionAudio]:` describe la forma de grabar y guardar en el servidor un archivo de audio.
- `[cita]:` permite guardar y gestionar una cita telefónica con una extensión gracias a los archivos callfile.
- `[emergencia]:` grabación de un archivo de audio por parte de un cliente, copia en el servidor de la red contraria y ejecución de todos los archivos de subscripción de ambas redes mediante llamada a una extensión concreta.
- `[subscripcion]:` subscripción al servicio de emergencia y creación de un callfile que ejecute el mensaje grabado por el cliente.

A continuación se muestra la configuración del plan de marcación:

extensions.conf de la red 1:

```
[general]
     static = yes
     writeprotect = no
     autofallthrough = yes
     clearglobalvars = no

[busquedadundi]
     switch => DUNDi/conexserv

[compartidas]
     exten => 1001,1,NoOp
     exten => 1003,1,NoOp
     exten => 1101,1,NoOp
     exten => 1103,1,NoOp
     exten => 1234,1,NoOp      ;meetme
     exten => 888,1,NoOp       ;IVR
     exten => 1121,1,NoOp      ;llamada a callfiles desde la red 2

[entrantesdundi]
     exten => 1234,1,Goto(meetme,1234,1)
     exten => 888,1,Goto(llamadaIVR,888,1)
     exten => 1121,1,Goto(emergencia,1121,1)
     exten => _1XXX,1,Goto(prueba12,${EXTEN},1)

[prueba12]
     include => meetme
     include => parkedcalls
     include => llamadaIVR
     include => grabacionAudio
     include => cita
     include => emergencia
     include => subscripcion
     include => busquedadundi

     exten => 1001,1,Dial(SIP/1001,50,mhktwx)
     exten => 1001,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
     exten => 1001,n(unavail),Voicemail(1001@buzon-sip,u)
     exten => 1001,n,Hangup
     exten => 1001,n(busy),Voicemail(1001@buzon-sip,b)
     exten => 1001,n,Hangup
```

```
     exten => 1003,1,Dial(SIP/1003,50,mhktwx)
     exten => 1003,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
     exten => 1003,n(unavail),Voicemail(1003@buzon-sip,u)
     exten => 1003,n,Hangup
     exten => 1003,n(busy),Voicemail(1003@buzon-sip,b)
     exten => 1003,n,Hangup

     exten => 2999,1,Answer()
     exten => 2999,n,VoiceMailMain(${CALLERID(num)}@buzon-sip)
     exten => 2999,n,Hangup

     exten => 1101,1,Dial(IAX2/1101,50,hktwx)
     exten => 1101,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
     exten => 1101,n(unavail),Voicemail(1101@buzon-iax,u)
     exten => 1101,n,Hangup
     exten => 1101,n(busy),Voicemail(1101@buzon-iax,b)
     exten => 1101,n,Hangup

     exten => 1103,1,Dial(SIP/1103,50,hktwx)
     exten => 1103,n,GotoIf($[“${DIALSTATUS}”=”BUSY”]?busy:unavail)
     exten => 1103,n(unavail),Voicemail(1103@buzon-iax,u)
     exten => 1103,n,Hangup
     exten => 1103,n(busy),Voicemail(1103@buzon-iax,b)
     exten => 1103,n,Hangup

     exten => 3999,1,Answer()
     exten => 3999,n,VoiceMailMain(${CALLERID(num)}@buzon-iax)
     exten => 3999,n,Hangup

;------------------------------------------------------------
;    MEETME
;------------------------------------------------------------

[meetme]
     exten => 1234,1,Answer()
     exten => 1234,n,meetme(,pcs)

;------------------------------------------------------------
;    RESPUESTA DE VOZ INTERACTIVA - IVR
;------------------------------------------------------------

[llamadaIVR]
     exten => 888,1,Answer()
     exten => 888,n,Wait(1)
     exten => 888,n,Goto(IVR,s,1)
```

```
        [IVR]
             exten => s,1,Wait(1)
             exten => s,n,Background(/var/lib/asterisk/sounds/en/IVR-prueba/elegirRed)
             exten => s,n,WaitExten()

             exten => 1,1,Goto(IVR1,s,1)
             exten => 2,1,Goto(IVR2,s,1)

             exten => i,1,Playback(invalid)
             exten => i,2,Hangup
             exten => t,1,Goto(IVR,s,1)
             exten => h,1,Hangup

        [IVR1]
             exten => s,1,Wait(1)
             exten => s,n,Background(/var/lib/asterisk/sounds/en/
IVR-prueba/elegirExtenRed1)
             exten => s,n,WaitExten()

             exten => 1,1,Goto(prueba12,1001,1)
             exten => 2,1,Goto(prueba12,1003,1)
             exten => 3,1,Goto(IVR,s,1)

             exten => i,1,Playback(invalid)
             exten => i,2,Hangup
             exten => t,1,Goto(IVR1,s,1)
             exten => h,1,Hangup

        [IVR2]
             exten => s,1,Wait(1)
             exten => s,n,Background(/var/lib/asterisk/sounds/en/
IVR-prueba/elegirExtenRed2)
             exten => s,n,WaitExten()

             exten => 1,1,Goto(busquedadundi,2001,1)
             exten => 2,1,Goto(busquedadundi,2003,1)
             exten => 3,1,Goto(IVR,s,1)

             exten => i,1,Playback(invalid)
             exten => i,2,Hangup
             exten => t,1,Goto(IVR,s,1)
             exten => h,1,Hangup
```

```
        ;------------------------------------------------------------
        ;    GRABACIÓN DE ARCHIVOS DE AUDIO
        ;------------------------------------------------------------

        [grabacionAudio]
             exten => _66X,1,Answer()
             exten => _66X,n,Wait(2)
             exten => _66X,n,Record(/temporal/audio${EXTEN:2:3}.wav)
             exten => _66X,n,Wait(2)
             exten => _66X,n,Playback(/temporal/audio${EXTEN:2:3})
             exten => _66X,n,Wait(2)
             exten => _66X,n,Hangup

        ;------------------------------------------------------------
        ;    CITA TELEFÓNICA
        ;------------------------------------------------------------

        [cita]
             exten => 4567,1,Answer()
             exten => 4567,n,Wait(1)

             exten => 4567,n(inicio),Set(ID=${CALLERID(num)})
             exten => 4567,n,Playback(/var/lib/asterisk/sounds/en/cita/tfno)
             exten => 4567,n,Read(numero,,15,,2,10)
             exten => 4567,n,SayDigits(${numero})
             exten => 4567,n,Wait(1)
             exten => 4567,n,Playback(/var/lib/asterisk/sounds/en/cita/anomesdia)
             exten => 4567,n,Read(fecha,,8,,2,10)
             exten => 4567,n,SayDigits(${fecha})
             exten => 4567,n,Playback(/var/lib/asterisk/sounds/en/cita/horamin)
             exten => 4567,n,Read(hora,,4,,2,,10)
             exten => 4567,n,SayDigits(${hora})

             exten =>
4567,n,Playback(/var/lib/asterisk/sounds/en/cita/confirmacioncita)
             exten => 4567,n,Read(sino,,1,,,5)

             exten => 4567,n,GotoIf($["${sino}"="1"]?cita:inicio)

             ;cita
             exten => 4567,n(cita),GotoIf($["${ID:1:1}"="0"]?citasip:citaiax)

             ;citasip
             exten => 4567,n(citasip),System(echo Channel:SIP/
${ID}>>/callfiles/callback)
             exten => 4567,n,Goto(callfile)
```

```
            ;citaiax
            exten => 4567,n(citaiax),System(echo Channel:IAX2/
${ID}>>/callfiles/callback)

            ;callfile
            exten => 4567,n(callfile),System(echo Callerid:Cita>>/callfiles/callback)
            exten => 4567,n,System(echo WaitTime:30>>/callfiles/callback)
            exten => 4567,n,System(echo Maxretries:3>>/callfiles/callback)
            exten => 4567,n,System(echo RetryTime:300>>/callfiles/callback)
            exten => 4567,n,System(echo Account:${ID}>>/callfiles/callback)
            exten => 4567,n,System(echo Context:prueba13>>/callfiles/callback)
            exten => 4567,n,System(echo Extension:${numero}>>/callfiles/callback)
            exten => 4567,n,System(echo Priority:1>>/callfiles/callback)

            exten => 4567,n,System(touch -t ${fecha}${hora} /callfiles/callback)
            exten => 4567,n,System(mv /callfiles/callback
/var/spool/asterisk/outgoing/${fecha}${hora})
            exten => 4567,n,Hangup

        ;--------------------------------------------------------------
        ;    EMERGENCIA
        ;--------------------------------------------------------------

        [emergencia]
            exten => 112,1,Answer()
            exten => 112,n,Wait(1)
            exten => 112,n,Playback(/var/lib/asterisk/sounds/en/emergencia/bienvenida)
            exten => 112,n,Wait(1)
            exten => 112,n,Record(/var/lib/asterisk/sounds/en/emergencia/mensaje.wav)
            exten => 112,n,Wait(1)
            exten => 112,n,Playback(/var/lib/asterisk/sounds/en/emergencia/mensaje)
            exten => 112,n,Wait(1)
            exten =>
112,n,Playback(/var/lib/asterisk/sounds/en/emergencia/confirmacion)
            exten => 112,n,Read(conf,,1,,1,5)
            exten => 112,n,GotoIf($["${conf}"="1"]?envio:cancelar)

            ;envio
            exten => 112,n(envio),
System(scp /var/lib/asterisk/sounds/en/emergencia/mensaje.wav
root@192.168.2.2:/var/lib/asterisk/sounds/en/emergencia/mensaje.wav)
            exten => 112,n,Wait(20)
            exten => 112,n,System(cp /callfiles/subscripcion/* /callfiles/temp/)
            exten => 112,n,System(mv /callfiles/temp/* /var/spool/asterisk/outgoing/)
            exten => 112,n,Goto(busquedadundi,1122,1)
            exten => 112,n,Hangup
```

```
            exten => 112,n,(cancelar),Hangup

            exten => 1121,1,Answer()
            exten => 1121,n,System(cp /callfiles/subscripcion/* /callfiles/temp/)
            exten => 1121,n,System(mv /callfiles/temp/* /var/spool/asterisk/outgoing/)
            exten => 1121,n,Hangup

        ;--------------------------------------------------------------
        ;    SUBSCRIPCIÓN
        ;--------------------------------------------------------------
        [subscripcion]
            exten => 555,1,Answer()
            exten => 555,n,Set(id=${CALLERID(num)})
            exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/intro)
            exten => 555,n,Read(confirmar,,1,,,5)
            exten => 555,n,GotoIf($["${confirmar}"="1"]?sub:nosub)

            ;sub
            exten => 555,n(sub),GotoIf($["${id:1:1}"="0"]?subsip:subiax)

            ;subsip
            exten => 555,n(subsip),System(echo Channel:SIP/
${id}>>/callfiles/subscripcion/${id})
            exten => 555,n,Goto(subscrip)

            ;subiax
            exten => 555,n(subiax),System(echo Channel:IAX2/
${id}>>/callfiles/subscripcion/${id})

            ;subscrip
            exten => 555,n(subscrip),System(echo
Callerid:Emergencia>>/callfiles/subscripcion/${id})
            exten => 555,n,System(echo WaitTime:30>>/callfiles/subscripcion/${id})
            exten => 555,n,System(echo Maxretries:3>>/callfiles/subscripcion/${id})
            exten => 555,n,System(echo RetryTime:300>>/callfiles/subscripcion/${id})
            exten => 555,n,System(echo Account:${id}>>/callfiles/subscripcion/${id})
            exten => 555,n,System(echo Application:Playback>>/callfiles/subscripcion/
${id})
            exten => 555,n,System(echo
Data:/var/lib/asterisk/sounds/en/emergencia/mensaje>>/callfiles/subscripcion/${id})
            exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/fin1)

            ;nosub
            exten => 555,n(nosub),System(rm -f /callfiles/subscripcion/${id})
            exten => 555,n,Playback(/var/lib/asterisk/sounds/en/subscripcion/fin0)
            exten => 555,n,Hangup
```

A continuación se detallarán los cambios de la configuración del plan de marcación del servidor de la red 1 contexto a contexto:

- [general]:

No sufre cambios

- [busquedadundi]:

No sufre cambios

- [compartidas]:

Se publicarán las extensiones SIP e IAX definidas en el servidor de la red 1, `1001`, `1003`, `1101` y `1103`.

Así como las usadas para meetme (`1234`), llamada con respuesta de voz interactiva (`888`) y la necesaria para publicar los callfiles desde la red 2 (`1121`)

- [entrantesdundi]:

Puesto que se han publicado las extensiones que se comparten, habrá que definir qué contexto corresponde a cada una.

Para la extensión `1234` corresponde el contexto `[meetme]`

Para la `1121`, el contexto `[emergencia]`, dónde se volcarán los archivos callfiles para que sean leídos por Asterisk

Para la extensión `888`, se saltará al contexto `[IVR]`

Y para cualquier extensión que comience por 1, es decir, que sea de la forma `1XXX` saltará al contexto `[prueba12]`, que es dónde se definen las llamadas propias de la red.

- [prueba12]:

En éste caso no es necesario separar los contextos por internos y externos, puesto que para acceder a cualquier extensión que no pertenezca a la red se hará gracias al contexto `[busquedadundi]`.

Además, se ha optado por unir los contextos diferenciados por los protocolos SIP e IAX en uno sólo que abarca todas las extensiones, tanto SIP como IAX.

Así pues, en éste contexto se definirá el funcionamiento de las extensiones `1001`, `1003`, `1101` y `1103`.

También quedarán definidas en éste contexto las llamadas de acceso al buzón de voz, que en éste caso necesitarán de una numeración distinta en función de que el llamante utilice el protocolo SIP o IAX.

Quedarán fijadas las extensiones `2999` para el acceso al buzón de voz de cualquier extensión SIP y la `3999` para las extensiones IAX.

- [meetme]:

Definido en la red 1.

No sufre cambios.

- [llamadaIVR]

Definido en la red 1.

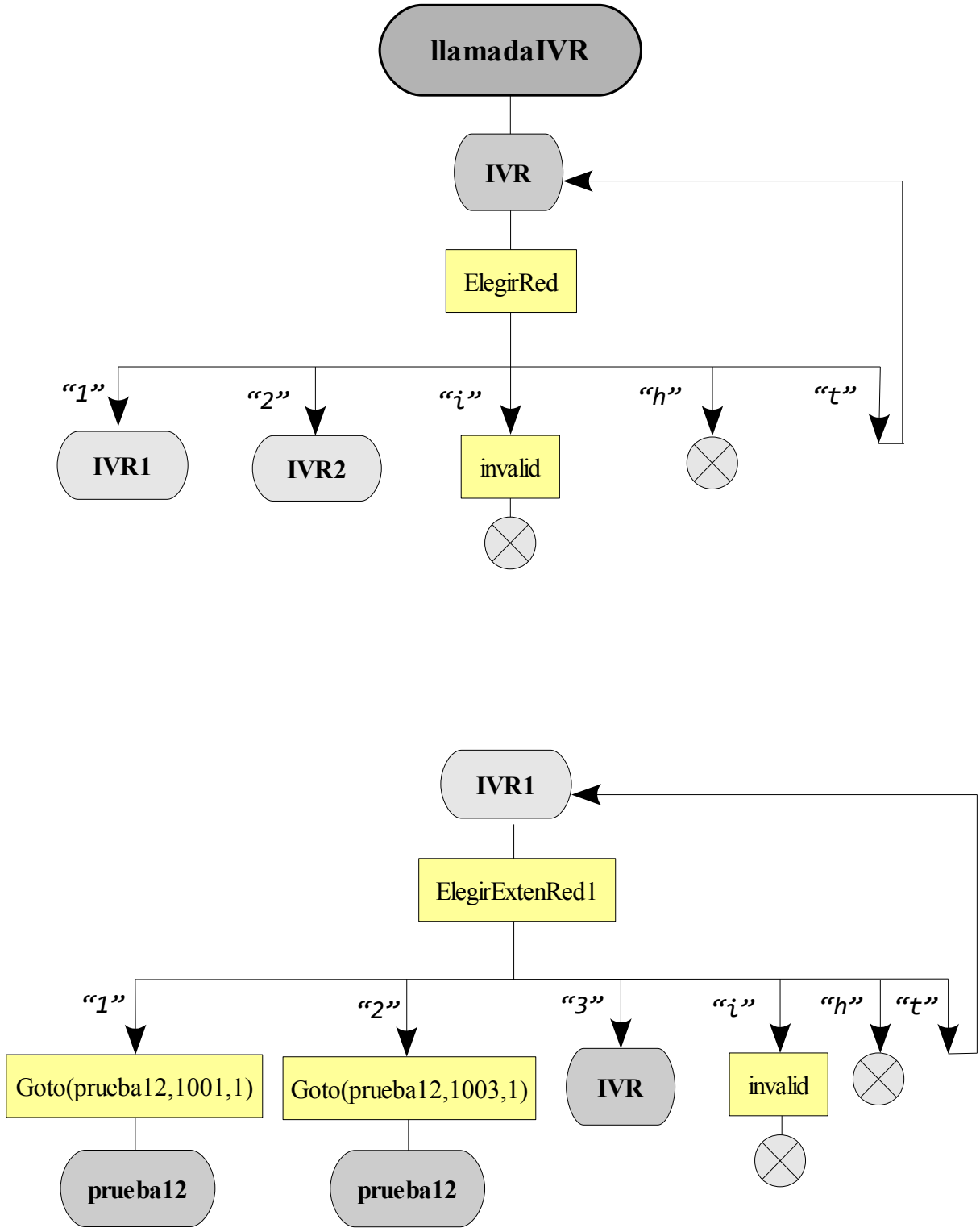
La respuesta de voz interactiva permite que una extensión pueda llamar a una u otra en base a una serie de archivos de audio que nos muestran varias opciones para llegar a la extensión determinada.

Mediante el IVR es posible acceder a cualquier extensión SIP tanto de la red 1 como de la red 2.

El principal cambio surge a raíz de que ya no existe el cliente `[red2]` que apuntaba al servidor de la red contraria y que mediante una ruta concreta permitía acceder a una extensión. Por tanto, para solucionar esa modificación se ha hecho uso de un salto al contexto `[busquedadundi]` en caso de que la extensión pertenezca a la red 2.

Dicho contexto se encargará de solicitar la información pertinente en el servidor de la red 2 y reconducirá la llamada a la extensión SIP que se había solicitado inicialmente.

Se han usado únicamente extensiones SIP por simplificar el código, sin embargo la modificación para albergar también a las extensiones IAX sería muy rápida.


llamadaIVR
IVR
ElegirRed
"1"
"2"
"i"
"h"
"t"
IVR1
IVR2
invalid
IVR1
ElegirExtenRed1
"1"
"2"
"3"
"i"
"h"
"t"
Goto(prueba12,1001,1)
Goto(prueba12,1003,1)
IVR
invalid
prueba12
prueba12

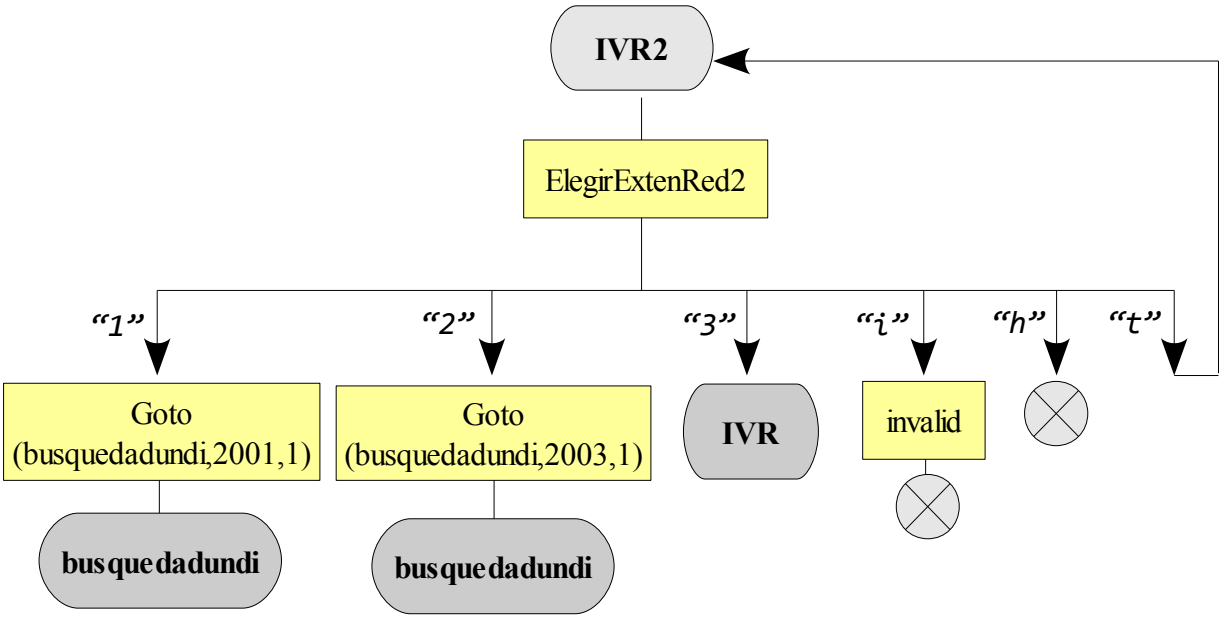


Figura 54: "Diagrama de flujo del contexto [llamadaIVR] con DUNDi"

- [grabacionAudio]:

No sufre cambios

- [cita]:

El contexto `[cita]` permite guardar los datos de un usuario al que se quiere llamar y la hora concreta y el propio sistema Asterisk lo ejecutará cuándo llegue el momento oportuno.

Para ello se usan archivos callfiles, que se moverán al directorio */var/spool/asterisk/outgoing/* y se ejecutarán cuándo el tiempo establecido coincida con la hora del sistema.

El archivo tendrá como destino el cliente que solicitó la cita y luego ejecutará la acción correspondiente, que en éste caso será una llamada a la extensión con la que se quería concertar dicha cita.

Anteriormente, esta llamada a la extensión con la que concertar la cita se calculaba mirando los dígitos de la extensión para averiguar en primer lugar si era una extensión de la red 1 o la red 2, y en segundo lugar buscando el protocolo de la extensión, para así hacer una llamada mediante la aplicación `Dial()` que permitiera llegar al destino.

En caso de que la extensión de la cita perteneciera a la red 2 se hacía uso del cliente `[red2]` para poder contactar con el servidor de dicha red y que pudiera dirigir la llamada a la extensión correspondiente.

Sin embargo, el uso del cliente `[red2]` queda descartado, el procedimiento para llamar a cualquier extensión que no pertenezca a la red es un salto al contexto `[busquedadundi]`, el cuál buscará en el otro servidor la extensión solicitada.

En lugar de ejecutar una aplicación como `Dial()` con la ruta a la extensión final, se efectúa por tanto un salto al contexto definiendo la extensión final y la prioridad.

Con éste procedimiento se evita tener que buscar la red a la que pertenece la aplicación y el protocolo asociado. Con un simple salto a un contexto es suficiente.

En éste caso se ha optado por el contexto `[prueba12]`, y la extensión buscada será la guardada en la variable `${numero}`, introducida por el cliente como extensión con la que concertar la cita.

De este modo se buscará en primer lugar la extensión en el contexto `[prueba12]`, si pertenece a la red 1, se ejecutará la llamada pertinente asociada a la extensión. En caso de que pertenezca a la red 2, cómo el contexto `[busquedadundi]` está incluido dentro del contexto `[prueba12]`, buscará ahí como última opción. En el otro servidor se ejecutará la llamada asociada a la extensión solicitada si existe.

Una vez finalizado el archivo callfile se modificarán las condiciones temporales según la fecha y hora definidas por el cliente. A continuación se moverá al directorio */var/spool/asterisk/outgoing/* con el nombre *${fecha}${hora}.call* para que sea fácil localizar los archivos y cuándo se ejecutarán.

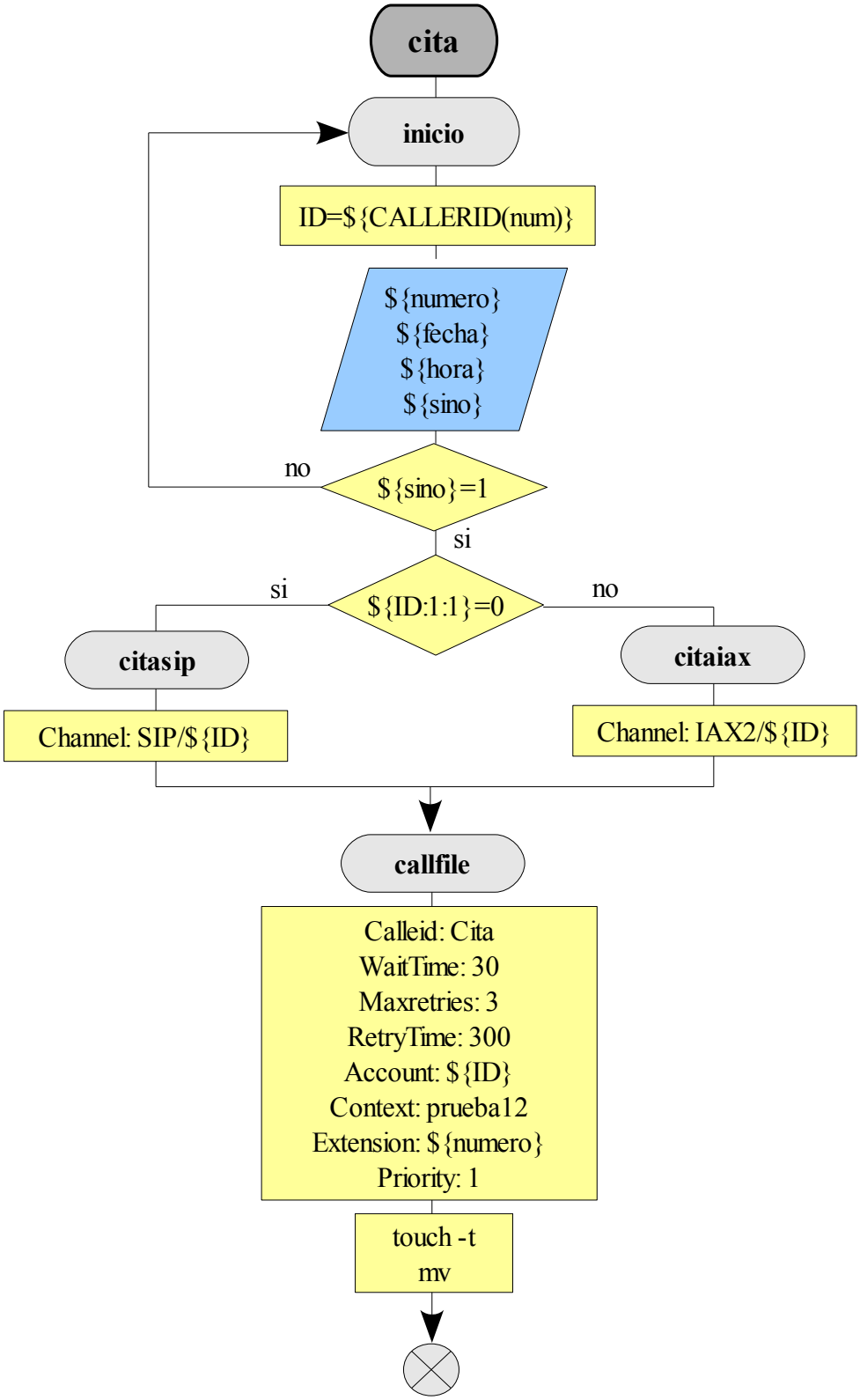


Figura 55: “Diagrama de flujo del contexto [cita] con DUNDi”

- [emergencia]:

El contexto `[emergencia]` permite que cualquier cliente, al llamar a la extensión `112`, pueda grabar un mensaje de emergencia que se reproducirá de forma inmediata en todas aquellas extensiones que estén subscritas a dicho servicio.

Anteriormente se realizaba una copia de los callfiles subscritos que se enviaba al directorio */var/spool/asterisk/outgoing/*, y Asterisk se encargaba de ejecutarlos instantáneamente.

En el directorio dónde se almacenaban los callfiles de la subscripción había tanto de extensiones de la red 1 como extensiones de la red 2 que habían sido generados al llamar a una extensión definida para tal fin en esta red, cuándo se subscribía una extensión en la red contraria.

Para aquellas extensiones que pertenecían a la otra red, la forma de acceder a ellas era mediante la configuración de la primera línea del archivo callfile, es decir, con la definición de `Channel`, dónde se indicaba el protocolo y el cliente al que llamar. En caso de que fuera una extensión de la red 2, habría que hacer uso del cliente `[red2]` para hacer llegar la llamada al destino.

Una vez eliminado el cliente `[red2]`, habiendo mejorado y simplificado el código, se encuentra el problema de que no es posible configurar `Channel` para hacer llegar la llamada al destino.

Después de intentar múltiples combinaciones, como utilizar la variable `${CALLERID(num)}`, o similar, para obtener la ruta de acceso a la red contraria o al menos la extensión destino, o después de intentar buscar la solución considerando `${EXTEN}` como la extensión que se subscribe en la red 2. E incluso haciendo uso del cliente troncal `enlace`, como si se tratara de `[red2]`, se ha encontrado una solución alternativa distinta, que se basa en copiar el mensaje grabado por el cliente al otro servidor y mover los callfiles de ambas redes al directorio correspondiente para que sean ejecutados.

Sin embargo, un requisito imprescindible para realizar una copia al otro servidor de la grabación, es que se haga de forma segura y encriptada.

Para ello se hará uso del comando `scp`, el cuál usa el protocolo SSH, que solicitará una contraseña que habrá que gestionar para que se introduzca de forma automática.

Para que no sea necesario introducir constantemente la contraseña, y por tanto Asterisk pueda funcionar sin una supervisión permanente se ha optado por:

- Activar el demonio ssh mediante los comandos:
  `#/sbin/service sshd start`
- Descomentar "`PubkeyAuthentication=yes`" en el archivo */etc/ssh/sshd_config*
- Generar clave en el servidor local con el comando en: */root/.ssh/id_rsa*
  `#ssh_keygen -t rsa`, que dará lugar a los archivos *id_rsa* e *id_rsa.pub*
- Una vez copiado el archivo *id_rsa.pub* al servidor remoto habrá que ejecutar el comando:

```
#mv id_rsa.pub .ssh/authorized_keys
```

De ésta forma se podrá copiar el mensaje de emergencia de una forma segura.

Una vez copiado al directorio correspondiente de la red contraria, que deberá coincidir con el de la red original, se moverá una copia de los archivos callfiles de las subscripciones para que sean inmediatamente ejecutados.

Además se realizará una llamada a una extensión concreta en la red contraria que repetirá el mismo proceso, mover una copia de las extensiones subscritas al directorio */var/spool/asterisk/outgoing/* para que se ejecuten.

De la misma forma, será necesario describir en ésta red una extensión que permita que la red contraria pueda mover los archivos callfiles cuándo se haya generado un mensaje urgente que haya que reproducir en todas las extensiones subscritas, ya sean de la red 1 o de la red 2.

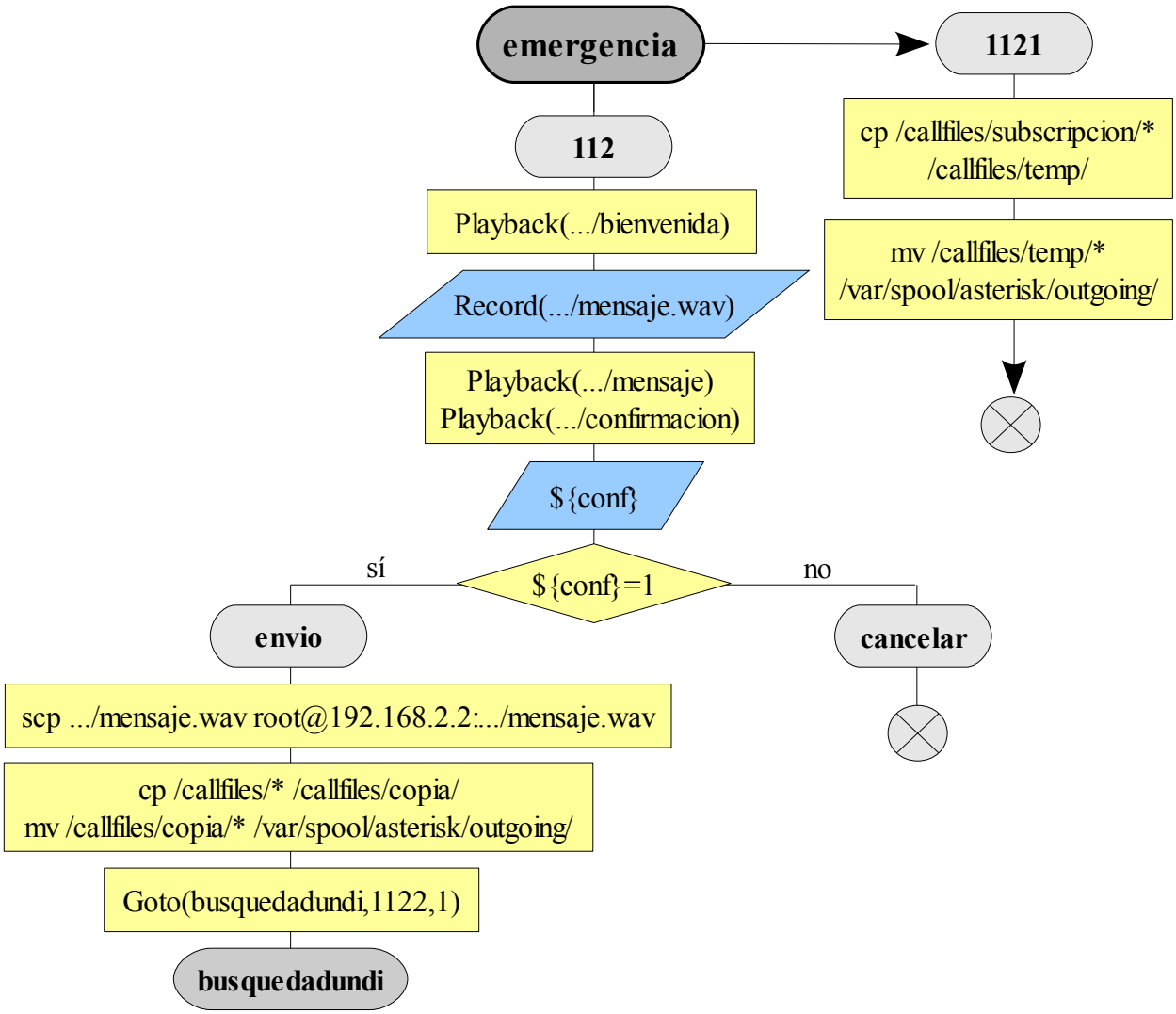


Figura 56: "Diagrama de flujo del contexto [emergencia] con DUNDi"

- [subscripcion]:

El servicio de subscripción permite subscribirse o cancelar el acceso al servicio de llamadas de emergencia con sólo llamar a una extensión determinada, en éste caso a la `555`.

Éste servicio crea un archivo callfile por cada extensión de la red que se subscribe a él.

Dicho archivo definirá la forma de acceso a cada extensión, por tanto la primera línea definirá en `Channel` el protocolo y la extensión que se desea subscribir.

La acción a ejecutar será la reproducción del mensaje grabado por el cliente.

Estos archivos tendrán el nombre de la extensión y estarán guardados en un directorio a la espera de que sea grabado el mensaje y por tanto sea necesario ejecutarlos todos.

Para cancelar la subscripción se hará mediante el borrado del callfile correspondiente.

En la configuración anterior se usaron extensiones que duplicaran las subscripciones de la red contraria, guardándose a la vez en aquella red y en ésta, con los cambios pertinentes en la línea `Channel` del archivo callfile.

Sin embargo, no se puede realizar una llamada desde un callfile a una red contraria, a menos que se haga saltando al contexto que permite buscar una extensión. La línea `Channel` especifica el formato a usar y éste no concuerda con una posible llamada a la red contraria, por tanto se ha optado por suprimir esta duplicación de archivos de subscripciones y copiar el mensaje urgente a la otra red, y que a la misma vez se muevan los archivos de subscripción de la red 2 al directorio correspondiente para que sean ejecutados.

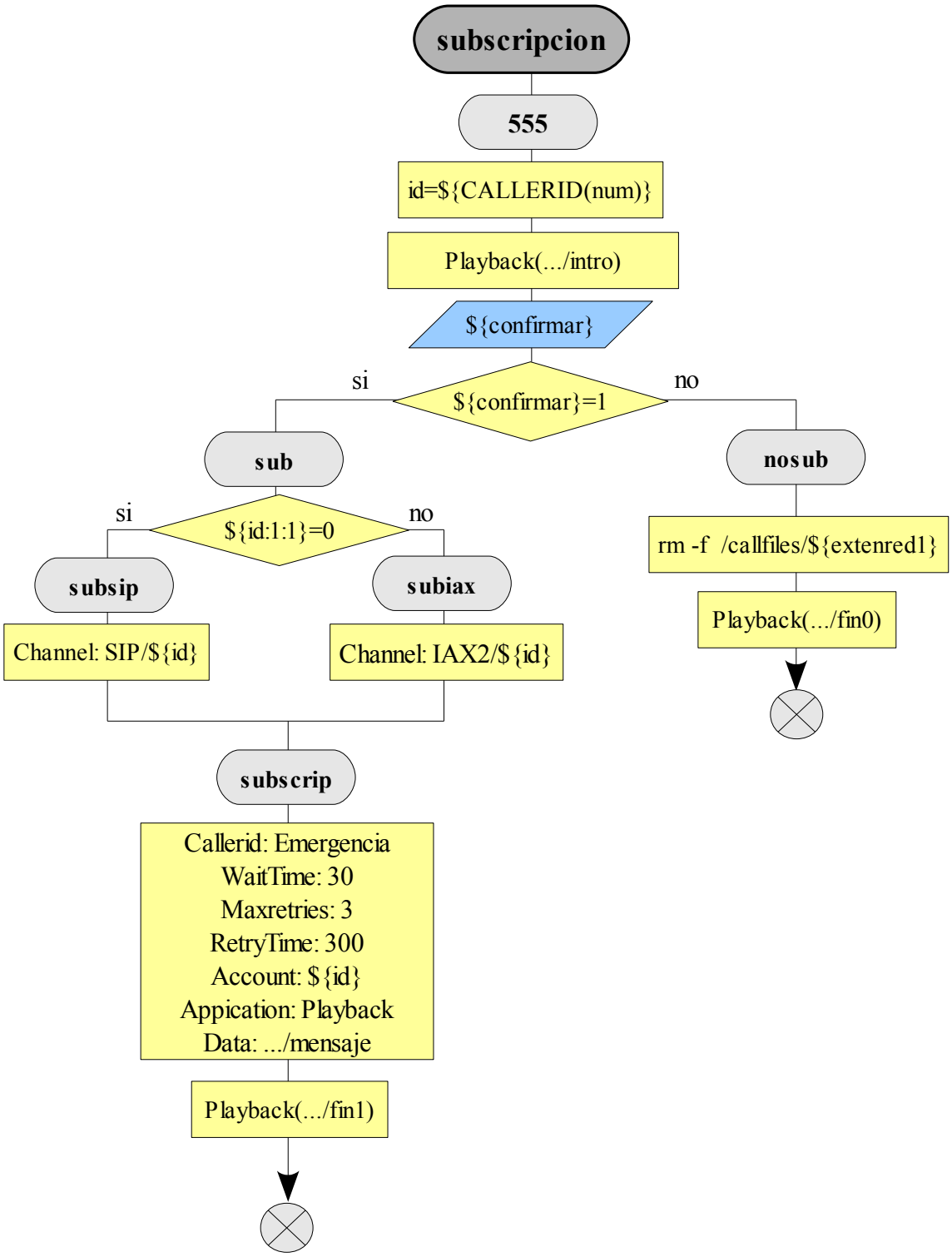


Figura 57: “Diagrama de flujo del contexto [subscripcion] con DUNDi”

dundi.conf:

Como en el caso anterior, el archivo de configuración *dundi.conf* permite definir la forma de comunicación entre ambos servidores.

Para ello se configurará el recurso DUNDi `conexserv` con el contexto asociado `[compartidas]` para indicar las extensiones que se desean publicar hacia el exterior y que por tanto serán visibles por el servidor de la red contraria.

Además, habrá de especificarse el cliente IAX que ejerce de conexión para la intercomunicación, así como la variable que almacena la contraseña temporal y la dirección IP del servidor.

También habrá que definir el identificador de la red con la que se desea compartir la información. En éste caso será la dirección MAC del servidor de la red 2, que definirá a su vez la dirección IP, las claves necesarias y los permisos asociados.

En la red 2, el archivo será prácticamente igual, salvo que será el identificador MAC del servidor de la red 1 lo que se describa.

Tanto el recurso DUNDi como el contexto asociado, así como el nombre del cliente que hace de puente tienen el mismo nombre, la diferencia en `[mappings]` estriba en la dirección IP del servidor, que en éste caso será la `192.168.2.2`.

Por último en el contexto `[general]` se usará como identificador la dirección MAC del servidor de la red 2.

dundi.conf:

```
[general]
     department = proyectoVoIP
     organization = UPCT
     locality = CT
     stateprov = MU
     country = SP
     email = mi@email.com
     phone = +666666666

     bindaddr = 0.0.0.0
     port = 4520
     entityid = 00:0F:FE:77:C1:F5
     cachetime = 30
     ttl = 5
     autokill = yes
     secretpath = dundi
```

```
        [mappings]
            conexserv => compartidas,0,IAX2,enlace${SECRET}@192.168.1.2/$
{NUMBER},nopartial

        [00:0F:FE:77:C6:21]
            model = symmetric
            host = 192.168.2.2
            inkey = llave
            outkey = llave
            include = all
            permit = all
            order = primary
```

Con éste último caso se dan por finalizadas todas las pruebas que se han considerado pertinentes a la hora de configurar un sistema Asterisk y sus prestaciones más comunes.

Capítulo 6: LÍNEAS FUTURAS

En el basto mundo que ofrece Asterisk es difícil imaginar las fronteras y las posibilidades de mejora.

Ofrece un abanico tan extenso que cualquier ampliación resultaría válida, si bien es cierto que se han encontrado posibles mejoras en cuánto al desarrollo de aplicaciones.

La más evidente es la posibilidad de integrar distintas herramientas.

Existe la posibilidad de mejorar Asterisk integrando la telefonía de VoIP con Skype, de forma que sea posible realizar llamadas que contengan vídeo a una calidad alta.

Aunar distintos calendarios e integrarlos en un mismo sistema es una necesidad primordial hoy día. Asterisk en la versión 8 ofrece esta posibilidad mediante el archivo *calendar.conf*, que permite sincronizar la agenda incluso desde el teléfono móvil.

Esto es especialmente útil cuándo se desea gestionar un IVR según los horarios de oficina, festividades y demás excepciones.

Convertir un archivo de texto en uno de audio (text to speech) permite a cualquier invidente usar los servicios que proporciona Asterisk.

Festival ofrece la posibilidad de asociar el sistema de interpretación de texto al conectarlo a Asterisk, incluso este software libre tiene voces definidas en español.

MySQL se ha posicionado como una de las mejores opciones a la hora de manejar bases de datos. En sistemas dónde es importante la tarificación y el control de la información resultará imprescindible contar con una base de datos que de una forma simple y ordenada permita el acceso a las distintas opciones a consultar. Asterisk, contiene archivos preconfigurados, que de ser configurados correctamente, permitirían la comunicación con MySQL en caso de que sea necesario.

Otra de las posibilidades es el uso de programas externos (AGI) que permitan ampliar las prestaciones de la centralita, o que incluso puedan gestionarla de forma externa. Para definirlos se podrá hacer uso de múltiples lenguajes de programación, como PHP, C, etc.

Por último, una posible mejora a tener en cuenta es la interconexión con la red de telefonía básica o PSTN. Ésto es posible gracias a distintos conversores que, con sus distintas interfaces, permiten las distintas posibilidades de comunicación.

Capítulo 7: ÍNDICE DE FIGURAS

FIGURAS **Páginas**

Capítulo 8: GLOSARIO

ADSL (Asymmetric Digital Subscriber Line). Método para aumentar la velocidad de transmisión en un cable de cobre. ADSL facilita la división de capacidad en un canal con velocidad más alta para el suscriptor, típicamente para transmisión de vídeo, y un canal con velocidad significativamente más baja en la otra dirección.

Aplicación. Instrucción asociada a una extensión. Tipos: generales, de mantenimiento de llamada, de control de flujo, de manipulación de variables, de gestión de sonidos, de uso de buzón de voz y conferencias.

Asterisk. Aplicación de código abierto que permite la administración de una central telefónica en una red de área local usando tecnología VoIP

Asynchronous Transfer Mode (ATM) es una tecnología de conmutación de red que utiliza celdas de 53 bytes, útil tanto para LAN como para WAN, que soporta voz, vídeo y datos en tiempo real y sobre la misma infraestructura. Utiliza conmutadores que permiten establecer un circuito lógico entre terminales, fácilmente escalable en ancho de banda y garantiza una cierta calidad de servicio (QoS) para la transmisión. Sin embargo, a diferencia de los conmutadores telefónicos, que dedican un circuito dedicado entre terminales, el ancho de banda no utilizado en los circuitos lógicos ATM se puede aprovechar para otros usos.

ATA (Analog Telephone Adapter). Adaptador telefónico para usar un teléfono analógico en una red IP. Habitualmente tienen un conector FXS para teléfono analógico normal y envían por VoIP a través del conector LAN, soportan SIP normalmente.

Canal. Conexión que distribuye una llamada entrante o saliente dentro del sistema de Asterisk PBX. Puede ser una conexión con la telefonía analógica, digital o VoIP. Tipos: IAX2, SIP, Skinny, Zap, mISDN

Códec, Algoritmos de Compresión/Descompresión. Se utilizan para reducir el tamaño de los datos multimedia, tanto audio como vídeo. Compactan (codifican) un flujo de datos multimedia cuando se envía y lo restituyen (decodifican) cuando se recibe. Entre los códecs de audio más extendidos se encuentran: GSM (Global Standard for Mobile Communications), ADPCM, PCM, DSP TrueSpeech, iLBC, g711. Y entre los códecs de vídeo tenemos THEORA, H.264, H.263+, y MPEG4.

Conmutación de circuitos. Es un tipo de comunicación que establece o crea un canal dedicado (o circuito) durante la duración de una sesión. Después de que es terminada la sesión (e.g. una llamada telefónica) se libera el canal y éste podrá ser usado por otro par de usuarios. El ejemplo más típico de este tipo de redes es el sistema telefónico. Los sistemas de conmutación de circutos son ideales para comunicaciones que requieren que los datos/infiormación sean transmitidos en tiempo real.

Conmutación de paquetes. Técnica de conmutación en la cual los mensajes se dividen en paquetes antes de su envío. Cada paquete se transmite de forma individual y puede incluso seguir rutas diferentes hasta su destino. Una vez que los paquetes llegan a éste se agrupan para reconstruir el mensaje original. En los sistemas de conmutación de paquetes el canal es compartido por muchos usuarios simultáneamente. La conmutación de paquetes es más eficiente y robusta para datos que pueden ser enviados con retardo en la transmisión (no en tiempo real), tales como el correo electrónico, paginas web, archivos, etc.

Contexto. Engloba un determinado número de extensiones en su interior y define un comportamiento concreto dentro del dialplan. Tipos: general, globals, definidos por el usuario, etc.

Dialplan. Define el comportamiento de la centralita Asterisk PBX en cuánto a llamadas entrantes, salientes y forma de uso. Se encuentra definido en el archivo extensions.conf, ubicado en /etc/asterisk/. El contenido está dividido en contextos y éstos a su vez en extensiones. Los contextos habituales son: general, global y aquellos definidos por el usuario.

DNS (Domain Name System). Sistema de asignación de nombre a equipos y servicios de red que se organiza en una jerarquía de dominios. La asignación de nombres DNS se usa en las redes TCP/IP, como Internet, para localizar equipos y servicios con nombres descriptivos. Cuando un usuario escriba un nombre DNS en una aplicación, los servicios DNS podrán traducir el nombre a otra información asociada con el mismo, como una dirección IP.

Extensión: Número que tiene asociado una lista de aplicaciones a ejecutar. Normalmente corresponde con un cliente telefónico, otras veces es el destino de un salto desde una aplicación y gestiona un proceso a ejecutar. Tipos de extensiones: literales, predefinidas, o de patrón.

FXO. Puerto que recibe las señales del puerto FXS. Un teléfono no tiene un puerto FXO. No envía señales de tono timbrado, sólo recibe las señales que envía los FXS. Funciona como terminal de línea.

FXS. Puerto usado por las líneas de telefonía analógica (también denominados POTS), este puerto envía señales de timbre y tono para teléfonos analógicos. Es decir, emulan una línea telefónica tradicional.

Gatekeeper. Entidad de red H.323 que proporciona traducción de direcciones y controla el acceso a la red de los terminales, pasarelas y MCUs H.323. Puede proporcionar otros servicios como la localización de pasarelas. Actúa en conjunto con varios gateways y se encarga de autentificar usuarios, controlar el ancho de banda, encaminamiento IP, etc. Es el cerebro de la red de telefonía IP.

Gateway. Dispositivo empleado para conectar redes que usan diferentes protocolos de comunicación de forma que la información puede pasar de una a otra. En VoIP existen dos tipos de pasarelas: la Pasarela de Medios (Media Gateways), para la conversión de datos (voz), y la Pasarela de Señalización (Signalling Gateway), para convertir información de señalización. Puede hacer de puente entre la red telefónica convencional (PSTN) y la red IP, convirtiendo la señal analógica en un caudal de paquetes IP y viceversa.

H.323, es la recomendación global (incluye referencias a otros estándares, como H.225 y H.245) de la Unión Internacional de Telecomunicaciones (ITU). Define las diferentes entidades que hacen posible estas comunicaciones multimedia: endpoints, gateways, unidades de conferencia multipunto (MCU) y gatekeepers, así como sus interacciones.

IAX (Inter-Asterisk eXchange protocol). Fue diseñado como un protocolo de conexiones VoIP entre servidores Asterisk, aunque actualmente sirve para conexiones entre clientes y servidores que soporten el protocolo. Actualmente la versión usada es la IAX2, diseñada y pensada para su uso en conexiones de VoIP aunque también puede soportar conexiones de vídeo por ejemplo.

ISDN. Ver RDSI

Jitter (variación de retardo). Es un término que se refiere al nivel de variación de retado que introduce una red a la hora de entregar los paquetes.

MGCP. IETF 2705. Protocolo maestro/esclavo de control de dispositivos dónde un gateway esclavo (MG, Media Gateway) es controlado por un maestro (MGC, Media Gateway Controller o Call Agent). Se compone por un MGC, uno o más MG y uno o más SG (Signaling Gateway)

Private Branch Exchange (PBX), Centralita telefónica privada utilizada en compañías y organizaciones, para manejar llamadas externas e internas. La ventaja es que la compañía no necesita una línea telefónica para cada uno de sus teléfonos. Además las llamadas internas no salen al exterior y por tanto no son facturadas.

PSTN. Red conmutada de telefonía pública basada en la conmutación de circuitos para el transporte de llamadas de voz codificadas. La realización de una comunicación requiere el establecimiento de un circuito físico durante el tiempo que dura ésta, lo que significa que los recursos que intervienen en la realización de una llamada no pueden ser usados en otra hasta que la primera no finalice, incluso durante los silencios que se suceden dentro de una conversación típica.

Pulse Code Modulation (PCM), Convierte una señal analógica (sonido, voz normalmente) en digital para que pueda ser procesada por un dispositivo digital, normalmente un ordenador. Si, como ocurre en Telefonía IP, nos interesa comprimir el resultado para transmitirlo ocupando el menor ancho de banda posible, necesitaremos usar además un códec.

RDSI (Red Digital de Servicios Integrados). Red que da soporte a varios canales digitales. La RDSI básica tiene 2 de 64 kbps y uno de 16 kbps para señales de control. Ideal para imagen, sonido y multimedia.

Router, Un dispositivo físico, o a veces un programa corriendo en un ordenador, que reenvía paquetes de datos de una red LAN o WAN a otra. Basados en tablas o protocolos de enrutamiento, leen la dirección de red destino de cada paquete que les llega y deciden enviarlo por la ruta más adecuada (en base a la carga de tráfico, coste, velocidad u otros factores). Trabajan en la capa de Red de la pila de protocolos.

RTCP (RTP Control Protocol). Protocolo de comunicación que proporciona información de control asociada a un flujo de datos RTP. La función principal es informar de la calidad de servicio proporcionada por RTP. Recoge estadísticas de la conexión e información como bytes enviados, paquetes enviados o perdidos o jitter.

RTP (Real-time Transport Protocol). Define un formato de paquete estándar para el envío de audio y vídeo sobre Internet. Es definido en el RFC 1889.

SIP (Session Initiation Protocol). Es un estándar definido para establecer, enrutar y modificar sesiones de comunicaciones a través de redes IP. Usa el modelo de Internet para el mundo de las telecomunicaciones, usando protocolos como HTTP y SMTP. También usa una estructura de dirección URL. SIP no depende del dispositivo y no hace distinción entre voz y datos. Es un protocolo para el establecimiento, terminación y modificación de sesiones. Está basado en un sistema de petición/respuesta.

TDM (Time-Division Multiplexing). Es una forma de entablar comunicaciones a través de la PSTN utilizando dos o más canales de señalización para la transmisión de mensajes de voz.

VoIP: Voz sobre Protocolo de Internet, también llamado Voz sobre IP. Es la tecnología que permite la transmisión de la voz a través de protocolo IP por medio de muestreo y codificación hasta convertirla en una trama de datos.

Zaptel: Controladores para la aplicación Asterisk que permiten la configuración y soporte de tarjetas FXS y FXO.

Capítulo 9: BIBLIOGRAFÍA

1 ARROYO ALMAGRO, Federico. *Estudio, evaluación, implantación y configuración de aplicaciones VoIP. Software Libre. Casos Prácticos.* Director: Juan Carlos Sánchez Aarnoutse. Cartagena: Federico Arroyo Almagro, 2010 Documento en pdf de acceso sólo desde UPCT

2 ASTERISK [Web en línea] <http://www.asterisk.org/> [Consultada 04/09/2013]

3 CARDOZO F. Joel. *Sistemas de telecomunicaciones. Concepto de IP en las nuevas redes Integradas* [Web en línea] s.l. : Monografías.com, 2006 <http://www.monografias.com/trabajos33/telecomunicaciones/telecomunicaciones.shtml> [Consultada 04/092013]

4 CUEVA, Marco. *Aspectos generales acerca de la voz sobre IP* [Web en línea] s. l. : Monografias.com, 2011 <http://www.monografias.com/trabajos87/voz-ip/voz-ip.shtml> [Consultada 19/09/2013]

5 DAVISON, Jonathan et al. *Fundamentos de voz sobre IP* Madrid: Cisco, 2001. ISBN 84-205-3190-1

6 DUNDi [Web en linea] <http://www.dundi.com> [Consultada 21/03/2012]

7 Ekiga [Web en línea] <http://ekiga.org/> [Consulta 25/09/2013]

8 IBARRA CORRETGÉ, Saúl. *Asterisk: el futuro de la telefonía y la VoIP ha llegado* [Web en línea] <http://www.slideshare.net/saghul/curso-de-asterisk-everano-2007> [Consultada 13/03/2013]

9 IRONTEC [Web en línea] <http://www.irontec.com/formacion_cursos.html> [Consulta 24/01/2013]

10 KEAGY, Scott. *Integración de Redes de Voz y Datos* Madrid : Cisco Systems, 2001 84-205-3187-1

11 KIAX ver2 [Web en línea] <http://www.forschung-direkt.eu/kiax2/#features> [Consultada 18/08/2013]

12 MAHLER, Paul *VoIP Telephony with Asterisk: a technical overview of the Open Source PBX.* [Libro en línea]. s l.: Signate. LL, 2004 [Consulta 25/09/2013] ISBN 09759992-0-6

13 MARTINEZ, Evelio. Conmutación de circuitos y paquetes [Web en línea] <http://www.eveliux.com/mx/conmutacion-de-circuitos-y-paquetes.php> [Consultada 15/08/2012]

14 MEGGELEN J, Van. et al. *Asterisk. The Future of Telephony* [Libro en línea], Sebastopol ; O'Reilly Media Inc, 2007, [Consulta 25/09/2013] ISBN e-Book: 10: 0-596-51048-9

15 MENÉNDEZ, Julián J. *Usando la red DUNDi en Asterisk* [Web en línea] <http://www.julianmenendez.es/usando-dundi-asterisk/> [Consultada 23/02/2011]

16 Monografías.com [Web en línea] <http://www.monografias.com/> [Consulta 11/01/2013]

17 Interconexión a la PSTN [Web en línea] <http://elastixtech.com/fundamentos-de-telefonia/interconexion-a-la-pstn/> [Consultada 23/02/2012]

18 ROSARIO VILLAREAL, M. A. *El Estándar VoIP - Redes y servicios de banda ancha* [Web en línea]. s.l. : Monografías.com. 2006 <http://www.monografias.com/trabajos33/estandar-voip/estandar-voip.shtml> [Consulta 19092013]

19 Sinologic [Web en línea]<http://www.sinologic.net/blog/> [Consultada 23/02/2013]

20 Sourceforge [Web en línea] <http://sourceforge.net/projects/kiax/files/> [Consulta 25/09/2013]

21 SPENCER, M et al. *IAX: Inter-Asterisk eXchange Version 2* [Web en línea] s. l. : RFC Editor, 2010 <http://www.rfc-editor.org/pdfrfc/rfc5456.txt.pdf> [Consultada 03/05/2012]

22 VIVAS, Katherine. *VoIP. Voz sobre IP* [Web en línea] s.l. : Monografias.com, 2005 <http://www.monografias.com/trabajos23/voz-sobre-ip/voz-sobre-ip.shtml> [Consultada 04/09/2013]

23 *VOIPFORO* [Web en línea] <http://www.voipforo.com/> [Consulta 20/09/2013]

24 *VOIP info* [Web en línea] <http://www.voip-info.org/> [Consulta 15/08/2013]

25 *VOZ sobre Protocolo de Internet* [Web en línea] <http://es.wikipedia.org/wiki/VoIP> [Consultada 25/11/2012]

26 *VOZ to voice: talking around the World* [Web en línea] <http://www.voztovoice.org/> [Consultada 08/10/2012]