Sistema de autenticación de usuarios y validación de llamadas telefónicas

Daniel Britos, Jorge Kleinerman, Laura Vargas
Laboratorio de Redes y Comunicaciones de Datos
Facultad de Ciencias Exactas, Físicas y Naturales, Universidad Nacional de Córdoba
(5000) Avda. Vélez Sársfield 1611, Córdoba, Argentina
dbritos@gmail.com, jkleinerman@gmail.com, lvargas@efn.uncor.edu,

Abstract

This work presents a software that complements an IP-PBX (Internet Protocol - Private Branch Exchange) implemented with an Asterisk server. Its functionalities allow authenticate users, give them permissions or impose restrictions on the calls they make to the external telephony network, in a flexible and personal way.
This software also stores in a database detailed records of the calls saving date and time, duration, called phone number, caller's name. Furthermore the database can be queried at any time by using SQL (Structured Query Language).
It is important to say that this software is developed to cover true requirements of a department at the National University of Cordoba so as to control the use of telephone lines and lower expenses.

Resumen

Este trabajo presenta un software que complementa una IP-PBX (Internet Protocol- Private Branch eXchange) instalada con un servidor Asterisk. Mediante el mismo es posible autenticar usuarios, otorgarles a los mismos permisos o imponerles restricciones en las llamadas que realicen a la red telefónica externa, en forma flexible y personalizada.
Este sistema registra en una base de datos fecha y hora de las llamadas que hace cada usuario, duración de las mismas, número destino, pudiendo en cualquier momento ser consultada con flexibilidad usando SQL (Structured Query Language).
Cabe destacar que este software es desarrollado para cubrir requerimientos reales de una dependencia en la Universidad Nacional de Córdoba, con el fin de controlar el uso de las líneas telefónicas y disminuir gastos.

Palabras clave: VoIP, Asterisk, PBX, SQL, control de llamadas.

1. Introducción

Un problema habitual que sufren las organizaciones que comparten líneas telefónicas entre sus usuarios para realizar llamadas a la red de telefonía pública, es el uso desmedido de las mismas, lo cual deriva en altos gastos en los abonos mensuales y falta de disponibilidad de las líneas telefónicas.
Muchas organizaciones solucionan este problema estableciendo tiempos máximos para las llamadas que se realizan a la red de telefonía pública. Esto tiene como desventaja que usuarios que realmente necesitan hacer una llamada por un tiempo mayor que el fijado tienen que volver a discar y su conversación se ve interrumpida. Además, con este sistema no se tiene un control de las llamadas innecesarias.
Otras organizaciones establecen bloqueos a diferentes tipos de llamadas como las de larga distancia, o a celulares. Este mecanismo tampoco es lo suficientemente flexible, ya que imposibilita la realización de este tipo de llamadas a usuarios que realmente las necesitan para cumplir con sus tareas.
Hoy en día, muchas organizaciones están reemplazando sus centrales telefónicas analógicas PBX (Private Branch eXchange) [1] por centrales telefónicas que utilizan IP (Internet Protocol) [2]. Asterisk [3] es un software de código de fuente abierto de una central telefónica IP-PBX [4].
Si bien este sistema incluye entre sus características la posibilidad de dejar registros de cada llamada y hacer restricciones de acuerdo al teléfono desde el cual la llamada es originada, el mismo deja sus registros en un archivo o en una base de datos pero con una estructura predefinida [5]. Además, en el registro donde queda asentado quien originó la llamada, se especifica el interno desde el cual la misma fue realizada, pero no el usuario que la hizo. En estos casos, cuando se encuentra en los registros una llamada innecesaria, comúnmente, el responsable del interno alega que otra persona hizo uso del teléfono.
El objetivo de este trabajo ha sido complementar lo ofrecido por Asterisk y crear un perfil para cada usuario, independiente del teléfono desde el cual se realizó la llamada, también establecer en el mismo las restricciones y permisos del usuario de acuerdo al tipo de llamadas que necesita realizar para desempeñar sus tareas, así como dejar registro de todas las llamadas en duración, número destino, usuario que realizó la llamada e interno desde el cual se efectuó.
Para lograr esto se vinculó a Asterisk con un software que interactúa con un motor de base de datos, de forma tal que en cada llamada se autentifique el usuario que está realizando la llamada, sus permisos y, si la misma fue permitida, se deje registro en la base de datos de los detalles de ésta. De aquí en adelante se llamará a este software CallControl.
Este artículo se organiza de la siguiente forma: en el punto 2 se explican los detalles de la implementación del software, el punto 3 está dedicado a la Interface del Gateway Asterisk, la sección 4 describe el entorno de desarrollo mientras que la 5 presenta un esquema del sistema, la 6 los distintos pasos del desarrollo y la 7 vincula el software creado con el plan de numeración de Asterisk. Finalmente se presentan las conclusiones y trabajos futuros.

2. Implementación de CallControl

Para facilitar la comprensión del funcionamiento del software desarrollado se describirá brevemente el mecanismo con el cual Asterisk maneja cada llamada.
En Asterisk el comportamiento que se le da a las llamadas, tanto entrantes como salientes de cualquiera de los canales o protocolos, es manejado por el plan de numeración (dial plan). Con éste se realiza el control de flujo de ejecución para todas las operaciones. El mismo está definido en uno de sus archivos

de configuración a través de un “lenguaje” propio que se divide en contextos, extensiones, prioridades y aplicaciones.

- **Contextos**

En cada contexto se define un conjunto de extensiones visibles entre sí. Extensiones de distintos contextos no se pueden ver entre sí salvo que importemos un contexto dentro de otro. La separación de las extensiones en distintos contextos permite regular el acceso de distintos clientes a distintos tipos de llamadas.
Un contexto se define en el archivo **extensions.conf** poniendo el nombre del contexto entre corchetes. Por ejemplo:

[llamadas_entrantes]

Todas las instrucciones situadas después de una declaración como la anterior, pertenecen a ese contexto hasta que se defina otro.

- **Extensiones**

Una extensión es un conjunto de instrucciones que Asterisk seguirá cuando alguien digite el número de esta extensión dentro del contexto al cual pertenece. Como se dijo, dentro de cada contexto se pueden definir muchas extensiones.
La sintaxis para definir una extensión es la siguiente:

exten => *número,prioridad,aplicación()*

Cuando algún cliente, configurado para que sus llamadas lleguen al contexto donde está definida la extensión, digite el número ***número***, se ejecutara la aplicación ***aplicación()***.
En la misma extensión puede haber más de una aplicación. Las mismas se ejecutarán secuencialmente, de acuerdo a la prioridad.

- **Prioridades**

Cada extensión puede tener muchas instrucciones, las cuales se ejecutarán en orden descendente de su prioridad. Ejemplo:

exten => 500,1,Answer( )
exten => 500,2,Hangup( )

- **Aplicaciones**

Cada aplicación, tales como las mencionadas Answer() o Hangup() realiza una determinada acción sobre un canal Existen muchas más. Una de las más importantes es la aplicación Dial(). La misma permite conectar al cliente que digitó la extensión, con el canal que se le pase como argumento. Ejemplo:

exten => 123,1,Dial(SIP/cliente1)

3. AGI - Asterisk Gateway Interface

Una de las funcionalidades que ofrece esta IP-PBX, y que se aprovechará a lo largo de este trabajo para lograr el objetivo, es la integración del plan de numeración con otro software. El mismo puede ser

llamado desde el plan de numeración cuando el usuario realiza alguna llamada en particular o se da alguna situación en especial.
Este mecanismo, a través del cual Asterisk se comunica con un software independiente, se llama AGI - Asterisk Gateway Interface [6]. El mismo consiste en un conjunto de interfaces que le permiten al programa, comunicarse y controlar el comportamiento del sistema. Uno de los usos más comunes de esta característica es vincular este software a un motor de base de datos a los fines de lograr la integración del sistema a una base de datos relacional.
Un programa que se vincula con Asterisk debe ser llamado de la siguiente manera desde el plan de numeración:

exten => 727,1,AGI(nombreAplicacion,arg1,arg2,…,argN)

En este caso se ha hecho uso de esta vinculación para tener un control flexible y dinámico de las restricciones y permisos de los usuarios con una base de datos montada en un motor SQLite [7], variante de SQL.

4. Entorno de Desarrollo

CallControl fue desarrollado en el lenguaje de programación Python [8], que posee una fuerte orientación a objetos, es flexible y permite rápidos desarrollos de aplicaciones. Se hizo uso de las librerías Pyst [9], conjunto de interfaces escritas en Python, para interconectar el plan de numeración a la nueva aplicación.
Python además de darle al software la orientación a objetos deseada permite su total integración con el sistema operativo Linux y las librerías elegidas para trabajar con la base de datos (SQLite).
SQLite no es un sistema de gestión de base de datos cliente-servidor, ni es un proceso independiente con el que el programa principal se comunica, sino un conjunto de librerías que se enlaza con la aplicación, pasando a ser parte integral de la misma. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un único fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.
Como el servidor Asterisk que corre este aplicación no está pensando para un elevado número de usuarios, no es necesario hacer uso del modelo cliente servidor. De esta manera, teniendo en cuenta el bajo consumo de recursos de este “motor” y la fácil integración con Python, queda justificada la elección de SQLite.

5. Esquema del Sistema

En la figura 1 se muestran las partes constitutivas del sistema. Los tres bloques básicos son:

- Servidor Asterisk, con el plan de numeración.
- CallControl, aplicación desarrollada y descripta en este trabajo.
- Base de datos relacional SQLite.

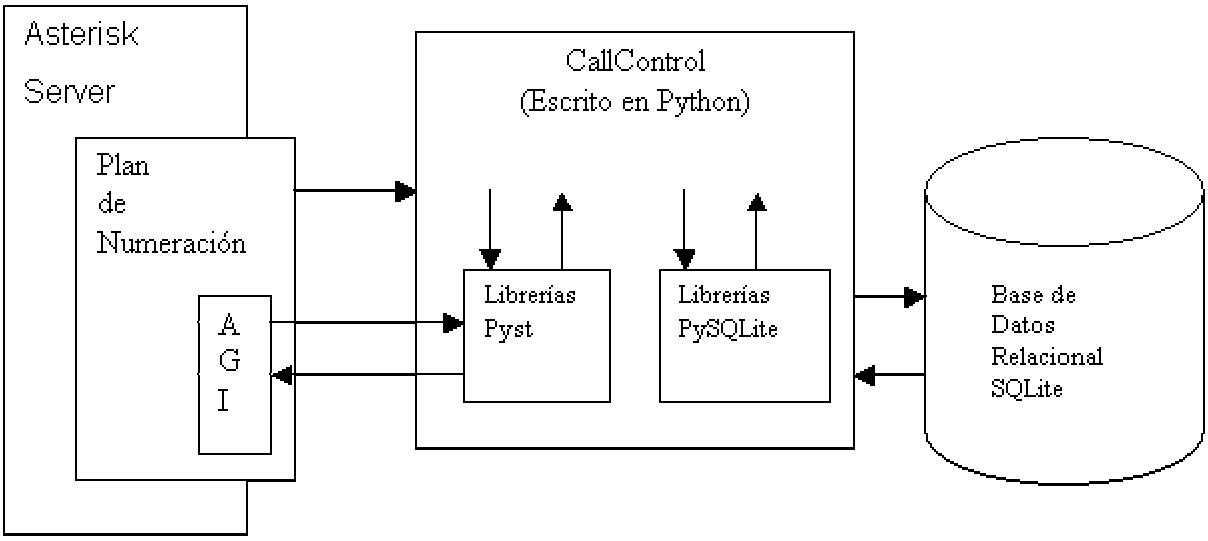

Figura 1- Esquema del Sistema

6. Desarrollo del Sistema

6.1 Obtención de los requerimientos

Para la obtención de los requerimientos se hizo uso del proceso JAD **(Joint Application Development),** que es una técnica exploratoria popular desarrollada inicialmente por IBM al final de 1970 y que luego se expandió y perfeccionó [10]. La misma incluye a los usuarios como participantes activos en el proceso de desarrollo. En este caso el usuario es la persona que administra los sistemas de telefonía de la Universidad Nacional de Córdoba, quien conoce precisamente los requerimientos usuales de los usuarios finales.
Es importante destacar los fundamentos principales de este proceso.

- Las personas que llevan a cabo un trabajo tienen la mejor comprensión del mismo.
- Las personas con conocimientos de las tecnologías tienen la mejor comprensión de las posibilidades de esas tecnologías.
- Los procesos del negocio y las tecnologías raramente se encuentran aislados y generalmente trascienden los límites de cualquier sistema u oficina y afectan el trabajo en departamentos relacionados. La gente que trabaja en estas áreas relacionadas tiene una percepción valiosa del papel del sistema en el entrono donde deberá funcionar.

6.2 Documentos de requerimientos

Mediante el proceso de obtención de los requerimientos se confeccionó el documento de requerimientos. En el mismo se describen los requerimientos funcionales y no funcionales que debe tener el sistema. Los primeros hacen referencia a las interacciones de los usuarios con el mismo y los segundos a algunas características deseables en cuanto a rendimiento y disponibilidad.

6.3 Casos de uso del sistema

Del documento de requerimientos del sistema se plasmaron los casos de uso.
En la redacción de los documentos se evitó el lenguaje técnico para que puedan ser completamente entendidos por usuarios finales.
Se utilizaron los casos de uso como herramientas simples para describir el comportamiento deseado del sistema. En la figura 2 se puede ver un diagrama de casos de usos que muestra las interacciones del sistema completo con el usuario final.

Figura 2- Casos de Uso

En las figuras 3, 4 y 5 se pueden observar las interacciones de Asterisk con el plan de numeración y el software desarrollado (CallControl) en distintos modos de funcionamiento.

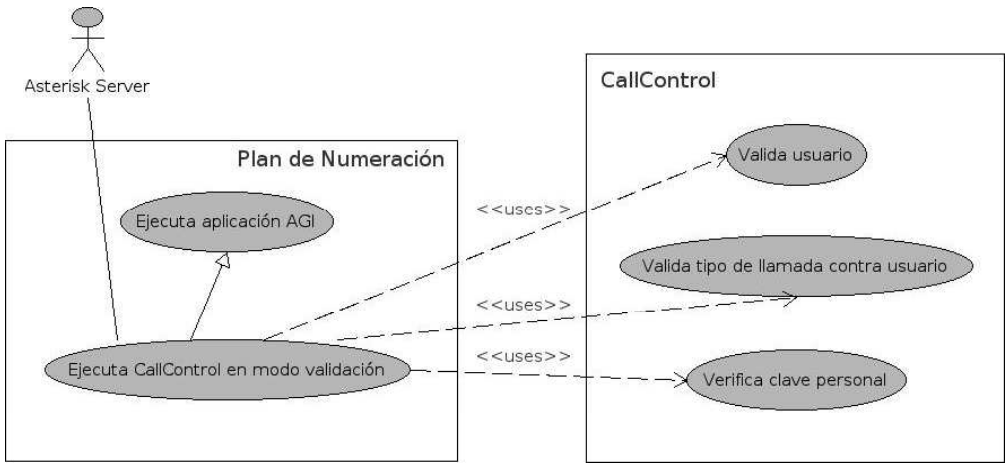

Figura 3- Caso de uso de CallControl: validación de usuarios

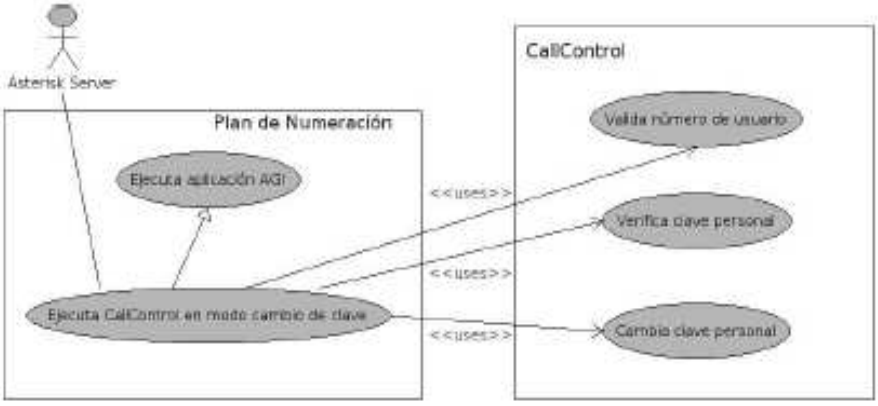

Figura 4- Caso de uso de CallControl: cambio de clave

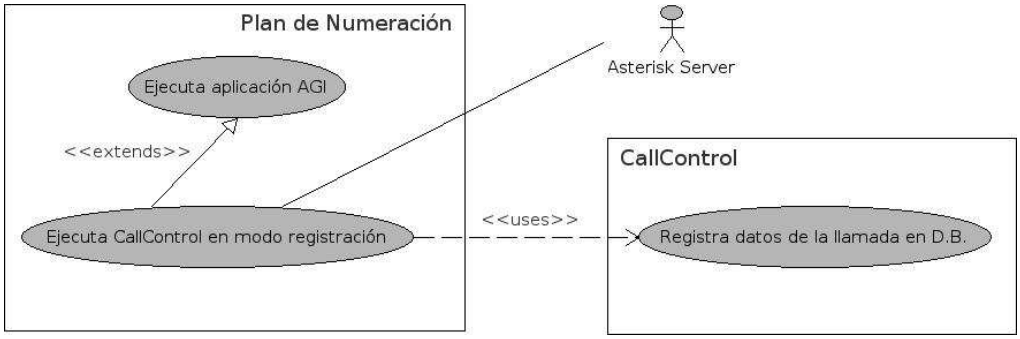

Figura 5- Caso de uso de CallControl: registro de datos de la llamada

A través de los casos de uso se puede apreciar que CallControl debe ser capaz de funcionar de tres maneras independientes:

- Cuando un usuario realiza una llamada que requiere ser autenticada se llama a la aplicación CallControl desde el plan de numeración de Asterisk pasándole como argumento el tipo de llamada que el usuario está intentando realizar.
  Entonces, la aplicación CallControl deberá realizar los siguientes pasos:
  1. Solicitarle al usuario que ingrese su número de usuario.
  2. Verificar si este es un usuario que se encuentra en el sistema.
  3. Verificar si el tipo de llamada que quiere realizar le está permitida.
  4. Solicitarle su clave personal.
  5. Verificar la validez de la clave ingresada.
  6. Devolverle al plan de numeración algún valor que le informe si el usuario podrá o no realizar la llamada.
  7. En caso que la llamada pueda ser realizada, la aplicación CallControl deberá informar al plan de numeración el número de usuario que va a realizar la llamada.

- Cuando un usuario quiere cambiar su clave personal llama a un número predefinido. Este suceso llama a la aplicación CallControl desde la extensión correspondiente a ese número del plan de numeración, en otro modo de ejecución. Los pasos que seguirá CallControl en este caso serán:
    1. Solicitarle al usuario que ingrese su número de usuario y clave personal actual.
    2. Verificar la validez de los datos previamente ingresados.
    3. Solicitarle al usuario que ingrese su nueva contraseña.
    4. Informarle al usuario que su clave ha sido cambiada.

- Una vez que el usuario terminó una llamada, la aplicación CallControl es llamada pasándole como argumentos la fecha/hora de inicio, fecha/hora de fin, duración de la misma, número de usuario que realizó la llamada, interno desde la cual fue realizada y número destino. Esta información deberá ser vertida por CallControl en la base de datos.

7. Vinculación entre CallControl y el Plan de Numeración

Para describir la forma en que usará el software desarrollado, se debe saber conocer qué es y cómo funciona una macro en Asterisk.

7.1 Macros en Asterisk

Una macro es un tipo especial de contexto, definido por el usuario con el prefijo `macro-`. Éstos son patrones reutilizables de ejecución, como los procedimientos en un lenguaje de programación [11].

Ejemplo:

```
[macro-contador]
exten => s,1,Set(COUENTA=${ARG1})
exten => s,2,While($[ ${COUENTA} > 0])
exten => s,3,SayNumber(${COUENTA})
exten => s,4,Set(COUNT=$[ ${COUENTA} - 1 ])
exten => s,5,EndWhile( )

[micontexto]
exten => 123,1,Macro(contador,10)
```

En el ejemplo anterior se define una macro que cuenta en forma descendente el número que se le pasó como argumento. La macro debe ser llamada con la aplicación `Macro()`, como se puede ver en el contexto de ejemplo: `[micontexto]`. El primer argumento de esta aplicación, es el nombre de la macro a ejecutar, seguido de los argumentos que se le pasan a la macro.
Los argumentos **arg1**, **arg2**, …, **argn** que se le pueden pasar a la macro se convierten en la variables **${ARG1}**, **${ARG2}**, …,**${ARGN}**, dentro del contexto de la macro.
La macro retorna -1 si falla en cualquiera de los pasos, o 0 en otro caso.

7.2 Vinculación de los modos de ejecución de la aplicación CallControl con una macro

Con la breve explicación anterior del funcionamiento de una macro se puede entender cómo se utilizó esta herramienta para que las llamadas que requieren autenticación por parte de los usuarios pasen a través de la aplicación CallControl.
La macro está codificada de la siguiente forma:

```
 [macro-authLogCall]

exten => s,1,Set(ORIG=${CHANNEL:11:3})
exten => s,2,AGI(callcontrol.py,vu,${ARG1})
exten => s,3,GotoIf($[${VALIDCALL} = 0]?4:7)
exten => s,4,Dial(${ARG2}/${ARG3},10)
exten => s,5,Goto(s-${DIALSTATUS},1)
exten => s,6,Hangup( )
exten => s,7,Playback(vm-nopermison)
exten => s,8,Hangup( )

exten => s-BUSY,n,Playback(vm-isunavail)
exten => s-BUSY,n,Hangup( )

exten => s-NOANSWER,n,Playback(vm-isunavail)
exten => s-NOANSWER,n,Hangup( )

exten => s-CHANUNAVAIL,n,Playback(vm-isunavail)
exten => s-CHANUNAVAIL,n,Hangup( )

exten => h,1,GotoIf($[$[${VALIDCALL} = 0] & $["${DIALSTATUS}" = "ANSWER"]]?2:3)
exten => h,2,DeadAGI(callcontrol.py,lc,"${USERNUMBER}","${CDR(start)}",
                          "${CDR(end)}",${CDR(duration)},1,${ARG3:7:3},${ORIG})
exten => s,3,Hangup( )
```

Se comienza salvando en la variable `ORIG` el interno desde el cual fue originada la llamada. Luego se llama a CallControl pasándole el primer argumento de la macro, que es el tipo de llamada que el usuario intenta realizar. Al terminar la aplicación, CallControl dejará inicializada la variable `VALIDCALL`, indicándole a la macro si la llamada es válida o no. En caso afirmativo, se llamará a la aplicación `Dial()`, pasándole como argumentos el canal al que se llamará y el número a discar.
En el caso que la variable `VALIDCALL` indique que la llamada no es válida, el flujo de ejecución saltará a la prioridad 7, donde se le indicará al usuario que no puede realizar esa llamada, y luego se cortará el canal.
Cuando la aplicación `Dial()` termina, se inicializa la variable `DIALSTATUS` con alguno de los siguientes valores: `ANSWER`, `BUSY`, `NOANSWER` o `CHANUNAVAIL` de acuerdo a la condición del canal, luego se llama a la aplicación `goto()`, pasándole como argumento `s–${DIALSTATUS}`. Si `${DIALSTATUS}` tiene el valor `ANSWER`, el flujo de ejecución no salta y se ejecuta la próxima aplicación, que simplemente corta el canal y finaliza la llamada normalmente. En el caso de que `${DIALSTATUS}` tenga alguno de los otros valores, se llevará el flujo de ejecución a alguna de las siguientes extensiones: `s-BUSY`, `s-NOANSWER` o `s-CHANUNAVAIL`. En cualquiera de estas extensiones se le informará al usuario que el canal no esta disponible y luego se cortará la llamada.
En cualquier momento el flujo de ejecución de la macro puede saltar a la extensión **h** [12], si se da alguna de las siguientes condiciones: el usuario que originó la llamada o el que la recibió, la corta; o la

macro llega a un punto que llama la aplicación `Hangup()`. En esta extensión se llama a la aplicación CallControl en modo registración, pasándole las fechas y horas de inicio y fin de la llamada, la duración, el número de usuario que la realizó, el interno desde el cual fue realizada (que fue previamente guardada en la variable `ORIG`), y el número de destino. En este caso se usa la aplicación `DeadAgi()`para llamar a CallControl.
Antes de llamar a la aplicación CallControl en esta extensión, se pregunta si se pudo realizar la llamada verificando las variables `VALIDCALL` y `DIALSTATUS`, para no cargar la base de datos con llamadas que no se llegaron a realizar.
De igual forma se puede crear una macro que no requiera de la autenticación del usuario para realizar llamadas, pero que sí deje registro en la base de datos de todos los datos, menos el número de usuario. La misma se define de la siguiente forma.

```
[macro-onlyLogCall]

exten => s,1,Set(ORIG=${CHANNEL:11:3})
exten => s,n,Dial(${ARG1}/${ARG2},10)
exten => s,n,Goto(s-${DIALSTATUS},1)
exten => s,n,Hangup( )

exten => s-BUSY,n,Playback(vm-isunavail)
exten => s-BUSY,n,Hangup( )

exten => s-NOANSWER,n,Playback(vm-isunavail)
exten => s-NOANSWER,n,Hangup( )

exten => s-CHANUNAVAIL,n,Playback(vm-isunavail)
exten => s-CHANUNAVAIL,n,Hangup( )

exten => h,1,GotoIf($["${DIALSTATUS}" = "ANSWER"]?2:3)
exten => h,2,DeadAGI(callcontrol.py,lc,"000","${CDR(start)}",
"${CDR(end)}",${CDR(duration)},1,${ARG2:7:3},${ORIG})
exten => s,3,Hangup( )
```

7.3. Uso de las macros macro-authLogCall y macro-onlyLogCall

A continuación se muestra cómo se utilizan las macros previamente definidas con un ejemplo.

```
[intsAndPstnRes]

include => onlyInts

exten => _0351XXXXXXX,1,Macro(authLogCall,1,Zap/g1,${EXTEN:4:7})
exten => _0XXXXXXX,1,Macro(authLogCall,1,Zap/g1,${EXTEN:1:7})

exten => _035115XXXXXXX,1,Macro(authLogCall,4,Zap/g1,${EXTEN:4:7})
exten => _015XXXXXXX,1,Macro(authLogCall,4,Zap/g1,${EXTEN:1:7})

exten => _011XXXXXXXX,1,Macro(authLogCall,3,Zap/g1,${EXTEN:4:7})
exten => _01115XXXXXXXX,1,Macro(authLogCall,5,Zap/g1,${EXTEN:4:7})
```

```
[intsAndPstnUnres]

include => onlyInts

exten => _0351XXXXXXX,1,Macro(onlyLogCall,Zap/g1,${EXTEN:4:7})
exten => _0XXXXXXX,1,Macro(onlyLogCall,1,Zap/g1,${EXTEN:1:7})

exten => _035115XXXXXXX,1,Macro(onlyLogCall,4,Zap/g1,${EXTEN:4:7})
exten => _015XXXXXXX,1,Macro(authLogCall,Zap/g1,${EXTEN:1:7})

exten => _011XXXXXXXX,1,Macro(onlyLogCall,Zap/g1,${EXTEN:4:7})
exten => _01115XXXXXXXX,1,Macro(authLogCall,Zap/g1,${EXTEN:4:7})
```

En este ejemplo, se definen dos contextos, `[intsAndPstnRes]` y `[intsAndPstnUnres]`. En los mismos, se definen patrones de extensiones. Cuando un usuario realiza una llamada a la red de telefonía pública, el número marcado concordará con alguno de los patrones de acuerdo al prefijo de la localidad o tipo de número que el usuario digite.
En cualquiera de los patrones de extensión que se corresponda con la llamada, se ejecutará la aplicación macro, que a su vez será la encargada de llamar a la macro previamente definida.
De acuerdo al contexto donde llegue la llamada que está realizando el usuario, se ejecutará la macro `authLogCall` o `onlyLogCall.` En el caso que se llame a la macro `authLogCall`, se le pasará como primer argumento la categoría de llamada que se está realizando. Por ejemplo, si se está realizando una llamada urbana, se le pasará 1, si se está realizando una llamada a teléfono móvil, se le pasará 2, etc. Esto se corresponde con una tabla de la base de datos, donde están todas las categorías de llamadas. Existe otra tabla en la base, donde se encuentran los identificadores (IDs) de usuarios e IDs de categoría. Esta tabla es consultada por la aplicación CallControl para verificar si el usuario puede realizar la llamada. El segundo argumento que recibe esta macro es el canal sobre el cual se realizará la llamada y el tercero es la parte útil de la extensión que digitó el usuario, que será utilizada para discar sobre el canal previamente pasado como argumento. Por ejemplo, si el usuario digitó 03514815648 para una llamada urbana, sobre el canal se discará 4815648 ya que el prefijo 0351 no es necesario.
En el caso que se llame a la macro `onlyLogCall`, se le pasarán los mismos argumentos que a la macro anterior menos la categoría de llamada, ya que esta macro no autentica usuarios, solamente deja registros en la base de datos.
Las llamadas originadas en los internos llegarán a los contextos `[intsAndPstnRes]` o `[intsAndPstnUnres]` de acuerdo a cómo estén configurados en el archivo `sip.conf` de Asterisk.
Por último, en ambos contextos se incluye el contexto `[onlyInts]`, que permite a cualquier interno alcanzar otro interno sin necesidad de autenticar este tipo de llamada.

8. Conclusiones y Trabajos Futuros

Mediante el software desarrollado se cubre un servicio necesario en los sistemas de telefonía compartidos, logrando control por parte del mismo usuario en el uso de las líneas telefónicas.
A lo largo del trabajo se mostró como se vinculó la aplicación CallControl con los servicios y aplicaciones que brinda Asterisk junto a la base de datos, de manera sencilla y dejando puertas abiertas a implementaciones futuras de mayor escala.

Se verificó el adecuado funcionamiento de las interacciones de los usuarios con el sistema, mediante el uso de casos de prueba, que arrojaron resultados satisfactorios.
Es importante destacar que este sistema ya está siendo utilizado en una dependencia de la Universidad Nacional de Córdoba con excelentes resultados hasta el momento. Se llegó a la conclusión de que el sistema puede ser instalado y usado por otras unidades académicas.
Se pretende agregar al sistema otras funcionalidades, tales como "salas de conferencias", donde los usuarios se pueden encontrar y dialogar juntos. También "directorio", que permite en sistemas con muchos usuarios buscarlos a través del nombre con que están registrados en archivos.
Actualmente se esta trabajando en la implementación de una interfaz web que permita a un administrador con pocos conocimientos del sistema consultar la base de datos extrayendo estadísticas del uso de las líneas telefónicas, como así también detectar algún mal uso de las mismas.
También se pretende lograr que esta interfaz permita agregar, gestionar y quitar información de los usuarios como así también restricciones.

Referencias

[1] B. Khasnabish, *"Implementing Voice over IP"*, Wiley, 2003.
[2] RFC 791 – Internet Protocol.
[3] J. Van Meggelen, J. Smith y L. Madsen, *"Asterisk, the future of telephony"*, O'Reilly Media, 2005.
[4] C. Krishna Sumanth, J. How, *"Integration of Open Source and Entreprise IP-PBXs"*, Dalhousie University, Halifax, 2006.
[5] Asterisk.org *"Features and Architecture of Asterisk PBX"*, http://www.asterisk.org/features.
[6] Voip-Info.org, *"A reference guide to all things VOIP"*, http://www.voip-info.org/wiki-Asterisk+AGI
[7] SQLite, Official Website, http://www.sqlite.org
[8] Python Programming Language, Official Website, http://python.org
[9] Pyst: Python for Asterisk, http://sourceforge.net/projectx/pyst
[10] Mei Yatco, *"Joint Application Design/Development"*, University of Missouri, 1999, http://www.umsl.edu/~sauterv/analysis/JAD.html
[11] VoIP en Español – El Dialplan de Asterisk, http://voip.megawan.com.ar/doku.php/asterisk_configuracion_extensions.conf
[12] Voip-Info.org, *A reference guide to all things VoIP*, http://www.voip-info.org/wiki/view/Asterisk+h+extension