ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

INTEGRACIÓN DE UN SISTEMA PARA CONSULTA DE NOTAS VÍA LÍNEA TELEFÓNICA CON RECONOCIMIENTO DE VOZ

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

FUENTES MEDINA ANDRÉS FERNANDO

DIRECTOR: MSc. XAVIER CALDERÓN

Quito, Diciembre 2007

DECLARACIÓN

Yo, Andrés Fernando Fuentes Medina, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Andrés Fuentes

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Andrés Fuentes, bajo mi supervisión.

_____

MSc. Xavier Calderón

DIRECTOR DEL PROYECTO

AGRADECIMIENTO

A quien me regaló una vida llena de amor y comprensión, mi madre; y de manera especial a dos grandes personas que compartieron un espacio de sus vidas conmigo, mis tíos Narciza y Wilmer.

DEDICATORIA

Por las buenas costumbres que me inculcaste cuando vivimos juntos y por tantas lecciones que me diste con tu partida, este trabajo es para ti papá.

INDICE DE CONTENIDOS

INDICE DE ANEXOS

INDICE DE FIGURAS

INDICE DE TABLAS

INDICE DE ESPACIOS DE CÓDIGO

RESUMEN

El presente proyecto trata sobre el diseño e implementación de un sistema de atención automática vía línea telefónica que incorpora reconocimiento y síntesis de voz, para hacer conocer las calificaciones a los estudiantes de la Carrera de Ingeniería en Electrónica y Redes de Información de la Escuela Politécnica Nacional, además de información asociada a sus planes de proyectos de titulación y certificados de prácticas preprofesionales.

En el ***Capítulo I*** se estudia las prestaciones y aceptación que están ganando los sistemas automatizados que involucran la poca o nula intervención del ser humano en la creación de servicios que pueden accederse a través de la telefonía tradicional. Se da una breve descripción de las tecnologías actuales de reconocimiento natural de voz. También, se hace un estudio de los lenguajes de etiquetado más utilizados que se han diseñado para la creación de estas aplicaciones: SRGS (*Speech Recognition Grammar Specification*), SISR (*Semantic Interpretation for Speech Recognition*), VXML (*Voice Extensible Markup Language*) y SALT (*Speech Application Language Tags*). Al final se describe brevemente las características del software de código abierto que ha revolucionado los sistemas de telefonía, Asterisk; y también dos motores de reconocimiento natural de voz.

A lo largo del ***Capítulo II*** se diseña el sistema integrado de consulta de notas y servicios adicionales. Se parte describiendo las necesidades que se deberá solventar a partir de las cuales se crea el flujo del diálogo guiado. Se analizan los patrones de reconocimiento tomando en cuenta las distintas maneras de

pronunciar el número de identificación de los estudiantes en la Universidad. El acceso a la base de datos que almacena la información de los estudiantes y sus notas se lo hace a través de procedimientos almacenados, que se describen junto con el diseño de una nueva base de datos que almacenará la información de planes de proyectos de titulación y certificados de prácticas preprofesionales. Al final se incorporan todos los elementos anteriores en el sistema de atención automática, que ofrece una interacción con el usuario ya sea mediante reconocimiento natural de voz o la pulsación de las teclas del teléfono.

Todo este diseño se plasma en un sistema real en el ***Capítulo III*** en donde se trata la implementación del mismo. Partiendo de la descripción de los paquetes de software necesarios para desarrollar la aplicación, se analizan los requerimientos de hardware. Luego se describe paso a paso la instalación de cada paquete así como también la configuración de la base de datos para que soporte los nuevos servicios. Se analiza con detalle la implementación del sistema de atención automática en el servidor de telefonía, tanto el flujo del diálogo como la creación de la gramática de reconocimiento de la aplicación y la interacción con la base de datos. Al final se da una descripción del funcionamiento del sistema, los resultados de las pruebas realizadas y el análisis de costos.

En el ***Capítulo IV*** se dan las conclusiones y recomendaciones más relevantes, fruto de la elaboración del presente proyecto.

PRESENTACIÓN

Ante el limitado acceso al Internet en países como el nuestro, las empresas han buscado nuevas formas de llegar a sus clientes y han acertado en hacerlo a través del dispositivo de comunicación más común del planeta, el teléfono.

Sin duda alguna, la interfaz que ofrece una llamada telefónica es muy limitada si se la compara con el Internet, sin embargo, en la mayoría de ocasiones lo que se necesita es solamente una respuesta inmediata, conocer un saldo bancario, inscribirse en un curso, conocer los resultados de un examen, solicitar servicio de comidas a domicilio o reservar una cita con el médico, y si estamos en la calle es más fácil hacerlo desde un teléfono celular, o desde un teléfono fijo si estamos en casa o en la oficina.

Con la creación de sistemas automatizados se logra incrementar los horarios de atención, disminuyendo significativamente los costos de mantener los mismos sistemas con recursos humanos. Aunque el descontento por parte de los usuarios se manifiesta enseguida debido a los menús repetitivos hasta alcanzar el servicio que se desea utilizar, las limitaciones del teclado del teléfono vuelven la comunicación tediosa y poco amigable.

El objetivo es convertir la interacción entre el usuario y la operadora automática como una conversación entre dos personas, hecho que se logra desde el momento que el sistema se pueda adaptar a la manera de comunicarse del ser humano, el lenguaje oral. A pesar del continuo desarrollo de la ciencia y la tecnología queda aún mucho camino por recorrer para alcanzar excelentes

resultados en este campo, puesto que a los ordenadores aún no se les logra incorporar la capacidad de interpretar el entorno con la sencillez que nosotros los humanos lo hacemos. Sin embargo, en la actualidad ya es posible añadir en los sistemas de atención automática la funcionalidad de reconocimiento natural de voz, siempre y cuando se lleve al usuario por un diálogo guiado.

Este avance da a los sistemas de atención automática una mayor flexibilidad para entender lo que el usuario necesita, de una manera directa y mucho más ágil. El problema que surge es su costo elevado, que hace que estos sistemas estén generalmente destinados al mercado de las grandes empresas.

Con el afán de lograr una mejor integración de las soluciones de distintos fabricantes se ha estandarizado mediante lenguajes de etiquetado la manera de programar estas aplicaciones, además, la popularidad que en la actualidad va ganando el software de código abierto mediante las comunidades de desarrolladores y usuarios a nivel mundial, permite crear grandes sistemas telefónicos a costos relativamente bajos, accesibles por las empresas de nivel medio, que son las más comunes en nuestro país.

En las siguientes páginas se da una alternativa a aquellas soluciones costosas, con resultados similares pero a un precio mucho menor. El sistema que se presenta tiene por objetivo proporcionar las calificaciones a los estudiantes de la carrera de Electrónica y Redes de Información de la Escuela Politécnica Nacional. Se proporcionará también información correspondiente a los planes de proyectos de titulación y certificados de prácticas preprofesionales que se entregan en la coordinación de la carrera para que sean revisados y aprobados.

Capítulo I

1 RECONOCIMIENTO NATURAL DE VOZ EN SISTEMAS DE ATENCIÓN AUTOMÁTICA

1.1 EVOLUCIÓN DE LOS SISTEMAS TELEFÓNICOS DE ATENCIÓN AUTOMÁTICA

Desde la invención del Internet se ha podido acceder a la información de una manera ágil y oportuna, las empresas han encontrado en éste un excelente medio para promocionar sus productos y servicios, sin embargo su penetración en la población mundial continúa a paso lento. Según las Estadísticas Mundiales del Internet (*Internet World Stats*[1]), hasta agosto del 2007, apenas el 12,8% de la población del Ecuador tiene acceso al Internet. Este hecho limita la promoción de productos a través de este medio desde que las empresas están empeñadas en incorporar nuevos mercados y brindar más y mejores servicios a sus clientes.

Por otra parte, la telefonía tradicional ha logrado una gran difusión en todas las regiones del planeta y se ha puesto empeño en explotar aún más esta tecnología. Así, los primeros sistemas de atención al público lo constituyen los llamados *"Call Centers"* en los que simplemente se tiene un conjunto de operadores humanos que atienden a los llamantes mediante un sistema establecido de colas. Varios inconvenientes surgen al atender a una gran cantidad de usuarios, el aumento de operadores se vuelve poco práctico a la vez que insostenible, además los usuarios no encontraban satisfactorio el servicio, la espera excesiva en la cola la abandonaba el mismo llamante al no tener una pronta respuesta, se suma a esto la inflexibilidad del horario de atención, pues depende siempre del horario de trabajo de los operadores.

La idea ha sido ofrecer un servicio continuo las veinticuatro horas del día durante los siete días de la semana, pero un servicio así con operadores humanos excedería significativamente el presupuesto de cualquier empresa. Se ha optado por implementar operadores virtuales que suplan en la medida que fuese posible

[1] http://www.internetworldstats.com

las acciones humanas, este fue el inicio de la integración de la telefonía con la computación. Mayores inconvenientes surgen ahora cuando queda en consideración la limitada interfaz que ofrece una llamada telefónica al tratar de comunicar un ser humano con una máquina. El uso del teclado limita digitar uno o varios números cada vez que el sistema solicita escoger una opción, ingresar un código o una cantidad; esto convierte a la interacción hombre-máquina en una repetición de menús poco flexibles que por su monotonía el usuario prefiere no volver a utilizar.

La implementación de un sistema de atención a usuarios debería enmarcar el principio de que el sistema debe adaptarse al usuario y no el usuario al sistema. Considerando esto se llega a la necesidad de interactuar de la manera más habitual para el ser humano, el habla. El reconocimiento del habla es una tecnología en evolución y está ganando gran aceptación a nivel mundial, los usuarios simplemente ordenan al sistema que es lo que necesitan y se evitan el recorrido por menús redundantes, reduciendo el tiempo efectivo para conseguir sus propósitos.

A todo lo anterior se suma el hecho de integrar a la telefonía el poder del Internet: acceso a bases de datos, promoción de bienes, encuestas, servicios de noticias, correo electrónico e información en general, mediante la creación de sistemas unificados de servicios a partir de tecnologías Web.

1.1.1 SISTEMAS DE RESPUESTA DE VOZ INTERACTIVA

Un sistema de respuesta de voz interactiva (IVR, *Interactive Voice Response*) es un sistema computarizado que permite a un usuario ser atendido por una operadora virtual a través de una línea telefónica. El sistema interactúa con el usuario mediante la reproducción de archivos de audio o voz sintetizada (TTS, *Text to Speech*), y el usuario con el sistema mediante la pulsación de las teclas del teléfono y la pronunciación de órdenes, que es la detección de tonos *DTMF* (*Dual Tone Multi Frecuency*) y el reconocimiento del habla respectivamente.

En la actualidad los IVRs se han convertido en la interfaz de los *Call Centers* para los usuarios, reciben su información y dependiendo de lo que deseen hacer atienden de manera eficiente la solicitud o transfieren la llamada a un operador humano. Un IVR puede constituir un sistema de acceso a la banca electrónica en donde los usuarios se identifiquen y realicen consultas de saldos, transferencias, pagos de servicios, etc. o un simple sistema de directorio telefónico de una empresa en donde se levante el teléfono, se marque un número de acceso al IVR y se diga el nombre de la persona con la que se desea comunicar, para que el sistema automáticamente enrute la llamada.

1.1.1.1 Ventajas

Un IVR comparado con un tradicional *Call Center* presenta las siguientes ventajas:

- Acceso inmediato
- Horario extendido
- Reducción de costos de operación
- Incremento en la calidad del servicio

Lo más tedioso para los usuarios de los tradicionales *Call Centers* es la espera en la cola hasta que la primera operadora se desocupe y pueda atenderlos. Este intervalo depende del número de usuarios potenciales, el número de operadores activos, el horario en el que se concentren el mayor número de llamadas y el tiempo de atención promedio por usuario. Cuando el tiempo de espera supera la paciencia del llamante es muy común que él concluya la llamada mostrando efectivamente su descontento con el servicio, lo que hace que deje de utilizarlo.

Con un IVR no existen las tediosas colas, el usuario ingresa directamente a ser atendido, sin embargo, el número máximo de llamantes utilizando el servicio simultáneamente depende de las características de hardware y software del sistema, por ejemplo, puede suceder que un motor de reconocimiento de voz limite el número de conexiones simultáneas dependiendo de las licencias que se

hayan adquirido o que cuando se alcance cierto límite de conexiones el hardware empiece a mostrar retardos que disminuyan la calidad del IVR.

Es usual que un *Call Center* ofrezca su servicio durante un horario limitado, el mismo que está determinado por las horas de trabajo de los operadores. Ofrecer a los usuarios un horario extendido equivale a contratar más operadores, no es buena idea incrementar las horas de trabajo de los mismos empleados pues el cansancio humano empieza a manifestarse y puede suceder que las llamadas dejen de ser contestadas o la atención no sea cordial y amable, por ende, una disminución en la calidad del servicio. Un IVR puede trabajar las veinticuatro horas del día durante los siete días de la semana sin interrupción, atendiendo el mismo número de usuarios en cualquier horario.

Los puntos anteriores se suman en un incremento en la calidad de servicio del IVR, a la vez que reduce significativamente los costos de mantenimiento, pues se optimizan los recursos. El servicio es continuo y uniforme durante todo el día, durante todos los días.

1.1.1.2 Desventajas

Aunque un IVR presenta mejoras en el servicio y una optimización de recursos puede también presentar serios inconvenientes. Un motor de reconocimiento poco fiable mantendría al usuario repitiendo una misma palabra o frase varias veces hasta conseguir un resultado acertado, un usuario podría desconocer si su teléfono está configurado para marcar por tonos o si marca solamente por pulsos y no obtener resultado alguno cuando ingresa un número pulsando las teclas del mismo, también, el sintetizador de voz (o conversor de texto a voz) quizá no reproduzca un lenguaje entendible a partir del texto. Estos factores conducirían a un descontento total por parte de los usuarios y al no cumplimiento de los objetivos del sistema, pues lo que se trata de ofrecer es un diálogo lo más natural posible, como si se tratara de una conversación entre dos personas. También vale mencionar que un IVR en su más baja apreciación es un sistema compuesto por hardware y software que pueden sufrir fallas o interrupciones del servicio

repentinamente, quizá sin aviso alguno. Implementar un servicio redundante y tolerante a fallos seguramente requerirá una inversión mayor.

1.1.1.3 Consideraciones de diseño

La calidad de un IVR se basa en un diseño adecuado y éste en la consideración de los siguientes puntos:

1.1.1.3.1 Reconocimiento flexible de voz

Una misma palabra en un lenguaje determinado generalmente no es pronunciada de la igual manera por personas provenientes de distintas regiones, o un número telefónico o código de identificación tampoco es memorizado de la misma manera por distintas personas, algunas pueden decirlo dígito a dígito o en combinaciones de dígitos, decenas y centenas. El objetivo básico de un IVR es reconocer la información que el usuario ingresa a través de su voz y para esto el diseñador debe crear los algoritmos o configuraciones necesarias en el motor de reconocimiento para obtener los mejores resultados independientemente de la edad del usuario o su manera de pronunciar las palabras.

1.1.1.3.2 Navegación directa

En los sistemas de atención automática sin reconocimiento de voz el usuario está obligado a digitar varias veces las teclas de su teléfono mientras ingresa por los menús hasta llegar al punto deseado, esta situación se convierte en monótona cuando se necesita ingresar al servicio continuamente. El propósito de la incorporación del reconocimiento de voz a estos sistemas es precisamente para combatir este problema, por ejemplo, si un usuario desea conocer el saldo de su cuenta de ahorros en el servicio telefónico de un banco quizá deberá pulsar tres veces la tecla número *“1”* de su teléfono mientras ingresa: al menú inicial en donde le pueden ofrecer *servicios bancarios*, *indicadores económicos*, *últimas promociones*, y *atención por una operadora*, luego al siguiente menú dentro de *servicios bancarios* que puede contener *consultas*, *transferencias* y *pagos pendientes*; y finalmente *saldos* o *detalle de los últimos movimientos* dentro de *consultas*. Con un buen esquema de navegación por el menú de un IVR con

reconocimiento de voz el usuario quizá solamente tenga que pronunciar *“Consulta de saldos”* y evitarse todo lo anterior.

1.1.1.3.3 Interacción con el usuario

Para que el usuario pueda ingresar correctamente la información requerida debe preguntársele de manera clara y concreta lo que se necesita, la calidad de los archivos de audio y la voz sintetizada debe ser bastante alta, el lenguaje utilizado debe ser lo más natural y sencillo posible. Se deben establecer intervalos cortos de espera por una respuesta antes de insistir en ella y ofrecer asistencia en cualquier momento de la comunicación, quizá con solo pronunciar la palabra *“ayuda”* se debería ingresar a escuchar varias indicaciones del funcionamiento del sistema.

1.1.1.3.4 Limitaciones de hardware y software

La capacidad del hardware y software juegan un papel muy importante en la calidad del servicio de un IVR, si los retardos son altos al reconocer una palabra o una frase el usuario llegaría a sentir que el sistema está consumiendo el tiempo que él quisiera ahorrar, no sería adecuado colocar música en espera mientras el motor reconoce la entrada del usuario. De igual manera si el motor de reconocimiento no ofrece un resultado preciso, esto sucede cuando reconoce un patrón de palabras habiéndose pronunciado efectivamente algo distinto. Si el *IVR* proporciona información almacenada en una base de datos externa pueden existir retardos al consultar o procesar esa información.

1.1.1.4 Parámetros de medición de calidad

La calidad de un IVR se mide básicamente por dos parámetros:

1.1.1.4.1 Nivel de logro de objetivos (NLO)

Corresponde al porcentaje de usuarios que lograron su objetivo al usar el sistema, es el parámetro más importante para medir la calidad de servicio que el *IVR* ofrece.

$$NLO = \frac{\#\text{ de usuarios que fueron atendidos exitosamente}}{\#\text{ de usuarios totales que ingresaron al sistema}} * 100\% \qquad \text{(ecuación 1.1)}$$

Si un usuario ingresa al sistema y no puede ser entendido por el operador virtual o se le proporciona información errónea, obviamente se mostrará molesto y preferirá ya no volver a utilizarlo. Según *Natural Vox*, proveedor de tecnologías y soluciones IVR de España, un *NLO* bajo el 95% debería ser la calidad más baja que un IVR podría ofrecer[2].

1.1.1.4.2 Nivel de diálogo virtual (NVD)

Es la comparación entre el tiempo medio que llevaría atender exitosamente a un usuario con una operadora virtual y el tiempo medio que llevaría al mismo usuario ser atendido por una operadora humana.

$$NDV = \left(\frac{\text{duración media de llamada con operadora virtual}}{\text{duración media de llamada con operadora humana}} \right) \qquad \text{(ecuación 1.2)}$$

Es un parámetro un tanto difícil de medir pero se suele tener un promedio de duración de llamada cuando se usa el servicio de un *Call Center* y se lo compara con el tiempo de duración de las llamadas en el IVR. Un *NDV* de 1 equivale a decir que la atención con operadora virtual y operadora humana consumen el mismo tiempo, en la práctica es recomendable tener un *NDV* menor que 1, con esto se garantiza que tener el IVR es más eficiente que ofrecer el mismo servicio mediante un *Cal Center*.

1.1.2 NAVEGADORES DE VOZ

La tendencia actual de los *IVR* es proporcionar todo el poder del Internet a través de una llamada telefónica, el *Consorcio de la WWW* (W3C[3], *World Wide Web*

---

[2] El *Nivel de Logro de Objetivos* también se lo denomina *Nivel de Consecución de Objetivos*, http://www.natvox.com/faq.html

[3] http://www.w3.org

Consortium) trabaja arduamente para proporcionar una plataforma con interfaz de voz para acceder al Internet mediante la especificación de varios documentos basados en el lenguaje *XML* (*eXtensible Markup Languaje*) para cubrir requerimientos de diálogos de voz, voz sintetizada, reconocimiento de voz, control de llamadas para navegadores de voz, etc.

Posibles aplicaciones incluyen el acceso a servicios de negocios como banca electrónica, reservación de boletos en aerolíneas, servicios a domicilio, asesoramiento especializado; servicios comunitarios como noticias, estado del tiempo, tráfico vehicular; e incluso servicios personales como correo electrónico, agenda personal, contactos, etc.

La *figura 1.1* ejemplifica la posible estructura de un *IVR* con acceso a tecnologías Web:

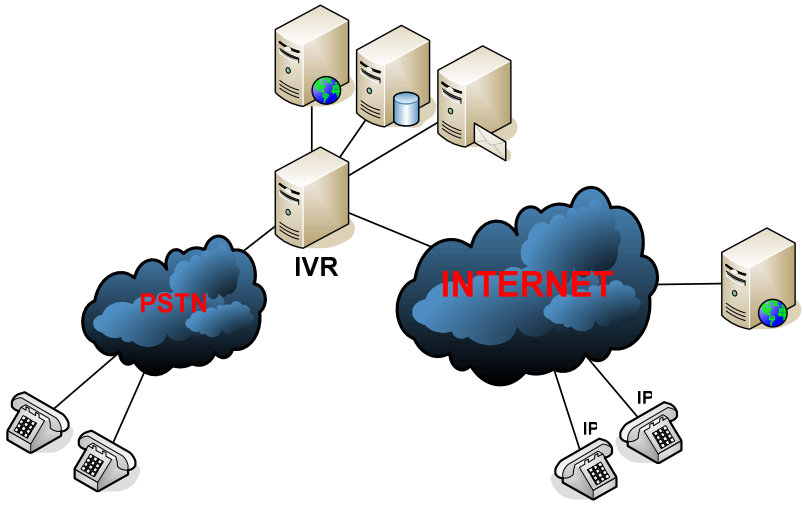

figura 1.1: Diagrama de un IVR con acceso a la PSTN y al Internet

Los usuarios pueden comunicarse a través de la telefonía tradicional (PSTN, *Public Switched Telephone Network*) o mediante teléfonos IP directamente desde el Internet, acceder al sistema del IVR y éste desde ahí proporcionar información de correo electrónico, información específica de bases de datos e incluso contenido Web. El IVR es capaz de conectarse a más recursos del Internet para receptar y enviar información o transferir las llamadas de sus usuarios.

El *W3C* ha proporcionado recomendaciones especializadas en control de llamadas y diálogos, gramática de reconocimiento de voz, interpretación semántica del reconocimiento, entre otras. En la *sección 1.3* se tratará con detalle estas recomendaciones.

1.2 TECNOLOGÍAS DE RECONOCIMIENTO NATURAL DE VOZ

El reconocimiento de voz surge de la necesidad de facilitar la interacción del hombre con los sistemas computarizados de una manera mucho más ágil que con los medios que se tiene actualmente (teclado, ratón, etc.). La tendencia es a utilizar el lenguaje oral de forma natural y cotidiana como nos comunicamos diariamente entre seres humanos, pues una persona puede pronunciar muchas más palabras que las que pueda digitar sobre un teclado en una misma cantidad de tiempo, de ahí la necesidad de incorporar en los sistemas computarizados el *reconocimiento natural de voz*.

El *reconocimiento natural de voz* es el proceso mediante el cual una máquina descifra qué palabras ha pronunciado una persona, partiendo de una señal acústica que representa los sonidos emitidos por ella. No llega hasta el punto de comprender lo que el usuario desea, esto se lo deja a aplicaciones que generalmente se centran en actividades específicas y toman como entrada la salida del reconocedor de voz que comúnmente es una pieza de software a la que se denomina *motor de reconocimiento de voz*. Existen discrepancias en la gramática de este campo de la investigación, pues se suele mencionar como *sistemas de reconocimiento de voz* o *detección de voz* a los sistemas biométricos de autenticación a través de las características sonoras de la voz de cada persona y *sistemas de reconocimiento automático del habla* a los sistemas que detectan las palabras pronunciadas por las personas en general, particularmente en este proyecto se hará mención a los *sistemas de reconocimiento de voz* como todo sistema capaz de detectar las palabras o frases pronunciadas por un usuario cualquiera.

1.2.1 CLASIFICACIÓN DE LOS SISTEMAS DE RECONOCIMIENTO NATURAL DE VOZ

Los seres humanos somos capaces de percibir nuestro entorno de una manera natural, espontánea y con una agilidad impresionante; un sistema computarizado en cambio muestra cierta torpeza al tratar de realizar estas mismas actividades, su especialidad está en realizar repetidamente cálculos matemáticos y lógicos a velocidades muy altas. Esto radica en la naturaleza de la información que procesa cada entidad, mientras para los seres humanos el comunicarse a través del lenguaje oral entre sí es algo que se experimenta desde muy temprana edad, para una máquina requeriría empezar por procesos sencillos como medición y muestreo de señales hasta procesos todavía complejos en la actualidad como el razonamiento y el aprendizaje.

Los sistemas de reconocimiento natural de voz según la clasificación del proveedor de soluciones Lumenvox[4] se dividen en dos grupos, dependientes del locutor e independientes del locutor.

1.2.1.1 Sistemas dependientes del locutor

Un sistema de reconocimiento natural de voz dependiente del locutor requiere de un entrenamiento previo, los usuarios deben almacenar en una base de datos las grabaciones de sus voces pronunciando las palabras que luego el sistema deberá reconocer. Cuando se llama al motor de reconocimiento para que detecte que palabra ha pronunciado un usuario, se compara ésta con las señales acústicas almacenadas en su base de datos correspondientes a las palabras o frases programadas y la respuesta se emite en base al nivel de semejanza. Existen serias limitaciones con este tipo de motores de reconocimiento, el estado de ánimo de las personas puede cambiar en determinadas situaciones y esto se reflejará en el tono de voz o la velocidad con que se pronuncia las palabras, el sistema no detectará que se haya pronunciado una palabra que está en su base de datos si esta tiene distinto tono, velocidad o volumen aunque sea pronunciada

[4] http://www.lumenvox.com

por el mismo usuario. Evidentemente el sistema se limitaría a ser usado solo por las personas que tuvieron su entrenamiento respectivo.

Un sistema así puede cumplir los requisitos para ciertas aplicaciones que requieran de un número mínimo de usuarios, por ejemplo, algunos teléfonos celulares que suelen incorporar un módulo de reconocimiento de voz para acceder al directorio telefónico pronunciando simplemente el nombre del contacto, no será problema que el usuario programe una sola vez el nombre de todos sus contactos y use luego su sistema las veces que desee.

1.2.1.2 Sistemas independientes del locutor

Los sistemas de reconocimiento natural de voz independiente del locutor pueden detectar lo que cualquier usuario pronuncie, no necesitan un entrenamiento previo y están destinados a aplicaciones específicas de *sistemas de respuesta de voz interactiva* en los que no se hace diferencia entre usuarios de distinta edad procedencia o sexo. Con estos sistemas se pueden crear básicamente dos tipos de aplicaciones, las de *lenguaje natural* y *diálogo guiado*. En las primeras se pueden formular preguntas abiertas al usuario como por ejemplo *¿Qué desea hacer?*, las posibles respuestas dependen del servicio, en una empresa que comercialice autos nuevos podría obtenerse: *“Comprar un auto”*, *“Conocer el estado del crédito de mi auto”*, *“Saber de nuevos modelos”, “Hablar con una operadora”* e incluyo *“Necesito ayuda”*. En las aplicaciones de diálogo guiado se lleva al usuario por menús de opciones determinadas, en el ejemplo anterior se podría tener: *¿Por favor diga que necesita hacer, conocer sobre las últimas promociones, revisar el estado de su crédito o hablar con una operadora?*. En cualquiera de los dos casos la aplicación debe ser lo suficientemente capaz de diferenciar las respuestas y tomar la acción debida, generalmente se analiza palabras claves y no exactamente las respuestas completas.

Un sistema de reconocimiento de voz independiente del locutor conlleva más complejidad que uno que dependa de él, el proceso de reconocimiento ya no se basa simplemente en comparar las señales acústicas de la voz sino que debe

establecer modelos acústicos promedio de los usuarios, pues la señal de una misma palabra pronunciada por un niño, un hombre adulto o una mujer seguramente tendrá un patrón distinto.

Existe también una clasificación de los sistemas de reconocimiento natural de voz de acuerdo a la cantidad de palabras que pueden reconocer, se mencionan los de *vocabularios pequeños*, *medianos*, y *grandes* o *ilimitados*. En la práctica son los proveedores de software quienes ofrecen módulos especializados para determinadas aplicaciones, por ejemplo, un motor que reconozca números pronunciados en un determinado idioma o palabras que se usan en la banca, etc.

1.2.2 ESTRUCTURA DE UN MOTOR DE RECONOCIMIENTO NATURAL DE VOZ

El reconocimiento natural de voz es un proceso estadístico basado en el cálculo de probabilidades de semejanza entre patrones de señales acústicas pregrabadas en el motor y la señal que proviene de la pronunciación del usuario, en otras palabras, los programadores deben entrenar el sistema con una gran cantidad de señales acústicas de voces que representen los valores promedio de entonación, volumen, etc. y su respectivo significado en un lenguaje en particular, por ejemplo, se pueden trabajar con voces de niños, adultos y ancianos de diferentes sexos, así se creará un modelo con el que el motor relacionará a cada usuario nuevo de acuerdo a las características de su voz.

En el proceso se tiene un conjunto de señales acústicas que representan sonidos particulares al pronunciar vocales y consonantes y también un conjunto de palabras que pueden formarse con esos sonidos en un lenguaje determinado. El reconocimiento natural de voz se basa en determinar la máxima probabilidad de obtener una determinada secuencia de palabras (o una sola palabra) habiéndose detectado en la entrada una determinada secuencia acústica, matemáticamente:

$$L = \arg\max_L P(L \mid A) \qquad \text{(ecuación 1.3)}$$

En donde *“L”* corresponde al conjunto de palabras que posiblemente pronunciará el usuario y *“A”* al conjunto de señales acústicas que representan todas las palabras en determinado lenguaje. La palabra o secuencia de palabras reconocida será la que maximice la probabilidad de que ésta coincida con la señal acústica receptada. Según la *fórmula de Bayes* se puede reescribir la probabilidad de la siguiente manera:

$$P(L \mid A) = \frac{P(A \mid L)P(L)}{P(A)}$$

$$L = \arg\max_L \frac{P(A \mid L)P(L)}{P(A)}$$

En donde probabilidad $P(A|L)$ corresponde a la probabilidad de obtener la señal acústica *“A”* habiéndose pronunciado la secuencia de palabras *“L”*, $P(L)$ es la probabilidad de la secuencia de palabras *“L”*, y $P(A)$ la probabilidad de la señal acústica *“A”*. Esta última probabilidad no tiene efecto al momento de maximizar el resultado pues el conjunto de señales acústicas está ya almacenado en el motor independientemente de la palabra o secuencia de palabras a reconocerse. Al final el resultado se puede expresar como:

$$L = \arg\max_L P(A \mid L)P(L) \qquad \text{(ecuación 1.4)}$$

La probabilidad $P(A|L)$ relaciona datos acústicos con sonidos de palabras en particular y $P(L)$ depende estrictamente del lenguaje, al primero se lo denomina *modelo acústico* y al segundo *modelo del lenguaje* y son las dos partes fundamentales que conforman un motor de reconocimiento natural de voz.

La *gramática de la aplicación* es también una parte importante al momento de realizar el reconocimiento natural de voz, ésta corresponde al conjunto de palabras o frases que el usuario va a pronunciar, es decir, las opciones que tiene en determinado instante. La *figura 1.2* muestra la estructura de un motor de reconocimiento natural de voz tomando en cuenta cada paso del proceso de reconocimiento.

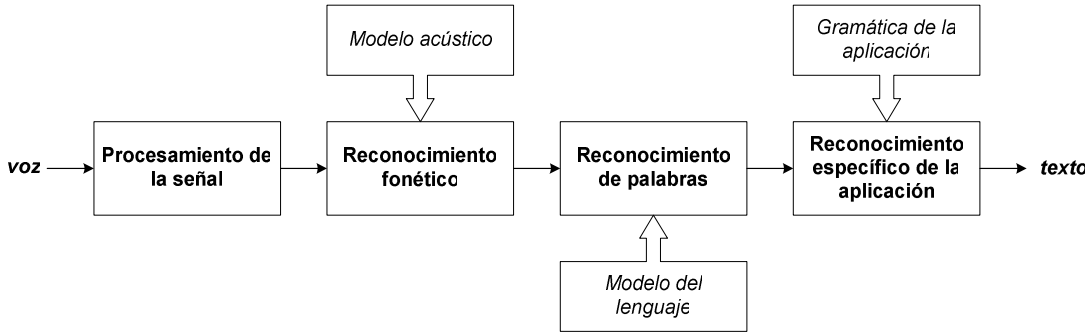

figura 1.2: Estructura de un motor de reconocimiento natural de voz

1.2.2.1 Modelo Acústico

El modelo acústico presente en los motores de reconocimiento natural de voz depende del entrenamiento que se le haya propiciado a este, básicamente el número de voces de diferentes características fonéticas pronunciando todos los posibles sonidos que se puedan generar con la voz, el modelo guarda una gran base de datos con la representación fonética de señales acústicas producidas por estas voces. La representación fonética indica como se pronuncia una vocal o consonante en determinada lengua y suele representarse por un alfabeto de símbolos especiales, un ejemplo de esto se lo puede apreciar en los diccionarios de traducción entre dos idiomas distintos, para indicar la pronunciación de alguna palabra.

Lo que hace el modelo acústico es traducir las señales de voz a una representación fonética que no depende del lenguaje, simplemente es la representación de los sonidos que se produjeron en la entrada del usuario.

1.2.2.2 Modelo del Lenguaje

El modelo del lenguaje define las características de cómo se articulan las palabras en un lenguaje determinado, eliminando posibles errores en el reconocimiento que no podrían pronunciarse en dicha lengua, por ejemplo en el idioma español es casi imposible que una persona articule la palabra *“HWVI”*, además que ésta no define ningún elemento gramatical del idioma (sustantivo, verbo, etc.). La conjugación de los verbos, la pronunciación de algunas letras bajo ciertas

condiciones en una palabra (la *“h”* cambia la pronunciación de la *“c”* cuando se forma *“ch”*), el orden de los elementos gramaticales (primero artículo después sustantivo), los hiatos, diptongos, entre otras reglas; definen las características propias de cada lenguaje y aportan una enorme ayuda al proceso de reconocimiento natural de la voz.

La representación fonética de la señal de voz del usuario que sale del modelo acústico es la entrada del modelo del lenguaje, aquí se analiza a que vocales y consonantes pertenecen dichos símbolos y luego se va armando las palabras y frases. Algunos motores de reconocimiento tienen la capacidad de utilizar al mismo tiempo dos o más modelos de lenguajes distintos, esto se lo maneja directamente desde la gramática de aplicación, pues se debe tomar en cuenta la orientación que se le da al sistema, un ejemplo claro se lo puede apreciar al considerar que el mismo lenguaje español no es igualmente hablado por personas de distintas regiones, los motores de reconocimiento pueden proporcionar varios modelos de lenguaje español, para usuarios latinoamericanos, españoles, mexicanos, etc. e incluso podría considerarse un modelo de lenguaje español para hablantes nativos de otro idioma a los que normalmente les cuesta articular las palabras pero que presentan un dialecto similar.

1.2.2.3 Gramática de la Aplicación

Una aplicación que use reconocimiento natural de voz generalmente tiene un fin específico y un rango limitado de palabras o frases que va a receptar, así por ejemplo, un servicio de consulta de planillas telefónicas solamente necesitará reconocer los dígitos del número en cuestión más unas cuantas palabras para navegar por el menú; indicarle al motor de reconocimiento esta gramática limitada seguramente disminuirá la carga de procesamiento e incrementará la tasa de aciertos en la detección de las palabras.

La gramática de aplicación es la parte que aporta el programador y define las secuencias válidas de palabras que podrán ser receptadas, dependiendo de la certeza en el reconocimiento de éstas se puede continuar con el curso adecuado

del sistema. Se debe diferenciar entre el texto que el motor de reconocimiento detecta en la entrada y el texto que este puede emitir en la salida, por ejemplo si el motor detecta un número de tres cifras como *"ciento ochenta y nueve"* quizá para un procesamiento más adecuado de esta información el programador necesite un *"189"*, o que cuando se detecte en la entrada *"Quiero consultar mi saldo"* se necesite solamente *"saldo"*; a esto se le denomina *interpretación semántica*. En la siguiente sección se analizarán los estándares que existen para la escritura de archivos de gramática de aplicación y la interpretación semántica en el reconocimiento natural de la voz.

1.3 ESTÁNDARES USADOS EN EL RECONOCIMIENTO NATURAL DE VOZ

1.3.1 SRGS (SPEECH RECOGNITION GRAMMAR SPECIFICATION)

La especificación de gramática de reconocimiento del habla (SRGS[5] por sus siglas en inglés) ha sido desarrollada por el Grupo de Trabajo de Navegadores de Voz del W3C (W3C Voice Browser Working Group[6]) y describe la sintaxis gramatical para definir las palabras o patrones de palabras que serán interpretados por un motor de reconocimiento natural de voz.

Esta sintaxis se puede escribir de dos maneras, mediante archivos ABNF (*Augmented BNF*) o mediante archivos XML. Cualquiera de las dos formas supone una traducción similar para el motor de reconocimiento, lo que deja a elección del programador la estructura a la que mejor se adapte. En este estudio se va a analizar como estructurar archivos XML.

El uso de una estructura gramatical en una aplicación de voz se debe principalmente para indicarle al motor de reconocimiento cuales son las palabras

[5] http://www.w3.org/TR/speech-grammar/

[6] http://www.w3.org/Voice/Group/

o patrones de palabras que el usuario pronunciará, además del lenguaje en el que se establece la comunicación. La especificación SRGS no define como enrutar una llamada, ni tampoco la manera de fijar ciertas características del motor de reconocimiento. Existen tecnologías para determinar el lenguaje en el que una persona habla o la autenticación de usuarios mediante el reconocimiento de voz; esto no está soportado por la especificación ni tampoco es parte de los objetivos de este proyecto.

1.3.1.1 Reglas

Una regla o instrucción define una palabra, frase o conjunto de palabras que el usuario pronunciará, al cual se le asocia un nombre que debe ser único en un archivo XML. Por ejemplo, si un usuario tendría que describir una casa de manera que especifique una parte de la misma (paredes, techo, puertas, etc.) y su color, podríamos crear una regla que se llame *"parte"* y otra *"color"*, así en nuestro archivo de gramática XML podríamos llamar a la regla *"parte"* seguida de la regla *"color"*, lo que evitaría especificar todas las combinaciones posibles.

Para definir una regla se usa el elemento <*rule*> y se le asigna un identificador mediante el atributo *"id"*, este nombre o identificador no debe contener los caracteres *"."* (punto), *":"* (dos puntos) y "-" (guión), y tampoco debe ser uno de los nombres de las reglas especiales (NULL, VOID, GARBAGE) que se explicará posteriormente. El ámbito de una regla puede ser *"public"* o *"private"*, y se define mediante el parámetro *"scope"*. Una regla de ámbito *"private"* puede ser referenciada solamente en el mismo archivo XML, a diferencia del ámbito *"public"* que puede ser referenciada también desde otros archivos. La definición de una regla se muestra en el *espacio de código 1.1*.

```
<rule id="mi_regla" scope="public">
   ……
   ……
</rule>
```

espacio de código 1.1: Definición de una regla en SRGS

Las reglas se definen para darle modularidad a la gramática a ser implementada y se las construye mediante elementos como: tokens, referencias a reglas, etiquetas, secuencias, alternativas y expresiones repetitivas.

1.3.1.1.1 Tokens

Se considera como una entidad gramatical del lenguaje, puede ser una o varias palabras que un motor de reconocimiento puede traducir a su representación fonética. Un token corresponde al texto que se encuentra entre las etiquetas <*token*> a excepción del contenido entre las etiquetas <*tag*>.

Un token puede estar compuesto de una o más palabras delimitadas o no por comillas dobles, inclusive puede formarse por caracteres alfanuméricos. Se aplica la normalización de espacios en blanco de modo que una cadena de estos, una tabulación o un salto de línea se convierten en un solo espacio en blanco.

En el *espacio de código 1.2* se tienen distintas definiciones para el token *"Escuela Politécnica Nacional"* pero lo que se tiene en el *espacio de código 1.3* son distintos tokens.

```
<token>Escuela Politécnica Nacional</token>
<token>"Escuela Politécnica Nacional"</token>
<token>   Escuela       Politécnica
                              Nacional</token>
<token> " Escuela        Politécnica Nacional"  </token>
<token> "   Escuela     Politécnica       Nacional    "</token>
```

espacio de código 1.2: Distintas definiciones del mismo token

```
<token>Escuela Politécnica Nacional</token>
<token>"Escuela_Politécnica_Nacional"</token>
<token>   "Escuela      Politécnica"
                               Nacional</token>
<token>EscuelaPolitécnicaNacional</token>
```

espacio de código 1.3: Definiciones de distintos tokens

Cuando se indica al motor el texto a ser reconocido mediante uno o varios tokens se debe tener muy en cuenta la verdadera pronunciación de las palabras, inclusive el dialecto de cada región, por ejemplo, si el sistema a desarrollar va a funcionar en un almacén de repuestos de vehículos y si en algún momento se espera que el cliente pronuncie la palabra *"llanta"*, los tokens indicados en el *espacio de código 1.4* pueden ser válidos y es más, deberían ser incluidos para obtener un óptimo resultado en el reconocimiento:

```
<token> llanta </token>
<token> yanta </token>
<token> shanta </token>
<token> lianta </token>
```

espacio de código 1.4: Distintas pronunciaciones de una misma palabra

Así mismo se debe evitar los símbolos y abreviaturas, cambiar *"Ing."* por *"Ingeniero"*, *"CIEEPI"* por *"ciepi"*, *"AEIE"* por *"a e i e"*, etc. Aunque el motor de reconocimiento aceptase caracteres alfanuméricos es preferible indicar en el token la pronunciación de dichas cantidades, por ejemplo, cambiar *"1000"* por *"mil"*, *"989"* por *"novecientos ochenta y nueve"*, *"nueve ocho nueve"*, *"nueve ochenta y nueve"*, etc. Se puede especificar el lenguaje que se usará en el reconocimiento exclusivamente para el token en cuestión mediante el parámetro *xml:lang* (*espacio de código 1.5*).

```
<token xml:lang="es"> electrónica </token>
```

espacio de código 1.5: Definición del lenguaje para un token

1.3.1.1.2 Referencias a reglas

Una referencia a regla constituye una llamada a otra regla que puede estar definida en el mismo archivo, en un archivo distinto e incluso a reglas especiales que se definirán más adelante. Una referencia a una regla local se la define como se indica en la primera línea del *espacio de código 1.6*.

```
<ruleref uri=“#color”/>
<ruleref uri=“otro_archivo#otro_color”/>
```

espacio de código 1.6: Definición de referencias a reglas

La regla definida con el nombre *“color”* debe existir en mismo archivo desde donde se la llama. El elemento *“ruleref”* identifica una referencia a regla y el atributo *“uri”* lleva asignado el nombre de la regla. Se debe notar que el nombre de la regla debe estar precedido del símbolo *“#”*, de lo contrario el analizador (*parser*) dará un mensaje de error.

Si la regla no se encuentra en el archivo actual debemos referenciar a dicho archivo anteponiendo su nombre al símbolo *“#”* (segunda línea del *espacio de código 1.6*).

Como se mencionó antes, existen reglas especiales predefinidas en la especificación y que la gramática no deberá redefinir, son tres: *NULL*, *VOID* y *GARBAGE*. La primera define una regla que coincide con cualquier palabra o conjunto de palabras que fuesen pronunciadas, la segunda con ninguna (significa que el motor de reconocimiento no emitirá una respuesta) y la tercera es similar a la primera pero funciona mientras no se detecte una nueva palabra o conjunto de palabras reconocidas.

```
<rule id=“pais_capital”>
   <ruleref special=“GARBAGE”/>
   <ruleref uri=“#pais”/>
   <ruleref special=“GARBAGE”/>
   <ruleref uri=“#capital”/>
</rule>
```

espacio de código 1.7: Ejemplo de uso de la regla especial GARBAGE

Las reglas especiales *NULL* y *VOID* son raramente usadas, en cambio *GARBAGE* muestra una gran funcionalidad, por ejemplo, si se espera que un usuario pronuncie un país con su capital y se le da la flexibilidad de que lo haga de la

manera que desee conservando el orden *“país-capital”* podemos crear las reglas que correspondan a todos los países y todas las capitales y escribir otra regla de la manera que se indica en el *espacio de código 1.7*.

Así, el motor de reconocimiento devolvería *“Ecuador Quito”* indistintamente de las siguientes oraciones:

“La capital del Ecuador es Quito”
“Ecuador tiene de capital a la ciudad de Quito”
“País Ecuador capital Quito”

En el ejemplo se aprecia además que para las referencias a reglas especiales en lugar del utilizar el atributo *“uri”* se utiliza el atributo *“special”*, un elemento <*ruleref*> debe contener uno de ellos pero jamás los dos.

1.3.1.1.3 Secuencias y Encapsulamiento

Si se usan las reglas *“país”* y *“capital”* del ejemplo anterior y no se daría flexibilidad al usuario, sino que debería pronunciar el país, seguido de la palabra *“capital”* y luego el nombre la capital, se lo podría hacer como se indica en el *espacio de código 1.8*.

```
<item>
   <ruleref uri=“#pais”/>
   capital
   <ruleref uri=“#capital”/>
</item>
```

espacio de código 1.8: Ejemplo de una secuencia encapsulada

Así podemos secuenciar referencias a reglas y tokens (además de otros elementos que se verá posteriormente como *“one-of”* y *“tag”*) para crear determinados patrones a reconocerse. El elemento <*item*> sirve para encapsular estas secuencias y cuando contiene solamente palabras (tokens) da la misma funcionalidad que el elemento <*token*>.

1.3.1.1.4 Alternativas

Para definir alternativas existe el elemento <*one-of*>, en donde cada alternativa se encapsula en el elemento <*item*>, por ejemplo, si se crea la regla antes mencionada que corresponde a pronunciar un color, se debería escribir lo que está en el *espacio de código 1.9*.

```
<rule id="color">
   <one-of>
      <item> azul </item>
      <item> amarillo </item>
      <item> rojo </item>
      <item> verde </item>
      <item> negro </item>
      <item> blanco </item>
      ……
   </one-of>
</rule>
```

espacio de código 1.9: La regla *"color"*

Al llamar a la regla *"color"* se debe corresponder con uno y solo uno de los colores mencionados (los puntos suspensivos enseñan que pueden ser más colores, de lo contrario es un error de sintaxis). Se puede indicar el lenguaje específico para el elemento <*one-of*> mediante el atributo *"xml:lang"* de la misma manera que se lo hace para un token, el lenguaje de éste último es el de mayor prioridad en caso de existir ambos.

1.3.1.1.5 Expresiones repetitivas

En ocasiones el usuario puede repetir una misma regla o token varias veces, limitada, acotada e incluso opcionalmente. El atributo *"repeat"* del elemento <*item*> ayuda a este propósito y debe ser asignado a una de las expresiones siguientes:

n	Se repite *n* veces
0-n	Puede repetirse *n* veces, el <*item*> es opcional
m-n	Se repite entre *m* y *n* veces
m-	Se repite *m* veces o más

1.3.1.1.6 Etiquetas para la interpretación semántica

El elemento <*tag*> puede ser incluido dentro del contenido de los elementos <*rule*> e <*item*> y es utilizado para indicarle al motor de reconocimiento los valores a devolver cuando se ha reconocido una determinada expresión.

```
<rule id="digito">
   <one-of>
      <item> cero   <tag> 0 </tag> </item>
      <item> uno    <tag> 1 </tag> </item>
      <item> dos    <tag> 2 </tag> </item>
      <item> tres   <tag> 3 </tag> </item>
      <item> cuatro <tag> 4 </tag> </item>
      <item> cinco  <tag> 5 </tag> </item>
      <item> seis   <tag> 6 </tag> </item>
      <item> siete  <tag> 7 </tag> </item>
      <item> ocho   <tag> 8 </tag> </item>
      <item> nueve  <tag> 9 </tag> </item>
   </one-of>
</rule>
```

espacio de código 1.10: La regla *"digito"* con interpretación semántica

La regla del *espacio de código 1.10* le dice al motor de reconocimiento que cuando se reconozca un dígito se devuelva su valor numérico. La funcionalidad de estas etiquetas viene dada por la recomendación de Interpretación Semántica para Reconocimiento del Habla (SISR, *Semantic Interpretation for Speech Recognition*) o por el motor de reconocimiento.

1.3.1.2 Estructura de Archivos de Gramática

Un archivo de gramática escrito con la recomendación SRGS debe tener un encabezado bien definido, por lo menos con los elementos que se indican en el *espacio de código 1.11*.

La primera línea es la declaración del archivo XML, su atributo *"version"* determina la versión del lenguaje XML que se utiliza en el documento, es obligatorio empezar el archivo con esta línea. Lo que viene luego es el prólogo, se define por el elemento <*grammar*> y su primer atributo obligatorio es también *"version"*, que corresponde con la versión de la recomendación SRGS con la que se está

trabajando. El atributo <*xml:lang*> define el lenguaje utilizado en el reconocimiento de todo el archivo, este puede ser redefinido en cada elemento <*item*>, <*token*>, o <*ruleref*>. Un archivo de gramática puede estar orientado a reconocer voz o tonos DTMF (Dual Tone Multi Frecuency) especificando *"voice"* o *"dtmf"* respectivamente en el atributo *"mode"*. Debe existir una regla principal (raíz) en la cual inicia el reconocimiento mediante la gramática especificada en el archivo, esto se lo hace mediante el atributo *"root"*.

```
<?xml version= "1.0"?>

<grammar version="1.0"
           xml:lang="es"
            mode="voice"
            root="regla_raiz"
            tag-format= "semantics/1.0"
            xmlns="http://www.w3.org/2001/06/grammar"
```

espacio de código 1.11: Estructura de un archivo de gramática

Como se mencionó anteriormente, es usual que el motor de reconocimiento implemente el funcionamiento de las etiquetas <*tag*> que se incluyen en la gramática para manejar los valores de retorno cuando se reconoce un token, con el parámetro *"tag-format"* se le dice al motor como debe interpretar estas etiquetas. El atributo *"xmlns"* corresponde al espacio de nombres (*namespace*) que se debe definir para el archivo de gramática, es un parámetro necesario y por defecto se le asigna *"http://www.w3.org/2001/06/grammar"*, que es el espacio de nombres definido para los archivos de gramática con formularios XML.

1.3.2 SISR (SEMANTIC INTERPRETATION FOR SPEECH RECOGNITION)

Un motor de reconocimiento natural de voz generalmente devuelve la última expresión coincidente dentro del archivo de gramática que el usuario ha pronunciado, esta información no es quizá la que se necesita para seguir procesando una aplicación. Por ejemplo, si se espera una respuesta afirmativa del usuario este podría decir *"si"*, *"okey"*, *"sip"*, *"claro"*, *"aja"*, etc. pero para la

aplicación bastaría con obtener un resultado común de todas estas alternativas, que podría ser un *"si"* un *"true"* o simplemente un *"1"*.

En la sección anterior se trató sobre la especificación de gramática de reconocimiento del habla (SRGS) y se señaló brevemente las etiquetas del tipo <*tag*>, que permiten indicarle al motor de reconocimiento el valor de retorno para cada regla del archivo de gramática. Es justamente aquí en donde se aplica la recomendación de la Interpretación Semántica para el Reconocimiento del Habla (SISR[7], por sus siglas en inglés).

La recomendación SISR al igual que SRGS ha sido desarrollada por el Grupo de Trabajo de Navegadores de Voz de la W3C y define la estructura de las expresiones usadas dentro de las etiquetas <*tag*> en los archivos de gramática que usan la recomendación SRGS y que se utilizan para interpretar semánticamente la información. La semántica es una rama del lenguaje que estudia el significado y uso de las palabras y frases, en el caso del reconocimiento del habla se refiere a la traducción del lenguaje humano (lo que el motor reconoce) a expresiones más entendibles por la aplicación.

1.3.2.1 Etiquetas para la Interpretación Semántica

En el prólogo de un archivo de gramática que usa la recomendación SRGS se debe indicar dentro del elemento <*grammar*> el atributo *"tag-format"*, que define la sintaxis a ser utilizada para la interpretación del contenido de las etiquetas <*tag*>. La recomendación SISR define dos tipos de sintaxis, *"Script"* y *"Literal de Cadena"*.

Para la sintaxis de interpretación semántica a manera de *"Script"* se asigna al atributo *"tag-format"* el valor *"semantics/1.0"*, indicando al motor de reconocimiento que el contenido de las etiquetas <*tag*> debe ser un programa

[7] http://www.w3.org/TR/semantic-interpretation/

ECMAScript[8] de perfil compacto (ECMA-327[9]) y para la sintaxis a manera de *"Literal de Cadena"* se debe asignar el valor *"semantics/1.0-literals"*, indicando que el contenido de las etiquetas son cadenas de caracteres. Un ejemplo sencillo para entender estas dos estructuras de interpretación semántica sería una regla que permita seleccionar al usuario un número entre uno y tres, en los *espacios de código 1.12* y *1.13* se indica esta regla con interpretación semántica a manera de *"Literal de Cadena"* y *"Script"* respectivamente.

```
<rule id="uno_dos_tres">
   <one-of>
      <item> uno <tag> 1 </tag> </item>
      <item> dos <tag> 2 </tag> </item>
      <item> tres <tag> 3 </tag> </item>
   </one-of>
</rule>
```

espacio de código 1.12: Interpretación semántica como de literal de cadena

```
<rule id="uno_dos_tres">
   <one-of>
      <item> uno <tag> out="1"; </tag> </item>
      <item> dos <tag> out="2"; </tag> </item>
      <item> tres <tag> out="3"; </tag> </item>
   </one-of>
</rule>
```

espacio de código 1.13: Interpretación semántica como *script*

La variable *"out"* corresponde al valor que el motor de reconocimiento devuelve cuando ha detectado una coincidencia entre la entrada del usuario y alguna regla del archivo de gramática.

---

[8] ECMAScript es un estándar de la "*European Computer Manufacturers Association"* que define un lenguaje de *"scripting"* general que se basa en lenguajes como JavaScript de Netscape y JScript de Microsoft. A lo largo de esta sección se conocerá elementos básicos de este lenguaje.

[9] La especificación ECMA-327 define un perfil compacto de ECMAScript, que es un subconjunto de la especificación ECMA-262, que corresponde a la tercera edición de la especificación del lenguaje ECMAScript.

Aunque el ejemplo muestra la misma funcionalidad para los dos tipos de estructuras, la segunda a manera de *"Script"*, provee de una mayor flexibilidad si se desea hacer algo más que devolver otro valor diferente al reconocido. El motor de reconocimiento puede implementar su propia estructura de interpretación semántica.

1.3.2.2 Variables y valores semánticos

Un valor semántico es el resultado de interpretar el contenido de las etiquetas <*tag*> y es asignado a una variable que puede pertenecer a la regla en cuestión, a otra regla o a una variable global. Existen variables predefinidas para cada regla y el usuario también puede crear nuevas variables.

En el estándar *ECMA-327* se definen variables a manera de *números*, *booleanos*, *cadenas* al estilo de Java, y un tipo de *objeto* similar a una clase de C++, entre otros. Este último se define como un conjunto no ordenado de propiedades, cada una de las cuales puede contener un valor primitivo, objeto o función. Por ejemplo, un objeto *"auto"* puede tener tres propiedades, *"marca"*, *"modelo"* y *"color"*, a las que se accede o asigna de la manera que se indica en el *espacio de código 1.14*.

```
auto.marca = "peugeot";
auto.modelo = "407";
auto.color = "rojo";
```

espacio de código 1.14: Acceso a variables

1.3.2.2.1 Variables de una regla

La variable propia de cada regla se denomina *"out"* y corresponde al valor de retorno de la misma cuando haya sido detectada en la entrada del usuario, antes de lo cual no tiene valor alguno. Para referenciar a la variable *"out"* de otra regla que no sea la regla en cuestión se lo hace como en el *espacio de código 1.15*. Adicional a cada regla se tiene dos variables más que no son modificables por el usuario desde el ámbito de la misma, *"text"* y *"score"*. La primera es de tipo

cadena y almacena el valor de la expresión que fue reconocida por el motor, su valor es similar al de la variable *“out”* cuando no se usa la interpretación semántica en la regla en cuestión. La segunda es del tipo *número* y almacena un valor referencial del grado de certidumbre con la que el motor reconoció una expresión. El *“score”* o puntuación es un parámetro que lo mide el motor de reconocimiento y depende de su capacidad el propiciar valores acertados, puede suceder que para un valor *“alto”* se haya reconocido una expresión errónea o para un valor *“bajo”* una expresión correcta. Las definiciones de *“alto”* y *“bajo”* también están a cargo del motor, puede definir un rango entre cero y uno, cero y cien, cien y mil, etc.

```
rules.nombre_de_la_regla
```

espacio de código 1.15: Acceso a la variable *out* de otra regla

Las variables *“text”* y *“score”* no son parte de la variable *“out”* ni de ninguna variable de la regla en sí, se accede a ellas mediante el objeto *“meta”* (*espacio de código 1.16*). Las dos primeras expresiones corresponden a la regla en cuestión, las dos siguientes corresponden a la última regla detectada anterior a la regla en cuestión y las dos últimas para referenciar a las variables de cualquier regla en archivo de gramática (*nombre_de_la_regla* es el valor que la regla lleva en el atributo *“id”*). Solamente en el primer caso las variables *“text”* y *“score”* no son modificables por el usuario, pero pueden ser modificadas cuando desde otra regla se referencie a ellas.

```
meta.current().text
meta.current().score

meta.latest().text
meta.latest().score

meta.nombre_de_la_regla.text
meta.nombre_de_la_regla.score
```

espacio de código 1.16: Acceso a las variables *text* y *score*

1.3.2.2.2 Variables Globales

Se puede definir variables globales en el encabezado de un archivo de gramática y tendrán efecto solamente en archivos que tengan asignado una interpretación semántica a manera de *"Script"*, la definición e inicialización puede ser de la manera que se indica en el *espacio de código 1.17.*

```
<tag> var miNum = 1; </tag>
<tag> var miCad = "hola"; </tag>
```

espacio de código 1.17: Variables globales

No existe un orden estricto entre la definición de las variables globales y los elementos del encabezado en el archivo de gramática.

1.3.2.2.3 Asignación por defecto

Se asigna por defecto el valor de *"meta.current().text"* a la variable *"out"* cuando no existen etiquetas *<tag>* en la regla, de lo contrario se asigna el valor de *"rules.latest()"*.

Para aclarar el uso de variables dentro de un archivo de gramática que corresponda a la recomendación *SRGS* supongamos una empresa que se dedique a la venta de autos nuevos de una marca en particular y que tenga un servicio para que los clientes puedan llamar y reservar uno de ellos mediante dos características, su modelo y su color; el archivo de gramática sería el del *espacio de código 1.18*.

La regla principal es *"auto"* y acepta que el usuario diga el modelo y el color en cualquier orden, además se pasará por alto cualquier pronunciación adicional al inicio o entre estos dos parámetros mediante las reglas especiales *"GARBAGE"*. Al asignarle propiedades a la variable *"out"* de la regla *"auto"* se convierte esta en tipo *objeto*, en cambio las variables *"out"* de las reglas *"modelo"* y *"color"* son de tipo *"cadena"*. En la regla *"color"* se retorna un código de tres letras mayúsculas

que lo identifican, en cambio en la regla *“modelo”* se retorna una sola *cadena* independientemente si son una o dos las palabras que lo identifican.

```
<grammar    version=“1.0”
           xmlns=“http://www.w3.org/2001/06/grammar”
            xml:lang=“es”
           tag-format=“semantics/1.0”
           root=“auto”>

<rule id=“auto” scope=“public”>
   <ruleref special=“GARBAGE”/>
   <one-of>
      <item> <ruleref uri=“#modelo”/> </item>
      <item> <ruleref uri=“#color”/> </item>
   </one-of>
   <ruleref special=“GARBAGE”/>
   <one-of>
      <item> <ruleref uri=“#modelo”/> </item>
      <item> <ruleref uri=“#color”/> </item>
   </one-of>
   <tag> out.mod=rules.modelo;
         out.col=rules.color;
   </tag>
</rule>

<rule id=“modelo” scope=“private”>
   <one-of>
      <item> corsa </item>
      <item> grand vitara <tag> out=“vitara”; </tag> </item>
      <item> luv dmax <tag> out= “luv”; </tag> </item>
      <item> trail blazer <tag> out=“blazer”; </tag> </item>
      <item> spark </item>
      <item> aveo </item>
   </one-of>
</rule>

<rule id=“color” scope=“private”>
   <one-of>
      <item> rojo <tag> out=“ROJ”; </tag> </item>
      <item> azul <tag> out=“AZU”; </tag> </item>
      <item> negro <tag> out=“NEG”; </tag> </item>
      <item> blanco <tag> out=“BLA”; </tag> </item>
      <item> gris <tag> out=“GRI”; </tag> </item>
      <item> verde <tag> out=“VER”; </tag> </item>
   </one-of>
</rule>

</grammar>
```

espacio de código 1.18: Ejemplo de gramática para una empresa que vende de autos

Si un usuario entra al sistema y pronuncia *“Deseo un auto grand vitara de color rojo”* y otro usuario pronuncia *“Quiero comprar un auto blanco modelo spark”*, las

variables y sus valores corresponderían a las de los *espacios de código 1.19* y *1.20* respectivamente.

```
rules.auto.mod = rules.modelo = “vitara”
rules.auto.col = rules.color = “ROJ”
meta.modelo.text = “grand vitara”
meta.color.text = “rojo”
```

espacio de código 1.19: Ejemplo de variables asignadas por defecto (a)

```
rules.auto.mod = rules.modelo = “spark”
rules.auto.col = rules.color = “BLA”
meta.modelo.text = “spark”
meta.color.text = “blanco”
```

espacio de código 1.20: Ejemplo de variables asignadas por defecto (b)

1.3.3 VXML (VOICE EXTENSIBLE MARKUP LANGUAGE)

VoiceXML es el lenguaje recomendado por el W3C (*World Wide Web Consortium*) para desarrollar aplicaciones vocales integradas a todos los servicios que provee la Web. Está diseñado para crear diálogos hombre-máquina que incorporen reconocimiento de voz y tonos DTMF, reproducción de audio digital, grabación de conversaciones, voz sintetizada (TTS, *Text to Speech*) y telefonía en general. Como parte de las recomendaciones del Grupo de Trabajo de Navegadores de Voz del W3C es quizá el punto de partida para la creación de un sistema interactivo de respuesta vocal, define el flujo del diálogo con el usuario para receptar información de este y enviarla a la aplicación para su procesamiento.

Una aplicación puede comprender uno o varios archivos *VoiceXML* enlazados entre si, que contienen una secuencia de diálogos. Un usuario se encontrará siempre en un diálogo a la vez y de la información que provea al sistema se definirá la siguiente transición, cuando un diálogo no define su próxima transición o el usuario cuelga la llamada se termina la ejecución de la aplicación.

1.3.3.1 Prompts

La aplicación puede proporcionar información al usuario ya sea mediante la conversión de texto a voz (*Text to Speech*) o mediante archivos de audio pregrabados, para esto se utiliza el elemento <*prompt*>. Generalmente se utiliza un *prompt* cuando se desea que el usuario ingrese alguna información ya sea mediante su voz o la digitación de las teclas del teléfono, si se asigna el valor *"true"* al atributo *"bargein"* se detendrá la ejecución del *prompt* y el sistema se prestará a procesar la entrada del usuario.

También se puede especificar un intervalo máximo para esperar la entrada del usuario mediante el atributo *"timeout"* (*espacio de código 1.21*).

```
<prompt bargein="true" timeout="10">
   Por favor diga o digite su número de cédula
</prompt>
```

espacio de código 1.21: Definición de *prompt* en VoiceXML

Para reproducir un archivo de audio se define el elemento <*audio*> dentro del <*prompt*> con el atributo *"src"* indicando el nombre del archivo (*espacio de código 1.22*).

```
<prompt>
   Su numero de cedula es
   <audio src="numeros/cero.wav">
   <audio src="numeros/cuatro.wav">
   <audio src="numeros/cero.wav">
   <audio src="numeros/uno.wav">
   <audio src="numeros/trescientos.wav">
   <audio src="numeros/quince.wav">
   <audio src="numeros/siete.wav">
</prompt>
```

espacio de código 1.22: Reproducción de archivos de audio

1.3.3.2 Diálogos

Existen dos tipos de diálogos, *formularios* y *menús*. El primero pretende recolectar información del usuario y el segundo presentarle un conjunto de alternativas.

1.3.3.2.1 Formularios

Un formulario se define por el elemento <*form*> y puede tener dos atributos: *"id"* y *"scope"*, el primero corresponde al nombre del formulario para ser referenciado desde otro diálogo y el segundo es el ámbito de los archivos de gramática de reconocimiento de voz que se referencian desde dicho diálogo, se le puede asignar un valor *"dialog"* o *"document"* para que los archivos de gramática estén activos en el diálogo o en el documento respectivamente.

Dentro de un formulario existen elementos de entrada, el más común <*field*> y generalmente pregunta al usuario que acción desea tomar para luego invocar al motor de reconocimiento de voz o tonos DTMF mediante archivos de gramática, para así detectar la respuesta y continuar con la aplicación. Otro elemento es <*record*>, el cual permite grabar lo que el usuario dice, es muy utilizado para aplicaciones de correos de voz. El elemento <*subdialog*> permite hacer las llamadas a subdiálogos, que se las considera como un tipo de funciones para realizar alguna acción específica y luego retornar al diálogo desde donde fueron invocados.

El elemento <*transfer*> permite transferir la llamada a otro número telefónico, <*block*> se usa para reproducir un mensaje o realizar una acción sin esperar una respuesta del usuario y <*prompt*> reproduce un mensaje *"Text to Speech"*. Para entender mejor la estructura de un formulario imaginemos una aplicación telefónica que permita verificar si se tiene mensajes en el buzón de una oficina postal que aún no han sido retirados, el flujo del diálogo podría ser el del *espacio de código 1.23.*

Cuando un usuario ingrese al sistema se le dará la bienvenida con el mensaje *"Bienvenido a Correos del Ecuador"*, luego se le preguntará *"Cuál es su número*

de buzón", el resultado se obtendrá mediante la invocación al motor de reconocimiento de voz con el archivo *"noBuzon.grxml"* que obviamente tendrá las reglas necesarias para detectar la entrada de un número válido de buzón. El elemento <*submit*> permite enviar la información a un servidor de aplicaciones, en el caso del ejemplo anterior verificará si el buzón con el número obtenido contiene mensajes nuevos, el resultado será devuelto al usuario y la aplicación llegará a su fin.

```
<form id="correo">
   <block> Bienvenido a Correos del Ecuador. </block>
   <field name="buzon">
      <prompt> Cuál es su número de buzón? </prompt>
      <grammar src="noBuzon.grxml"
               type="application/srgs+xml"/>
      <block>
      <submit next="/servlet/est_buz" namelist="buzon"/>
      </block>
   </field>
```

espacio de código 1.23: Ejemplo de diálogo basado en un formulario

Sin duda alguna corresponde a una sencilla aplicación que no ofrece flexibilidad al usuario, si es la primera vez que se ingresa en el sistema puede que se necesite ayuda, la calidad del canal de voz puede ser tan mala que confunda al motor de reconocimiento y no se pueda detectar la entrada del usuario, el número de buzón que se detectó puede ser incorrecto, etc. Estos eventos pueden controlarse mediante el tratamiento de excepciones, más adelante se detallará como hacerlo.

1.3.3.2.2 Menús

Un menú puede ser considerado como un formulario que presenta varias alternativas al usuario y toma una acción determinada cuando se escoge una de ellas. Para simplificar este tipo de formularios que suelen ser muy comunes se ha creado el elemento <*menu*>, que puede ser utilizado de la manera que se indica en el *espacio de código 1.24.* Cada elemento <*choice*> es una alternativa del menú, su atributo *"next"* especifica que acción tomar, que generalmente es un diálogo que está en otro archivo *VoiceXML*. En los dos últimos ejemplos se hace referencia a archivos de gramática mediante el elemento <*grammar*>, su atributo

“src” corresponde a la dirección del archivo y *“type”* al tipo, que generalmente es *“application/srgs+xml”* para archivos escritos bajo la recomendación *SRGS* en formato *XML*. En este caso los archivos de gramática referenciados deben contener las reglas para reconocer cualquiera de las opciones que el usuario escoja. Cuando un usuario ingrese al sistema se le dirá *“Bienvenido a nuestra empresa, por favor elija servicio al cliente, ventas, servicios”* y según la respuesta se continuará con el diálogo en el archivo *VoiceXML* correspondiente.

```
<menu>
   <prompt>
      Bienvenido a nuestra empresa, por favor elija
   </prompt>
   <choice next=“clientes.vxml”>
      <grammar src=“clientes.grxml”
               type=“application/srgs+xml”/>
      servicio al cliente
   </choice>
   <choice next=“ventas.vxml”>
      <grammar src= “ventas.grxml”
               type=“application/srgs+xml”/>
      ventas
   </choice>
   <choice  next= “servicios.vxml”>
      <grammar src=“servicios.grxml”
               type=“application/srgs+xml”/>
      servicios
   </choice>
</menu>
```

espacio de código 1.24: Ejemplo de diálogo basado en un menú

El elemento *<menu>* puede contener los atributos *“id”* y *“scope”* con el mismo significado que en un formulario, también puede contener el atributo *“dtmf”* para aceptar como respuestas pulsaciones de las teclas del teléfono, se le debe asignar el valor *“true”* para que cada opción del menú corresponda con los números 1, 2, 3, etc. según su orden. Cada *<choice>* puede especificar su propio valor *DTMF* y no ser tomado en cuenta para la asignación automática de números, cabe resaltar que solamente se podrán tener hasta nueve opciones y no se podrán asignar las teclas *, #, ni el número 0.

1.3.3.3 Variables

Dentro de un archivo *VoiceXML* se puede tener fragmentos de código ECMAScript, para lo cual se deben definir variables, el elemento <*var*> se encarga de esto. En el *espacio de código 1.25* se define una variable llamada *"cedula"* a la que se le asigna el valor *"0401300157"*, luego se le asigna un nuevo valor y finalmente se elimina dicho valor. El nombre de la variable se lo especifica en el tributo *"namelist"* y si son dos o más variables que se desea dejar sin valor se las escribe seguidamente separadas por un espacio en blanco.

```
<var name="cedula" expr="0401300157"/>
<assign name="cedula" expr="0401447776"/>
<clear namelist="cedula"/>
```

espacio de código 1.25: Creación y asignación de variables

El ámbito de una variable depende del lugar en el documento *VoiceXML* en donde fue declarada, puede ser de *sesión*, *aplicación*, *documento*, *diálogo* y de *contexto local*. Las variables de *sesión* no son declaradas en los archivos *VoiceXML* sino que las provee la plataforma que los interpreta y generalmente sirven para indicar ciertos parámetros propios de dicha plataforma, estas variables son de solo lectura y pueden ser referenciadas desde cualquier aplicación. Las variables de *aplicación* y *documento* hacen referencia a las declaradas dentro del elemento <*vxml*> en un documento raíz de una aplicación o en documentos secundarios respectivamente, y como su nombre lo indica, se puede acceder a ellas desde la aplicación o solamente desde el documento en cuestión. El ámbito de *diálogo* corresponde a las variables declaradas dentro de un formulario o menú respectivamente y el *contexto local* a las que se declara dentro de algún elemento adicional a los mencionados, como puede ser dentro de una excepción.

1.3.3.4 Excepciones

Cuando un usuario no responde, cuando el motor de reconocimiento no puede entender su respuesta, cuando solicita ayuda, o cuando no se puede enrutar una llamada, entre otras, la plataforma que soporta la aplicación lanzará excepciones.

Para capturarlas existe el elemento <*catch*> que tiene por atributos *“event”*, *“count”* y *“cond”* y puede ser incluido dentro de un documento o un diálogo.

El atributo *“event”* especifica el evento o eventos que serán capturados en caso de que ocurran, por ejemplo:

- *“error”* cuando ha ocurrido un error general, se puede especificar el tipo de error mediante sus propiedades, así *“error.noresource”* es lanzado cuando en la aplicación se referencia a un recurso que no está disponible o *“error.semantic”* cuando un error de ejecución se ha encontrado en el documento *VoiceXML*.
- *“noinput”* cuando el usuario no ha pronunciado ninguna respuesta.
- *“nomatch”* cuando el motor de reconocimiento no entiende la respuesta del usuario.
- *“help”* cuando el usuario ha pronunciado *“ayuda”*.

Para capturar dos o más eventos en un mismo <*catch*> se los debe especificar en el atributo *“event”* separados por un espacio en blanco (*espacio de código 1.26*). Además se puede abreviar el uso de estas etiquetas para capturar eventos, <*error*> es un equivalente a <*catch event=“error”*>, <*nomatch*> es un equivalente a <*catch event=“nomatch”*>, etc.

```
<catch event=“noinput nomatch”>
```

espacio de código 1.26: Captura de excepciones

Un evento puede ser lanzado varias veces en un diálogo, el usuario puede pronunciar más de una vez su respuesta sin que el motor de reconocimiento tenga éxito en comprenderla; para manejar la captura de eventos de acuerdo al número de veces que hayan ocurrido se utiliza el atributo *“count”*, por ejemplo, si se desea que al tercer intento de detección de una respuesta no válida se acabe la aplicación se debería tener una estructura similar a la del *espacio de código 1.27*. Al atributo *“cond”* se le debe asignar el valor *“true”* si se desea que se

capture la excepción, pero por defecto lo tiene asignado, por esta razón suele ser omitido.

```
<catch event=“nomatch” count=“1”>
  <prompt> Lo siento, no le entendí <prompt>
</catch>

<catch event=“nomatch” count=“2”>
  <prompt> Lo siento, no le entendí <prompt>
</catch>

<catch event=“nomatch” count=“2”>
  <exit/>
</catch>
```

espacio de código 1.27: Captura de excepciones por el número de ocurrencias

Además de los eventos predefinidos por la recomendación se puede lanzar eventos creados el usuario, para esto existe el elemento *<throw>* que tiene por atributos *“event”* y *“message”* entre otros, el primero identifica el tipo de evento que es lanzado y el segundo una descripción del mismo.

1.3.3.5 Estructura de un documento VoiceXML

Un documento *VoiceXML* debe tener una estructura bien definida, un encabezado, los diálogos y menús de la aplicación y de ser el caso, las referencias o transiciones a otros documentos.

El encabezado de un documento *VoiceXML* puede tener los elementos del *espacio de código 1.28*. La primera línea necesaria en todo documento *XML* define la versión del lenguaje que estamos utilizando, además de la codificación de caracteres. El elemento *<vxml>* define los parámetros necesarios del documento *VoiceXML*, su primer atributo *“xmlns”* es obligatorio y corresponde al espacio de nombres designado para este tipo de documentos. Generalmente un intérprete *VoiceXML* necesita de un servidor Web de documentos, con el atributo *“xml:base”* se especifica este servidor, para que todas las referencias a archivos se basen en esa dirección. El atributo *“xml:lang”* es opcional y define el lenguaje que se utilizará para la aplicación, por ejemplo se capturará el evento *<help>*

cuando el usuario pronuncie *"ayuda"* si el lenguaje está definido como *"es"*. La versión del lenguaje *VoiceXML* que se está utilizando en el documento se especifica con el atributo *"version"*. Los elementos *<meta>* son opcionales y sirven para definir ciertos parámetros del documento como el autor, los derechos de autor, una dirección de correo electrónico de soporte, etc.

```
<?xml version="1.0" encoding="UTF-8"?>

<vxml xmlns="http://www.w3.org/2001/vxml"
      xml:base="http://www.epn.edu.ec/aplicaciones/vxml/"
      xml:lang="es"
      version="2.0">

<meta name="author" content="Andrés Fuentes"/>
<meta name="maintainer" content="afuentes@epn.edu.ec"/>
……
……
</vxml>
```

espacio de código 1.28: Estructura de un documento *VoiceXML*

1.3.4 SALT (SPEECH APPLICATION LANGUAGE TAGS)

El lenguaje de etiquetas para aplicaciones de voz (SALT por sus siglas en inglés) es un trabajo conjunto de un sinnúmero de empresas que forman el denominado *"Salt Forum"*[10] (predominan Cisco Systems, Comverse, Intel, Microsoft, Philips y SpeechWorks), para agregar una interfaz estándar de lenguaje hablado a los servicios y aplicaciones Web, tanto para navegadores de voz como para navegadores multimodales. Se conoce como navegadores de voz a navegadores que permiten al usuario acceder a un recurso Web a través de su voz, puede ser utilizando un teléfono o un micrófono. Los navegadores multimodales por otra parte proveen al usuario varias maneras para acceder a la información, se puede mencionar un teclado, un ratón, un teléfono, un lápiz óptico, un lector de caracteres impresos, etc.

[10] www.saltforum.org

SALT es un conjunto de elementos XML (etiquetas) con atributos, propiedades, eventos y métodos que pueden ser usados conjuntamente con una aplicación basada en etiquetas (una página HTML por ejemplo) para agregar una interfaz de voz a dicha aplicación.

Se tienen etiquetas especializadas para la entrada y salida, a continuación una breve descripción de cada una de ellas.

1.3.4.1 Entrada

Se utiliza el elemento <*listen*> tanto para reconocimiento de voz como para grabación de audio. Cuando se necesita reconocer la voz de entrada se deben agregar documentos de gramática en línea o referenciados mediante la etiqueta <*grammar*>, de lo contrario se usa <*record*> para establecer los parámetros de la grabación. Dentro de un elemento <*listen*> se puede tener cualquiera de los dos e incluso el reconocimiento y la grabación simultáneamente.

Entre los parámetros que tiene el elemento <*listen*> están: *"id"* para especificarle un identificador único en el documento, *"initialtimeout"* para establecer un intervalo de tiempo en milisegundos desde que se pide la entrada del usuario hasta que éste pronuncie algo, *"babbletimeout"* para establecer el intervalo que va a durar el reconocimiento, entre otros. El elemento <*listen*> también contiene tres propiedades de solo lectura: *"recoresult"*, *"text"* y *"status"*; la primera corresponde al valor de retorno del reconocimiento, la segunda a la palabra o conjunto de palabras reconocidas y la tercera al estado del reconocimiento, lleva el valor *"0"* si éste sucedió exitosamente.

Un elemento <*listen*> contiene varios métodos para controlar la entrada del usuario: *"Start()"*, *"Stop()"*, *"Cancel()"*, *"Activate()"*, *"Deactivate()"*. Los tres primeros sirven para iniciar, parar y cancelar la ejecución del contenido SALT respectivamente y los dos últimos para activar y desactivar documentos de gramática. Para ilustrar un poco como funciona todo lo mencionado hasta ahora imaginemos una página HTML que contenga un cuadro de texto y un botón que

se llame *“ojos”*, al pulsar el botón, el usuario tendrá que pronunciar cual es el color de sus ojos, existirá un documento de gramática en el que se incluirá una lista de todos los colores posibles; una vez que se ha reconocido un color éste será mostrado en el cuadro de texto.

```
<form id=“paginaHTML”>
   <input name=“color” type=“text”/>
   <input name=“ojos”
          type=“button”
          onClick=“reconocerColor.Start();”/>
</form>

<salt:listen id=“reconocerColor”>
   <salt:grammar src=“./colores.grxml”/>
   <salt:bind targetelement=“color”
              targetmethod=“submit”/>
</salt:listen>
```

espacio de código 1.29: Aplicación que combina HTML con SALT

En el *espacio de código 1.29* se puede apreciar la manera como interaccionan documentos HTML con SALT para presentar una aplicación multimodal al usuario, el elemento *“bind”* es quien se encarga de esto.

El usuario puede también ingresar información a la aplicación mediante el teclado de su teléfono, es decir, tonos DTMF, para esto existe el elemento <*dtmf*> que funciona de manera muy parecida a <*listen*>, dentro de él también se especifican archivos de gramática de detección de tonos mediante <*grammar*> y se los asigna a objetos del documento principal mediante <*bind*>.

1.3.4.2 Salida

La salida al usuario puede ser a través de voz sintetizada o la reproducción de archivos de audio, para cualquiera de las dos se utiliza el elemento <*prompt*>. Al igual que listen contiene el atributo *“id”*; el atributo *“bargein”* es una bandera que cuando se le asigna el valor *“1”* permite al usuario detener la reproducción si pronuncia algo o si pulsa alguna tecla de su teléfono.

En el contenido del elemento <*prompt*> se puede tener directamente texto, referencias a variables que contengan texto o referencias a archivos de audio, en el *espacio de código 1.30* está un ejemplo que contiene a los tres.

```
<prompt id=“RespuestaCancion”>
   La canción
   <value targetelement=“TopTen1”
          targetattribute=“value”/>
   numero uno de nuestro TopTen de la semana es la siguiente
   <content href="/canciones/topten1.wav" />
</prompt>
```

espacio de código 1.30: Tipos de contenido del elemento <*prompt*>

El ejemplo anterior supone una aplicación a la que se llama para escuchar las canciones que han tenido buena acogida durante cada semana, en *“TopTen1”* se guarda el nombre de la canción que ha ocupado el primer lugar y la misma canción se reproduce desde el directorio *“/canciones”*. Como se puede ver, el elemento <*value*> sirve para referenciar valores que se obtienen desde la aplicación (*“TopTen1”* pudo haber sido un cuadro de selección en la página Web en donde el usuario pulsó la canción que ocupa el primer lugar) y <*content*> para referenciar a archivos de audio.

1.4 LA PBX OPEN SOURCE ASTERISK

Según las páginas del manual[11], Asterisk es un servidor de telefonía altamente integrado que ofrece servicios de PBX (Private Branch eXchange), Respuesta de Voz Interactiva (IVR), Distribuidor Automático de Llamadas (Automatic Call Distributor), Gateway de Voz sobre IP (VoIP), Sistemas de Teleconferencia y una gran cantidad de aplicaciones para un amplio rango de dispositivos telefónicos tanto de voz empaquetada como los de la telefonía tradicional.

[11] El contenido que resulta al digitar el comando “$ man asterisk” en un servidor Linux al que ya se ha instalado Asterisk.

Asterisk es una plataforma de código abierto que ha revolucionado la telefonía de los últimos tiempos, creado por Mark Spencer[12] y diseñado inicialmente para ser ejecutado sobre el sistema operativo GNU/Linux (también es compatible con OpenBSD, FreeSD, Mac OS X y Windows) se ha convertido en la pieza de software fundamental para el desarrollo de aplicaciones telefónicas, integra la mayoría de las funcionalidades que ofrecen los equipos propietarios que se limitan a una o pocas de ellas, con gran flexibilidad y reducción de costos significativa.

1.4.1 ARQUITECTURA

Asterisk obedece a una arquitectura modular formada en su núcleo por cuatro APIs (Application Programming Interface) que sirven de interfaz con todas sus funcionalidades[13].

1.4.1.1 El API de Canal

Permite la conmutación entre la *telefonía tradicional* y distintos protocolos de *Voz sobre IP* como SIP (Session Initiation Protocol), H.323, MGCP (Media Gateway Control Protocol), SCCP (Skinny Client Control Protocol) y un protocolo sumamente liviano diseñado también por Mark Spencer llamado IAX (Inter Asterisk eXchange).

Para integrarse con la telefonía tradicional se usan tarjetas denominadas *"Pseudo TDM"* pues dejan la mayor parte del procesamiento de señal al mismo núcleo de Asterisk, reduciendo con esto el costo de fabricación de las mismas.

---

[12] Mark Spencer es el creador de Asterisk y propietario de la compañía Digium, innovador del hardware compatible con Asterisk para realizar la interfaz con la telefonía tradicional.

[13] http://www.asterisk.org/architecture

1.4.1.2 El API Traductor de Codecs

Provee una manera flexible para trabajar con voz codificada bajo distintos esquemas, soporta codecs como G.711, G.723.1, G.726, G.729ª, GSM, entre otros y permite la conversión entre ellos (*transcoding*).

1.4.1.3 El API de Formato de Archivo

Permite reproducir y guardar archivos de audio con distintos formatos entre los que predominan wav y mp3.

1.4.1.4 El API de Aplicación

Establece la interfaz entre las aplicaciones de usuario y toda la funcionalidad de Asterisk, permite la configuración de la central telefónica, su plan de numeración y la creación de aplicaciones específicas que se comuniquen con recursos externos como acceso a contenidos y bases de datos.

1.4.2 UN SISTEMA DE RESPUESTA DE VOZ INTERCTIVA CON ASTERISK

Al iniciarse Asterisk lee un conjunto de archivos de configuración en donde se indica todos los parámetros para su funcionamiento y las tareas específicas a realizarse. En el archivo *"extensions.conf"* se programa las aplicaciones específicas para las que será usada la plataforma, puede ser un simple plan de numeración, un *Call Center* que maneje colas, un IVR, etc.

En el caso específico de un IVR se programa el sistema como una secuencia de aplicaciones que se irán ejecutando dependiendo de su orden, por ejemplo, si se desea que cuando un usuario marque a la extensión *"1234"* se le conteste inmediatamente la llamada, se le reproduzca un archivo de audio llamado *"canción.wav"* y luego se cuelgue automáticamente, se debería escribir lo que está en el *espacio de código 1.31*. El número que le sigue a la extensión se le conoce como prioridad y establece el orden de ejecución de las aplicaciones que van al final, las que son similares a funciones o métodos usados en un lenguaje

de programación. El ejemplo es ciertamente sencillo pero Asterisk incorpora decenas de aplicaciones y funciones para el control y distribución de llamadas, control de flujo de operación, aplicaciones específicas como *correo de voz*, *colas*, *conferencias*, manejo de variables, ejecución de comandos del sistema, entre otras, con las cuales se puede crear grandes sistemas de telefonía. Diversas compañías de software se han interesado en ofrecer sus productos como complementos de Asterisk y generalmente incorporan conectores que no son más que un conjunto de nuevas funciones que se llamarán desde el API.

```
exten => 1234,1,Answer()
exten => 1234,2,PlayBack(canción)
exten => 1234,3,HangUp()
```

espacio de código 1.31: Ejemplo de aplicación en Asterisk

Funciones más complejas como acceso a base de datos o contenidos Web necesitan de una interfaz más eficiente que la mencionada anteriormente, para esto Asterisk cuenta con una aplicación adicional llamada AGI.

1.4.3 AGI (ASTERISK GATEWAY INTERFACE)

Es la interfaz de comunicación entre Asterisk y programas externos que pueden ser escritos en uno de los lenguajes de programación soportados (Perl, PHP, Python, Java, entre otros). Desde estos programas se puede controlar todo el flujo de ejecución de Asterisk y combinarlo con el acceso a información externa, se pueden también crear subrutinas que tengan funciones específicas que se usen repetidamente, ejecutar comandos del sistema operativo e interactuar con programas que exploten todas las características que ofrecen los lenguajes de programación mencionados.

Como su nombre lo indica, se trata de una puerta de enlace hacia otras aplicaciones informáticas con las que se logra completamente la integración entre la telefonía y la computación.

1.5 MOTORES DE RECONOCIMIENTO NATURAL DE VOZ

1.5.1 LUMENVOX

Según sus creadores[14], el motor de reconocimiento natural de voz que ofrece Lumenvox es un interfaz para programas de aplicación que reconoce el habla de cualquier audio indistintamente de la fuente que provenga. A través de su interfaz en lenguaje de programación C y C++ puede integrarse con cualquier aplicación, soporta tanto plataformas Windows como Linux.

Entre sus modelos de lenguaje cuenta con el inglés, español y francés, en los dos primeros también ha desarrollado modelos específicos para el reconocimiento de dígitos. Es independiente del usuario y del hardware que se use, su módulo reductor de ruido le permite adaptarse dinámicamente a las condiciones acústicas del usuario, así aumenta la certeza en la detección de actividad de voz (VAD, *Voice Activity Detection*) y el propio reconocimiento, mejorando notablemente los resultados al desarrollar aplicaciones por un canal sumamente ruidoso como es el canal telefónico.

1.5.1.1 Arquitectura

Trabaja en una arquitectura cliente servidor y es software de licencia comercial, provee de dos versiones, una denominada *"Lite"* que limita el rango de palabras a reconocerse simultáneamente a 500 y la versión completa que sube significativamente este valor a 12.000.

Básicamente consiste de dos servidores que pueden ejecutarse en distintas máquinas, uno de licencias y el otro que es el propio motor de reconocimiento, el número de licencias adquiridas corresponde al número de puertos que pueden abrirse simultáneamente, es decir el número máximo de usuarios que podría soportar una aplicación. También posee un módulo para convertir al motor de reconocimiento de Lumenvox en un servidor MRCP.

---

[14] http://www.lumenvox.com

1.5.1.2 Estándares que soporta

Soporta el Protocolo de Control de Recursos Multimedia (MRCP, *Media Resource Control Protocol*), que es el lenguaje que permite trabajar de una manera similar con todo tipo de información multimedia (texto, gráficos, video, sonido, voz, etc.) proveniente de cualquier dispositivo, así se logra que el usuario pueda escoger lo mejor de cada vendedor para implementar en su aplicación y no estar limitado a las *"cajas cerradas"* que ofrecen todos los servicios en uno pero no siempre de manera eficiente. El estudio de este protocolo y su implementación quedan fuera del alcance de este proyecto.

También soporta la especificación de gramática para el reconocimiento del habla (SRGS) y su interpretación semántica (SISR). El soporte para *VoiceXML* es limitado, pues aunque Lumenvox ofrece una plataforma completa de la que el motor de reconocimiento es solamente su núcleo, este no es capaz de controlar hardware de interfaz telefónica que le permita enrutar llamadas. Existe también soporte para el lenguaje etiquetado de aplicaciones vocales (SALT).

1.5.1.3 Interpretación semántica

Lumenvox soporta la interpretación semántica para el reconocimiento del habla (SISR) tanto en su versión de borrador publicada el 1 de abril del 2003 y la versión final de la misma, se debe asignar *"lumenvox/1.0"* o *"semantics/1.0"* al atributo *"tag-format"* de un archivo de gramática (SRGS) respectivamente.

Para algunas aplicaciones es suficiente el uso de *"lumenvox/1.0"*, la variable de una regla que ha coincidido con la entrada del usuario se denomina *"$"* y el resultado de la última regla se guarda en *"$$"*. El nivel de certeza de reconocimiento o *"score"* varía entre 0 y 1000, según recomendaciones del fabricante un valor mayor a 600 se debería aceptar como acierto, entre 200 y 600 se debería pedir una confirmación y menor a 200 como un fallo; en la realidad esto depende de la extensión del archivo de gramática y de la confusión que provocaría al motor la similitud de las palabras a reconocerse.

1.5.2 NUANCE

Nuance[15] es hoy en día la marca pionera en ofrecer soluciones de voz en más de cuarenta idiomas y dialectos, el motor de reconocimiento de Nuance o *"Nuance Recognizer"* es el núcleo de la plataforma de voz basada en *VoiceXML*, junto con *"Nuance Vocalizer"* para *Text to Speech* y *"Nuance Verifier"* para autenticación de usuarios por voz.

Provee reconocimiento de voz independiente del usuario, presenta algunos modelos de lenguaje español, el latinoamericano, colombiano, argentino y el nativo de España, entre otros. El primero es una generalización del dialecto usado en los países latinoamericanos en donde se habla el idioma español, los dos siguientes son modelos más específicos para dichos países en donde Nuance ha ganado su mercado.

1.5.2.1 Arquitectura

Nuance Recognizer es también un software de licencia comercial, cada puerto activo de reconocimiento es una licencia en uso. Dependiendo de los requerimientos de la aplicación se puede contar con una capacidad determinada de reconocimiento, según sus creadores se puede trabajar con archivos de gramática de hasta cien millones de palabras simultáneas.

Es mayormente utilizado como servidor MRCP, donde se necesita de un cliente que procese una aplicación de telefonía y haga las solicitudes de reconocimiento al motor o adquirir la plataforma completa de aplicaciones de voz a Nuance. Es compatible con interfaces de telefonía E1/T1, ISDN PRI y en general con todo dispositivo que trabaje con protocolos SIP[16] (Session Initiation Protocol) y RTP (Real Time Protocol). Puede trabajar sobre los sistemas operativos Windows, Sun Solaris e IBM-AIX.

---

[15] http://www.nuance.com

[16] El protocolo de inicio de sesión (SIP) junto con el protocolo en tiempo real (RTP) se han convertido en los predominantes de la telefonía IP, por eso es muy común que todo dispositivo actual que trabaje con este tipo de telefonía soporte estos protocolos.

1.5.2.2 Estándares que soporta

Soporta MRCP, *VoiceXML*, SRGS, SISR, el estándar emergente del W3C Extensible Multimodal Annotation Markup Language (EMMA) para el acceso multimodal a aplicaciones Web y Natural Language Semantics Markup Language (NLSML) para el procesamiento de información en servidores de voz.

El acceso multimodal corresponde a la interacción del usuario con las aplicaciones mediante cualquier medio de los usados hasta ahora, puede ser el teclado, ratón, un lápiz óptico, un teléfono, la voz, texto impreso, etc. EMMA ha sido creado para interpretar semánticamente todo tipo de información a la entrada y proveer datos comunes para poder procesarlos. NLSML en cambio está limitado para acceso mediante el habla y texto escrito, por eso su último borrador se publicó el 20 de noviembre del 2000 y no se ha continuado trabajando en éste.

1.5.2.3 Interpretación semántica

Soporta la especificación SISR aunque ha agregado cierta funcionalidad en las etiquetas <*tag*> de los archivos de gramática SRGS para intercambiar información con las aplicaciones *VoiceXML*, esto fue muy usado antes de que la especificación SISR empezara a desarrollarse. El nivel de certeza se lo conoce en Nuance como *“confidence”* y varía entre 0 y 100.

Capítulo II

2 DISEÑO DEL SISTEMA DE CONSULTA DE NOTAS CON RECONOCIMIENTO DE VOZ

2.1 DESCRIPCIÓN DEL PROBLEMA

La Carrera de Electrónica y Redes de Información, al igual que todas las carreras de la Escuela Politécnica Nacional, maneja una base de datos que contiene información acerca del desenvolvimiento académico de los estudiantes durante el periodo lectivo (semestre) en curso. Esta base de datos se almacena en un servidor que contiene *SQL Server 7* sobre el sistema operativo *Windows 2000*, ambos de *Microsoft*. Es la *Unidad de Gestión de Información* de la EPN (UGI) quien administra las bases de datos de todas las carreras y permite el acceso a la secretaría de cada una solamente para incluir las notas correspondientes según vaya transcurriendo cada bimestre. La base de datos se denomina SAE, que significa *Sistema Académico Estudiantil*.

Es natural la preocupación que tienen los estudiantes acerca de sus calificaciones y por ende su concurrencia a la secretaría, sobretodo cuando ha finalizado un bimestre o cuando se haya rendido exámenes supletorios. En un inicio bastaba con una carpeta que contenía las listas de calificaciones por materia, a la cual se podía acudir; el número de estudiantes no presentaba mayor problema. Con el pasar del tiempo y la evolución de la tecnología se ha intentado optimizar este asunto, considerando también que la cantidad de estudiantes en la carrera crece cada vez más.

Se ha creado un servicio a través de la página Web de la EPN para acceder directamente al SAE de la carrera y consultar las notas a través del Internet, basta autenticarse como estudiante ingresando el número único junto con el número de cédula y se obtiene la información correspondiente. Este sistema es muy utilizado por los estudiantes aunque presenta las dificultades de toda aplicación Web, se necesita de un computador conectado a Internet para acceder al sistema.

La idea es contribuir con el servicio de consulta de notas creando un sistema adicional que sea de fácil acceso, sencillo de utilizar, disponible las veinticuatro horas del día, los siete días de la semana y que ejemplarice la aplicación de nuevas tecnologías. Se ha elegido un servicio de consulta de notas a través de una línea telefónica, tomando en cuenta que la difusión de la telefonía tradicional se ha extendido a la mayor parte de la población y que la telefonía celular incorpora cada vez más y más usuarios.

El sistema elegido consiste en un IVR al que se pueda acceder marcando un número telefónico común (y quizá una extensión adicional) desde cualquier teléfono fijo o celular, el sistema será automático y no necesitará de intervención humana, solicitará el número único al estudiante que llame y una vez confirmado realizará la consulta a la base de datos para extraer la información correspondiente y entregársela al usuario. Para una mayor comodidad de este último, el sistema deberá devolver información específica por bimestre o materia. La interacción del usuario con el sistema debe ser lo más sencilla posible y para esto se ha decidido incorporar tecnologías actuales de reconocimiento natural de voz mediante la creación de diálogos guiados.

El uso de las tecnologías de reconocimiento natural de voz aporta una gran funcionalidad en sistemas en donde el paso por menús repetitivos y el uso de las teclas del teléfono vuelven monótona la interacción con el usuario, pero también sería desaprovechar todas sus características si se lo usa en un sistema como el que se plantea en donde la consulta de notas es el único objetivo. Tomando en cuenta lo anterior y un mejor servicio a los estudiantes, se ha decidido incorporar en el sistema la consulta de estado de planes de proyecto de titulación y certificados de prácticas preprofesionales que continuamente son revisados y aprobados por la Subcomisión Académica de la Carrera, para esto se deberá crear una base de datos adicional en el mismo servidor en el que se encuentra el SAE. Aunque esta base de datos adicional será sencilla tanto en su estructura como en su extensión (esto debido a la cantidad de información que deberá manejar) se ha decidido crearla aparte para tener una independencia entre el proyecto y el SAE de la carrera, además que la UGI es responsable del

funcionamiento del SAE y limita el acceso al mismo, pues es un sistema estándar que se utiliza en todas las carreras de la Universidad; inclusive las consultas que se deberá hacer serán a través de procedimientos almacenados para comprometer en lo mínimo la integridad del SAE.

El sistema deberá reducir los costos de soluciones propietarias y para esto utilizará en cuanto sea posible software libre, sin descuidar por otro lado la calidad de servicio que se brinde al usuario. En el caso particular del motor de reconocimiento de voz se ha elegido la solución propietaria de la firma Lumenvox, que como la mayoría de soluciones que existen, vende sus licencias por puertos, es decir, número de usuarios simultáneos. Esto hace que sea necesario tener un sistema alternativo de interacción con el usuario solamente mediante tonos DTMF (para cuando el número de usuarios simultáneos sobrepase el número de licencias disponibles), en secciones posteriores se tratará con más detalle este asunto.

2.2 DISEÑO DEL FLUJO DE OPCIONES DEL IVR

Como se ha mencionado en el primer capítulo un IVR debe ser de rápido acceso y sencillo de utilizar, los diálogos deben ser claros y en lo posible permitir al usuario llegar directamente a la información que requiera, sin embargo el diseño del IVR depende del objetivo que tenga la aplicación y en ocasiones se debe jugar entre la sencillez que el sistema presente al usuario y el nivel de consecución de objetivos del mismo.

2.2.1 CONSIDERACIONES DE DISEÑO

Para estructurar el IVR se va a considerar los siguientes puntos:

- La información que se maneja, a pesar de ser de carácter personal no se la considera confidencial, por tal motivo no es necesario un sistema de autenticación de usuarios.

- La interacción del sistema con el usuario es a través de una línea telefónica que en ocasiones puede presentar niveles muy altos de ruido y degradar la certeza en el reconocimiento de voz, para esto se debe tener como alternativa la interacción a través de tonos DTMF para realizar las mismas funciones a través del recorrido por el sistema, sin embargo, presentar una interfaz en la que el diálogo guiado sería por ejemplo *"Para consulta de saldos diga consultas o digite uno, para estado de préstamos diga préstamos o digite dos, … "* llevaría a una interacción repetitiva y el uso del reconocimiento de voz casi no tendría sentido. Además, un motor de reconocimiento natural de voz hace la distinción entre fonemas de acuerdo a las alternativas vigentes en dicho momento, por ejemplo si en cierto punto del diálogo el usuario debe elegir entre tres opciones distintas quizá la certeza del reconocimiento no se degrade demasiado a pesar del ruido en la línea.
- Se diferenciará claramente entre un sistema que incorpore reconocimiento de voz y tonos DTMF con uno que sea puramente de tonos DTMF. En el primero, el principal agente y quien lleva el control de la entrada del usuario es el motor de reconocimiento de voz, quien a la vez es capaz de detectar tonos DTMF, esto quiere decir que se mantendrá activo un archivo de gramática de voz y uno de tonos DTMF simultáneamente (*mode="voice"* y *mode="dtmf"* respectivamente) y se obtendrá la respuesta del usuario independientemente de lo que éste haya decidido utilizar. Como se puede apreciar el usuario quizá decida utilizar enteramente el teclado de su teléfono y el motor de reconocimiento deberá estar activo consumiendo una licencia o puerto.
- Tomando en cuenta el punto anterior se limitaría el número de usuarios al número de licencias adquiridas del motor de reconocimiento, sin embargo, es posible mantener a la vez un *"IVR paralelo*" que maneje solamente tonos DTMF, así, cuando se hayan agotado las licencias el usuario tendrá la opción de utilizar el sistema con el teclado de su teléfono. Esto se puede hacer debido a que generalmente un motor de reconocimiento de voz es un paquete agregado a un sistema de telefonía en donde se puedan crear IVRs, ya sea una *PBX* en hardware o software, o un intérprete de *VXML*.

- Para que un usuario ingrese en el sistema debería identificarse de algún modo, se tienen dos alternativas, el número de cédula y el número único. Tomando en cuenta que la certeza de un motor de reconocimiento de voz depende de la semejanza entre las palabras o frases que se esperan al mismo tiempo del usuario y en este caso al ser un número de varios dígitos habrá más posibilidades o combinaciones mientras más dígitos existan, debido a esto se ha elegido el número único, pues a diferencia del número de cédula de diez dígitos este tiene como máximo nueve.
- El número único será el requisito para ingresar en el sistema, éste podrá ser pronunciado o digitado en el teclado del teléfono; para el primer caso es conveniente pedir una confirmación al usuario.
- La información del estado de los certificados de prácticas preprofesionales y de los planes de proyectos de titulación se la proporcionará directamente al usuario, las calificaciones por otra parte deberían proporcionarse ya sea por bimestre o por materia, según elija el usuario.
- El sistema permitirá al usuario volver a menús anteriores para utilizar otros servicios, por ejemplo en el caso de las calificaciones para consultar de otro bimestre o materia.

2.2.2 FLUJO DE OPCIONES DEL IVR

De acuerdo a las consideraciones anteriores el flujo del IVR empezará con una bienvenida al usuario para enseguida solicitarle el ingreso de su número único, para el caso de que este haya sido detectado a través del reconocimiento de voz se pedirá una confirmación. Una vez identificado el usuario se le preguntará que desea hacer, consultar sus notas, el estado de los certificados de prácticas preprofesionales o del plan de proyecto de titulación. En los dos últimos casos se le proporcionará la información correspondiente y se concluirá el diálogo si el usuario así lo desea o puede utilizar otro servicio de los disponibles y regresar al menú que aparece después de identificarse con su número único. Para el caso de la consulta de notas se ingresa en un menú adicional en donde se le preguntará al usuario si desea conocer todas sus notas, las de un bimestre particular o

supletorio, o las específicas de una materia. Si existiese error en la detección se solicitará nuevamente el ingreso de la información.

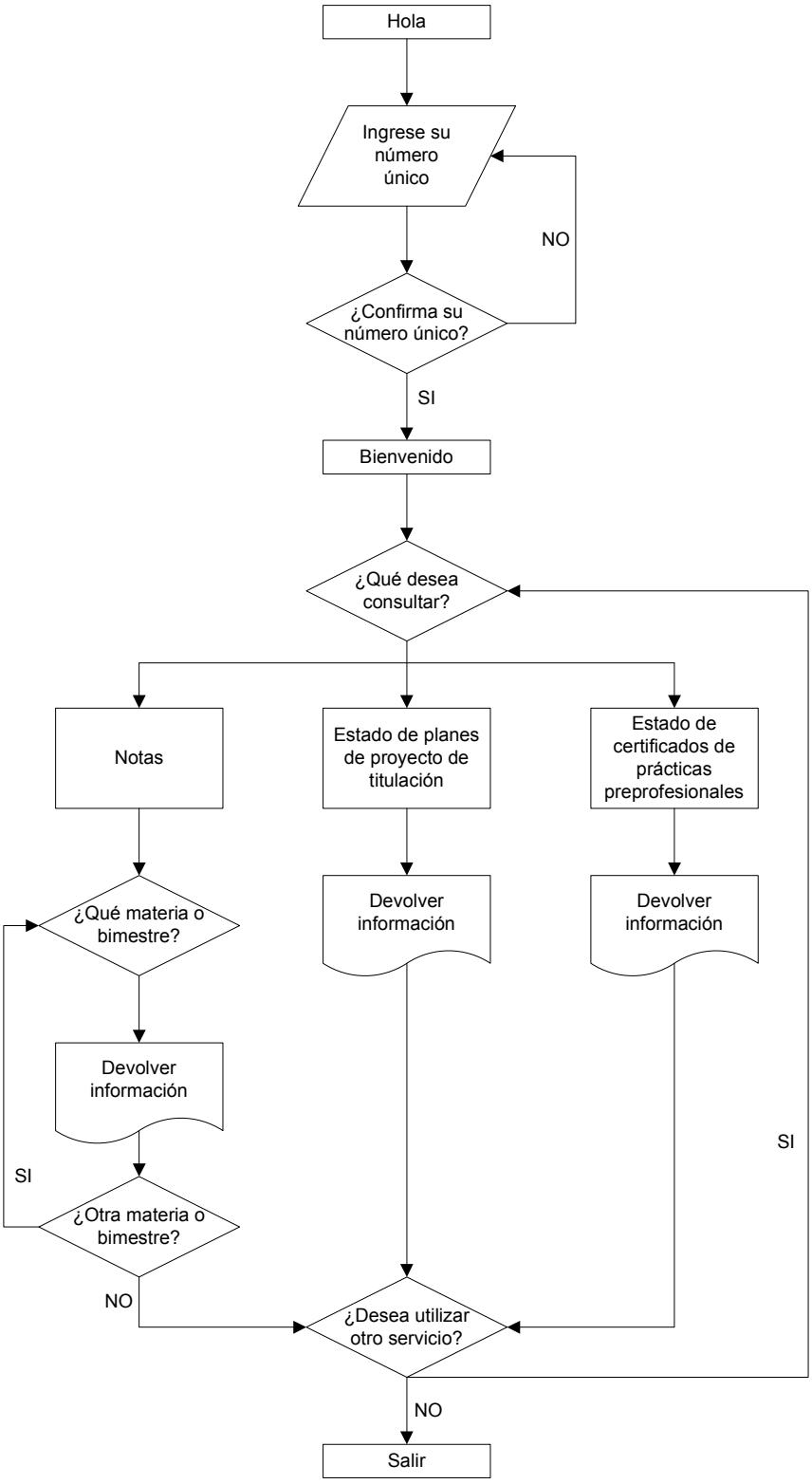

figura 2.1: Flujo del IVR

La *figura 2.1* muestra el flujo del *IVR* de la manera como se le presentará al usuario, a lo largo de este capítulo y tomando en cuenta consideraciones adicionales se irá analizando este esquema.

2.3 ANÁLISIS DE LOS PATRONES DE PALABRAS QUE DEBERÁN SER RECONOCIDOS

El reconocimiento natural de voz es un proceso estadístico en el que se compara la señal acústica de las palabras o frases que el usuario pronuncia con un conjunto de señales que se guardan en una base de datos llamada modelo acústico, el mismo que se relaciona con otro conjunto de patrones que dependen del idioma y dialecto de una región particular llamado modelo del lenguaje; en conjunto determinan la relación que existe entre la pronunciación de determinadas palabras y las señales acústicas que producen las mismas al ser articuladas por una persona independientemente de su edad o el volumen de su voz. La comparación se realiza en base a la cantidad de tokens[1] que se mantengan activos en cada instante, en otras palabras, es la cantidad de posibilidades que se le da al usuario para responder, por ejemplo, si se pide al usuario decir el color de sus ojos se debe tener un archivo de gramática en donde se especifique todos los colores existentes a manera de alternativas, lo que hace el motor de reconocimiento es comparar la señal acústica de la pronunciación del usuario con todas las posibles señales que se producirían al pronunciar cada uno de los colores en el lenguaje en el que hablará el usuario; la mejor coincidencia será el resultado que genere el motor.

Entre un conjunto de alternativas existe distinta complejidad en la discriminación por parte del motor, los tokens que se esperan pueden tener cierta semejanza y degradarán la calidad del reconocimiento, por ejemplo las palabras *“hombro”* y *“hombre”* se diferencian solamente en su última letra y éste será el único punto en donde el motor pondrá énfasis para hacer la distinción entre ambas. Si un token

[1] Se considera como token a una palabra o frase que puede ser pronunciada por el usuario en determinado momento.

contiene a otro, por ejemplo *“treinta”* y *“treinta y tres”*, también causará problemas al motor, sobretodo si el nivel de ruido es alto. La cantidad de tokens activos al momento de realizar el reconocimiento juega también un papel importante, entre más alternativas tenga el motor menor será su nivel de certeza. Se debe tener muy en cuenta que las alternativas que posea el usuario en determinado momento sea el mismo número de alternativas que tenga el motor de reconocimiento, es común que se dejen activos archivos de gramática que ya no serán usados. Para clarificar este asunto retomemos el caso en el que se pregunta el color de ojos del usuario, si luego de haber detectado un resultado con un nivel bajo de certeza se le pide una confirmación para que responda *“si”* o *“no”* y no se ha desactivado el archivo de gramática en el que se hallan todos los colores el motor no discriminará la respuesta solamente entre los patrones de las palabras *“si”* y *“no”* sino también entre los patrones de todos los colores del archivo anterior.

Las consideraciones mencionadas anteriormente ayudarán a mejorar la calidad del sistema en cuanto a respuesta, pero por otra parte, hay que considerar también la facilidad que se le de al usuario para utilizarlo, en lo posible se ajustará el sistema al usuario y no viceversa.

Según el esquema de la *figura 2.1* que muestra el flujo del *IVR* para este proyecto, se analizará por separado el reconocimiento del número único que evidentemente es un identificador para cada estudiante y el recorrido por los demás menús que se traduce en el reconocimiento de palabras solamente.

2.3.1 ANÁLISIS DEL NÚMERO ÚNICO

El número único es un código de identificación de los estudiantes en la Escuela Politécnica Nacional y es asignado desde el momento que se ingresa a la misma. Se hace preciso su análisis en el presente proyecto porque el usuario necesitará pronunciarlo o digitarlo para poder realizar las consultas en el sistema.

2.3.1.1 Estructura del número único

El número único además de ser un identificador de cada estudiante es también un código que muestra el año y periodo de ingreso a la universidad. Hasta el año 2002 se trabajaba con una longitud de siete dígitos, desde el año siguiente a este se aumentó dos dígitos más, sin mayor modificación en su estructura básica.

Cuando se tiene siete dígitos de longitud, los dos primeros corresponden a los dos últimos dígitos del año de ingreso del estudiante a su respectiva carrera, el siguiente dígito corresponde al periodo lectivo en el que ingresa, *"1"* para el periodo *octubre-marzo* y *"2"* para el periodo *abril-septiembre* de cada año, finalmente los últimos cuatro dígitos corresponden a una numeración secuencial de los estudiantes, por ejemplo el número *"0210017"* corresponde a un estudiante que ingresó en el primer periodo del año 2002 (*octubre 2001-marzo 2002*) y que fue el estudiante número 0017 que se registró en la Escuela Politécnica Nacional.

Cuando se tienen nueve dígitos la situación es similar, se agregan dos dígitos al inicio para indicar el año con cuatro dígitos y no solamente con dos, así el número *"200420315"* corresponde a un estudiante que ingresó en el segundo periodo del año 2004 y se le asignó un número de registro igual a 0315.

2.3.1.2 Análisis probabilístico del número único

El número único compuesto de siete o nueve dígitos es un identificador más que una cantidad, por tal motivo no existen reglas al momento de pronunciarlo como se lee generalmente una cantidad, con esto se quiere decir que distintas personas pueden leer un mismo número de distinta manera, por ejemplo para el número *"0121678"* son válidas entre otras, cualquiera de las siguientes expresiones:

"cero uno dos uno seis siete ocho"
"cero doce dieciséis setenta y ocho"
"cero ciento veintiuno seis siete ocho"
"cero ciento veintiuno sesenta y siete ocho"
"cero uno doscientos dieciséis siete ocho"

“cero uno dos mil seiscientos setenta y ocho”

Se puede apreciar claramente que un identificador como este puede expresarse en combinaciones de dígitos, decenas, centenas, miles, etc. y un buen sistema debería aceptar cualquiera de estas entradas y generar el mismo resultado a la salida.

Pedir que se pronuncie dígito a dígito causa una mala aceptación del sistema por parte de los usuarios debido a que cada uno tenemos una manera particular de memorizar números similares a este, lo practicamos muy a menudo con el número de la cédula de identidad, números telefónicos, etc. Además no tendría sentido el usar el reconocimiento natural de voz; pulsar cada dígito en el teclado del teléfono lograría el mismo objetivo y quizá de manera más eficiente. De todas maneras se hará un análisis del número único desde el punto de vista combinatorio tanto para el caso que contenga siete dígitos como nueve.

Combinaciones de dígitos, decenas, centenas, miles, etc. pueden formar una cantidad de determinadas cifras, por ejemplo, para el caso del número único de siete cifras se pueden formar las siguientes combinaciones de dígitos, decenas y centenas:

7 dígitos
5 dígitos, 1 decena
3 dígitos, 2 decenas
1 dígito, 3 decenas
4 dígitos, 1 centena
1 dígito, 2 centenas
2 dígitos, 1 decena, 1 centena
2 decenas, 1 centena

Todas las combinaciones suman siete cifras en total pero cada una contiene distintas permutaciones con repetición de elementos, para aclarar este asunto se

considera la segunda combinación (5 dígitos, 1 decena), para la cual, todas las siguientes permutaciones corresponderían:

1 decena, 5 dígitos
1 dígito, 1 decena, 4 dígitos
2 dígitos, 1 decena, 3 dígitos
3 dígitos, 1 decena, 2 dígitos
4 dígitos, 1 decena, 1 dígito
5 dígitos, 1 decena

El número de permutaciones para cada combinación que forme el número único se puede determinar con la fórmula para el cálculo de permutaciones de un conjunto de *n* elementos en donde K_i elementos se repiten:

$$P_{n,(k_1,k_2,k_3,...,k_n)} = \frac{n!}{k_1!k_2!k_3!...k_n!} \qquad \text{(ecuación 2.1)}$$

Para el ejemplo en cuestión en donde se tienen *"5 dígitos y 1 decena"* *n* sería la suma de los elementos, o sea *5+1=6* y los elementos q se repiten son los *5* dígitos:

$$P_{6,(5,1)} = \frac{6!}{5!.1!}$$

$$P_{6,(5,1)} = 6$$

Que son justamente el número de permutaciones que se indicó antes.

Con esta fórmula se puede calcular la cantidad de alternativas que tendría un usuario para pronunciar su número único, la *tabla 2.1* muestra el caso cuando se tienen siete números tomando en cuenta solamente dígitos, decenas y centenas. Para el caso de nueve dígitos el resultado es similar, la *tabla 2.2* muestra todas las posibilidades.

	Dígitos	***Decenas***	***Centenas***	***Perm.***	***Total***
7 dígitos	7	0	0	1	10000002
5 dígitos, 1 decena	5	1	0	6	600546
3 dígitos, 2 decenas	3	2	0	10	91010
1 dígito, 3 decenas	1	3	0	4	2916044
4 dígitos, 1 centena	4	0	1	5	54505
1 dígito, 2 centenas	1	0	2	3	2430033
2 dígitos, 1 decena, 1 centena	2	1	1	12	13080
2 decenas, 1 centena	0	2	1	3	27003
			Total	**44**	**16132223**

tabla 2.1: Permutaciones del número único de siete dígitos

	Dígitos	***Decenas***	***Centenas***	***Perm.***	***Total***
9 dígitos	9	0	0	1	1000000002
7 dígitos, 1 decena	7	1	0	8	80000728
5 dígitos, 2 decenas	5	2	0	21	2270121
3 dígitos, 3 decenas	3	3	0	20	14600020
1 dígito, 4 decenas	1	4	0	5	328050055
6 dígitos, 1 centena	6	0	1	7	7006307
3 dígitos, 2 centenas	3	0	2	10	8110010
3 centenas	0	0	3	1	729000002
4 dígitos, 1 decena, 1 centena	4	1	1	30	329700
2 dígitos, 2 decenas, 1 centena	2	2	1	30	273000
3 decenas, 1 centena	0	3	1	4	2919604
1 dígito, 1 decena, 2 centenas	1	1	2	12	9721200
			Total	**149**	**2182280749**

tabla 2.2: Permutaciones del número único de nueve dígitos

Ahora bien, un usuario podría tener su número único de siete o nueve dígitos, entonces el número total de permutaciones sería:

$$P_T = 44 + 149 = 193$$

Sin embargo lo que se está analizando son las posibles permutaciones que se dan para una cantidad de siete o nueve dígitos, el motor de reconocimiento no debe discriminar entre ciento cuarenta y nueve alternativas, hay que tomar en

cuenta que un dígito comprende 10 posibilidades (desde el 0 hasta el 9), una decena 90 (desde el 10 hasta el 99) y una centena 900 (desde el 100 hasta el 999), entonces para contabilizar el número total de alternativas o posibilidades se deberá multiplicar el número de permutaciones por diez elevado al número de dígitos, noventa elevado al número de decenas y 900 elevado al número centenas que se puedan tener. La última columna de la *tabla 2.1* y *tabla 2.2* muestran estos resultados, tomando en cuenta que no se considera ninguna restricción debido a la estructura del número único, por ejemplo, que en la tercera posición de un número de siete dígitos no se tendría que considerar que se tienen diez posibilidades sino solamente dos, el número *"1"* o *"2"* que corresponde al periodo en el que el estudiante ingreso a la Escuela Politécnica Nacional; de todos modos el cálculo muestra la cantidad de alternativas que se podrían tener si el número único no tuviera una estructura definida y habría la posibilidad de formar cualquier combinación con los siete o nueve dígitos; el número total de alternativas se muestra a continuación.

$$A = 16132223 + 2182280749 = 2198412972$$

La cantidad obviamente es demasiado grande y no se debe olvidar que se han considerado combinaciones de hasta centenas, quizá aumentaría si se consideraran unidades y decenas de mil, aunque es común que una persona recuerde un número telefónico o de cédula de identidad en combinaciones de hasta tres cifras, los demás casos suceden de manera muy particular. En la realidad, la probabilidad de que el motor acierte al reconocer una de las combinaciones no es exactamente 1/2198412972, pues utiliza mecanismos que ayudan a discriminar la respuesta según va sucediendo la señal acústica, los mismos que están fuera del alcance de este proyecto. Sin embargo, éste análisis nos da una idea de la cantidad de posibilidades que puede existir y con los resultados obtenidos se puede afirmar que el motor de reconocimiento enfrentará grandes problemas, la cantidad de alternativas con las que debe trabajar es demasiado extensa. Para mejorar este asunto se considerará solamente las permutaciones de dígitos y decenas, la *tabla 2.3* muestra los resultados:

	Dígitos	***Decenas***	***Centenas***	***Perm.***	***Total***
7 dígitos	7	0	0	1	10000002
5 dígitos, 1 decena	5	1	0	6	600546
3 dígitos, 2 decenas	3	2	0	10	91010
1 dígito, 3 decenas	1	3	0	4	2916044
9 dígitos	9	0	0	1	1000000002
7 dígitos, 1 decena	7	1	0	8	80000728
5 dígitos, 2 decenas	5	2	0	21	2270121
3 dígitos, 3 decenas	3	3	0	20	14600020
1 dígito, 4 decenas	1	4	0	5	328050055
			Total:	**76**	**1438528528**

tabla 2.3: Permutaciones considerando solamente dígitos y decenas

El número total de posibilidades disminuye en un 35% aproximadamente, pero el costo es que se debe pronunciar el número único máximo en combinaciones de dos cifras. Por una parte está la inconformidad del usuario ante el sistema que le obliga a decir su número de esta manera, pero por otra parte, el sistema será menos confiable al emitir su resultado si se considera el caso de la inclusión de las centenas. Tomando en cuenta que la cantidad de usuarios que entren en el segundo caso es una fracción del total de usuarios potenciales y que en el primer caso la mala calidad del sistema afectaría a todos los usuarios, se optará por manejar combinaciones de hasta dos cifras.

Sin embargo esta cantidad de alternativas con las que deberá trabajar el motor de reconocimiento de voz se puede disminuir si se considera la estructura del número único. Para el caso de los siete dígitos y revisando la base de datos del SAE de la carrera de Electrónica y Redes de Información se tienen registros de estudiantes que ingresaron desde 1993, por lo que el primer dígito sería *"9"* o *"0"*[2], es decir hay solo dos posibilidades. El segundo dígito deberá ser entre *"0"* y *"9"*, pero el tercero será solamente *"1"* o *"2"*, nuevamente existen solo dos posibilidades.

[2] Se considera los dos últimos dígitos de los años entre 1993 y 2002

Para el caso del número único de nueve dígitos la situación es bastante parecida, los cuatro primeros corresponden al año de ingreso del estudiante y se considera que los dos primeros son números fijos (el "20" corresponde al inicio del año), el tercero sería *"0"* o *"1"* (el sistema sería válido hasta el año 2019), el cuarto podría ser cualquier dígito entre *"0"* y *"9"* y el quinto será *"1"* o *"2"*. Sin embargo aquí se puede hacer una consideración adicional, al empezar el número único con el año de ingreso hace que en la mayor parte de los casos y por nuestra naturaleza humana empecemos diciendo el año como lo pronunciamos habitualmente, esto haría que el análisis de permutaciones se haga solamente con los cinco últimos dígitos y si se prevé que el sistema tenga una vida útil de cinco años desde su implementación se tendrían solamente diez posibilidades (desde el año 2003 hasta el 2012). De todas maneras es importante observar que resultado se obtiene, la *tabla 2.4* muestra que se disminuyó tanto el número de permutaciones como el número total de posibilidades con las cuales el motor de reconocimiento va a trabajar, esto es un punto a favor para la efectividad del sistema, además que se le facilita al usuario el ingreso de la información.

	Dígitos	***Decenas***	***Centenas***	***Perm.***	***Total***
7 dígitos	7	0	0	1	10000002
5 dígitos, 1 decena	5	1	0	6	600546
3 dígitos, 2 decenas	3	2	0	10	91010
1 dígito, 3 decenas	1	3	0	4	2916044
5 dígitos	5	0	0	1	100002
3 dígitos, 1 decena	3	1	0	4	4364
1 dígitos, 2 decenas	1	2	0	3	24333
			Total:	**29**	**14894592**

tabla 2.4[3]: Permutaciones considerando solamente los cinco últimos dígitos

El resultado es bastante alentador, se disminuyó por debajo del 1% la cantidad de posibilidades comparando con el número inicial en el que se incluían centenas.

[3] El resultado de la celda inferior derecha se calcula sumando el número de las permutaciones para el caso de siete dígitos más diez veces el número de permutaciones para cinco dígitos.

#	***Posición de la cifra:***	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1	1 digito, 1 decena, 1 digito, 1 decena, 1 digito							
2	1 digito, 1 decena, 2 dígitos, 1 decena							
3	1 digito, 1 decena, 4 dígitos							
4	1 digito, 2 decenas, 2 dígitos							
5	1 digito, 3 decenas							
6	2 dígitos, 1 decena, 1 digito, 1 decena							
7	2 dígitos, 1 decena, 3 dígitos							
8	2 dígitos, 2 decenas, 1 digito							
9	3 dígitos, 1 decena, 2 dígitos							
10	3 dígitos, 2 decenas							
11	4 dígitos, 1 decena, 1 digito							
12	5 dígitos, 1 decena							
13	7 dígitos							
14	1 decena, 1 digito, 1 decena, 2 dígitos							
15	1 decena, 1 digito, 2 decenas							
16	1 decena, 2 dígitos, 1 decena, 1 digito							
17	1 decena, 3 dígitos, 1 decena							
18	1 decena, 5 dígitos							
19	2 decenas, 1 digito, 1 decena							
20	2 decenas, 3 dígitos							
21	3 decenas, 1 digito							

tabla 2.5: Permutaciones de siete dígitos considerando la estructura del número único

#	***Posición de la cifra:***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***
1	1 digito, 1 decena, 2 dígitos									
2	1 digito, 2 decenas									
3	2 dígitos, 1 decena, 1 digito									
4	3 dígitos, 1 decena									
5	5 dígitos									
6	1 decena, 1 digito, 1 decena									
7	1 decena, 3 dígitos									
8	2 decenas, 1 digito									

tabla 2.6: Permutaciones de nueve dígitos considerando la estructura del número único

Ahora bien, se tomará en cuenta las consideraciones en la estructura del número único para disminuir aún más el conjunto de posibilidades totales, para esto es necesario escribir todas las permutaciones en combinaciones de dígitos y

decenas que pueden existir, en las *tablas 2.5* y *2.6* se muestra estas combinaciones de una manera más explícita, las columnas de la derecha indican las combinaciones, de color gris claro se muestran las decenas y de un gris un poco más oscuro las unidades o dígitos como se los ha denominado.

#	Posición de la cifra:	1	2	3	4	5	6	7	
1	1 digito, 1 decena, 1 digito, 1 decena, 1 digito	2	20		10	90		10	360000
2	1 digito, 1 decena, 2 dígitos, 1 decena	2	20		10	10	90		360000
3	1 digito, 1 decena, 4 dígitos	2	20		10	10	10	10	400000
4	1 digito, 2 decenas, 2 dígitos	2	20		90		10	10	360000
5	1 digito, 3 decenas	2	20		90		90		324000
6	2 dígitos, 1 decena, 1 digito, 1 decena	2	10	20		10	90		360000
7	2 dígitos, 1 decena, 3 dígitos	2	10	20		10	10	10	400000
8	2 dígitos, 2 decenas, 1 digito	2	10	20		90		10	360000
9	3 dígitos, 1 decena, 2 dígitos	2	10	2	90		10	10	360000
10	3 dígitos, 2 decenas	2	10	2	90		90		324000
11	4 dígitos, 1 decena, 1 digito	2	10	2	10	90		10	360000
12	5 dígitos, 1 decena	2	10	2	10	10	90		360000
13	7 dígitos	2	10	2	10	10	10	10	400000
14	1 decena, 1 digito, 1 decena, 2 dígitos	10		2	90		10	10	180000
15	1 decena, 1 digito, 2 decenas	10		2	90		90		162000
16	1 decena, 2 dígitos, 1 decena, 1 digito	10		2	10	90		10	180000
17	1 decena, 3 dígitos, 1 decena	10		2	10	10	90		180000
18	1 decena, 5 dígitos	10		2	10	10	10	10	200000
19	2 decenas, 1 digito, 1 decena	10	20			10	90		180000
20	2 decenas, 3 dígitos	10	20			10	10	10	200000
21	3 decenas, 1 digito	10	20			90		10	180000
								Total:	6190000

tabla 2.7: Permutaciones de siete dígitos con su respectiva probabilidad

Cada posición en el número de siete o nueve cifras tiene una probabilidad específica determinada por la estructura del número único, la *tabla 2.7* muestra esta probabilidad para el caso de los números de siete cifras. Como se mencionó anteriormente, el primer dígito puede ser *"9"* o *"0"*, es decir hay dos posibilidades, el segundo dígito puede estar entre *"0"* y *"9"*, entonces si el número empieza con una decena se pueden tener solamente los números entre *"90"* y *"99"*, es decir, solamente diez posibilidades. El tercer dígito puede ser solamente *"1"* o *"2"*,

nuevamente dos posibilidades, pero si se sigue el cuarto dígito (entre “0” y “9”) para formar una decena se tendría números entre *“10”* y *“29”*, o sea, veinte posibilidades. Al final se multiplica cada una de las probabilidades en cada posición para obtener el resultado final.

De igual manera se procede con la cantidad de nueve cifras, tomando en cuenta que las primeras cuatro cifras aportan solamente con diez posibilidades, desde el año *“2003”* hasta el *“2012”*. La *tabla 2.8* muestra los resultados.

#	***Posición de la cifra:***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	
1	1 digito, 1 decena, 2 dígitos	**10**				**2**	**90**		**10**	**10**	180000
2	1 digito, 2 decenas	**10**				**2**	**90**		**90**		162000
3	2 dígitos, 1 decena, 1 digito	**10**				**2**	**10**	**90**		**10**	180000
4	3 dígitos, 1 decena	**10**				**2**	**10**	**10**	**90**		180000
5	5 dígitos	**10**				**2**	**10**	**10**	**10**	**10**	200000
6	1 decena, 1 digito, 1 decena	**10**				**20**		**10**	**90**		180000
7	1 decena, 3 dígitos	**10**				**20**		**10**	**10**	**10**	200000
8	2 decenas, 1 digito	**10**				**20**		**90**		**10**	180000
										Total:	**1462000**

tabla 2.8: Permutaciones de nueve dígitos con su respectiva probabilidad

Al final sumamos el total de posibilidades tanto para siete como nueve cifras y observamos la cantidad de alternativas con las que va a trabajar el motor de reconocimiento de voz, el último resultado obtenido se redujo aproximadamente a la mitad.

$$A_T = 6190000 + 1462000 = 7652000$$

Este análisis de las posibles combinaciones que podría tener el número único más que proporcionar un estudio probabilístico real con el que se determine la efectividad del motor de reconocimiento de voz (tema que sobrepasa los objetivos de este proyecto), ayuda a comprender de que manera se le agrega complejidad al sistema al darle mayor flexibilidad al usuario, si se obligara al mismo pronunciar dígito a dígito este estudio estaría demás. También, proporciona información

acerca de cómo estructurar la gramática de reconocimiento que se va a utilizar para el número único, con la especificación *SRGS* que se estudió en el capítulo uno es posible estructurar un archivo que comprenda todas las permutaciones posibles, incluso instruir al motor de reconocimiento cuales alternativas se tienen en cada cifra del número único. Al final, lo que se obtiene es un sistema con mayor certeza en el entendimiento con el usuario, incrementando la consecución de los objetivos del mismo al utilizar el IVR.

2.3.2 ANÁLISIS DE LA NAVEGACIÓN POR EL MENÚ

La navegación por el menú no lleva un análisis tan minucioso como lo fue el número único, se trata simplemente de configurar en el sistema las posibles entradas del usuario de un conjunto muy pequeño de alternativas. Según el diagrama de la *figura 2.1* se distinguen dos submenús, cuando se le pregunta al usuario ¿Qué desea consultar?, al que se lo llamará *"menú principal"*, y cuando éste indica al sistema si desea saber sus notas por materia, bimestre o la totalidad de ellas; a este submenú se lo llamará *"menú de notas"*.

2.3.2.1 Menú principal

El sistema en cuestión en este proyecto tiene por objetivo brindar a los estudiantes de la carrera de Electrónica y Redes de Información de la Escuela Politécnica Nacional los siguientes servicios:

- Consulta de notas.
- Consulta de estado de planes de proyectos de titulación.
- Consulta de estado de certificados de prácticas preprofesionales.

Una vez confirmado el número único que el estudiante pronunció o una vez recibido un número válido (de siete o nueve dígitos) mediante tonos DTMF, se debe preguntar al usuario mediante un audio pregrabado *"¿Qué desea consultar?"*, y en el archivo de gramática obviamente deberán constar las tres alternativas, tomando en cuenta que la respuesta se puede discriminar

simplemente con el uso de palabras claves: *“notas”* para el primer caso, *“planes de proyectos”* (en incluso *“tesis”*) para el segundo y *“prácticas”* para el tercero.

El archivo de gramática no necesariamente debe contener tres alternativas, podemos ayudar al motor a obtener una mejor certeza en el reconocimiento agrupando posibles respuestas que apunten al mismo objetivo, por ejemplo para la consulta de estado de planes de proyectos de titulación después de preguntar al usuario *“¿Qué desea consultar?”* se podrían tener entre otras las siguientes respuestas:

- Planes de proyectos de titulación.
- Proyectos de titulación.
- Tesis.
- Consultar planes reproyectos de titulación.
- Consultar proyectos de titulación.
- Consultar tesis.

Las tres últimas opciones pueden suceder debido a que en el lenguaje común acostumbramos a responder repitiendo la pregunta o parte de ésta.

Para las seis o más opciones que se podrían tener se debe generar una misma respuesta al sistema que es claramente la consulta de planes de proyectos de titulación, esto se lo hace mediante *interpretación semántica* y la recomendación SISR que se estudió en el capítulo uno. Con las otras dos alternativas se debe proceder de manera similar.

2.3.2.2 Menú de notas

Para el menú de notas la situación es semejante aunque aquí se considera algo adicional. Existe un conjunto de materias registradas en la carrera que se dictan habitualmente y un estudiante recibe un subconjunto de ellas, sería incorrecto crear un solo archivo de gramática en donde consten todas las materias, el motor podría en muchos casos equivocarse y obtener materias que el estudiante ni

siquiera toma, el sistema debería volver a preguntar y crear con esto disconformidad al usuario.

La solución sería tener archivos de gramática dinámicos, que se creen y se activen justo antes de preguntarle a usuario y que incorporen solamente las materias que él tome (en base a una consulta previa al SAE) además de *"primer bimestre"*, *"segundo bimestre"*, *"supletorio"* y *"todas"* (o *"todas las materias"*). Esta solución obviamente consumirá recursos y quizá sea innecesaria si se considera que las primeras notas están disponibles al final del primer bimestre y también que luego de matrículas extraordinarias un estudiante no puede inscribirse en nuevas materias o anular la inscripción de las que ya tiene en su matrícula.

Con esto se quiere decir que los archivos de gramática que contengan las materias correspondientes para cada estudiante se pueden crear una vez que se hayan cerrado los procesos de matriculación, pues se supone que en este punto aún no hay registros de notas y luego de él un estudiante ya no podrá anular o matricularse en nuevas materias. El proceso se ejecutará una vez al inicio del semestre y los archivos estarán disponibles para todo este periodo.

Se considera también que algunas materias suelen pronunciarse de diversas maneras, por ejemplo la materia *"Teoría de Comunicaciones"* suele conocerse más como *"TC"*, incluso algún estudiante quizá la llame *"Teoría"* y otro *"Comunicaciones"*, será preciso incluir todas estas variaciones en el archivo de gramática, recordando siempre no utilizar palabras similares o de pronunciación semejante que al final confundan al motor de reconocimiento.

2.4 INTEGRACIÓN CON LA BASE DE DATOS SQL SERVER

La *Unidad de Gestión de Información* de la Escuela Politécnica Nacional (UGI) trabaja con una estructura estándar de base de datos denominada SAE, la misma que se encuentra instalada en la secretaría de cada carrera, generalmente en un servidor con sistema operativo *Windows 2000* y el sistema de gestión y administración de bases de batos *SQL Server 7* de *Microsoft*. Esta base de datos

se actualiza cada semestre en todas las carreras, es decir se crea una nueva con el nombre *“SAEIERI”* y como sufijo el periodo lectivo.

En su mayoría el SAE contiene información confidencial de la Universidad, datos de los estudiantes y de las carreras, por tal motivo la UGI tiene la enorme responsabilidad del buen funcionamiento, integridad y administración. Solamente ellos tienen un completo acceso y procuran en lo mínimo conceder permisos de lectura o escritura a este sistema, solamente las secretarias de cada carrera pueden ingresar información, específicamente las calificaciones de los estudiantes.

Este hecho es de vital importancia para el desarrollo del presente proyecto pues la consulta al SAE es una parte fundamental, es así que se ha propuesto realizar las consultas a través de procedimientos almacenados, los mismos que solamente accedan a información precisa y necesaria. Además, se trabaja con información adicional que no consta en la estructura del SAE de la carrera y se precisa que se debe crear una base de datos adicional en el mismo servidor *SQL Server 7* para manejar lo que tiene que ver con la consulta del estado de planes de proyectos de titulación y certificados de prácticas preprofesionales.

2.4.1 DISEÑO DE LA BASE DE DATOS PARA EL MANEJO DE SERVICIOS ADICIONALES

Como se mencionó antes, servicios adicionales corresponden a la consulta del estado de planes de proyectos de titulación y certificados de prácticas preprofesionales. Un estudiante que está matriculado en la materia *“Proyecto de Titulación”* antiguamente conocida como *“Tesis”* está en la capacidad de presentar ante la Subcomisión Académica de la carrera su plan para que sea revisado. Un plan de proyecto de titulación puede o no ser aprobado, de no serlo, es muy común que se necesite realizar ciertos cambios en el plan antes de ser aprobado.

De manera similar sucede con los certificados de prácticas profesionales, una vez que un estudiante haya concluido su periodo de práctica en alguna empresa, ésta tiene el deber de certificar el tiempo que estuvo el estudiante, que actividades desarrolló y su desenvolvimiento en la empresa. Este certificado es revisado también por la Subcomisión Académica y si no existe ninguna incongruencia será aprobado, de lo contrario se lo devuelve para su modificación y más tarde una nueva revisión.

En ambos casos, luego que cada uno de dichos documentos es revisado pasa a tener uno de los siguientes tres estados:

- No aprobado.
- Pendiente de aprobación.
- Aprobado.

Para el segundo caso se deja en constancia cuales cambios se deben hacer o el motivo para dejar pendiente su aprobación, de todas formas el objetivo de este proyecto, aparte de la consulta de notas, es proporcionar dicha información a los estudiantes para evitar su concurrencia hasta la secretaría de la carrera.

Como se ha mencionado en secciones anteriores se va a crear una base de datos para almacenar esta información, y se debe partir de la lista de estudiantes que cumplirían con los requisitos para presentar su plan de proyecto de titulación o el certificado de prácticas preprofesionales. Para el primer caso se debe estar matriculado en esa materia y para el segundo se necesita tener aprobado al menos el 70% de los créditos de toda la carrera aunque pueden existir excepciones. Lo primero es crear una tabla que almacene este grupo de estudiantes, que será un subconjunto de todos los que consten en la base de datos del SAE.

A esta tabla se la llamará desde ahora *“ESTUDIANTES”* y la única manera de ingresar datos en ella será mediante registros completos que se obtendrán a través de las respectivas consultas al SAE, con esto se garantizará la integridad

en la información de estudiantes. No se necesita almacenar toda la información que guarda el SAE de cada estudiante, sino simplemente la que se utilice para la implementación del servicio en cuestión; la *tabla 2.9* muestra las columnas que formarán la tabla *"ESTUDIANTES"*. El número único es una cadena caracteres numéricos de longitud siete o nueve, el nombre del estudiante una cadena de caracteres que tiene como límite cincuenta, el número máximo de créditos aprobados está alrededor de 215, pues depende de cuantas materias optativas haya aprobado el estudiante durante el transcurso de su carrera, de todos modos se lo ha considerado 230. El último campo indicará si el estudiante está o no matriculado en el *"Proyecto de Titulación"*. Todos los campos deberán ser obligatorios, es decir, no podrán contener valores nulos.

ESTUDIANTES	***Tipo de dato***	***Longitud máxima***	***Obligatorio***
Número único	cadena de dígitos	9	Si
Nombre del estudiante	cadena de caracteres	50	Si
Créditos aprobados	entero	230	Si
Matrícula en tesis	booleano	-	Si

tabla 2.9: Descripción de los campos de la tabla ESTUDIANTES

Un estudiante puede presentar más de un plan de proyecto de titulación si no se le ha aprobado uno anteriormente, o puede contener más de un certificado de prácticas preprofesionales si realizó sus pasantías en distintas empresas, pero en determinado instante solamente un certificado o plan será el que esté pendiente de revisión en la Subcomisión Académica y el IVR proporcionará la información solamente de éste.

Considerando lo anterior se necesitan dos tablas adicionales, una que almacene la información de los planes de proyectos de titulación y la otra los certificados de prácticas preprofesionales, a la primera se la llamará *"PROYECTOS"* y a la segunda *"PRACTICAS"*. La tabla *"ESTUDIANTES"* se relacionará con cada una de éstas mediante el campo *"número único"* y el tipo de relación será de uno a varios.

A diferencia de la tabla *"ESTUDIANTES"*, estas dos últimas tablas deberán ser llenadas por la persona encargada luego de cada reunión de la Subcomisión Académica y en base a las decisiones que ahí se hayan tomado, la integridad de la información ingresada se revisará de acuerdo al tipo de datos de cada campo. Las *tablas 2.10* y *2.11* muestran la estructura de las tablas *"PROYECTOS"* y *"PRACTICAS"* respectivamente, en ambas es necesario un identificador por cada registro que sirva como clave principal (el número único que es la clave principal de la tabla *"ESTUDIANTES"*), la fecha de recepción, última revisión del documento y el estado del mismo. Se almacena todos éstos campos para tener una buena consistencia de la información guardada.

PROYECTOS	***Tipo de dato***	***Longitud máxima***	***Obligatorio***
Identificador	entero	-	Si
Número único	cadena de dígitos	9	Si
Tema del plan	cadena de caracteres	50	Si
Nombre del director	cadena de caracteres	30	Si
Fecha de recepción	fecha	-	No
Fecha de revisión	fecha	-	No
Estado	cadena de caracteres	1000	Si

tabla 2.10: Descripción de los campos de la tabla PROYECTOS

PRACTICAS	***Tipo de dato***	***Longitud máxima***	***Obligatorio***
Identificador	entero	-	Si
Número único	cadena de dígitos	9	Si
Nombre de la empresa	cadena de caracteres	30	Si
Número de horas	entero	-	Si
Fecha de recepción	fecha	-	No
Fecha de revisión	fecha	-	No
Estado	cadena de caracteres	1000	Si

tabla 2.11: Descripción de los campos de la tabla PRÁCTICAS

Cada una de estas tablas contienen información precisa para el funcionamiento del IVR, pues toda la información correspondiente a los estudiantes se almacena en el SAE, por tal motivo todos los campos serán obligatorios a excepción de las fechas de recepción y revisión. En la *figura 2.2* se muestra la estructura de la base de datos para manejar servicios adicionales.

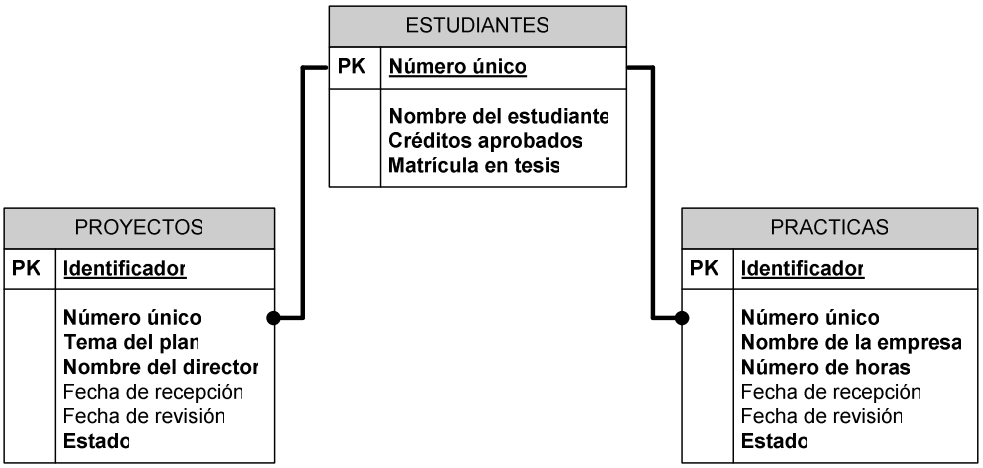

figura 2.2: Estructura de la base de datos para manejar servicios adicionales

2.4.2 COMUNICACIÓN CON LA BASE DE DATOS DEL SAE DE LA CARRERA

La interacción con el SAE de la carrera de Electrónica y Redes de Información es estricta por parte de la UGI y solamente se debe consultar la información necesaria para el funcionamiento del IVR. Los procedimientos almacenados a crearse pueden ser usados para la consulta de notas o para incorporar la información de estudiantes en la base de datos de servicios adicionales.

Un estudiante podría equivocarse al pronunciar o digitar su número único, si corresponde a un número único inexistente el sistema obviamente advertirá de este hecho y solicitará nuevamente el ingreso del mismo, pero también puede ocurrir que el número único errado coincida con el de otro estudiante, en este caso el sistema continuará y la información que se provea al usuario no será la que le corresponda al que está utilizando el sistema. En el caso del IVR con reconocimiento de voz se pedirá una confirmación al usuario repitiéndole el número único que ingresó[4], de ser afirmativa la respuesta se continúa al siguiente

[4] El nivel de certeza del motor de reconocimiento al detectar un token correcto mediante voz no es comparable si se detecta mediante tonos DTMF, para el segundo caso los motores suelen aplicar un nivel de certeza del 100%.

paso, así el estudiante y el sistema tendrán la certeza de que están interactuando correctamente. Para el caso de la detección de tonos DTMF es innecesaria esta confirmación por cuanto el sistema detecta exactamente lo que el usuario digitó, pero ante la posible equivocación al digitar el número único se puede dar la bienvenida pronunciando el nombre del usuario, así estará consciente que su interacción con el sistema es correcta; necesitamos entonces crear el primer procedimiento almacenado.

Para este proyecto la UGI ha permitido la consulta de solamente ciertos campos de las tablas *"ESTUDIANTES"* y *"CALIFICACIONES"*[5] del SAE de la carrera de Electrónica y Redes de Información. La primera obviamente contiene un registro por cada estudiante y lleva información correspondiente a todo su trayecto desde que ingresó a la universidad, como por ejemplo su nombre, su número único, el número de créditos aprobados, su dirección, teléfono, e incluso su fecha de graduación si fuese el caso, etc. La clave principal de esta tabla es el número único y será el parámetro de entrada para el primer procedimiento almacenado. La tabla *"calificaciones"*, en cambio, tiene tantos registros por estudiante según el número de materias en las que esté matriculado.

El primer procedimiento almacenado a ser llamado durante el diálogo del usuario con el IVR es el que permita consultar el *"nombre"* del estudiante dado un *"número único"* específico. Además, a la consulta se puede agregar el *"sexo"* del estudiante para dar una bienvenida diferenciada que puede ser *"Bienvenido señor"* o *"Bienvenida señorita"* seguido de su nombre.

En la tabla en mención constan tanto estudiantes matriculados como retirados y los ya graduados, la información que provee el IVR es solamente para los primeros por lo que es necesario incluir en la consulta el *"estado"* de matrícula del estudiante, la primera consulta tendrá la estructura siguiente:

---

[5] Las tablas realmente no llevan esos nombres pero durante este capítulo las llamaremos así cuando nos refiramos a ellas.

seleccionar *estado de la matrícula*, *nombre*, *sexo*
de la tabla *estudiantes*
en donde el *número único* coincida con " … "

Los puntos suspensivos al final son reemplazados con el número único de siete o nueve dígitos obtenido mediante el reconocimiento de voz o tonos DTMF. Una vez en el menú principal del IVR el estudiante deberá escoger que servicio va a utilizar, la consulta de notas se la hará en el SAE y los servicios adicionales en la base de datos adicional que se creará.

2.4.2.1 Consulta de Notas

El acceso a la tabla *"calificaciones"* ha sido también restringido a ciertos campos solamente, y entre ellos, los necesarios para el proyecto en cuestión corresponden a registros con el *"código de la materia"*[6] y las tres calificaciones (primera nota, segunda nota y nota del examen supletorio) de un estudiante identificado por su número único. Existen campos adicionales como la suma de sus notas y si el estudiante ha sido exonerado o no pero esta información puede ser calculada en base a las calificaciones, además, tomando en cuenta que se debe minimizar el tráfico en la red por cada consulta a la base de datos, esta información no será consultada. Sin embargo existe un campo que es preciso incluirlo en la consulta y es el que indica el *"numero de calificaciones"* que un estudiante ya tiene registrado, éste ayuda a interpretar hasta que nota se debe proporcionar al usuario al momento de la entrega de la información. A continuación se muestra la estructura de esta consulta, de igual manera los puntos suspensivos corresponden al número único:

seleccionar *código de la materia*, *número de calificaciones,*
primera calificación, *segunda calificación, examen supletorio*
de la tabla *calificaciones*
en donde el *número único* coincida con " … "

[6] El nombre de la materia se encuentra en la tabla *"materias"* a la que la UGI no concedió permiso alguno para este proyecto.

A diferencia de la primera consulta esta última no solo devolverá un registro sino tantos según el número de materias que un estudiante tome en el presente semestre.

2.4.2.2 Servicios Adicionales

Para agregar la información de estudiantes a la base que almacena los planes de proyectos de titulación y certificados de prácticas preprofesionales se necesita consultar al SAE el número único, el nombre del estudiante y el número de créditos aprobado, además, se debe consultar si está o no matriculado en el *"Proyecto de Titulación"*. Los tres primeros campos se consultan en la tabla *"ESTUDIANTES"* y la estructura de este procedimiento almacenado se tiene a continuación:

> seleccionar *número único, nombre*, *créditos aprobados*
> de la tabla *estudiantes*
> en donde los *créditos aprobados* superen los 120

El valor de 120 es menor que el 70% de créditos totales de la carrera, requisito para los estudiantes que deseen realizar prácticas preprofesionales, pero se lo ha dejado así porque pueden existir excepciones. Por cada estudiante se deberá hacer otra consulta para saber si está o no matriculado en el *"Proyecto de Titulación"*. Esto se lo debe hacer en la tabla *"CALIFICACIONES"*, el procedimiento almacenado es el que se muestra a continuación:

> seleccionar *número de registros* de la tabla *calificaciones*
> en donde el *número único* coincida con " … "
> y *materia* coincida con *"Proyecto de Titulación"*

De igual manera los puntos suspensivos identifican el parámetro de entrada del procedimiento almacenado y corresponde al número único de cada estudiante que fue seleccionado según el procedimiento anterior. Esta consulta devuelve *"1"* en caso de haber una coincidencia y sirve solamente para diferenciar si está

matriculado o no en el *“Proyecto de Titulación”*, si la consulta no devuelve ningún valor se considera que el estudiante no está matriculado.

A lo largo de esta sección se ha analizado la interacción con el SAE existente y la creación de la base de datos para servicios adicionales tomando en cuenta los requerimientos del IVR que se va a crear. El diseño de la base de datos y los procedimientos almacenados se plasmará en lenguaje SQL y acorde a lo verdaderos nombres de tablas y campos en el siguiente capítulo, en donde se explica la implementación del sistema.

2.5 INTEGRACIÓN DE TODO EL SISTEMA

Hasta ahora se ha estudiado cada uno de los componentes del IVR para consulta de notas y servicios adicionales de una manera aislada y sin mencionar que aplicaciones se van a usar y de que manera se van a integrar, así se ha logrado un análisis general que no esté ligado a los alcances y limitaciones de aplicaciones particulares. En esta sección se explicará a breves rasgos qué componentes se necesitan y como estará estructurado el sistema.

Hay que recordar que un IVR es un servicio de atención automática que suele incorporarse en los sistemas de telefonía, por eso es muy común que exista manera de crear uno en las PBX que se venden hoy en día, las que generalmente son cajas cerradas con hardware especializado para manejar telefonía tradicional y actualmente telefonía IP, que pueden o no incorporar otras funciones, las cuales en su mayoría tienen altos costos. Con esto no se descarta que para un IVR como el de éste proyecto pueda existir hardware o software que esté especializado solamente en la creación de IVRs y que no maneje toda la funcionalidad de una PBX, de todos modos el punto de partida de la aplicación de este proyecto requiere de una PBX que recepte las llamadas entrantes de los estudiantes y que pase el control al IVR correspondiente para que ejecute el diálogo guiado y devuelva la información solicitada.

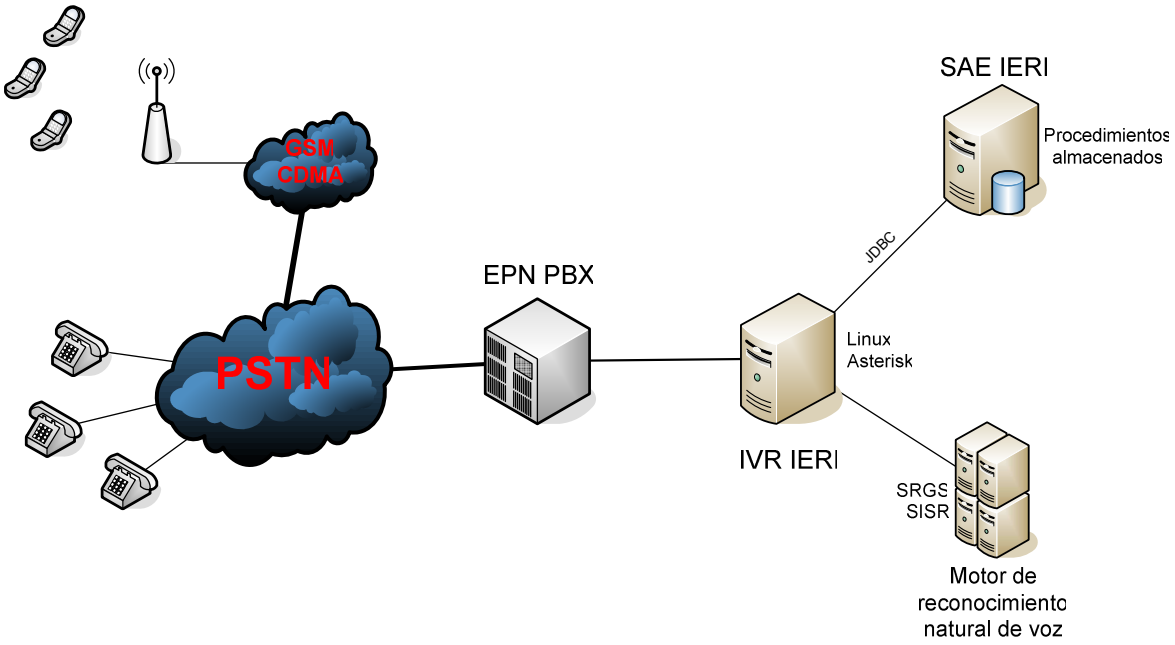

figura 2.3: Componentes del IVR

En la *figura 2.3* se indica los tres componentes del IVR que se va a crear, el software de código abierto Asterisk, que es una PBX de gran capacidad que incluye algunas funciones entre las que está la creación de IVRs, el motor de reconocimiento natural de voz y la base de datos en donde se encuentra la información de estudiantes, calificaciones, temas de proyectos de titulación y prácticas preprofesionales.

Se ha elegido Asterisk por su flexibilidad al momento de desarrollar aplicaciones telefónicas para distintos propósitos y sobre todo porque es software de código abierto que funciona principalmente sobre sistemas operativos Linux, que en conjunto soporta una gran variedad de plataformas de hardware. Como se mencionó anteriormente, la PBX es el punto en donde se inicia el IVR, pues recibe las llamadas que llegan desde la red de telefonía tradicional (la que también se interconecta con operadoras de telefonía celular) y ejecuta cada uno de los pasos que tendrá el sistema de atención automática. Entre los pasos o acciones del IVR se encuentra el reconocimiento de voz para extraer la información de la persona que llama, para esto se necesita el motor de reconocimiento natural de voz que debe soportar las recomendaciones *SRGS* y *SISR*, para que desde la PBX se le indique el audio, los archivos de gramática y la

interpretación semántica para que ejecute la detección del más probable de los tokens que el usuario haya pronunciado. Se ha indicado el motor de reconocimiento como un componente aparte porque puede o no ser incorporado en una PBX, pues su funcionalidad está especializada para sistemas como el que se plantea en este proyecto y no necesariamente es parte de un sistema básico de telefonía.

Otro componente es la base de datos que guarda la información, la misma que ya está creada y que solamente va a ser consultada, es un servidor aparte con el que la PBX deberá comunicarse, aquí juega un papel importante la flexibilidad de la PBX que se utilice y para este caso Asterisk ofrece una puerta de comunicación con cualquier tipo de aplicación informática en un lenguaje de programación determinado, como se indicó en la *sección 1.4.3*, esta interfaz de comunicación se llama *AGI* y el lenguaje que se ha escogido para la implementación es Java porque permite tanto la comunicación con Asterisk como la comunicación con bases de datos SQL Server 7. No ha sido la intención adelantarse a la implementación del sistema, tema que será tratado con detalle en el siguiente capítulo, sino que se ha visto oportuno en este punto dar una idea del sistema que se está construyendo.

Es momento de regresar a analizar la *figura 2.1* en donde se indicó como primera instancia el flujo de opciones del IVR, e incorporar la funcionalidad de todos los componentes que formarán el sistema, tanto las consultas a la base de datos como las llamadas a las funciones del motor de reconocimiento natural de voz.

Aunque el IVR podrá reconocer la voz del llamante o detectar tonos DTMF es mejor entender el funcionamiento del mismo como si fuesen dos sistemas que cumplen el mismo objetivo pero que tienen limitaciones distintas. Aunque ambos ejecutan las mismas consultas a la base de datos, ambos no necesitan de un motor de reconocimiento.

2.5.1 IVR CON RECONOCIMIENTO NATURAL DE VOZ

En páginas anteriores se hizo mención de la diferencia entre un sistema que soporte reconocimiento de voz y tonos DTMF con uno que soporte solo tonos DTMF, en esta sección se va a explicar de mejor manera este punto.

En la *figura 2.4* se indica el diagrama de flujo de diálogo del *IVR* con reconocimiento de voz y *"en ciertos casos"* tonos DTMF. En cada bloque que se necesite solamente la detección de voz se indica mediante *ASR*[7] y es solamente en la recepción del número único en donde se habilita también la detección de tonos DTMF, se lo hace así para aprovechar la funcionalidad del motor de reconocimiento y no volver repetitivo el diálogo, lo que se evita con esto son mensajes del tipo *"Si desea consultar notas diga notas o digite uno, para estados de planes de proyectos de titulación diga proyectos o digite dos, para certificados de prácticas preprofesionales diga prácticas o digite tres"*. Al usar solamente reconocimiento de voz se logra que el diálogo sea fluido y el usuario llegue rápidamente a la información que necesita, pero en el caso del número único al existir un gran número de posibilidades, la detección de tonos DTMF puede dar mejores resultados.

En este *IVR* la detección de tonos DTMF la hace el motor de reconocimiento, es decir, que deberá tener activos tanto archivos de gramática de voz (*mode="voice"*) como de DTMF (*mode="dtmf"*).

Se indica también cada consulta a la base de datos del SAE y también a la que se ha denominado *"PRAPRO"* que es la que almacena la información de planes de proyectos de titulación y certificados de prácticas preprofesionales. Algo que no se indica en el diagrama es el estado en el que quedará el diálogo el momento que la base de datos no esté disponible, en este caso la llamada deberá finalizar previo un mensaje de indicación al usuario de dicho suceso.

---

[7] Del inglés Automatic Speech Recognition, que significa reconocimiento automático del habla

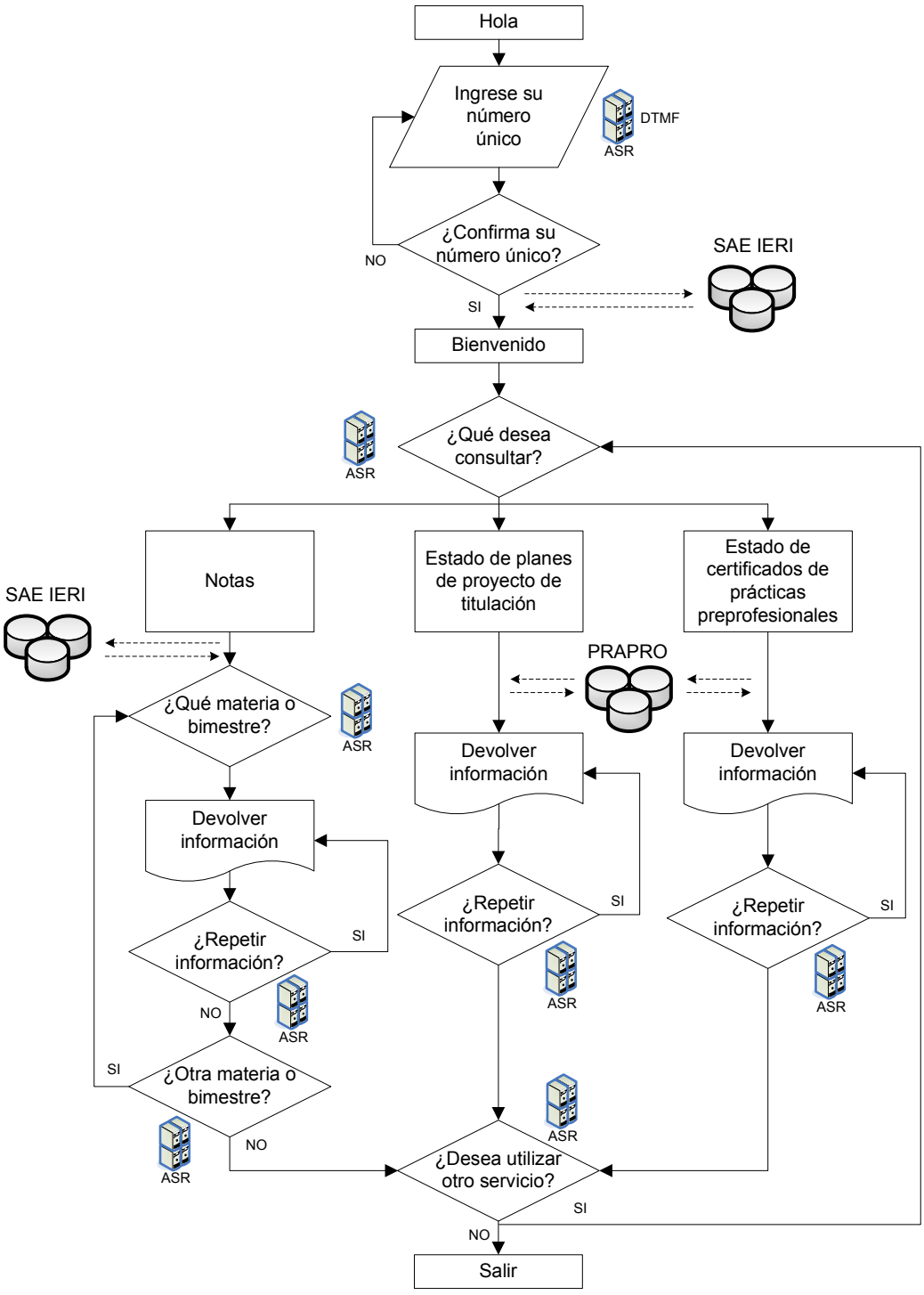

figura 2.4: IVR con reconocimiento natural de voz

Respecto al diagrama inicial de la *figura 2.1* se ha agregado también el bloque de decisión que pregunta al usuario si desea repetir la información que se le ha proporcionado y también la bienvenida después de hacer la primera consulta a la

base de datos para preguntar el nombre, sexo y estado de la matrícula del estudiante cuyo número único coincida con el que se detectó previamente.

2.5.2 IVR CON RECONOCIMIENTO DE TONOS DTMF

A diferencia del reconocimiento natural de voz en un sistema de atención automática, el usar solamente el teclado de un teléfono limita la comunicación y por ende la consecución de objetivos pero es de gran validez cuando se tienen líneas con mucho ruido o cuando las personas que lo usan tienen dificultades para comunicarse oralmente o simplemente no pueden hacerlo. La detección de tonos DTMF es una función primordial en una PBX pues es la manera de decirle a la central el destino de la comunicación que se desee realizar.

Un IVR que detecte solamente las entradas del teclado de un teléfono puede tener resultados aceptables si el diálogo con el usuario no es extenso, es decir, cuando no se trabaje con demasiada información. Configurar un *IVR* como el planteado en la *figura 2.4* pero solamente con detección de tonos DTMF puede tener una estructura bastante similar, y en este proyecto se necesita que sea así para facilitar el uso por parte del usuario que quizá estuvo acostumbrado al reconocimiento de voz y por alguna razón se le debe obligar a usar solamente el teclado de su teléfono.

La *figura 2.5* muestra el *IVR* con detección de tonos DTMF y lo que se ha suprimido es la confirmación del número único luego de haberlo detectado, pues como se mencionó en páginas atrás, no es necesario porque la detección de tonos DTMF tiene un nivel de certeza muy alto y se lo considera del 100%.

Aunque al final se ha suprimido el bloque *“salir”* se debe indicar que en ambos casos este paso es opcional, pues se puede dar la alternativa al usuario de salir del sistema siendo éste último el que concluya la llamada aunque lo más natural es que se espere que el usuario cuelgue el teléfono. De todos modos se lo ha suprimido en la *figura 2.5* porque el audio que indica las opciones disponibles en

determinado instante debe ser lo más corto posible y se logra esto suprimiendo la opción que le permite salir del sistema.

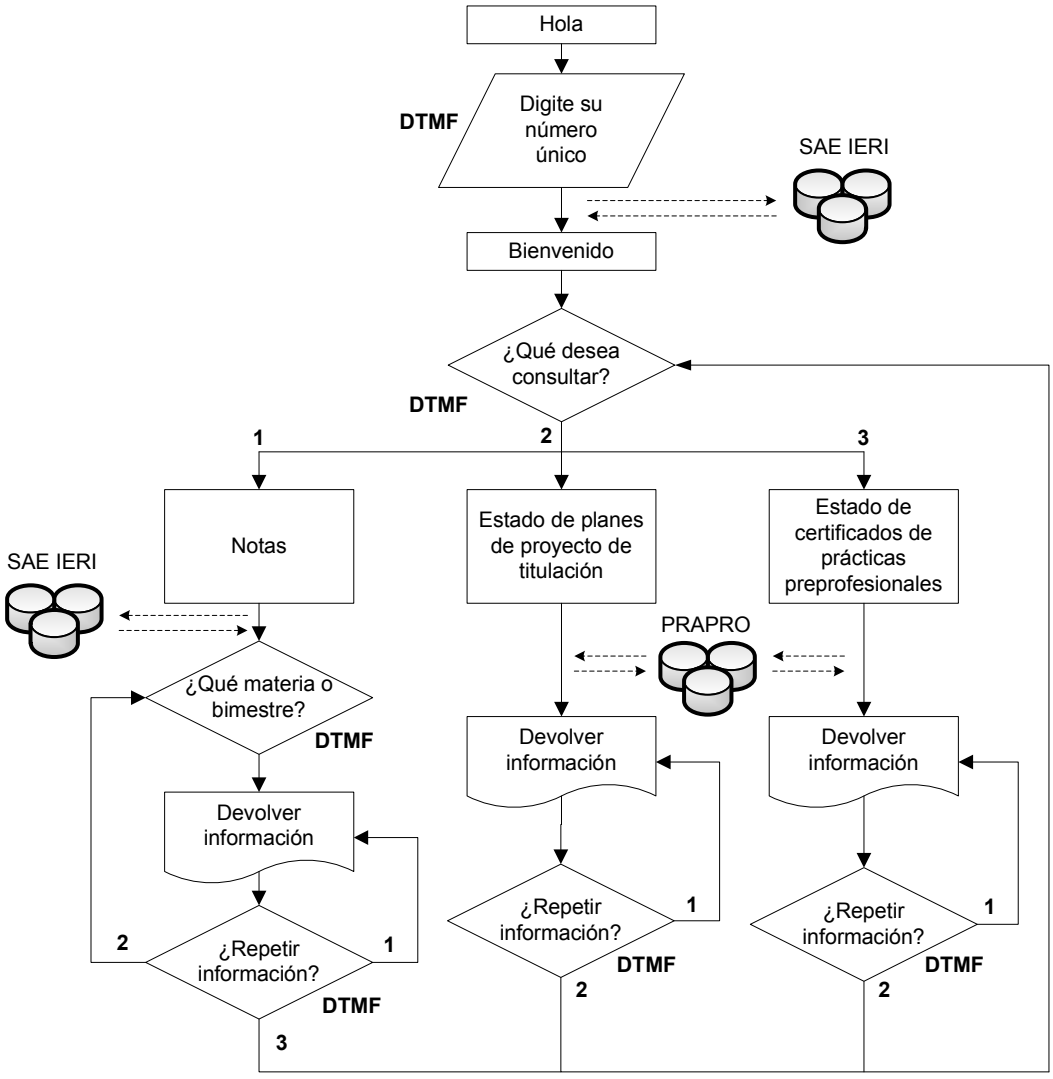

figura 2.5: IVR con detección de tonos DTMF

Si bien la detección de tonos DTMF está presente en la recolección del número único en ambos IVRs, se debe insistir en que el agente que está escuchando el canal para hacer determinada detección no es el mismo. Para el primer caso quien hace la detección es el motor de reconocimiento y una vez detectada la secuencia de dígitos pasa el control al IVR, pero en el segundo caso lo hace enteramente el IVR, el mismo que solamente detiene la ejecución de su diálogo cuando hace consultas a la base de datos.

Capítulo III

3 IMPLEMENTACIÓN Y PRUEBAS DEL SISTEMA

3.1 REQUERIMIENTOS DE SOFTWARE

Es momento de plasmar todo lo estudiado y analizado respecto a cada uno de los componentes del IVR de este proyecto (*figura 2.3*) en un sistema real, que incorpore las funcionalidades planteadas tomando en cuenta todas las recomendaciones que se ha indicado, equilibrando la disminución de costos con la calidad de servicio que se brinde al usuario.

La elección del software a utilizar es uno de los puntos importantes para lograr un sistema eficiente y estable, pero por otra parte juega un papel predominante en el costo final del mismo. La estrategia que se va generalizando a nivel mundial tiende a usar software de código abierto, que por su naturaleza puede tener costos sumamente bajos hasta el punto de distribuirse gratuitamente y que sobretodo está disponible a miles de desarrolladores a nivel mundial que buscan siempre robustecer su programación y crear sistemas lo más estables posible; éste ha sido el éxito de todas las comunidades de software libre que cada año ganan mercado y van estableciendo una cultura de igualdad, humanismo y difusión del conocimiento para todas las regiones del planeta. Este avance nace con la invención del Internet y en el proceso de adaptación de países poco desarrollados como el nuestro surgen problemas, principalmente en aplicaciones en donde el idioma juega un papel importante, tal es el caso de los motores de reconocimiento natural de voz, que muestran excelentes resultados cuando detectan el idioma Inglés, pero que su desarrollo en otros idiomas se ve limitado por la poca colaboración de comunidades de habla hispana.

La elección del software para este proyecto además de equilibrar la calidad del servicio con la disminución de costos debe considerar un punto adicional, que cada una de sus partes facilite su integración en un sistema robusto y confiable, pues existen situaciones en donde los estándares aún no se han establecido y la integración solamente sucede si cada parte lo permite. En esta sección se analizará cada uno de los paquetes de software que formarán el IVR.

3.1.1 SISTEMA OPERATIVO

El sistema operativo es la base sobre la cual se ejecutan todas las aplicaciones y en gran parte será responsable de la estabilidad que tenga el IVR. Para este proyecto se ha elegido el sistema operativo GNU/Linux, por su robustez, disponibilidad, estabilidad y el desarrollo constante en el cual se centra una de las comunidades más grandes de software libre a nivel mundial, lo cual facilita su actualización y la incorporación de nuevas funcionalidades.

Entre la diversidad de distribuciones de GNU/Linux se trabajará con Fedora[1], específicamente la versión 6, porque es uno de los proyectos de la compañía líder en la venta de sistemas GNU/Linux a nivel empresarial, Red Hat[2], que además de ser gratuito está orientado al uso en servidores, hecho que coincide perfectamente con el IVR de este proyecto.

3.1.2 SERVIDOR DE TELEFONÍA

Lo que ofrece un IVR es un servicio de telefonía y por ende se necesita del software y/o hardware apropiado para poder configurar este servicio, aunque no necesariamente puede ser así pues para el caso de este proyecto no hace falta el enrutamiento de llamadas o el manejo de codificadores de voz, características que son fundamentales en una PBX, sin embargo, son estos sistemas los que ofrecen funcionalidades de IVR.

Históricamente las PBX han sido vendidas como sistemas cerrados con hardware y software propietario a costos sumamente altos, con capacidades limitadas de telefonía y casi nada modulares. Este hecho cambió con la invención de procesadores de propósito general cada vez más rápidos y a menor costo, que han permitido trasladar la carga total o parcial de las funciones de hardware a

[1] http://fedoraproject.org

[2] http://www.redhat.com

paquetes de software especializados, éste es el caso del Proyecto de Telefonía Zapata[3] y su catalizador, Asterisk.

Para este proyecto se ha elegido Asterisk porque es un paquete de software con grandes capacidades de PBX, que es compatible con la mayoría de arquitecturas de hardware y aunque tiene soporte para diversos sistemas operativos entre los que se incluyen *Windows* y *Solaris*, está primordialmente hecho para GNU/Linux, pues su desarrollo es siempre mayor. Es distribuido como software de código abierto y ha ganado una increíble popularidad en todo el mundo porque se lo ha utilizado a nivel empresarial y mantiene en producción grandes sistemas de telefonía, además que puede ser descargado gratuitamente desde su sitio Web[4].

Su naturaleza modular y de código abierto ha facilitado que desarrolladores de software a nivel mundial creen paquetes que agregan funcionalidades adicionales a Asterisk, entre las cuales se destaca un *gateway* de comunicación con otras aplicaciones informáticas en diversos lenguajes de programación, este hecho facilita crear sistemas integrados como el que se trata en este proyecto.

3.1.3 MOTOR DE RECONOCIMIENTO NATURAL DE VOZ

El reconocimiento natural de voz es una tecnología que si bien no es nueva ha sido en los últimos años en los que ha alcanzado un desarrollo sustentable por la tendencia a utilizar sistemas automatizados en los que la interacción hombre máquina sea cada vez más natural para nosotros. Lamentablemente es en este tipo de aplicaciones en donde juega un papel importante el lenguaje en el cual se desarrollan las mismas, hecho que limita la capacidad de adaptarlas a nuestras necesidades.

Entre los motores de reconocimiento natural de voz que manejen el idioma Español ofreciendo buenos resultados se encuentran dos, el de la compañía

[3] http://www.zapatatelephony.org

[4] http://www.asterisk.org/downloads

Nuance[5] y el de *Lumenvox*[6]. Además de no ser software de código abierto ambos soportan sistemas operativos *Windows* y Linux y venden su producto a manera de puertos de comunicación, pero solamente el segundo ha afianzado su estrategia de poder agregarse a un sistema basado en Asterisk, proporcionándole un módulo adicional que incorpora aplicaciones para llamar directamente a las funciones de reconocimiento, hecho que motiva a utilizar el motor de Lumenvox para el IVR de éste proyecto.

3.1.4 SOFTWARE PARA CONVERSIÓN DE TEXTO A VOZ

Durante el desarrollo de este proyecto se ha mencionado a breves rasgos la necesidad de un software para conversión de texto a voz para situaciones en donde tener archivos de audio pregrabados sea poco eficiente. La elección del software adecuado presenta el mismo problema que el motor de reconocimiento natural de voz; el lenguaje Español no está completamente soportado.

Existe un paquete que viene incorporado en algunas distribuciones de GNU/Linux (entre ellas Fedora) para conversión de texto a voz llamado *Festival*[7], el cual da soporte al idioma Español pero la voz que genera los archivos de audio es demasiado robotizada y en ocasiones se vuelve poco entendible, hecho que destaca la falta de desarrollo en este lenguaje y obliga a buscar otra alternativa para este proyecto.

Entre el software propietario para reconocimiento natural de voz se ha elegido *Cepstral* porque incorpora un gran soporte para el idioma Español y sobretodo porque tiene la capacidad de generar archivos de audio directamente para aplicaciones de telefonía, es decir muestreados a 8 KHz. Al igual que los motores de reconocimiento natural de voz una limitante en la adquisición de software para conversión de texto a voz es que sus licencias se venden a manera de puertos, es

---

[5] http://www.nuance.com

[6] http://www.lumenvox.com

[7] http://www.cstr.ed.ac.uk/projects/festival

decir, un número máximo de canales que en determinado instante se mantengan activos.

3.1.5 CONECTOR CON SQL SERVER 7

Para poder leer la información del SAE es necesario un conector a dicha base de datos que proporcione una interfaz a Linux (específicamente Asterisk) y en la interacción de aplicaciones de dos sistemas operativos quizá la mejor solución es el lenguaje de programación Java, porque es continuamente desarrollado y ha logrado una excelente aceptación a nivel mundial, además que Asterisk lo soporta a través del módulo *asterisk-java*[8].

Un inconveniente con SQL Server 7 es que es un sistema de administración de bases de datos que ha dejado de ser soportado por su creador *Microsoft*, un punto que corrobora esto es que su propio conector JDBC denominado *sqljdbc*[9] desde su primera versión solamente soporta SQL Server 2000 y 2005. Afortunadamente existe el conector *jTDS*[10] que soporta las versiones 6.5, 7, 2000 y 2005 de SQL Server, lo mejor de todo es de código abierto y se lo puede utilizar libremente, razones suficientes para ser utilizado en este IVR.

3.2 REQUERIMIENTOS DE HARDWARE

Elegir el hardware adecuado para un sistema en producción es otro de los puntos fundamentales para obtener eficiencia y estabilidad. El rendimiento del software será óptimo cuando tenga a su disposición excelentes recursos de hardware entre los que predominan velocidad de procesamiento, cantidad de memoria y almacenamiento en disco. Dimensionar un servidor en base a la carga del software utilizado es un tema bastante complicado pero se pueden seguir algunas

---

[8] http://asterisk-java.org

[9] http://www.microsoft.com/sql/technologies/jdbc/default.mspx

[10] http://jtds.sourceforge.net

pautas. Los productores de software propietario suelen proporcionar requerimientos de hardware para su aplicación bajo determinadas condiciones de carga (lo que no sucede siempre con proyectos de software libre), pero generalmente son los mínimos para que ésta funcione, sin embargo, estos requerimientos pueden darnos una idea de que tantos recursos se consumirán en nuestro sistema. Por otra parte un servidor no solamente está dimensionado en base al software sino también a requerimientos físicos que dependen de la naturaleza del servicio a utilizar.

El servidor o servidores que albergarán el IVR de este proyecto serán dimensionados en base a la cantidad de líneas telefónicas que llegarán al mismo, lo que se traduce en el número máximo de usuarios simultáneos que se podrá tener, a partir de esta información se podrá determinar una carga estimada de los paquetes de software que se instalarán.

3.2.1 EN BASE A REQUERIMIENTOS FÍSICOS

Al decir requerimientos físicos se trata exactamente del número de puertos FXO que la central deberá tener para conectarse a la PSTN y servir de manera adecuada a todos los usuarios. Existen tarjetas para este propósito que son fabricadas específicamente para el funcionamiento con Asterisk en Linux, y su primer y principal fabricante es *Digium*[11], la compañía que está detrás del desarrollo de Asterisk, aunque con el pasar del tiempo otras compañías también han lanzado sus propias versiones de estas tarjetas. De todos modos existen modelos de cuatro, ocho y veinticuatro puertos FXO (que también pueden ser FXS, o combinados). La de cuatro puertos es la más usada en sistemas pequeños en donde no se requiera manejar gran cantidad de llamadas salientes o entrantes (porque además se debe pagar el costo de las líneas a la operadora de telefonía) y no necesariamente se deben usar los cuatro puertos, pues la estructura de la tarjeta se compone de una placa base en donde se agregan los módulos FXO, es decir se puede crecer en razón de un módulo a la vez aunque

[11] http://www.digium.com

suelen existir precios promocionales por comprar la tarjeta con todos sus módulos.

Es común que existan confusiones al momento de dimensionar la cantidad de líneas entrantes a un IVR, aunque su naturaleza se asemeja a la de un C*all Center* no es correcto trabajar con la fórmula de *Erlang C* debido a que no se manejan colas de espera. Un IVR es un sistema en donde cualquier cliente que ingrese deberá ser atendido, hecho que tiene limitaciones en redes de conmutación de circuitos como una PSTN pero que en redes de conmutación de paquetes, como un ambiente de telefonía IP, es completamente posible.

Una aproximación al dimensionamiento de un IVR es utilizar la fórmula de *Erlang B*, porque determina el número de líneas externas que se necesitarán para dar servicio a un determinado número de clientes (cantidad de tráfico) con un valor mínimo de pérdidas establecido. El número de líneas externas se traduce en la cantidad de puertos FXO de la tarjeta que estén activos (que tengan conectada una línea telefónica), el valor mínimo de pérdidas se lo conoce como *Grado de Servicio* (GoS) y se prevé conveniente para este proyecto un valor del 1% (una de cada cien llamadas no será atendida). El tráfico dependerá de la cantidad de estudiantes que consulten sus notas en determinado intervalo de tiempo, el que suele considerarse el más congestionado, y el tiempo promedio del servicio. El tráfico se mide en *Erlangs* y suele ser un parámetro conocido en el diseño de un *Call Center* pero para el caso del este IVR no se conoce con certeza la afluencia de usuarios a utilizar el servicio porque obviamente no ha sido implementado antes, pero según la apreciación de las personas que permanecen en la oficina de la Coordinación de la Carrera de Electrónica y Redes de Información son aproximadamente veinte estudiantes por hora los que acuden a consultar sus notas en los días posteriores a la culminación de los exámenes, periodo que es el de más afluencia.

Con la información anterior podemos calcular el tráfico de llamadas en la hora más congestionada (BHT, *Bussy Hour Traffic*), esto se lo hace con la fórmula siguiente:

$$BHT = \frac{n * t_m}{T} \qquad \text{(ecuación 3.1)}$$

En donde *n* es el número de llamadas entrantes por hora, t_m el tiempo medio de llamada, que resultado de las pruebas realizadas (*sección 3.9*) es aproximadamente 80 segundos, y *T* el intervalo de tiempo que estamos considerando, para este caso una hora. Entonces tenemos:

$$BHT = \frac{20 * 80}{3600} \; Erlangs$$

$$BHT = 0{,}44 \; Erlangs$$

Mediante una *Tabla de Erlang B* (*Anexo I*) y en base a un *GoS* del 1% se determina que son necesarias tres líneas telefónicas para poder atender a los estudiantes, número que hasta septiembre del año 2007 no sobrepasaba los 542 alumnos. Por el crecimiento uniforme de la población estudiantil de la Carrera de Electrónica y Redes de Información en los últimos tres años[12] se opta por tener cuatro líneas telefónicas, es decir, una tarjeta *Digium TDM400P* con cuatro módulos FXO.

3.2.2 EN BASE A LA CARGA DE SOFTWARE

Hasta la actualidad no existen estudios precisos para dimensionar un servidor de telefonía con Asterisk, y menos para un IVR como el que se trata en este proyecto. Lo que existe son referencias de casos implementados bajo ciertas condiciones (número de extensiones, líneas salientes, codificación de voz, compresión, etc.) que la comunidad de usuarios a nivel mundial ha instalado en determinados servidores y les ha funcionado sin ningún problema (o han detallado

[12] Según la Guía Académica para las Carreras de Pregrado de la Facultad de Ingeniería Eléctrica y Electrónica, periodo septiembre 2007 – febrero 2008, pág. 13, el crecimiento promedio ha sido 28 estudiantes por semestre.

los problemas que se les ha presentado)[13], y que sirven como recomendaciones de dimensionamiento. El problema es complejo porque la naturaleza de cada servidor es distinta dependiendo del ambiente en el cual funcione y los servicios que preste. De todos modos entre los parámetros que se toman en cuenta están el número de usuarios a atender, la codificación de voz que se utilice, si se hace o no conversión entre codecs de voz (*transcoding*), compresión de voz y si la cancelación de eco para las interfaces analógicas se hace por software o hardware.

Lamentablemente la naturaleza del IVR de este proyecto es distinta porque no se hace telefonía IP, sino que recibe las llamadas y se las procesa en el servidor, no existen redirecciones y tampoco conversión entre codecs de voz, pero si existen otros procesos adicionales que consumen recursos del sistema y que hay que tener muy en cuenta, los más importantes son el reconocimiento natural de voz y la conversión de texto a voz.

Un estudio un tanto avanzado de los sistemas de telefonía basados en Asterisk se da en el libro *"Asterisk, the Future of Telephony"* y de ahí se basa la afirmación que para un sistema que maneje cuatro usuarios es suficiente contar con un procesador de 400 MHz y 256 MB de memoria RAM[14].

El motor de reconocimiento natural de voz *Lumenvox* ofrece datos un tanto específicos dependiendo del número de canales que se tengan[15], en base a pruebas que han realizado los propios desarrolladores. Para el caso de 4 puertos de reconocimiento se necesitaría de 437 MB de memoria y un 6% de utilización de un procesador que según se indica es Intel de doble núcleo.

---

[13] http://www.voip-info.org/wiki-Asterisk+dimensioning

[14] "Asterisk, the Future of Telephony", pág. 10

[15] http://help.lumenvox.com/robo/projects/speechengine/FAQs/Hardware_Questions.htm, aunque las pruebas han sido desarrolladas bajo sistemas operativos Windows se considera válido tomarlas como referencias.

El conversor de texto a voz Cepstral también señala sus requerimientos mínimos[16] y aunque son reducidos vale indicarlos, un procesador Intel Pentium II o AMD K6 y 64 MB de RAM.

Sin duda alguna es el motor de reconocimiento de voz el paquete que más recursos consume y esto se explica porque debe tener en memoria tanto los archivos de gramática como los modelos acústicos y del lenguaje.

Tomando en cuenta todo lo anterior, el sistema debe ser lo suficientemente rápido para poder procesar la información de manera que no presente retardos molestosos al usuario, además no se debe dejar de lado la disminución del costo total de implementación de este proyecto; por lo tanto se utilizará un solo servidor en el que funcione tanto el IVR programado sobre Asterisk como el motor de reconocimiento de voz *Lumenvox* y el software de conversión de texto voz *Cepstral*, con las siguientes características:

- Procesador Intel i386 (o AMD simulando i386[17]) de doble núcleo, 1.8 GHz
- Memoria RAM de 1 GB
- Disco duro de 80 GB
- Tarjeta de red 10/100 Mbps
- Unidad de CD-ROM o DVD-ROM (para la instalación solamente)

El almacenamiento en disco no es primordial en el IVR porque no hay información que se guarda continuamente más que los mensajes del sistema y de las aplicaciones. La alimentación de energía para el servidor juega un papel importante porque es un sistema que debe trabajar las veinticuatro horas del día durante los siete días de la semana[18], por lo tanto, es imprescindible contar con un sistema de regulación y respaldo de energía eléctrica (UPS).

---

[16] http://www.cepstral.com/cgi-bin/support?page=faq&type=i386-linux#requirements

[17] Al momento de escribir este documento, el motor de reconocimiento *Lumenvox* no soporta plataformas de 64 bits.

[18] Para servicios en donde la alta disponibilidad juega un papel sumamente crítico es recomendable utilizar servidores adicionales que le ofrezcan redundancia al sistema.

3.3 INSTALACIÓN Y CONFIGURACIÓN DE SOFTWARE EN EL SERVIDOR GNU/LINUX

Antes de instalar los paquetes en el servidor hay que hacer una inspección para que todas las piezas de hardware funcionen correctamente, lo recomendable es ingresar en el BIOS (*Basic Input Output System*) de la tarjeta madre y revisar la configuración de cada dispositivo, deshabilitando los que no se utilicen, por ejemplo, tarjetas de sonido o módems.

Para la instalación del sistema operativo la opción más usada es hacerlo desde los CDs o DVDs de la distribución, por lo tanto debemos configurar que este dispositivo sea el primero en arrancar, pero una vez finalizada la instalación debemos arrancar desde el disco duro, esto optimiza el tiempo de encendido del servidor. Es también recomendable agregar una contraseña al BIOS para que usuarios no autorizados no puedan ingresar al sistema y dañar la configuración, además que podrían iniciar desde otro dispositivo y comprometer la integridad del sistema.

3.3.1 SISTEMA OPERATIVO FEDORA CORE

Se ha mencionado anteriormente que la distribución del sistema operativo GNU/Linux a instalar en el servidor es *Fedora*, específicamente la versión 6.

3.3.1.1 Configuración de los recursos

Una recomendación general sobre el tamaño de la partición en disco duro usada como memoria virtual (*swap*) señala que debe ser del doble de la memoria RAM y no siempre es así, depende enteramente de las aplicaciones que se usarán y para el funcionamiento del IVR de este proyecto no se necesitará más que reservar un espacio de 1 GB, pero aprovechando la gran capacidad del disco duro se seguirá la recomendación y se la dejará de 2 GB. Se creará una partición adicional de 100 MB para *"/boot"* y el resto del disco para *"/"*, ambas con sistema de archivos *ext3*.

El sistema está destinado para funcionar en un ambiente LAN que tiene conexión con la PSTN a través de líneas analógicas, por este motivo no es necesaria la instalación de un *firewall* en el servidor. Tampoco es necesaria la instalación del sistema *SE Linux* (*Security Enhanced Linux*). El gestor de arranque para *Fedora* es *Grub* y es conveniente configurar el tiempo de arranque del sistema operativo por defecto (en este caso solamente *Fedora*) en un valor bajo, será de 5 segundos para este servidor. Los puntos anteriores se han indicado debido a que son pasos necesarios en el proceso de instalación con ambiente gráfico de la distribución del sistema operativo.

3.3.1.2 Paquetes de instalación

Como la mayor parte de las distribuciones, Fedora ofrece algunas configuraciones predeterminadas de instalación y la denominada *"instalación mínima"* es suficiente para poner en funcionamiento el IVR de este proyecto, pues no se necesita un ambiente gráfico, sin embargo se puede instalar otros paquetes si en algún momento se necesitaría proveer de un servicio adicional. Los paquetes básicos para el funcionamiento de *Asterisk* y los demás aplicativos son:

- kernel-devel
- ncurses, ncurses-devel
- openssl y openssl-devel
- zlib y zlib-devel
- bison y bison-devel
- gcc
- libnewt y libnewt-devel

Problemas de incompatibilidad suelen darse con algunas versiones anteriores de la distribución, pero Fedora 6 es totalmente compatible con Asterisk y los demás aplicativos. Si se instalan paquetes adicionales a los indicados para el funcionamiento del IVR es siempre recomendable verificar que no tengan levantado ningún servicio a menos que se esté utilizando, pues consumen recursos del sistema innecesariamente.

3.3.2 JAVA DEVELOPMENT KIT

El ambiente de desarrollo Java es necesario para poder interactuar con la base de datos SQL Server 7 del SAE, y para instalarlo existen dos alternativas, la primera es instalar el paquete que viene con la distribución de GNU/Linux, y la segunda, descargarlo directamente de la Web[19]. El problema con la primera opción es que no se suele incorporar versiones actualizadas y preferiblemente se adoptará la segunda. La versión que se instalará es la 1.6, conocida como Java 6.

El archivo de instalación del JDK para sistemas GNU/Linux i386 es *jdk-6-linux-i586.bin* y se le debe asignar permisos de ejecución. Una vez ubicados en el directorio en donde se encuentra el archivo (el que para este proyecto será /usr/src) se debe digitar:

```
# chmod u+x jdk-6-linux-i586.bin
# ./jdk-6-linux-i586.bin
```

Se debe aceptar el acuerdo de licencia que aparece a continuación para proseguir con la instalación. Una vez finalizada la misma debemos asignar a las variables de entorno del sistema operativo los valores específicos del directorio en donde el JDK ha sido instalado. Esto se lo hace en el archivo /etc/profile y lo que se agrega es:

```
PATH=$PATH:/usr/src/jdk1.6.0/bin
CLASSPATH=/usr/src/jdk1.6.0:/usr/local/agi-java
JAVA_HOME=/usr/src/jdk1.6.0
export PATH CLASSPATH JAVA_HOME
```

Para probar que la instalación del JDK ha sido correcta se pueden ejecutar los siguientes comandos adicionales:

```
# source /etc/profile
# java –version
```

[19] http://java.sun.com/javase/downloads/index.jsp

El primer comando hace que los cambios en el archivo /etc/profile se hagan efectivos y la salida del segundo comando debería ser similar a la siguiente si la instalación ha sido exitosa:

```
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, sharing)
```

3.3.3 ASTERISK, OPEN SOURCE PBX

Para instalar Asterisk se debe necesariamente compilar su código fuente y crear los módulos de la aplicación para que puedan ser llamados desde el sistema operativo. Todo el proceso siguiente debe se debe hacer como usuario *root*. Los paquetes pueden ser descargados desde el sitio Web de Asterisk[20] y los necesarios para este proyecto son dos, los que se indican a continuación junto con las versiones que se utilizarán:

- zaptel-1.2.19.tar.gz
- asterisk-1.2.22.tar.gz

El primer paquete contiene los *drivers* de la tarjeta *Digium TDM400P* que se instalará y el segundo es la aplicación propiamente dicha. Una vez descargados se puede almacenarlos en el directorio /usr/src para proceder a su compilación e instalación.

3.3.3.1 Zaptel

Se debe cambiar al directorio /usr/src para desempaquetar, compilar e instalar el paquete *Zaptel*, los siguientes comandos se digitan en el orden indicado en la consola del sistema operativo:

```
# cd /usr/src
# tar -xvfz zaptel-1.2.19.tar.gz
```

[20] http://www.asterisk.org/downloads

```
# cd zaptel-1.2.19
# make
# make install
# make config
```

El último comando genera los *scripts* para poder iniciar, parar o reiniciar *Zaptel* como un servicio del sistema operativo. El archivo de configuración del paquete *Zaptel* es /etc/zaptel.conf y los parámetros se configuran en base a la tarjeta a utilizar, su número de puertos FXO y FXS. En el *Anexo A.1* se puede encontrar éste archivo con la configuración para la tarjeta *Digium TDM400P* con 4 puertos FXO. Los siguientes comandos se deben ejecutar para completar la configuración:

```
# ztcfg –vvv
# chkconfig zaptel on
```

El primero lee la configuración del archivo /etc/zaptel.conf y el segundo activa el servicio *Zaptel* para que se inicie automáticamente cuando arranca el sistema operativo. La salida del comando "*ztcfg –vvv"* debe indicar claramente que cuatro canales han sido configurados, luego puede digitarse *"modprobe zaptel"* para cargar el módulo en el *kernel* pero en lugar de esto es recomendable reiniciar el sistema.

3.3.3.2 Asterisk

El proceso de instalación de Asterisk es algo similar, en la línea de comandos del sistema operativo se debe ejecutar:

```
# cd /usr/src
# tar -xvfz asterisk-1.2.22.tar.gz
# cd asterisk-1.2.22
# make
# make install
# make config
# chkconfig asterisk on
```

Adicionalmente se puede ejecutar *"make samples"* para crear los archivos de configuración de ejemplo en el directorio /etc/asterisk que vienen en la aplicación, estos suelen comentar cada parámetro y nos dan una profunda visión de todas

las características de Asterisk. En el *Anexo A.2* se pueden encontrar los archivos de configuración de Asterisk que deben ser modificados para crear el IVR de este proyecto.

3.3.4 SPECH ENGINE LUMEVOX

El motor de reconocimiento natural de voz *Lumenvox* contiene dos módulos, un servidor de licencias y el motor propiamente dicho. Cada uno comprende un servicio y pueden ser instalados en distintas máquinas. El primero es el que almacena las licencias que se tengan del producto y atiende las solicitudes del motor cada vez que éste necesite hacer un reconocimiento.

La empresa *Lumenvox* ofrece sus productos como paquetes *RPM* para cada distribución soportada, los que pueden ser descargados si se ha comprado una o más licencias del mismo[21]. La recomendación es descargar las últimas versiones porque contienen actualizaciones en los modelos acústicos y del lenguaje, los que permiten obtener mejores resultados en el reconocimiento. Se instalarán los siguientes paquetes:

- LumenVoxLicenseServer-7.5-502.fc6.i386.rpm
- LumenVoxSRE-7.5-502.fc6.i386.rpm
- asterisk-1.2.x-lumenvox-support-b15-engine7.5.tar.gz

El último paquete es el conector que *Lumenvox* tiene para Asterisk. Se pueden copiar los archivos en el directorio /usr/src para luego ser instalados

3.3.4.1 Servidor de Licencias

Ubicándose en el directorio en donde se encuentran los paquetes de instalación se debe digitar:

[21] http://www.lumenvox.com/customers

```
# rpm -i LumenVoxLicenseServer-7.5-502.fc6.i386.rpm
```

La instalación es interactiva, luego de aceptar los términos de licencia se debe aceptar también el directorio de instalación por defecto (/opt/lumenvox/licenseserver_7.5/). Cada licencia adquirida debe ser instalada en el servidor y el método es el siguiente:

```
# cd /opt/lumenvox/licenseserver_7.5/bin
# ./getlvsystem_info
```

En el mismo directorio se crea un archivo llamado *"Info.bts"*, el mismo que almacena información de las características del servidor como las direcciones de hardware (MAC) de las interfaces de red, memoria, disco duro, procesador, entre otras, esto hace que la licencia solamente pueda ser instalada en dicho servidor[22]. Este archivo debe ser enviado al sitio Web de *Lumenvox* para que generen la correspondiente licencia[23], y en la misma página, una vez subido el archivo *"Info.bts"* se crea un enlace para descargar un nuevo archivo llamado *"LicenceXXX.bts"*, siendo las letras XXX un número variable y que puede ser de tres o más cifras. Este archivo debe ser copiado en el directorio /opt/lumenvox/licenseserver_7.5/bin/ y lo que se ejecuta es lo siguiente:

```
# mv LicenceXXX.bts Licence.bts
# ./license_mgr -h Licence.bts
```

Aparece un mensaje que indica que la licencia ha sido instalada correctamente. Por cada licencia que se desee instalar se debe repetir el mismo procedimiento.

3.3.4.2 Motor de reconocimiento

Para instalar el motor de reconocimiento de *Lumenvox* el procedimiento es sencillo, se deben ejecutar los comandos siguientes:

---

[22] Si alguna de estas piezas de hardware falla y debe ser reemplazada se debe contactar con *Lumenvox* para poder realizar nuevamente el proceso.

[23] http://www.lumenvox.com/customers

```
# cd /usr/src
# rpm -i LumenVoxSRE-7.5-502.fc6.i386.rpm
```

Nuevamente una vez aceptado los términos de licencia se debe aceptar también el directorio de instalación por defecto (/opt/lumenvox/engine_7.5/).

3.3.4.3 Conector con Asterisk

La instalación del conector para Asterisk requiere modificar (parchar) su código fuente y volver a compilarlo, para esto se deben seguir los siguientes pasos:

```
# tar -xvfz asterisk-1.2.x-lumenvox-support-b15-engine7.5.tar.gz
# cd asterisk-1.2.x-lumenvox-support-b15-engine7.5
# cp res_speech_lumenvox.so to /usr/lib/asterisk/modules
# cp lumenvox.conf to /etc/asterisk
# cp speech-1.2.patch /usr/src/asterisk-1.2.22
# cd /usr/src/asterisk-1.2.22
# patch -p0 < speech-1.2.patch
# make
# make install
```

Una vez ejecutados los comandos anteriores se debe modificar los archivos *"modules.conf"* y *"lumenvox.conf"*, ambos dentro de /etc/asterisk/, para que reflejen la configuración que se necesita para el servidor, los archivos se indican en el *Anexo A.3*.

El módulo que *Lumenvox* agrega a la instalación de Asterisk es *"res_speech_lumenvox.so"*, es por esto que se lo copia en el directorio /var/lib/asterisk/modules/ como se indica anteriormente, que es en donde se encuentran todos los módulos de Asterisk.

El archivo *"/usr/sbin/safe_asterisk"* es un *script* para arrancar Asterisk de manera persistente, es decir que si por algún motivo se para, éste insiste en subirlo nuevamente y es el que se ejecuta cuando se llama al comando *"service asterisk start | stop | restart"* o cuando se inicia el sistema. Para que las variables de entorno que agrega *Lumenvox* puedan ser vistas desde este *script* se necesita que sean exportadas, en el *Anexo A.4.1* se indica el archivo *"/usr/sbin/safe_asterisk"* con las modificaciones respectivas, que son solamente

líneas agregadas al inicio de la función *"run_asterisk()"* indicando la exportación de las variables mediante el comando *"export"*.

3.3.5 TEXT TO SPEECH CEPSTRAL

El software para conversión de texto a voz (TTS) Cepstral se compra como voces masculinas o femeninas en determinado idioma, la distinción se la hace por el nombre. En este proyecto se utilizará la voz masculina en español de *Miguel*, que está diseñada específicamente para aplicaciones telefónicas, porque puede codificar los sonidos a 8 KHz directamente. El archivo que se debe descargar desde su sitio Web[24] es:

- Cepstral_Miguel_i386-linux_4.0.3.tar.tar

De manera similar se lo puede copiar en el directorio /usr/src y se ejecuta lo siguiente:

```
# tar -xvfz Cepstral_Miguel_i386-linux_4.0.3.tar
# cd Cepstral_Miguel_i386-linux_4.0.3
# ./install.sh
```

Una vez instalado la aplicación que se agrega a la línea de comando es *"swift"* y mediante ésta se puede hacer la conversión de texto a voz. De manera similar al motor de reconocimiento se deben instalar las licencias a manera de puertos, lo que se debe ejecutar se muestra a continuación:

```
# swift --register
```

El registro de las licencias es también interactivo, pues se solicita el ingresar el nombre del usuario, compañía, nombre de la voz y la clave de licencia, datos que se conocen cuando se adquiere el producto. Se debe hacer el mismo proceso por cada licencia que se desee instalar.

---

[24] http://www.lumenvox.com/downloads

3.3.6 ADICIONALES

Si bien no son aplicaciones como las que hasta ahora se ha instalado, existen dos librerías que deben ser necesariamente incorporadas en el sistema, la primera corresponde al conector JDBC para lograr la conexión con la base de datos del SAE y la segunda logra tener una interfaz con Asterisk mediante pequeños programas escritos en lenguaje de programación Java.

3.3.6.1 jTDS

El proyecto jTDS se enfoca en la conexión con bases de datos Microsoft SQL Server y Sybase mediante lenguaje Java a través de un conector JDBC de código abierto. La librería puede ser descargada desde la Web[25] y se ha elegido para este proyecto, almacenarla en el directorio /usr/local/agi-java/. La librería se denomina *"jtds-1.2.1.jar"* y la variable de entorno CLASSPATH deberá apuntar a este paquete, más adelante se verá la manera de hacerlo.

3.3.6.2 Asterisk-Java

El paquete *Asterisk-Java* es un módulo AGI (*Asterisk Gateway Interface*) para Java y aunque todavía no se tiene la versión 1.0 del producto (porque se necesita soportar todas las aplicaciones del API de Asterisk), las versiones actuales son de mucha utilidad para este proyecto, debido a que las aplicaciones que se necesitan ya están incorporadas, además que el software es de libre distribución.

Para empezar, se lo debe descargar desde la página del proyecto[26] y lo que se tiene es también un paquete denominado *"asterisk-java-0.3.jar"* que debe almacenarse en /usr/local/agi-java/. La variable de entorno CLASSPATH debe apuntar también a ésta librería, en el archivo /etc/profile se debería tener:

```
CLASSPATH=/usr/src/jdk1.6.0:/usr/local/agi-java/jtds-1.2.1.jar:
  /usr/local/agi-java:/usr/local/agi-java/asterisk-java-0.3.jar
```

[25] http://jtds.sourceforge.net/

[26] http://asterisk-java.org/download/0.3

Se apunta también al directorio /usr/local/agi-java/ porque más adelante se observará que aquí estarán almacenados los *scripts* para la interacción entre Asterisk y la base de datos del SAE.

Para utilizar el módulo *Asterisk-Java* es necesario tener un servicio en ejecución permanentemente (org.asteriskjava.fastagi.DefaultAgiServer), y para esto se ha agregado el correspondiente comando en el archivo /etc/rc.local que se ejecuta al arrancar el sistema operativo una vez inicializados todos los servicios que estén configurados, en el *Anexo A.4.2* se muestra la configuración de este archivo.

Una vez hecho esto es recomendable reiniciar el sistema y verificar que todos los servicios se inicien correctamente, esto se lo puede hacer verificando la lista de procesos activos mediante el comando *"ps -ef"*.

3.4 CONFIGURACIÓN DEL SERVIDOR DE BASE DE DATOS SQL SERVER

En la *sección 2.4* se trató la interacción del IVR con el SAE, tanto la creación de la base de datos que manejará servicios adicionales como los procedimientos almacenados para la consulta de notas de los estudiantes. Se lo hizo de una manera general sin incluir sentencias en lenguaje SQL, solamente indicando las tablas de la base de datos y la información que se necesita consultar para este proyecto. Ahora se plasmará dicho análisis en la base de datos real, creando las secuencias de comandos en lenguaje SQL.

3.4.1 IMPLEMENTACIÓN DE LA BASE DE DATOS PARA SERVICIOS ADICIONALES

Como se ha mencionado anteriormente, por servicios adicionales se considera la consulta del estado de planes de proyectos de titulación y de certificados de prácticas preprofesionales, por lo tanto se deberá tener tres tablas, una para cada

servicio y otra adicional para manejar la información correspondiente a los estudiantes. La estructura de esta base de datos es la de la *figura 2.2* y los campos de cada tabla con su tipo de datos se lo tiene en las *tablas 2.9*, *2.10*, *2.11*, correspondientes a *"ESTUDIANTES"*, *"PROYECTOS"* y *"PRÁCTICAS"* respectivamente.

```
CREATE DATABASE PRAPRO
GO

USE PRAPRO
GO

CREATE TABLE ESTUDIANTES (
      codest varchar(9) NOT NULL,
      nomest varchar(50) NOT NULL,
      totcred int NULL
      mattes bit NULL
      constraint pk_estudiantes primary key(codest)
)
GO

CREATE TABLE PRACTICAS (
      pracid int IDENTITY(1, 1) NOT NULL,
      codest varchar(9) NOT NULL,
      nomemp varchar(30) NOT NULL,
      numhor int NOT NULL,
      fecrec smalldatetime NOT NULL,
      fecrev smalldatetime NOT NULL,
      estcer varchar(2000) NOT NULL
      constraint pk_practicas primary key(pracid),
      constraint fk_estpra foreign key(codest) references ESTUDIANTES
)
GO

CREATE TABLE PROYECTOS (
      proyid int IDENTITY(1, 1) NOT NULL,
      codest varchar (9) NOT NULL,
      templa varchar (300) NOT NULL,
      nomdir varchar (30) NULL,
      fecrec smalldatetime NOT NULL,
      fecrev smalldatetime NOT NULL,
      estpla varchar(2000) NULL
      constraint pk_proyectos primary key(proyid),
      constraint fk_estpro foreign key(codest) references ESTUDIANTES
)
GO
```

espacio de código 3.1: Comandos SQL para generar la base de datos adicional

La base de datos se la llamará *“PRAPRO”* y en el *espacio de código 3.1* se muestra la secuencia de comandos SQL para generarla, incluidas las reglas para establecer las referencias entre tablas. Lo que se indica es básicamente la creación de las tres tablas, en el *Anexo B.1* se puede observar la secuencia completa de comandos, incluyendo desencadenadores para verificar el correcto ingreso, actualización y eliminación de registros, además de procedimientos almacenados para la consulta de la información correspondiente a planes de proyectos de titulación y certificados de prácticas preprofesionales.

3.4.2 IMPLEMENTACIÓN DE LA COMUNICACIÓN CON EL SAE DE LA CARRERA

De manera similar, en la *sección 2.4.2* se indicó la información que debe ser consultada en la base de datos del SAE para proporcionar las notas el momento que un usuario utilice el sistema; se dio la estructura de los procedimientos almacenados necesarios para este efecto.

Son cuatro los procedimientos almacenados y la secuencia de comandos SQL para generarlos se tiene en el *espacio de código 3.2*. El primero es *“sp_reqNombre”* y consulta solamente información básica de un estudiante como su nombre, su sexo y si está matriculado en el presente semestre, esta información sirve para darle la bienvenida luego de que haya ingresado su número único. El siguiente procedimiento almacenado es *“sp_reqNotas”* y consulta todas las notas que el estudiante tiene registrado en la base de datos, a partir de este conjunto de calificaciones se seleccionan solamente las que necesite conocer. Estos dos procedimientos son los únicos que se ejecutan cada vez que el usuario ingresa en el sistema y desea consultar sus notas.

Los dos procedimientos almacenados que se crean a continuación se utilizan para copiar la información de la tabla *“ESTUDIANTES”* del SAE a la tabla *“ESTUDIANTES”* de la base de datos *“PRAPRO”*, esto se lo ha hecho así porque es más conveniente copiar esta información cada semestre (realizar una sola

consulta al SAE) que consultar cada vez que se necesite agregar un estudiante, además que las futuras consultas se la harán sobre esta nueva información, que es de extensión mucho más reducida y con menos registros, debido a que los estudiantes aptos para presentar un plan de proyecto de titulación deben estar matriculados en esa materia y para realizar prácticas preprofesionales deben tener al menos aprobado el 70% de los créditos de la carrera[27].

```
CREATE PROCEDURE  sp_reqNombre
      @numUni varchar(9)
AS
      SELECT estatue, nomest, sexoes FROM SAEESTUD
      WHERE codest = @numUni
GO

CREATE PROCEDURE  sp_reqNotas
      @numUni varchar(9)
AS
      SELECT codmat, numcal, calif1, calif2, calif3, calif4, calif5,
             sumato, aprueb FROM SAECALIF
      WHERE CODEST = @numUni
GO

CREATE PROCEDURE sp_reqEstudiantes
      @totCred int
AS
      SELECT codest, nomest, totcred FROM SAEESTUD
      WHERE estatue = 'MATRICULADO' AND totcred >= @totCred
GO

CREATE PROCEDURE sp_reqTesis
      @numUni varchar(9)
AS
      SELECT count(*) FROM SAECALIF
      WHERE codest = @numuni AND codmat = 'IRO920'
GO
```

espacio de código 3.2: Comandos SQL para generar los procedimientos almacenados

Hay que resaltar que en la base de datos del SAE se necesita crear un usuario con permisos de ejecución solamente para estos procedimientos almacenados, este trabajo lo realizan las personas que forman parte de la UGI.

---

[27] El valor 120 en el procedimiento almacenado *"sp_reqEstudiantes"* es más bajo que el 70% de los créditos necesarios para aprobar la carrera pero se lo ha dejado así porque suelen existir casos aislados en donde a un estudiante se le puede asignar ciertas prácticas preprofesionales.

3.5 IMPLEMENTACIÓN DEL IVR

En Asterisk, como en la mayoría de aplicaciones que funcionan sobre Linux, existen archivos de configuración que establecen el comportamiento de todas las funciones de la PBX, estos archivos se encuentran en /*etc*/*asterisk*/ y cada vez que se haga referencia a un archivo sin indicar su ruta absoluta se considerará que se encuentra dentro de este directorio.

Como se mencionó en la *sección 1.4.2*, es en el archivo *extensions.conf* en donde se establece el plan de numeración, el enrutamiento de llamadas y algunas funciones adicionales entre las que está la creación de IVRs. Existe una estructura definida para editar cada archivo y en el caso del *extensions.conf* se precisan de tres niveles jerárquicos. El primero de ellos se denomina *contexto*, y es un fragmento del archivo en el que se define todas las opciones que tiene un usuario en determinado instante dentro del IVR. Cada contexto empieza con su definición, asignándole un nombre encerrado entre corchetes, un contexto acaba cuando se define otro. A continuación se encuentran las *extensiones*, que pueden ser numéricas o literales, las primeras son muy utilizadas cuando se crean IVRs para detección de tonos DTMF y las otras cuando se agrega reconocimiento de voz, aunque esto no necesariamente se constituye en una regla. Por último se mencionan las *prioridades* que definen el orden de ejecución de las aplicaciones que componen el diálogo del IVR, por lo tanto, cada aplicación que corresponda a una extensión determinada dentro un contexto en el archivo *extensions.conf*, deberá tener una prioridad de ejecución. Existe una gran variedad de aplicaciones que se pueden utilizar dentro de un IVR, dependiendo de la naturaleza del mismo. En esta sección se explicará cada una de ellas mientras se las vaya utilizando, a la vez que se entenderá de mejor manera la estructura de *contextos*, *extensiones* y *prioridades*.

Nuevamente se hace la distinción entre el IVR con reconocimiento natural de voz y el IVR con detección de tonos DTMF, los mismos que en el archivo de configuración pueden estructurarse en uno o varios contextos, en el *Anexo A.2.1* se puede confirmar este asunto.

3.5.1 IVR CON RECONOCIMIENTO NATURAL DE VOZ

El IVR con reconocimiento natural de voz es el inicio del diálogo guiado que se le ofrece al usuario, por cuanto el IVR con detección de tonos DTMF se ejecuta solamente cuando se sobrepase el número de puertos de reconocimiento. La estructura de este IVR mostrada en la *figura 2.4* se divide en seis secciones para facilitar su programación, las mismas que se indican en la *tabla 3.1*.

Sección	***Aplicaciones***
A	Contestación de la llamada y recepción del número único
B	Confirmación del número único
C	Consulta del nombre del estudiante en el SAE y determinación del servicio a utilizar
D	Consulta de notas
E	Consulta del estado de certificados de prácticas preprofesionales
F	Consulta del estado de planes de proyectos de titulación

tabla 3.1: *Secciones del IVR con reconocimiento natural de voz*

Es muy común pensar en una extensión como un terminal de usuario que tenga un número telefónico de uno o más dígitos al que la persona llamante pueda marcar, pero en Asterisk para la creación de IVRs se utilizan también las extensiones como grupos de aplicaciones a las cuales se puede redireccionar una llamada en cierto instante, es así que para el IVR con reconocimiento natural de voz se crearán seis extensiones correspondientes a cada sección de la *tabla 3.1*.

3.5.1.1 Contestación de la llamada y recepción del número único

La primera extensión y punto de partida del diálogo guiado del IVR se muestra en el *espacio de código 3.3*. El contexto se llama *"IERI-IVR-VOZ"*, la extensión es *"a"*, y empieza con la prioridad *"1"*, las demás aplicaciones se ejecutan según el orden de las mismas en el archivo, esto es especificado por la prioridad *"n"* que llevan consigo.

```
[IERI-IVR-VOZ]

exten => a,1,Answer()
exten => a,n,Wait(1)
exten => a,n,SpeechCreate()
exten => a,n,GotoIf($["${ERROR}" != ""]?IERI-IVR-DTMF,s,3)
exten => a,n,Set(SPEECH_DTMF_MAXLEN=9)
exten => a,n,Set(CONTADOR=0)
exten => a,n,SpeechLoadGrammar(NumeroUnico|
                          /etc/asterisk/grammars/NumeroUnico.grxml)
exten => a,n,SpeechLoadGrammar(NumeroUnicoDTMF|
                          /etc/asterisk/grammars/NumeroUnicoDTMF.grxml)
exten => a,n,PlayBack(IERI-IVR/prompts/hola)
exten => a,n(getnu),SpeechActivateGrammar(NumeroUnico)
exten => a,n,SpeechActivateGrammar(NumeroUnicoDTMF)
exten => a,n(start),SpeechStart()
exten => a,n,SpeechBackground(IERI-IVR/prompts/diga_digite_nu,5)
exten => a,n,Macro(rcvRespuestaNuUn,start)
exten => a,n,SpeechDeactivateGrammar(NumeroUnico)
exten => a,n,SpeechDeactivateGrammar(NumeroUnicoDTMF)
exten => a,n,GotoIf($["${TYPGRAM}" = "voz"]?b,1:c,1)
```

espacio de código 3.3: Contestación de la llamada y recepción del número único

La primera aplicación es *“Answer”* y lo que hace es contestar la llamada en cuanto arribe a la central telefónica y *“Wait”* espera tantos segundos como se indiquen entre paréntesis antes de ejecutar la siguiente aplicación. *“SpeechCreate”* es una de las aplicaciones para reconocimiento de voz, lo que hace es verificar si hay licencias disponibles en el motor, en caso afirmativo ocupa una de ellas para el reconocimiento posterior y la libera cuando se cuelgue la llamada o se invoque a la aplicación *“SpeechDestroy”*. Cuando el motor no se encuentra activo o no hay licencias disponibles se lanza un error y para ser controlado se ejecuta la siguiente aplicación *“GotoIf”*, si existe error la llamada se direcciona a un contexto llamado *“IERI-IVR-DTMF”*, a la extensión *“s”* con prioridad *“3”*; mas adelante se verá que éste es el IVR con detección de tonos DTMF.

La aplicación *“Set”* sirve para asignar variables, sean propias de las aplicaciones o creadas por el usuario, un ejemplo de las primeras es *“SPEECH_DTMF_MAXLEN”* que indica al motor el número máximo de dígitos que va a pulsar el usuario. La variable *“CONTADOR”* es creada y asignada para controlar el número de intentos del usuario al pronunciar su número único.

A continuación se cargan en memoria dos archivos de gramática para detectar el número único, uno mediante reconocimiento de voz y el otro mediante tonos DTMF, la estructura de estos archivos se explica con detalle en la *sección 3.6*. En este punto se da la bienvenida al usuario, la aplicación *"PlayBack"* permite reproducir un archivo de audio[28] sin que pueda ser interrumpido ante la pulsación de alguna tecla del teléfono, en cuanto termina se activan los dos archivos de gramática que se cargaron en memoria para proceder al reconocimiento en cuanto empiece la aplicación *"SpeechBackground"* después de *"SpeechStart"*. Lo que se hace es reproducir un audio en donde se indica al usuario lo que debe pronunciar o digitar y en cuanto lo haga se detiene la reproducción y se procede con el reconocimiento.

Las etiquetas *"getnu"* y *"start"* que se encuentran entre paréntesis sirven para direccionar el diálogo a esa prioridad cuando se requiera.

La aplicación *"SpeechBackground"* lleva como primer parámetro el audio a reproducir y como segundo parámetro el número de segundos que sigue esperando luego de finalizada la reproducción. Cuando el motor de reconocimiento ha detectado una respuesta coincidente en los archivos de gramática o se hayan acabado los cinco segundos después de reproducir el audio se ejecuta la siguiente aplicación que es una *Macro*, es decir una sección de código que se halla en otro contexto y que puede recibir parámetros, algo similar a una función o método. El primer argumento es el nombre de la *Macro* y los demás parámetros son los que se hayan configurado por parte del usuario, en este caso la *Macro* se llama *"rcvRespuestaNuUn"*, lo que hace es determinar si hubo o no respuesta del usuario y si es un número único válido, el parámetro *"start"* sirve para volver a ejecutar la detección en caso de requerirse.

Todas las macros utilizadas en el IVR se crean también en el archivo *"extensions.conf"*, luego de la implementación de cada contexto. Para una mejor comprensión de las mismas se recomienda revisar el *Anexo A.2.1*, en donde cada

[28] En el *Anexo C* se tiene una lista de todos los archivos de audio que se utilizarán en el IVR.

macro corresponde a un contexto que empieza con la palabra *“macro”* seguida de un guión y el nombre que la identifica.

Si ya se tiene el número único detectado se desactivan los archivos de gramática y se discrimina si el reconocimiento fue mediante voz o tonos DTMF, en el primer caso para solicitar una confirmación por parte del usuario o en el segundo, simplemente para proseguir con la consulta a la base de datos. La variable *“TYPGRAM”* es asignada en la *Macro “rcvRespuestaNuUn”*.

3.5.1.2 Confirmación del número único

Cuando el motor de reconocimiento ha detectado un número único mediante reconocimiento de voz se hace necesario una confirmación del mismo, no siendo así en el caso de detección mediante tonos DTMF. La siguiente extensión es la *“b”* y se muestra en el *espacio de código 3.4*.

```
exten => b,1,PlayBack(IERI-IVR/prompts/su_nu_es)
exten => b,n,GotoIf($["${NUMUNI:0:2}" = "20"]?${n}+1:${n}+3)
exten => b,n,PlayBack(IERI-IVR/numeros/2000)
exten => b,n,PlayBack(IERI-IVR/numeros/${IF($["${NUMUNI:2:1}" = "0"]?
                                          ${NUMUNI:3:1}:${NUMUNI:2:2})})
exten => b,n,Set(BEGIN=${IF($["${NUMUNI:0:2}" = "20"]?5:0)})
exten => b,n,Set(CURSOR=$[${BEGIN}+1])
exten => b,n,Set(LENGTH=${LEN(${NUMUNI})})
exten => b,n,While($[${BEGIN} < ${LENGTH}])
exten => b,n,Set(CURSOR=${IF($["${NUMUNI:$[${BEGIN}+1]:1}" = "_"]?1:2)})
exten => b,n,PlayBack(IERI-IVR/numeros/${NUMUNI:${BEGIN}:${CURSOR}})
exten => b,n,Set(BEGIN=$[${BEGIN}+${CURSOR}+1])
exten => b,n,Set(CURSOR=1)
exten => b,n,EndWhile()
exten => b,n,Macro(rcvRespuestaSiNo)
exten => b,n,GotoIf($["${RESPSINO}" = "si"]?c,1)
exten => b,n,Macro(verificarMaxIntentos)
exten => b,n,Goto(a,getnu)
```

espacio de código 3.4: Confirmación del número único

Lo que se hace en esta extensión es repetir al usuario lo que el motor detectó y preguntarle si es correcto, en caso afirmativo se procede con la siguiente sección del diálogo del IVR pero si no es así se debe volver a la sección anterior y solicitar nuevamente se pronuncie o digite el número único. Es poco aceptable que el

sistema pida confirmación del número único pronunciado de manera distinta como lo hizo el usuario, consideremos el siguiente ejemplo:

Usuario: *“cero dos diez cero diecisiete”*
Sistema: *“su número único es: cero dos uno cero cero uno siete?”*

El motor de reconocimiento acertó en la detección del número único pero el usuario puede confundirse y negar la confirmación. La solución es preguntar al usuario de la misma manera que él pronunció el número único y para esto se utiliza la interpretación semántica en los archivos de gramática. Lo que devuelve el motor la cadena de caracteres que ha reconocido y para el IVR de este proyecto se necesita que esta cadena de retorno tenga la siguiente estructura:

“0_2_10_0_17_”

Así se logra identificar la combinación de unidades y decenas que utilizó el usuario. Lo que se hace en el *espacio de código 3.4* es primeramente identificar si el número único es de siete o nueve dígitos, si es de nueve seguramente empieza con un año, por lo tanto hay que reproducir el audio de ese año, los cinco dígitos restantes o todos los siete (si el numero es de siete dígitos), formarán combinaciones y cada unidad o decena detectada será precedida del símbolo *“_”*. El final del número único reconocido se lo puede conocer mediante la función *“LEN()”* que devuelve la longitud de una cadena.

Luego de preguntar si se pronunció un número único determinado, el usuario puede afirmar o negar dicha posibilidad, la discriminación de la respuesta se la hace en la macro *“rcvRespuestaSiNo”*. Cuando el sistema ha acertado y la respuesta del usuario es afirmativa se continúa con la siguiente sección que es determinar que servicio se desea utilizar; de lo contrario se incrementa un intento en el contador mediante otra macro llamada *“VerificarMaxIntentos”* y se vuelve a la sección anterior para solicitar nuevamente el número único.

3.5.1.3 Consulta del nombre del estudiante en el SAE y determinación del servicio a utilizar

Cuando se conoce el número único que el usuario ha ingresado se procede a determinar si se trata de un estudiante válido, en este punto se produce la primera interacción con la base de datos del SAE y para esto se llama a la aplicación *"Agi"* utilizando el script *"sqlReqNombre"*[29], a la cual se le envía como parámetro el número único detectado, esta sección se encuentra en el *espacio de código 3.5*.

```
exten => c,1,Agi(agi://localhost/sqlReqNombre?nu=${NUMUNI})
exten => c,n,Macro(verificarConexionBDD)
exten => c,n,GotoIf($[$["${NOMEST}" != ""] & $["${ESTMAT}" =
                                                "MATRICULADO"]]?${n}+3)
exten => c,n,PlayBack(IERI-IVR/prompts/nu_no_coincide)
exten => c,n,Goto(a,getnu)

exten => c,n,PlayBack(IERI-IVR/prompts/${IF($["${SEXOES:0:1}" = "M"]?
                                             bienvenido:bienvenida)})
exten => c,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
               –m text -o /var/lib/asterisk/sounds/IERI-IVR/nombres/
                                            ${NUMUNI}.wav "${NOMEST}")
exten => c,n,PlayBack(IERI-IVR/nombres/${NUMUNI})
exten => c,n(getan),SpeechLoadGrammar
                          (MenuIVR|/etc/asterisk/grammars/MenuIVR.grxml)
exten => c,n,SpeechActivateGrammar(MenuIVR)
exten => c,n(start),SpeechStart()
exten => c,n,SpeechBackground(IERI-IVR/prompts/menu_voz,5)
exten => c,n,Macro(rcvRespuestaMenu,start)
exten => c,n,SpeechDeactivateGrammar(MenuIVR)
exten => c,n,SpeechUnloadGrammar(MenuIVR)
exten => c,n,GotoIf($["${RESPUESTA}" = "notas"]?d,1)
exten => c,n,GotoIf($["${RESPUESTA}" = "practicas"]?e,1)
exten => c,n,GotoIf($["${RESPUESTA}" = "proyectos"]?f,1)
```

espacio de código 3.5: Consulta del nombre del estudiante en el SAE y determinación del servicio a utilizar

Se utiliza una macro llamada *"verificarConexionBDD"* siempre que se hace consultas a la base para determinar cualquier error de conexión. Si el número único detectado no coincide con ninguno registrado en la base de datos del SAE se vuelve a la extensión *"a"* del IVR, caso contrario se da la bienvenida al

[29] En el IVR de este proyecto se utilizan varios scripts llamados desde la aplicación *"Agi"* y todos ellos se analizan en la sección 3.6.

estudiante y con la variable *“NOMEST”* que contiene el nombre del mismo se crea el archivo de audio correspondiente, se lo hace mediante la aplicación *“swift”* del sistema operativo que se crea al instalar el software para conversión de texto a vox *Cepstral*.

Se reproduce el audio que contiene el nombre del estudiante una vez creado y a continuación se le pregunta qué servicio desea utilizar, consulta de notas, estado de planes de proyectos de titulación o certificados de prácticas preprofesionales. Para esto se carga en memoria y se activa el archivo de gramática *“MenuIVR.grxml”* y de acuerdo a la elección del usuario se va a la extensión *“d”*, *“e”* o *“f”* respectivamente.

3.5.1.4 Consulta de notas

La extensión correspondiente a la consulta de notas es la *“d”* y se muestra en el *espacio de código 3.6*. Indistintamente si el usuario elija recibir todas sus calificaciones o específicamente por algún bimestre o materia la consulta se hace de todas las notas disponibles de dicho estudiante, y la discriminación se hace en base a los datos obtenidos.

```
exten => d,1,Agi(agi://localhost/sqlReqNotas?nu=${NUMUNI})
exten => d,n,Macro(verificarConexionBDD)
exten => d,n(getan),SpeechLoadGrammar(MateriasAlumno|/etc/asterisk
                                    /grammars/materias/${NUMUNI}.grxml)
exten => d,n,SpeechActivateGrammar(MateriasAlumno)
exten => d,n(start),SpeechStart()
exten => d,n,SpeechBackground(IERI-IVR/prompts/menu_voz_notas,5)
exten => d,n,Macro(rcvRespuestaMenu,start)
exten => d,n,Set(RESPMAT=${RESPUESTA})
exten => d,n,SpeechDeactivateGrammar(MateriasAlumno)
exten => d,n,SpeechUnloadGrammar(MateriasAlumno)
exten => d,n(info),MacroIf($["${RESPMAT}" = "1_bimestre"]?
                                                  decirNotasBimestre,1)
exten => d,n,MacroIf($["${RESPMAT}" = "2_bimestre"]?decirNotasBimestre,2)
exten => d,n,MacroIf($["${RESPMAT}" = "supletorio"]?decirNotasBimestre,3)
exten => d,n,MacroIf($["${RESPMAT}" = "todasnotas"]?decirNotasTodas)
exten => d,n,MacroIf($["${LEN(${RESPMAT})}" = "6"]?
                                          decirNotasMateria,${RESPMAT})
exten => d,n,Macro(menuRepInfoVoz,info)
```

espacio de código 3.6: Consulta de notas

La consulta a la base de datos del SAE se la hace a través del script *“sqlReqNotas”* enviándole como parámetro el número único del estudiante. Se pregunta al usuario las notas que desea consultar y en base a su respuesta se ejecuta una macro específica mediante la aplicación *“MacroIf”*. Para primer bimestre, segundo bimestre y supletorio se utiliza una misma macro llamada *“decirNotasBimestre”* con un solo parámetro, *“1”*, *“2”* y *“3”* respectivamente, y para consultar todas las notas la macro es *“decirNotasTodas”*. Si se ha elegido consultar las notas de una materia en particular se ejecuta otra macro llamada *“decirNotasMateria”*.

Una vez proporcionada la información al usuario se consulta si desea que se repita dicha información, si desea consultar más notas o si desea regresar al menú principal para utilizar otro servicio, todo esto es manejado por una macro adicional, *“menuRepInfoVoz”* que recibe como parámetro la prioridad desde donde se repetirá la información.

3.5.1.5 Consulta del estado de certificados de prácticas preprofesionales

Para consultar el estado de certificados de prácticas preprofesionales se conecta con la base de datos PRAPRO, para esto se utiliza el script *“sqlReqPracticas”* nuevamente con el número único del estudiante. La información a proporcionar al estudiante es el estado del certificado y el nombre de la empresa en donde se hizo la práctica, este último para corroborar que los datos solicitados sean los correctos. Se crean los archivos de audio en base al contenido de las variables *“ESTCER”* y *“NOMEPM”* respectivamente (*espacio de código 3.7*).

En éste punto se controla también si el estudiante no tiene registrado ningún certificado de prácticas preprofesionales o si el mismo aún no ha sido revisado. Luego de proporcionar la información al usuario se ejecuta la macro *“menuRepInfoVoz”* que permite repetir los datos recibidos o regresar al menú principal para utilizar otro servicio.

```
exten => e,1,Agi(agi://localhost/sqlReqPracticas?nu=${NUMUNI})
exten => e,n,Macro(verificarConexionBDD)
exten => e,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                 -m text -o /var/lib/asterisk/sounds/IERI-IVR/practicas/
                                                  ${NUMUNI}.wav "${ESTCER}")
exten => e,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                 -m text -o /var/lib/asterisk/sounds/IERI-IVR/practicas/
                                                emp${NUMUNI}.wav "${NOMEMP}")
exten => e,n(prac),GotoIf($["${ESTCER}" = ""]?${n}+1:${n}+3)
exten => e,n,PlayBack(IERI-IVR/prompts/ninguna_practica)
exten => e,n,Goto(rep)
exten => e,n,PlayBack(IERI-IVR/prompts/rev_practicas)
exten => e,n,PlayBack(IERI-IVR/practicas/emp${NUMUNI})
exten => e,n,PlayBack(IERI-IVR/prompts/rev_estado)
exten => e,n,PlayBack(IERI-IVR/practicas/${NUMUNI})
exten => e,n(rep),Macro(menuRepInfoVoz,prac)
```

espacio de código 3.7: Consulta del estado de certificados de prácticas preprofesionales

3.5.1.6 Consulta del estado de planes de proyectos de titulación

El proceso es similar a la extensión anterior, la consulta se la hace en la base de datos PRAPRO a diferencia que el script de la aplicación *"Agi"* es *"sqlReqProyectos"* y que la información a proporcionar es el estado del plan y el tema del mismo, nuevamente para corroborar que los datos recibidos sean los correctos. El *espacio de código 3.8* corresponde a esta extensión.

```
exten => f,1,Agi(agi://localhost/sqlReqProyectos?nu=${NUMUNI})
exten => f,n,Macro(verificarConexionBDD)
exten => f,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                -m text -o /var/lib/asterisk/sounds/IERI-IVR/proyectos/
                                                  ${NUMUNI}.wav "${ESTPLA}")
exten => f,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                -m text -o /var/lib/asterisk/sounds/IERI-IVR/proyectos/
                                                tem${NUMUNI}.wav "${TEMPLA}")
exten => f,n(prac),GotoIf($["${ESTPLA}" = ""]?${n}+1:${n}+3)
exten => f,n,PlayBack(IERI-IVR/prompts/ningun_proyecto)
exten => f,n,Goto(rep)
exten => f,n,PlayBack(IERI-IVR/prompts/rev_proyectos)
exten => f,n,PlayBack(IERI-IVR/proyectos/tem${NUMUNI})
exten => f,n,PlayBack(IERI-IVR/prompts/rev_estado)
exten => f,n,PlayBack(IERI-IVR/proyectos/${NUMUNI})
exten => f,n(rep),Macro(menuRepInfoVoz,prac)
```

espacio de código 3.8: Consulta del estado de planes de proyectos de titulación

3.5.2 IVR CON DETECCIÓN DE TONOS DTMF

El IVR con detección de tonos DTMF tiene una estructura similar al IVR que incorpora reconocimiento de voz (*figura 2.5*), utiliza las mismas *macros* para realizar las consultas a la base de datos y también las mismas que proporcionan al usuario la información solicitada. Se diferencia básicamente en la manera que el usuario ingresa sus respuestas; a través de la pulsación de las teclas del teléfono.

El IVR en cuestión no puede formarse en un solo contexto como el anterior debido a que al iniciar el diálogo, luego de ingresar el número único, el usuario debe escoger entre tres opciones: consulta de notas, planes de proyetos de titulación y certificados de prácticas preprofesionales (teclas 1, 2 y 3 respectivamente); y si elige la primera ingresará en un nuevo submenú: primer bimestre, segundo bimestre, supletorio o todas las notas; nuevamente se repiten las tres primeras teclas del teléfono. En Asterisk la solución puede ser utilizar teclas contiguas (desde el 1 al 7) o simplemente crear un contexto por cada submenú, para este proyecto se ha elegido la segunda alternativa.

En el *Anexo A.2.1* luego del IVR con reconocimiento natural de voz encontramos dos contextos más, *"IERI-IVR-DTMF"* y *"IERI-IVR-DTMF-notas"*, éstos corresponden al IVR con detección de tonos DTMF.

Cuando la interfaz del usuario con el sistema es solamente el teclado del teléfono se utilizan extensiones predefinidas en Asterisk que se activan bajo determinado suceso. En el *espacio de código 3.9* se muestra el contexto *"IERI-IVR-DTMF-notas"*, que es el submenú al que ingresa el usuario cuando solicita consultar sus notas. La extensión *"s"* (start) es la primera que se ejecuta al ingresar en el contexto, *"i"* (invalid) es una extensión hacia la que se direcciona la llamada cuando el usuario ha ingresado una opción no válida, y las aplicaciones que existan en *"t"* (timeout) se ejecutan cuando se haya acabado el tiempo de espera (5 segundos por defecto) después de haber solicitado al usuario elija una opción. Suelen existir más opciones pero las indicadas son las que se utilizan en el IVR con detección de tonos DTMF.

```
[IERI-IVR-DTMF-notas]

exten => s,1,Agi(agi://localhost/sqlReqNotas?nu=${NUMUNI})
exten => s,n,Macro(verificarConexionBDD)

exten => s,n(mnotas),Background(IERI-IVR/prompts/menu_dtmf_notas)
exten => s,n,WaitExten()

exten => _[123],1,Macro(decirNotasBimestre,${EXTEN})
exten => _[123],n,Macro(menuRepInfoDTMF)

exten => 4,1,Macro(decirNotasTodas)
exten => 4,n,Macro(menuRepInfoDTMF)

exten => i,1,Playback(IERI-IVR/prompts/op_no_valida)
exten => i,n,Goto(s,mnotas)

exten => t,1,Playback(IERI-IVR/prompts/op_no_digitada)
exten => t,n,Goto(s,mnotas)
```

espacio de código 3.9: IVR con detección de tonos DTMF, consulta de notas

La aplicación *"WaitExten"* es la que permite al usuario ingresar una de las opciones que se le indica mediante el archivo de audio *"menu_dtmf_notas"*, el patrón *"_[123]"* se ejecuta siempre que el usuario pulse 1, 2 o 3, la variable *"${EXTEN}"* guarda esta extensión pulsada. Se tiene aparte la extensión 4 porque la acción a tomar es distinta. De todas maneras, una vez solicitado al usuario el ingreso de alguna opción, solamente podrá escoger entre 1, 2, 3 y 4, cualquier otra tecla lo llevará a *"i"* y si no pulsa ninguna a *"t"*.

A diferencia del IVR con reconocimiento natural de voz aquí el usuario no podrá escoger una sola asignatura para saber sus notas, pues es más tedioso tener que repetirle todas las materias en las que está matriculado, además que en ciertos casos podrá necesitar extensiones de dos dígitos (10, 11, etc.) y el sistema se volvería nada amigable.

La información que se ha proporcionado muestra las diferencias en la implementación de ambos IVRs, las aplicaciones usadas y la estructura es en su mayor parte similar, además se considera que la detección de tonos DTMF es un respaldo al sistema cuando existan problemas con el reconocimiento de voz.

3.6 IMPLEMENTACIÓN DE LOS ARCHIVOS DE GRAMÁTICA A SER RECONOCIDOS

En la *sección 2.3* se analizó detenidamente la estructura del número único y la manera de cómo se debería organizar el archivo de gramática para obtener un mejor nivel de certeza en el reconocimiento, también se mencionó brevemente el recorrido del usuario por el menú principal y el menú de notas. En esta sección se construirán dichos archivos de gramática tomando en cuenta primeramente la interpretación semántica que se debe dar a los patrones de reconocimiento.

3.6.1 INTERPRETACIÓN SEMÁNTICA

La interpretación semántica permite adaptar los resultados que obtiene el motor de reconocimiento a una forma más entendible para la aplicación. En la *sección 3.5.1.2* se adelantó un poco este asunto mencionando que si un usuario, por ejemplo, pronuncia el número único *"cero, dos, diez, cero, diecisiete"* el IVR debería proporcionar un resultado similar al siguiente:

"0_2_10_0_17_"

De esta manera se conocerá claramente que combinación de dígitos y decenas utilizó y así poder repetirle el mismo número para su confirmación. Como se puede apreciar, el reconocimiento del número único equivale a detectar cantidades de una y dos cifras que se pronuncien contiguamente, aunque en realidad, el motor no reconoce números sino palabras que más que tener la escritura gramatical correcta de un número, debe representar la pronunciación de dicha cantidad en nuestro dialecto.

Continuando con la interpretación semántica de los patrones de reconocimiento se observa que cualquiera que sea la pronunciación de una cantidad el motor debería retornar siempre su representación numérica seguida de un guión bajo ("_"). Para clarificar este asunto se ha tomado una serie de cantidades que podrían reconocerse (*espacio de código 3.10*) junto con su valor de retorno.

```
<item> treintaiuno      <tag> $="31_"; </tag> </item>
<item> trentiuno        <tag> $="31_"; </tag> </item>
<item> treintidos       <tag> $="32_"; </tag> </item>
<item> treintaidos      <tag> $="32_"; </tag> </item>
<item> trentidos        <tag> $="32_"; </tag> </item>
```

espacio de código 3.10: Interpretación semántica de los patrones de reconocimiento

Se ha escrito el ejemplo anterior con la recomendación SRGS que se trató en la *sección 1.3.1*. Es claro que pueden existir varias pronunciaciones para una misma cantidad pero el valor de retorno será único. De esta manera se establecerán todas las combinaciones posibles, tomando en cuenta que mientras se vaya formando el número único a partir de la señal acústica del usuario, el valor de retorno sea una suma de las cadenas representadas como números más un guión bajo (“_”).

3.6.2 NÚMERO ÚNICO

La estructura que debe ser plasmada en los archivos de gramática para el reconocimiento del número único fue analizada con anterioridad y se encuentra en las *tablas 2.7* y *2.8*, para siete y nueve cifras respectivamente. Lo primero que se debe hacer es crear subreglas que contengan conjuntos de cantidades que coincidan con la estructura del número único.

En el *espacio de código 3.11* se describen tres reglas: la primera que corresponde a pronunciar solamente dos dígitos (cero o nueve), la segunda que contiene todos los dígitos (desde el cero hasta el nueve) y la tercera que incluye solamente decenas que terminan en uno o dos. A partir de estas tres subreglas se puede empezar a construir las primeras combinaciones del número único.

El punto de partida son los registros de la *tabla 2.7* que forman el número único de siete cifras, al trasladar las dos primeras a lenguaje XML mediante la recomendación SRGS se tiene lo que está en el *espacio de código 3.12*.

```
<!-- Subregla: Dígitos (solamente 0 y 9) -->

<rule id="digito_09" scope="public">
        <one-of xml:lang="es-CO">
                <item> cero     <tag> $="0_"; </tag> </item>
                <item> nueve    <tag> $="9_"; </tag> </item>
        </one-of>
</rule>


<!-- Subregla: Dígitos (desde el 0 hasta el 9) -->

<rule id="digito" scope="public">
        <one-of xml:lang="es-CO">
                <item> cero     <tag> $="0_"; </tag> </item>
                <item> uno      <tag> $="1_"; </tag> </item>
                <item> dos      <tag> $="2_"; </tag> </item>
                <item> tres     <tag> $="3_"; </tag> </item>
                <item> cuatro   <tag> $="4_"; </tag> </item>
                <item> cinco    <tag> $="5_"; </tag> </item>
                <item> seis     <tag> $="6_"; </tag> </item>
                <item> siete    <tag> $="7_"; </tag> </item>
                <item> ocho     <tag> $="8_"; </tag> </item>
                <item> nueve    <tag> $="9_"; </tag> </item>
        </one-of>
</rule>


<!-- Subregla: Decenas (solamente terminadas en 1 y 2) -->

<rule id="decena_x12" scope="public">
        <one-of>
                <item> once             <tag> $="11_"; </tag> </item>
                <item> doce             <tag> $="12_"; </tag> </item>
                <item> veintiuno        <tag> $="21_"; </tag> </item>
                <item> veintidos        <tag> $="22_"; </tag> </item>
                <item> treintaiuno      <tag> $="31_"; </tag> </item>
                <item> treintaidos      <tag> $="32_"; </tag> </item>
                <item> cuarentaiuno     <tag> $="41_"; </tag> </item>
                <item> cuarentaidos     <tag> $="42_"; </tag> </item>
                <item> cincuentaiuno    <tag> $="51_"; </tag> </item>
                <item> cincuentaidos    <tag> $="52_"; </tag> </item>
                <item> sesentaiuno      <tag> $="61_"; </tag> </item>
                <item> sesentaidos      <tag> $="62_"; </tag> </item>
                <item> setentaiuno      <tag> $="71_"; </tag> </item>
                <item> setentaidos      <tag> $="72_"; </tag> </item>
                <item> ochentaiuno      <tag> $="81_"; </tag> </item>
                <item> ochentaidos      <tag> $="82_"; </tag> </item>
                <item> noventaiuno      <tag> $="91_"; </tag> </item>
                <item> noventaidos      <tag> $="92_"; </tag> </item>
        </one-of>
</rule>
```

espacio de código 3.11: Subreglas para la detección del número único

```
<item>
      <!-- 1 digito, 1 decena, 1 digito, 1 decena, 1 digito -->

      <item repeat="1"> <ruleref uri="#digito_09"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_x12"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
                        <tag> $+=$$; </tag> </item>
</item>

<item>
      <!-- 1 digito, 1 decena, 2 digitos, 1 decena -->

      <item repeat="1"> <ruleref uri="#digito_09"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_x12"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
                        <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
                        <tag> $+=$$; </tag> </item>
</item>
```

espacio de código 3.12: Combinaciones de siete cifras del número único

Las dos combinaciones anteriores utilizan las subreglas del *espacio de código 3.11* y una subregla adicional llamada *“decena”* en donde se especifican números desde el diez hasta el noventa y nueve.

La primera combinación (*1 dígito, 1 decena, 1 dígito, 1 decena, 1 dígito*) se justifica de la siguiente manera:

- ***1 dígito:*** el primer dígito puede ser solo 0 o 9 porque el SAE contiene información de estudiantes desde el año 1993 hasta el 2002 (desde el 2003 son 9 dígitos) y se consideran solamente los dos últimos dígitos.
- ***1 decena:*** Haciendo referencia al punto anterior, esta decena podría empezar con cualquier número pero termina solamente en 1 o 2, debido a que el tercer dígito del número único corresponde al periodo en el cual el estudiante ingresó a la Escuela Politécnica Nacional.

- ***1 dígito:*** No hay restricciones, cualquier dígito del 0 al 9
- ***1 decena:*** No hay restricciones, cualquier decena del 10 al 99
- ***1 dígito:*** No hay restricciones, cualquier dígito del 0 al 9

En la *sección 2.3.1.2* se encuentra información detallada del procedimiento anterior y los posibles valores que puede tomar cierta cifra dependiendo de su posición dentro del número único.

De esta manera se organiza todo el archivo de gramática para reconocer el número único, tomando en cuenta que existe una sola combinación que puede ser detectada a la vez, lo que quiere decir que se debe incorporar tanto combinaciones de siete como de nueve cifras. En el *Anexo D.1* se indica este archivo de gramática que pese a su extensión es simplemente un conjunto de alternativas de pronunciación para el usuario. En el IVR de este proyecto se ha elegido el directorio /*etc*/*asterisk*/*grammars*/ como repositorio de los archivos de gramática.

3.6.3 MENÚ PRINCIPAL

El archivo de gramática para manejar un menú de pocas opciones es sencillo de implementar, en el *espacio de código 3.13* se puede observar el archivo que se utiliza para la discriminación de la respuesta del usuario en el menú principal.
En él se incluyen todas las posibles respuestas que el usuario podría pronunciar, pero el valor de retorno del motor siempre será *“notas”*, *“prácticas”* o *“proyectos”*. De esta manera se facilita su posterior procesamiento en el IVR, lo que se puede observar en las tres últimas líneas del *espacio de código 3.5*, en donde se determina el servicio a utilizar mediante la discriminación de estas tres cadenas. La interpretación semántica posibilita también crear aplicaciones modulares en donde se facilitan realizar cambios sin afectar el comportamiento de toda la aplicación. En los *Anexos D.2* hasta *D.6* se detallan los archivos de gramática que son usados en el IVR de este proyecto.

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO"
         version="1.0"
         root="menu"
         mode="voice"
         tag-format="lumenvox/1.0">

   <rule id="menu" scope="public">
      <one-of>
         <item> notas                 <tag> $="notas";     </tag> </item>
         <item> consulta de notas     <tag> $="notas";     </tag> </item>
         <item> consultar notas       <tag> $="notas";     </tag> </item>
         <item> prácticas             <tag> $="practicas"; </tag> </item>
         <item> prácticas preprofesionales
                                      <tag> $="practicas"; </tag> </item>
         <item> consultar prácticas
                                      <tag> $="practicas"; </tag> </item>
         <item> consulta de prácticas preprofesionales
                                      <tag> $="practicas"; </tag> </item>
         <item> proyectos             <tag> $="proyectos"; </tag> </item>
         <item> proyectos de titulación
                                      <tag> $="proyectos"; </tag> </item>
         <item> consultar proyectos de titulación
                                      <tag> $="proyectos"; </tag> </item>
         <item> tesis                 <tag> $="proyectos"; </tag> </item>
      </one-of>
   </rule>
</grammar>
```

espacio de código 3.13: Archivo de gramática para el menú principal

3.6.4 MENÚ DE NOTAS

Un archivo de gramática que merece una consideración especial es el que se utiliza el momento que el usuario solicita consultar sus notas. Las alternativas que se deben tener son solamente *“primer bimestre”*, *“segundo bimestre”*, *“supletorios”*, *“todas las notas”*, y el conjunto de materias en las que el estudiante esté matriculado; lo que quiere decir que depende de cada alumno.

Una solución consiste en crear dinámicamente el archivo cada vez que el usuario ingrese en el sistema, lo que se traduce en repetidas consultas a la base de datos del SAE. Tomando en cuenta que luego de concluidas las matrículas extraordinarias ya no es posible matricularse o anular matrículas en cualquier materia, se puede crear estos archivos una sola vez y serán válidos durante todo

el semestre, para ésto se creará un *script* que haga la consulta al SAE de los estudiantes matriculados y las respectivas materias que están tomando en el periodo lectivo en consideración.

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO"
         version="1.0"
         root="Materias"
         mode="voice"
         tag-format="lumenvox/1.0">

<rule id="Materias" scope="public">

   <one-of>
      <item> Redes de Area Local Inalambricas
                                        <tag> $="IRC754" </tag> </item>
      <item> Comunicaciones Satelitales   <tag> $="IRC883" </tag> </item>
      <item> Satelitales                  <tag> $="IRC883" </tag> </item>
      <item> Redes de Area Extendida      <tag> $="IRO754" </tag> </item>
      <item> WAN                          <tag> $="IRO754" </tag> </item>
      <item> Evaluación de Redes          <tag> $="IRO763" </tag> </item>
      <item> Evaluation                   <tag> $="IRO763" </tag> </item>
      <item> Administración General       <tag> $="IRO813" </tag> </item>
      <item> Administración               <tag> $="IRO813" </tag> </item>
      <item> Comercialización             <tag> $="IRO823" </tag> </item>
      <item> Redes TECEPE IPE             <tag> $="IRO863" </tag> </item>
      <item> TECEPE IPE                   <tag> $="IRO863" </tag> </item>

      <item> primer bimestre          <tag> $="1_bimestre" </tag> </item>
      <item> segundo bimestre         <tag> $="2_bimestre" </tag> </item>
      <item> supletorio               <tag> $="supletorio" </tag> </item>
      <item> todas                    <tag> $="todasnotas" </tag> </item>
   </one-of>

</rule>

</grammar>
```

espacio de código 3.14: Archivo de gramática para el menú de notas

En el *espacio de código 3.14* se observa un ejemplo de cómo sería un archivo de gramática para un estudiante determinado, nótese que para determinadas materias se tiene más de una posible pronunciación pero un solo valor de retorno.

El *script* para crear los archivos de gramática por estudiante consulta los códigos de todas las asignaturas en las que esté matriculado y mediante una rutina de

comparación inserta los *tokens* que puedan corresponderse con el nombre de cada materia. Cada archivo tendrá por nombre el número único del estudiante seguido de la extensión *".grxml"* y estarán localizados en el directorio /*etc*/*asterisk*/*grammars*/*materias*/. El *script* ha sido creado en lenguaje Java y por ende utiliza el conector JDBC *jTDS*. En el *Anexo E.1* se encuentra este *script*, el mismo que debe ser compilado[30] antes de ser ejecutado y que en el servidor se lo localiza en /*usr*/*local*/*agi-java*/*XMLCreateGrammars.java*.

3.7 CONEXIÓN ENTRE ASTERISK Y LA BASE DE DATOS SQL SERVER

En la *sección 3.3.6* se indicó el procedimiento para instalar dos librerías adicionales que serán usadas en el sistema: *Asterisk-Java* y *jTDS*. La primera establece una interfaz entre el API de Asterisk y cualquier aplicación escrita en lenguaje Java, y la segunda corresponde a un conector JDBC para enlazarse con bases de datos SQL Server. Lo que se necesita en el IVR de este proyecto es trasladar la información desde la base de datos del SAE hasta el API de Asterisk para que sea procesada y proporcionada al usuario, por lo tanto se crearán los cuatro *scripts* que permitan realizar las consultas a la base de datos, los mismos que se mencionaron en la *sección 3.5.1* mientras se iba programando el IVR, y corresponden a la consulta de:

- El nombre, sexo y estado de la matrícula del estudiante
- Las notas del estudiante
- Si tiene pendiente algún certificado de prácticas preprofesionales
- Si tiene pendiente algún plan de proyecto de titulación

[30] Por definición un script es un archivo de texto (o fragmento de código) con instrucciones que se van ejecutando consecutivamente y existen lenguajes como Perl o PHP entre otros, que están más orientados a esto. Al utilizar Java quizá no se debería hablar de scripts porque el archivo de texto con la clase o programa (.java) no se ejecuta como tal sino que necesariamente debe ser compilado (.class). De todos modos se los ha llamado scripts porque son programas sencillos con instrucciones que se ejecutan de manera muy similar a estos.

El directorio en donde estarán estos archivos es /*usr*/*local*/*agi-java*/ y la aplicación de Asterisk que los ejecuta es *"AGI()"*. El llamado a estos scripts es a través de un mapeo de nombres que se guarda en el mismo directorio, en el archivo *"fastagi-mapping.properties"*, cuyo contenido se encuentra en el espacio de *código 3.15*.

```
sqlReqNombre    = SQLReqNombre
sqlReqNotas     = SQLReqNotas
sqlReqPracticas = SQLReqPracticas
sqlReqProyectos = SQLReqProyectos
```

espacio de código 3.15: Mapeo de scripts de Asterisk

Por lo tanto, el nombre de las clases Java que se deben crear corresponden a los de la columna derecha y el nombre de la izquierda es el identificador que sirve como parámetro de la aplicación *"AGI"* en el IVR.

La librería *Asterisk-Java* proporciona una clase base (*BaseAgiScript*) de la cual debemos derivar nuestras aplicaciones. Lo que se implementa es la función *"service (AgiRequest request, AgiChannel channel)"*, en donde el primer parámetro corresponde a un manejador de la solicitud desde el API de Asterisk y el segundo sirve para tener un control de la llamada desde la aplicación.

Lo que interesa lograr con estos scripts es guardar los valores que resultan de ejecutar los procedimientos almacenados en la base de datos del SAE en variables del IVR de Asterisk, el *espacio de código 3.16* muestra la manera de hacerlo.

```
while (rs.next()) {

      channel.setVariable("ESTMAT", rs.getString(1));
      channel.setVariable("NOMEST", rs.getString(2));
      channel.setVariable("SEXOES", rs.getString(3));
      channel.setVariable("ESTBDD", "ok");
}
```

espacio de código 3.16: Exportación de variables hacia Asterisk

El primer parámetro de la función *“setVariable”* es el identificador de la variable en el IVR y el segundo parámetro es el valor que se le asigna. Una vez ejecutado el archivo, las variables de Asterisk tienen los valores correspondientes a los que se les asignó desde el *script* escrito en Java.

El procedimiento es similar para las cuatro consultas a la base de datos, tomando en cuenta que para el nombre y las notas del estudiante se ejecutan los procedimientos almacenados en la base de datos del SAE, en cambio para consultar la información correspondiente a certificados de prácticas preprofesionales y planes de proyectos de titulación la consulta se la hace en la base de datos PRAPRO.

En el *Anexo E.2* se pueden encontrar los cuatro *scripts* que se han mencionado en esta sección, no se ha entrado en más detalles debido a que el procedimiento es similar en todos: realizar la consulta correspondiente a la base de datos y exportar dichos resultados como variables del IVR de Asterisk, en donde luego se procesa la información obtenida.

Hay que recordar que estos scripts deben ser compilados para poder ser ejecutados. También, se debe mantener activo un servicio que se incorpora en la librería *Asterisk-Java* (*org.asteriskjava.fastagi.DefaultAgiServer*) que hace el mapeo de *scripts* a identificadores que son llamados desde la aplicación *“AGI”* en Asterisk, para una mayor referencia se recomienda observar el *Anexo A.4.2* en donde está la configuración del archivo */etc/rc.local*, el mismo que activa el servicio mencionado cada vez que se arranca el sistema operativo.

3.8 FUNCIONAMIENTO DEL SISTEMA

Hasta este punto se ha integrado todo el IVR para consulta de notas, estado de certificados de prácticas preprofesionales y planes de proyectos de titulación, lo que resta es determinar si su dimensionamiento ha sido correcto, además de su programación. Mientras no se tenga certeza de los resultados que brinde el sistema no debería ser puesto en producción. Simular un ambiente real

generalmente tiene limitaciones, aunque sirve como una pauta para corregir posibles errores.

El sistema fue dimensionado para soportar como máximo cuatro usuarios simultáneos, por esta razón se ha decidido tener dos puertos de reconocimiento natural de voz, lo que quiere decir que cuando un tercer o cuarto usuario ingrese, será redireccionado al IVR con detección de tonos DTMF. Indistintamente de lo anterior se necesitan los cuatro puertos de conversión de texto a voz.

A continuación se describirá la instalación y el funcionamiento del sistema tanto en ambiente real como en ambiente de pruebas, para el segundo caso, los resultados se analizarán en la *sección 3.9*.

3.8.1 INSTALACIÓN EN UN AMBIENTE REAL

El ambiente real para el que fue diseñado e implementado el IVR de este proyecto se muestra en la *figura 3.1*.

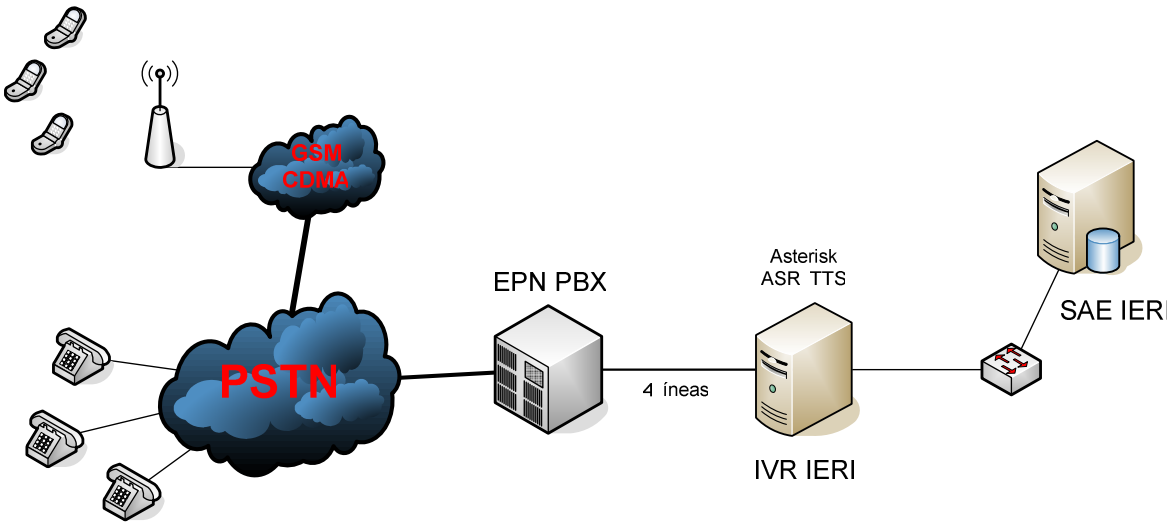

figura 3.1: Instalación del servidor en un ambiente real

El servidor en el cual funciona el IVR debe instalarse en una red LAN con el servidor que alberga la base de datos del SAE. Desde la PBX de la Escuela

Politécnica Nacional deberán salir cuatro líneas o extensiones hacia la tarjeta *Digium TDM400P*, y dependiendo de sus características se deberá configurar que las llamadas que arriben a un número determinado (que será el que los usuarios deberán marcar desde la PSTN) se redireccionen automáticamente al IVR, o en todo caso asignar una extensión de ingreso al sistema. Los usuarios podrán ingresar desde la PSTN o por intermedio de ésta, desde cualquier red de telefonía celular.

3.8.2 INSTALACIÓN EN UN AMBIENTE DE PRUEBAS

Simular un ambiente real como el de la *figura 3.1* requerirá tener ocho líneas telefónicas que se conecten a la PSTN, cuatro para ingreso al sistema y otras cuatro para poder llamar al mismo. Ante la poca factibilidad de poder construir un ambiente como el descrito debido al alto costo de tener ocho líneas telefónicas en un mismo lugar se opta por remplazar dos módulos FXS por dos FXO en la tarjeta *Digium TDM400P*, para tener una arquitectura como la que se muestra en la *figura 3.2*.

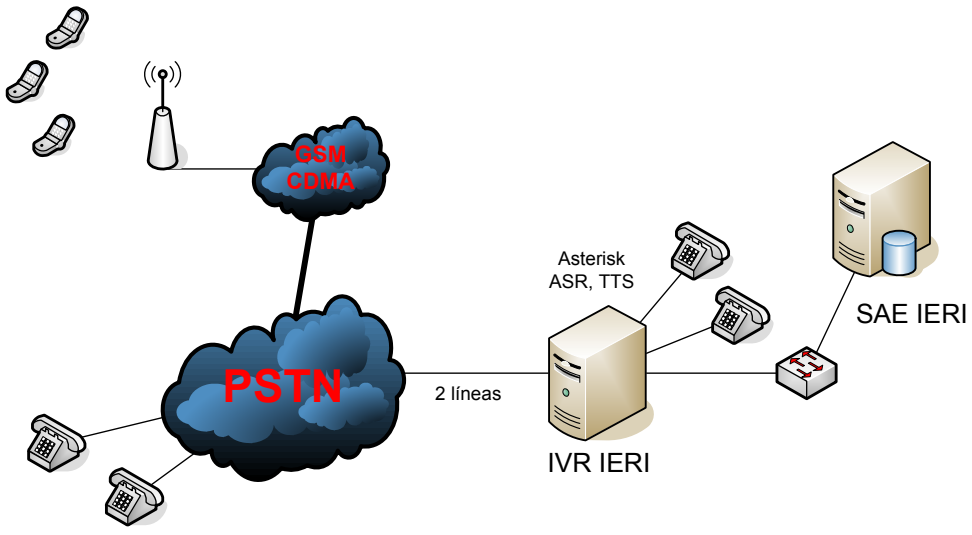

figura 3.2: Instalación del servidor en un ambiente de pruebas

Al final, se tiene un sistema similar al que se va a instalar en ambiente real, las dos líneas telefónicas para el acceso desde la PSTN permiten comprobar que el

sistema funcione correctamente cuando se conecte a esta red que contiene un alto índice de ruido. Los dos puertos FXS permiten conectar dos teléfonos analógicos para simular dos usuarios adicionales que en conjunto sumarían los cuatro que debe soportar el IVR.

Realizar pruebas directamente con la base de datos del SAE puede ser un tema un poco delicado, por cuanto es un sistema en donde se guarda información importante de los estudiantes y no debe ser comprometido de ninguna manera. Para solucionar este asunto se ha instalado un servidor adicional con sistema operativo Windows 2000, que contiene SQL Server 7 y una base de datos similar al SAE, al cual se direccionarán todas las consultas.

Un punto muy importante que no se ha mencionado hasta ahora es la manera como se va a trabajar con la base de datos PRAPRO que almacena la información correspondiente a certificados de prácticas preprofesionales y planes de proyectos de titulación. Como se trata de una base de datos que se almacena en el mismo DBMS al cual se tiene un acceso restringido se debe tener una aplicación adicional que permita realizar esta actividad y en este proyecto se ha optado por desarrollar un programa para Windows con las librerías de la MFC de Microsoft, hecha completamente en lenguaje C++, en Visual Studio 7.0. La aplicación es sencilla, fácilmente entendible para el usuario y está documentada en el *Anexo F*.

3.8.3 CONSIDERACIONES PARA REALIZAR LAS PRUEBAS

El buen funcionamiento de un sistema tiene que ver en gran parte a las modificaciones que se le haga, resultado de las constantes pruebas a las que se lo someta, las mismas que deben simular los peores casos de funcionamiento. Para este proyecto se consideran primordialmente dos situaciones, un alto nivel de tráfico y ruido excesivo en el canal telefónico (o también ruidos externos en el ambiente en el que se encuentra el usuario), lo que se traduciría en una sobrecarga al servidor y un pobre nivel de reconocimiento respectivamente.

Adicional a lo anterior se debe también comprobar el funcionamiento propio del sistema, que no exista un menú sin salida y que se proporcione la información correcta al usuario. En los siguientes párrafos se explicará la manera como realizar las pruebas para atacar todos estos puntos.

3.8.3.1 Alto nivel de tráfico

El sistema ha sido diseñado para soportar hasta cuatro usuarios simultáneos, por lo tanto se harán las pruebas ingresando desde la PSTN y desde los teléfonos que van conectados a los puertos FXS del servidor al mismo tiempo, tratando en lo posible de hacer simultáneos los procesos que sobrecargan el procesador, como son el reconocimiento de voz y la generación de los archivos de audio a partir de texto.

3.8.3.2 Ruido excesivo en el canal telefónico

El canal telefónico es ruidoso por naturaleza y este hecho depende de muchos factores externos que son difíciles de ser simulados, pero que al final afectan el reconocimiento de voz y degradan la calidad del IVR. Para tener un acercamiento a lo que podría ser un canal ruidoso se considerará ingresar al sistema llamando desde un teléfono celular en lugares en donde existan ruidos externos como puede ser el tráfico vehicular, el caminar y hablar de las personas, entre otros.

Adicional al ruido del canal telefónico es común que exista también eco en las líneas y que cause efectos similares. El eco suele ser eliminado ya sea por hardware o software, en efecto, Asterisk incorpora un cancelador de eco que se adapta al canal al inicio de cada llamada y que es transparente al usuario, de todos modos es importante verificar a pesar de que existiese eco en el auricular del llamante el reconocimiento de voz no tenga inconvenientes.

3.8.3.3 Recorrido por el IVR

Se comprobará todas las posibles interacciones del usuario con el sistema, ingresando a cada menú, saliendo de él y volviendo a entrar nuevamente a cada

uno de ellos. La idea es que un usuario pueda navegar fácilmente y utilizar todos los servicios sin tener que colgar y volver a llamar.

3.8.3.4 Nivel de logro de objetivos

En la *sección 1.1.1.4* se trató acerca de los parámetros que miden la calidad de un IVR, entre ellos el Nivel de Logro de Objetivos y el Nivel de Diálogo Virtual. Este último no es posible determinarlo en el IVR de este proyecto porque determina la relación entre el tiempo de atención con operadoras humanas y con el uso del IVR (operadoras virtuales), y no se tiene datos del primer parámetro porque nunca ha sido implementado.

El nivel de logro de objetivos se calculará como el porcentaje de llamadas que fueron atendidas exitosamente de un conjunto de usuarios que requieran consultar todos o algunos de los servicios.

3.9 RESULTADOS DE LAS PRUEBAS REALIZADAS

Para verificar el funcionamiento del IVR se ha tomando un conjunto de diez alumnos de la Carrera de Electrónica y Redes de Información de la Escuela Politécnica Nacional, se ha agregado la información correspondiente a cada uno, tanto sus datos como registros de notas, certificados de prácticas preprofesionales y planes de proyectos de titulación, en la base de datos experimental, como se muestra en la *figura 3.2*.

Se les ha permitido usar el sistema alternadamente desde líneas directas al servidor (puertos FXS), desde un teléfono fijo y también desde un teléfono celular (en exteriores en donde existe cierto nivel de ruido) de manera que cada uno haga consultas del siguiente tipo en cada llamada:

- Consulta de notas
- Consulta de certificados de prácticas preprofesionales
- Consulta de planes de proyectos de titulación

- Consulta de notas y certificados de prácticas preprofesionales
- Consulta de notas y planes de proyectos de titulación
- Consulta de certificados de prácticas preprofesionales y planes de proyectos de titulación
- Consulta de los tres servicios

El acceso ha sido simultáneo y constante por todos los usuarios de manera que cada uno haga dos llamadas por cada servicio de los señalados anteriormente. Los datos tabulados de estas pruebas se encuentran en el *Anexo G*.

Se ha dado una instrucción mínima al usuario para utilizar el sistema, solamente se le indicó que debe pronunciar su número único en combinaciones de hasta decenas y que si es de nueve dígitos debe empezar diciendo el año. También se dejó que él escogiera si consulta una nota, de un bimestre en especial o todas sus notas. A continuación se irá explicando los resultados de las pruebas que se hicieron.

3.9.1 ALTO NIVEL DE TRÁFICO

El dimensionamiento del servidor es correcto porque en ningún momento se presentaron retardos en el reconocimiento o en la conversión de texto a voz, tomando en cuenta que permanentemente estuvieron cuatro usuarios simultáneos. El desarrollo del diálogo guiado es normal y presenta una conversación sin ningún tipo de corte o demora, cada llamada que llega al servidor es inmediatamente atendida.

3.9.2 RUIDO EXCESIVO EN EL CANAL TELEFÓNICO

Algunas llamadas seguramente tuvieron mayor ruido en la línea que otras, sobretodo las que se hizo desde un teléfono celular en exteriores, esto fue una de las causas para que la detección del número único no se haya producido en el primer intento, en la *tabla 3.2* se observa el número de llamadas dependiendo de

cuantas veces el usuario tuvo que repetir su número único para ser detectado, o en otras palabras, cuántas veces tuvo que decir su número único hasta que el sistema lo reconozca correctamente. Por ejemplo, existieron 24 usuarios que al decir su número único a la primera vez el sistema erró pero al segundo intento lo detectó correctamente y luego pudo utilizar los servicios.

Intento	***# llamadas***
1	100
2	24
3	9
4	4
5	3
total	***140***

tabla 3.2: Intentos del usuario hasta detectar correctamente el número único

Al hacer las llamadas ningún usuario detectó eco, razón por la cual se puede asegurar que la cancelación por software que incorpora Asterisk está funcionando correctamente o que simplemente no hay eco en la línea.

3.9.3 RECORRIDO POR EL IVR

En ningún momento el IVR mostró menús sin salida, siempre existió la opción de volver al menú principal o regresar a la opción anterior. Esto es un tanto predecible debido a que el IVR contiene solamente un menú y un submenú, su estructura es simple y el recorrido es directamente a usar uno de los tres servicios.

3.9.4 NIVEL DE LOGRO DE OBJETIVOS

Según los resultados del *Anexo G* todos los usuarios fueron atendidos, pero en un ambiente real es poco deseable que el usuario deba intentar tantas veces hasta ingresar correctamente su número único. Considerando que con cuatro o más

intentos la llamada será no atendida y utilizando la *ecuación 1.1* y los datos de la *tabla 3.2*, tenemos:

$$NLO = \frac{100 + 24 + 9}{100 + 24 + 9 + 4 + 3} * 100\%$$

$$NLO = 95\%$$

El nivel de logro de objetivos es igual a 95%, es decir que de cada cien usuarios noventa y cinco logran utilizar correctamente el servicio, por lo tanto el IVR cumple con la recomendación de calidad mencionada en la *sección 1.1.1.4.1*.

3.10 ANÁLISIS DE COSTOS

Resultado de hacer un balance entre los costos del sistema y la calidad del mismo se optó por utilizar en algunos casos software gratuito de código abierto pero en otros se hizo necesario incorporar software propietario. En la *tabla 3.3* se muestra los costos referenciales del sistema en dólares norteamericanos.

cantidad	***Item***	***Precio unitario***	***precio total***
1	Servidor clon Intel Core 2 Duo 1.8 GHz, RAM 1 GB, HD 80 GB, DVD-ROM	$ 680	$ 680
1	Tarjeta Digium TDM 400P, con 4 puertos FXO	$ 350	$ 350
2	Puertos de reconocimiento de voz Lumenvox Lite	$ 245	$ 490
4	Puertos de conversión texto a voz Cepstral	$ 30	$ 120
1	Desarrollo de la aplicación	$ 800	$ 800
6	Horas técnicas para la instalación y configuración del servidor	$ 50	$ 300
4	Horas técnicas de mantenimiento semestral (dos por semestre)	$ 50	$ 200
		total	**$ 2940**

tabla 3.3: Costos de la solución en USD

Los precios del hardware son los del mercado local al momento de hacer este análisis de costos, las licencias de software en cambio, se compran directamente al proveedor mediante transacciones bancarias por Internet. No se toma en

cuenta el costo del sistema operativo *Windows* 2000 junto con *SQL Server* 7 y el servidor que los alberga porque son sistemas que existen en la carrera. Tampoco se ha tomado en cuenta impuestos por estas transacciones o el ancho de banda utilizado para descargar los paquetes de instalación.

El costo de desarrollo de la aplicación se ha calculado tomando en cuenta un valor estimado de dieciséis horas de trabajo, valorizadas igual que las horas de instalación y configuración. Se ha incluido el mantenimiento del sistema durante dos semestres, es decir, cada vez que se actualice la base de datos de estudiantes.

Capítulo IV

4 CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El reconocimiento de voz es un tema complejo debido a que los ordenadores aún no tienen la capacidad para percibir e interpretar el entorno con la facilidad que nosotros los humanos lo hacemos, pero la gran velocidad con la que pueden procesar la información realizando operaciones lógicas y matemáticas, ha logrado disminuir esta brecha.
- El conjunto de sonidos que emite una persona al hablar no solamente lleva la información del mensaje contenido en la frase pronunciada, sino también información del interlocutor (sexo, edad, región de procedencia, estado de ánimo, etc.) que influye negativamente en la tasa de aciertos de un sistema de reconocimiento de voz.
- La estandarización de lenguajes de etiquetado para tecnologías multimodales de acceso a Internet y aplicaciones en general, ha logrado que los productos de distintos fabricantes puedan interactuar, proporcionando a los programadores un panorama tecnológico más abierto y ya no estar sometidos a las habituales *cajas cerradas*, con las respectivas limitaciones de cada fabricante.
- Un sistema de respuesta de voz interactiva (IVR, *Interactive Voice Response*) como el que se ha diseñado e implementado en este proyecto no solamente brinda un buen servicio a los usuarios sino también da una mejor imagen a la empresa que lo adopte.
- Asterisk al ser un sistema telefónico completamente basado en software de código abierto, ha revolucionado la integración de la computación con la telefonía (CTI, *Computer Telephony Integration*) porque ofrece una interfaz flexible que permite crear sistemas de acceso telefónico que incorporan funcionalidades adicionales de la computación, como es el reconocimiento de voz, acceso a bases de datos e incluso acceso a contenido Web.
- La popularidad de aplicaciones de código abierto ha permitido a las empresas adoptar nuevos sistemas tecnológicos a costos relativamente bajos, pero en el caso que éstas dependan del idioma de las personas, los

resultados no son similares para todas las regiones del planeta, sobretodo en las que no se habla el inglés como lengua natal. El reconocimiento de voz y la conversión de texto a voz en el idioma español es un claro ejemplo en donde las aplicaciones de código abierto disponibles no ofrecen los resultados esperados y la tendencia es a utilizar software propietario. Lo anterior es debido a que las comunidades de desarrolladores de software generalmente provienen de países industrializados, en donde el idioma natal es en su mayor parte el inglés.

- La utilización de software de código abierto no implica necesariamente ninguna inversión, dependiendo del vendedor habrá ocasiones en que toque pagar por la aplicación. Además, en sistemas como el de este proyecto en donde la alta disponibilidad es primordial, es inevitable cubrir los gastos del hardware que brinde los resultados esperados.
- La invención de procesadores cada vez más rápidos y a menor costo para servidores de propósito general, ha logrado trasladar la mayor parte del procesamiento digital de señales, que generalmente realizaba el costoso hardware de telefonía, al software de las aplicaciones, disminuyendo significativamente el precio final de un sistema telefónico y logrando un mayor acceso por parte de las empresas de nivel medio.
- El costo por línea telefónica adicional para atender a un mayor grupo de usuarios mediante un sistema como el que se ha implementado continúa siendo relativamente alto y no siempre factible, debido a que las operadoras de telefonía deben proporcionar una nueva conexión física al servidor que también deberá tener un puerto de conexión disponible. Con la llegada de la telefonía IP estos costos podrían reducirse, por cuanto el mismo enlace físico podría transmitir un gran número de llamadas, sin embargo, en este caso se dependerá del ancho de banda disponible.
- La flexibilidad que se da al usuario para pronunciar su número único en combinaciones de unidades y decenas facilita la comunicación con el sistema, aunque todavía existe cierta limitación que depende de la capacidad de reconocimiento que tiene el motor. Con el avance de la tecnología se espera que cada vez se tenga una mejor tasa de aciertos y la interacción se vuelva más humana.

- Al utilizar los servicios del IVR desde un teléfono celular se logra un menor nivel de entendimiento con el sistema, sobretodo si se lo hace desde exteriores en donde se incorpora cierto ruido a la señal de voz. El ruido característico del canal telefónico también afecta la interacción entre el usuario y el sistema.
- El IVR que se ha implementado brinda eficientemente los servicios para los cuales fue diseñado, sin embargo no es el único punto de falla del sistema. Si el servidor que almacena la base de datos llegaría a tener problemas o no existiese comunicación entre ambos, de nada serviría la alta disponibilidad del hardware adquirido o del software desarrollado.

4.2 RECOMENDACIONES

- El servidor que se ha implementado mantiene activos solamente los servicios necesarios para hacer funcionar el IVR. Al no instalar ninguna interfaz gráfica se debe editar en modo texto los archivos de configuración, pero se logra aprovechar de mejor manera el hardware disponible; ésta es una forma de optimizar recursos.
- Antes de implementar un IVR como el de este proyecto se debe asegurar que todas las aplicaciones de software y librerías adicionales sean compatibles, por ejemplo, Asterisk ofrece interfaces para varios lenguajes de programación y algunas bases de datos pueden accederse solamente utilizando ciertos mecanismos; entonces se debe buscar un lenguaje que sea entendible por ambos.
- Actualizar el software utilizado es una recomendación general para cualquier sistema, pero en el caso de un IVR se debe dar prioridad a los módulos de reconocimiento de voz y de conversión de texto a voz, porque las nuevas versiones generalmente incorporan actualizaciones que incrementan el nivel de aciertos o la claridad de los audios producidos respectivamente.
- Siempre se debe desactivar los archivos de gramática que no se estén usando, las opciones que un usuario tenga en determinado instante deben

ser las mismas entre las cuales deba discriminar el motor de reconocimiento. Además, cuando se crea los archivos de gramática de la aplicación de debe incorporar todas las posibles respuestas del usuario, tomando en cuenta la manera de pronunciar cada palabra.

- La seguridad física del sistema es importante, en la medida que sea posible se debe instalar el servidor en un cuarto con ventilación, energía de respaldo, detector de humo, etc. Para sistemas en donde la alta disponibilidad sea un tema crítico se debe instalarlo en un cuarto de servidores, inclusive con hardware adicional que le brinde redundancia.
- El servidor debe tener conexión solamente con los recursos que forman parte del sistema, alejado de posibles amenazas de usuarios no autorizados. Para el IVR que se ha implementado solamente se necesita tener conexión con la base de datos y las cuatro líneas telefónicas de la PSTN.
- Aunque existen varios módulos de administración Web que se pueden agregar a Asterisk, no es recomendable usar ninguno de ellos porque están orientados a crear extensiones y planes de numeración tanto para telefonía tradicional como telefonía IP. Además de hacer muy difícil la programación de un IVR como el que se ha implementado pueden borrar accidentalmente dicha programación. Hay que respaldar los archivos de configuración periódicamente.
- Los archivos de audio que se utilicen en el sistema deben ser grabados de modo que sean claros y entendibles para el usuario, solamente con la información necesaria para que éste pueda moverse adecuadamente por el diálogo guiado.
- Antes de poner en producción un IVR como el de este proyecto se deben realizar suficientes pruebas hasta depurar todos los posibles errores que puedan darse, esto es, verificar que el menú no tenga lazos sin salida y que siempre se puedan ingresar a todos los servicios.
- Si se desea guardar registros de cada llamada hecha al sistema se puede usar el módulo adicional de Asterisk denominado *“asterisk-addons”*, que permite guardar dicha información en alguna base de datos relacional. En sistemas GNU/Linux es muy común utilizar *MySQL*.

REFERENCIAS BIBLIOGRÁFICAS

Libros

- MEGGELEN, Jim; SMITH, Jared; MADSEN, Leif, Asterisk, the Future Of Telephony, O'Reilly Media, United States of America, 2005.

- NEGUS, Christopher, Linux Bible, Wiley Publishing, Indianapolis, 2005.

- HUDSON, Andrew; HUDSON Paúl, Red Hat Fedora Core 6 Unleashed, Sams Publishing, Indeanapolis, 2006.

- BAIN, Tony; DAVIDSON Louis; DEWSON Robin; HAWKINS Chuck, SQL Server 2000 Stored Procedures Handbook, Apress, New York, 2003.

- SUNDERIC, Dejan, SQL Server 2000 Stored Procedured & XML Programming, Second Edition, McGraw Hill, California, 2003.

- SHEPHERD, George; KRUGLINSKI, David, Programación Avanzada con Microsoft Visual C++, McGraw Hill, España, 2003

- TANENBAUM, Andrew, Redes de Computadoras, 4ª Ed., Pearson Education, México, 2003.

Direcciones Electrónicas

- http://www.asterisk.org

- http://www.voip-info.org/wiki-Asterisk

- http://asterisk-java.org

- http://jtds.sourceforge.net
- http://www.lumenvox.com/
- http://www.cepstral.com/
- http://www.w3.org/TR/speech-grammar
- http://www.w3.org/TR/semantic-interpretation
- http://www.w3.org/TR/2000/NOTE-VoiceXML-20000505
- http://www.saltforum.org
- http://en.wikipedia.org/wiki/Natural_language_processing
- http://en.wikipedia.org/wiki/Speech_recognition
- http://java.sun.com/docs/books/tutorial/index.html

Anexos

ANEXO A: ARCHIVOS DE CONFIGURACIÓN

A.1 ZAPTEL

A.1.1 /etc/zaptel.conf

```
# Archivo de configuración para la tarjeta Digium TDM400P con cuatro
# puertos FXO

fxsks=1-4
loadzone=us
defaultzone=us
```

A.2 ASTERISK

A.2.1/etc/asterisk/extensions.conf

[general]

```
static=yes
writeprotect=yes
autofallthrough=yes
clearglobalvars=no
priorityjumping=no
```

[IERI-IVR]

```
exten => s,1,Goto(IERI-IVR-VOZ,a,1)
exten => s,2,HangUp()
```

[IERI-IVR-VOZ]

```
exten => a,1,Answer()
exten => a,n,Wait(1)
exten => a,n,SpeechCreate()
exten => a,n,GotoIf($["${ERROR}" != ""]?IERI-IVR-DTMF,s,3)
exten => a,n,Set(SPEECH_DTMF_MAXLEN=9)
exten => a,n,Set(CONTADOR=0)
exten => a,n,SpeechLoadGrammar
                  (NumeroUnico|/etc/asterisk/grammars/NumeroUnico.grxml)
exten => a,n,SpeechLoadGrammar
           (NumeroUnicoDTMF|/etc/asterisk/grammars/NumeroUnicoDTMF.grxml)
exten => a,n,PlayBack(IERI-IVR/prompts/hola)
exten => a,n(getnu),SpeechActivateGrammar(NumeroUnico)
exten => a,n,SpeechActivateGrammar(NumeroUnicoDTMF)
exten => a,n(start),SpeechStart()
```

```
exten => a,n,SpeechBackground(IERI-IVR/prompts/diga_digite_nu,5)
exten => a,n,Macro(rcvRespuestaNuUn,start)
exten => a,n,SpeechDeactivateGrammar(NumeroUnico)
exten => a,n,SpeechDeactivateGrammar(NumeroUnicoDTMF)
exten => a,n,GotoIf($["${TYPGRAM}" = "voz"]?b,1:c,1)


exten => b,1,PlayBack(IERI-IVR/prompts/su_nu_es)
exten => b,n,GotoIf($["${NUMUNI:0:2}" = "20"]?${n}+1:${n}+3)
exten => b,n,PlayBack(IERI-IVR/numeros/2000)
exten => b,n,PlayBack(IERI-IVR/numeros/${IF($["${NUMUNI:2:1}"
                                    = "0"]?${NUMUNI:3:1}:${NUMUNI:2:2})})
exten => b,n,Set(BEGIN=${IF($["${NUMUNI:0:2}" = "20"]?5:0)})
exten => b,n,Set(CURSOR=$[${BEGIN}+1])
exten => b,n,Set(LENGTH=${LEN(${NUMUNI})})
exten => b,n,While($[${BEGIN} < ${LENGTH}])
exten => b,n,Set(CURSOR=${IF($["${NUMUNI:$[${BEGIN}+1]:1}" = "_"]?1:2)})
exten => b,n,PlayBack(IERI-IVR/numeros/${NUMUNI:${BEGIN}:${CURSOR}})
exten => b,n,Set(BEGIN=$[${BEGIN}+${CURSOR}+1])
exten => b,n,Set(CURSOR=1)
exten => b,n,EndWhile()
exten => b,n,Macro(rcvRespuestaSiNo)
exten => b,n,GotoIf($["${RESPSINO}" = "si"]?c,1)
exten => b,n,Macro(verificarMaxIntentos)
exten => b,n,Goto(a,getnu)


exten => c,1,Agi(agi://localhost/sqlReqNombre?nu=${NUMUNI})
exten => c,n,Macro(verificarConexionBDD)
exten => c,n,GotoIf($[$["${NOMEST}" != ""] &
                                  $["${ESTMAT}" = "MATRICULADO"]]?${n}+3)
exten => c,n,PlayBack(IERI-IVR/prompts/nu_no_coincide)
exten => c,n,Goto(a,getnu)
exten => c,n,PlayBack(IERI-IVR/prompts/${IF($["${SEXOES:0:1}" =
                                          "M"]?bienvenido:bienvenida)})
exten => c,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                    -m text -o /var/lib/asterisk/sounds/IERI-IVR/nombres/
                                             ${NUMUNI}.wav "${NOMEST}")
exten => c,n,PlayBack(IERI-IVR/nombres/${NUMUNI})
exten => c,n(getan),SpeechLoadGrammar(MenuIVR|/etc/asterisk/grammars/
                                                        MenuIVR.grxml)
exten => c,n,SpeechActivateGrammar(MenuIVR)
exten => c,n(start),SpeechStart()
exten => c,n,SpeechBackground(IERI-IVR/prompts/menu_voz,5)
exten => c,n,Macro(rcvRespuestaMenu,start)
exten => c,n,SpeechDeactivateGrammar(MenuIVR)
exten => c,n,SpeechUnloadGrammar(MenuIVR)
exten => c,n,GotoIf($["${RESPUESTA}" = "notas"]?d,1)
exten => c,n,GotoIf($["${RESPUESTA}" = "practicas"]?e,1)
exten => c,n,GotoIf($["${RESPUESTA}" = "proyectos"]?f,1)


exten => d,1,Agi(agi://localhost/sqlReqNotas?nu=${NUMUNI})
exten => d,n,Macro(verificarConexionBDD)
exten => d,n(getan),SpeechLoadGrammar(MateriasAlumno|/etc/asterisk/
                                      grammars/materias/${NUMUNI}.grxml)
exten => d,n,SpeechActivateGrammar(MateriasAlumno)
exten => d,n(start),SpeechStart()
exten => d,n,SpeechBackground(IERI-IVR/prompts/menu_voz_notas,5)
exten => d,n,Macro(rcvRespuestaMenu,start)
```

```
exten => d,n,Set(RESPMAT=${RESPUESTA})
exten => d,n,SpeechDeactivateGrammar(MateriasAlumno)
exten => d,n,SpeechUnloadGrammar(MateriasAlumno)

exten => d,n(info),MacroIf($["${RESPMAT}" =
                                    "1_bimestre"]?decirNotasBimestre,1)
exten => d,n,MacroIf($["${RESPMAT}" = "2_bimestre"]?decirNotasBimestre,2)
exten => d,n,MacroIf($["${RESPMAT}" = "supletorio"]?decirNotasBimestre,3)
exten => d,n,MacroIf($["${RESPMAT}" = "todasnotas"]?decirNotasTodas)
exten => d,n,MacroIf($["${LEN(${RESPMAT})}" =
                                     "6"]?decirNotasMateria,${RESPMAT})
exten => d,n,Macro(menuRepInfoVoz,info)


exten => e,1,Agi(agi://localhost/sqlReqPracticas?nu=${NUMUNI})
exten => e,n,Macro(verificarConexionBDD)
exten => e,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                           -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                    practicas/${NUMUNI}.wav "${ESTCER}")
exten => e,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                           -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                 practicas/emp${NUMUNI}.wav "${NOMEMP}")
exten => e,n(prac),GotoIf($["${ESTCER}" = ""]?${n}+1:${n}+3)
exten => e,n,PlayBack(IERI-IVR/prompts/ninguna_practica)
exten => e,n,Goto(rep)
exten => e,n,PlayBack(IERI-IVR/prompts/rev_practicas)
exten => e,n,PlayBack(IERI-IVR/practicas/emp${NUMUNI})
exten => e,n,PlayBack(IERI-IVR/prompts/rev_estado)
exten => e,n,PlayBack(IERI-IVR/practicas/${NUMUNI})
exten => e,n(rep),Macro(menuRepInfoVoz,prac)


exten => f,1,Agi(agi://localhost/sqlReqProyectos?nu=${NUMUNI})
exten => f,n,Macro(verificarConexionBDD)
exten => f,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                           -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                    proyectos/${NUMUNI}.wav "${ESTPLA}")
exten => f,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                           -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                 proyectos/tem${NUMUNI}.wav "${TEMPLA}")
exten => f,n(prac),GotoIf($["${ESTPLA}" = ""]?${n}+1:${n}+3)
exten => f,n,PlayBack(IERI-IVR/prompts/ningun_proyecto)
exten => f,n,Goto(rep)
exten => f,n,PlayBack(IERI-IVR/prompts/rev_proyectos)
exten => f,n,PlayBack(IERI-IVR/proyectos/tem${NUMUNI})
exten => f,n,PlayBack(IERI-IVR/prompts/rev_estado)
exten => f,n,PlayBack(IERI-IVR/proyectos/${NUMUNI})
exten => f,n(rep),Macro(menuRepInfoVoz,prac)


exten => h,1,SpeechUnloadGrammar(NumeroUnicoDTMF)
exten => h,n,SpeechUnloadGrammar(NumeroUnico)
exten => h,n,SpeechDestroy()
exten => h,n,HangUp()


[IERI-IVR-DTMF]

exten => s,1,Answer()
exten => s,n,Wait(1)
```

```
exten => s,n,Set(TIMEOUT(digit)=3)
exten => s,n,Set(TIMEOUT(response)=6)
exten => s,n,Playback(IERI-IVR/prompts/hola)

exten => s,n(getnu),Read(NUMUNI,IERI-IVR/prompts/digite_nu,9,skip,3)
exten => s,n,GotoIf($["${NUMUNI}" = ""]?z,1)
exten => s,n,GotoIf($[$["${LEN(${NUMUNI})}" = "7"] |
                        $["${LEN(${NUMUNI})}" = "9"]]?${n}+1:${n}+4)
exten => s,n,Agi(agi://localhost/sqlReqNombre?nu=${NUMUNI})
exten => s,n,Macro(verificarConexionBDD)
exten => s,n,GotoIf($[$["${NOMEST}" = ""] | $["${ESTMAT}" !=
                                        "MATRICULADO"]]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/nu_no_valido)
exten => s,n,Goto(getnu)
exten => s,n,PlayBack(IERI-IVR/prompts/${IF($["${SEXOES:0:1}" = "M"]?
                                            bienvenido:bienvenida)})
exten => s,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                        -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                nombres/${NUMUNI}.wav "${NOMEST}")
exten => s,n,PlayBack(IERI-IVR/nombres/${NUMUNI})
exten => s,n(mdtmf),Background(IERI-IVR/prompts/menu_dtmf)
exten => s,n,WaitExten()


exten => 1,1,Goto(IERI-IVR-DTMF-notas,s,1)


exten => 2,1,Agi(agi://localhost/sqlReqPracticas?nu=${NUMUNI})
exten => 2,n,Macro(verificarConexionBDD)
exten => 2,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                        -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                practicas/${NUMUNI}.wav "${ESTCER}")
exten => 2,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                        -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                              practicas/emp${NUMUNI}.wav "${NOMEMP}")
exten => 2,n(prac),GotoIf($["${ESTCER}" = ""]?${n}+1:${n}+3)
exten => 2,n,PlayBack(IERI-IVR/prompts/ninguna_practica)
exten => 2,n,Goto(repra)
exten => 2,n,PlayBack(IERI-IVR/prompts/rev_practicas)
exten => 2,n,PlayBack(IERI-IVR/practicas/emp${NUMUNI})
exten => 2,n,PlayBack(IERI-IVR/prompts/rev_estado)
exten => 2,n,PlayBack(IERI-IVR/practicas/${NUMUNI})
exten => 2,n(repra),Macro(menuRepInfoDTMF,prac)


exten => 3,1,Agi(agi://localhost/sqlReqProyectos?nu=${NUMUNI})
exten => 3,n,Macro(verificarConexionBDD)
exten => 3,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                        -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                                proyectos/${NUMUNI}.wav "${ESTPLA}")
exten => 3,n,System(/usr/local/bin/swift -p audio/sampling-rate=8000
                        -m text -o /var/lib/asterisk/sounds/IERI-IVR/
                              proyectos/tem${NUMUNI}.wav "${TEMPLA}")
exten => 3,n(proy),GotoIf($["${ESTPLA}" = ""]?${n}+1:${n}+3)
exten => 3,n,PlayBack(IERI-IVR/prompts/ningun_proyecto)
exten => 3,n,Goto(repro)
exten => 3,n,PlayBack(IERI-IVR/prompts/rev_proyectos)
exten => 3,n,PlayBack(IERI-IVR/proyectos/tem${NUMUNI})
exten => 3,n,PlayBack(IERI-IVR/prompts/rev_estado)
exten => 3,n,PlayBack(IERI-IVR/proyectos/${NUMUNI})
```

```
exten => 3,n(repro),Macro(menuRepInfoDTMF,proy)


exten => i,1,Playback(IERI-IVR/prompts/op_no_valida)
exten => i,n,Goto(s,getnu)


exten => t,1,Playback(IERI-IVR/prompts/op_no_digitada)
exten => t,n,Goto(s,getnu)


[IERI-IVR-DTMF-notas]

exten => s,1,Agi(agi://localhost/sqlReqNotas?nu=${NUMUNI})
exten => s,n,Macro(verificarConexionBDD)

exten => s,n(mnotas),Background(IERI-IVR/prompts/menu_dtmf_notas)
exten => s,n,WaitExten()

exten => _[123],1,Macro(decirNotasBimestre,${EXTEN})
exten => _[123],n,Macro(menuRepInfoDTMF)

exten => 4,1,Macro(decirNotasTodas)
exten => 4,n,Macro(menuRepInfoDTMF)

exten => i,1,Playback(IERI-IVR/prompts/op_no_valida)
exten => i,n,Goto(s,mnotas)

exten => t,1,Playback(IERI-IVR/prompts/op_no_digitada)
exten => t,n,Goto(s,mnotas)


; Macro para detectar el número unico que ingresa el usuario (ivr voz)
[macro-rcvRespuestaNuUn]
; ${ARG1} - etiqueta de la prioridad desde donde empieza el dialogo

exten => s,1,NoOp( >> Speech: spoke = ${SPEECH(spoke)}  results =
                ${SPEECH(results)}  uterance = ${SPEECH_TEXT(0)}  score =
                                                    ${SPEECH_SCORE(0)})
exten => s,n,Set(NUMUNI=${SPEECH_TEXT(0)})
exten => s,n,GotoIf($[$["${NUMUNI}" = "ayuda"] & $[${SPEECH_SCORE(0)} >
                                                    600]]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/ayuda)
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})
exten => s,n,GotoIf($["${SPEECH_GRAMMAR(0)}" = "dtmf"]?dtmf:voz)
exten => s,n(dtmf),Set(TYPGRAM=dtmf)
exten => s,n,GotoIf($[$["${LEN(${NUMUNI})}" = "7"] |
       $[$["${LEN(${NUMUNI})}" = "9"] & $["${NUMUNI:0:2}" = "20"]]]?end)
exten => s,n,PlayBack(IERI-IVR/prompts/nu_no_valido)
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})
exten => s,n(voz),Set(TYPGRAM=voz)
exten => s,n,GotoIf($["${SPEECH(spoke)}" = "0"]?${n}+1:${n}+4)
exten => s,n,PlayBack(IERI-IVR/prompts/no_escuche)
exten => s,n,Macro(verificarMaxIntentos)
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})

exten => s,n,GotoIf($[$["${SPEECH(results)}" = "0"] |
                                $[${SPEECH_SCORE(0)} < 600]]?${n}+1:end)
exten => s,n,PlayBack(IERI-IVR/prompts/no_entendi)
exten => s,n,Macro(verificarMaxIntentos)
```

```
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})

exten => s,n(end),MacroExit()


; Macro para detectar la respuesta del usuario en el menu principal
; (ivr voz)
[macro-rcvRespuestaMenu]
; ${ARG1} - etiqueta de la prioridad desde donde empieza el dialogo

exten => s,1,NoOp( >> Speech: spoke = ${SPEECH(spoke)}  results =
                ${SPEECH(results)}  uterance = ${SPEECH_TEXT(0)}  score =
                                                        ${SPEECH_SCORE(0)})
exten => s,n,Set(RESPUESTA=${SPEECH_TEXT(0)})
exten => s,n,GotoIf($["${SPEECH(spoke)}" = "0"]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/no_escuche)
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})
exten => s,n,GotoIf($[$["${SPEECH(results)}" = "0"] |
                              $[${SPEECH_SCORE(0)} < 400]]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/no_entendi)
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})
exten => s,n,GotoIf($[$[${SPEECH_SCORE(0)} >= 400] &
                              $[${SPEECH_SCORE(0)} < 600]]?${n}+1:${n}+4)
exten => s,n,PlayBack(IERI-IVR/prompts/su_respuesta)
exten => s,n,SpeechBackground(IERI-IVR/respuestas/${RESPUESTA},1)
exten => s,n,Macro(rcvRespuestaSiNo)

exten => s,n,GotoIf($["${RESPSINO}" = "no"]?${n}+1:${n}+3)
exten => s,n,Set(RESPSINO=)
exten => s,n,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})
exten => s,n(end),MacroExit()


; Macro para detectar la respuesta del usuario cuando se pida
; una confirmacion (ivr voz)
[macro-rcvRespuestaSiNo]
; Sin argumentos

exten => s,1,SpeechLoadGrammar(SiNo|/etc/asterisk/grammars/SiNo.grxml)
exten => s,n,SpeechActivateGrammar(SiNo)  ; archivo de gramatica con dos
posibilidades, si o no
exten => s,n(start),SpeechStart()
exten => s,n,SpeechBackground(IERI-IVR/prompts/diga_si_no,5)      ;  diga
si o no
exten => s,n,NoOp( >> Speech: spoke = ${SPEECH(spoke)}  results =
                  ${SPEECH(results)}  uterance = ${SPEECH_TEXT(0)}
                  score = ${SPEECH_SCORE(0)})
exten => s,n,Set(RESPSINO=${SPEECH_TEXT(0)})

exten => s,n,GotoIf($[${SPEECH(spoke)} = 0]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/no_escuche)
exten => s,n,Goto(start)

exten => s,n,GotoIf($[$["${SPEECH(results)}" = "0"] |
                              $[${SPEECH_SCORE(0)} < 600]]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/no_entendi)
exten => s,n,Goto(start)

exten => s,n,SpeechDeactivateGrammar(SiNo)
exten => s,n,SpeechUnloadGrammar(SiNo)
```

```
exten => s,n(end),MacroExit()


; Macro para decir todas las notas al usuario (ivr voz y dtmf)
[macro-decirNotasTodas]
; Sin argumentos

exten => s,1,GotoIf($["${TOTCAL}" = "0"]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/ninguna_nota)
exten => s,n,Goto(end)
exten => s,n,PlayBack(IERI-IVR/prompts/notas_todas)
exten => s,n,Set(CONTMAT=1)     ; recorre cada materia
exten => s,n,While($["${CONTMAT}" <= "${NUMMAT}"])
exten => s,n,Set(NUMCAL[${CONTMAT}]=${NUMCAL[${CONTMAT}]:0:1})
exten => s,n,Set(CONTCAL=1)
exten => s,n,GotoIf($[$["${NUMCAL[${CONTMAT}]}" = "0"] |
                                  $["${NUMCAL[${CONTMAT}]}" = ""]]?otra)
exten => s,n,PlayBack(IERI-IVR/materias/${CODMAT[${CONTMAT}]})
exten => s,n(nota),Set(CALIF=${CALIF${CONTCAL}[${CONTMAT}]})
exten => s,n,GotoIf($["${CONTCAL}" < "3"]?${n}+2)
exten => s,n,PlayBack(IERI-IVR/prompts/supletorio)
exten => s,n,PlayBack(IERI-IVR/numeros/${IF($["${CALIF:1:1}" =
                                       "."]?${CALIF:0:1}:${CALIF:0:2})})
exten => s,n,PlayBack(IERI-IVR/numeros/punto)
exten => s,n,PlayBack(IERI-IVR/numeros/${IF($["${CALIF:1:1}" =
                                       "."]?${CALIF:2:1}:${CALIF:3:1})})
exten => s,n,Wait(1)
exten => s,n,Set(CONTCAL=$[${CONTCAL}+1])
exten => s,n,GotoIf($["${CONTCAL}" > "3"]?${n}+2)
exten => s,n,GotoIf($["${CALIF${CONTCAL}[${CONTMAT}]}" = ""]?${n}+1:nota)
exten => s,n(otra),Set(CONTMAT=$[${CONTMAT}+1])
exten => s,n,EndWhile()
exten => s,n(end),MacroExit()


; Macro para decir las notas al usuario, por bimestre (ivr voz y dtmf)
[macro-decirNotasBimestre]
; ${ARG1} - Bimestre del que se desea consultar las notas,
; 3 para supletorio

exten => s,1,GotoIf($["${TOT${ARG1}BI}" = "0"]?${n}+1:${n}+3)
exten => s,n,PlayBack(IERI-IVR/prompts/ninguna_nota)
exten => s,n,Goto(end)
exten => s,n,PlayBack(IERI-IVR/prompts/notas_${ARG1}bim)
exten => s,n,Set(CONTMAT=1)     ; recorre cada materia
exten => s,n,While($["${CONTMAT}" <= "${NUMMAT}"])
exten => s,n,Set(CALIF=${CALIF${ARG1}[${CONTMAT}]})
exten => s,n,GotoIf($["${CALIF}" = ""]?otra)
exten => s,n,PlayBack(IERI-IVR/materias/${CODMAT[${CONTMAT}]})
exten => s,n,PlayBack(IERI-IVR/numeros/${IF($["${CALIF:1:1}" =
                                       "."]?${CALIF:0:1}:${CALIF:0:2})})
exten => s,n,PlayBack(IERI-IVR/numeros/punto)
exten => s,n,PlayBack(IERI-IVR/numeros/${IF($["${CALIF:1:1}" =
                                       "."]?${CALIF:2:1}:${CALIF:3:1})})
exten => s,n,Wait(1)
exten => s,n(otra),Set(CONTMAT=$[${CONTMAT}+1])
exten => s,n,EndWhile()
exten => s,n(end),MacroExit()
```

```
; Macro para decir las notas al usuario, por materia (ivr voz)
[macro-decirNotasMateria]
; ${ARG1} - Codigo de la materia a consultar

exten => s,1,Set(CONTMAT=1)
exten => s,n,While($["${CODMAT[${CONTMAT}]}" != "${ARG1}"])
exten => s,n,Set(CONTMAT=$[${CONTMAT}+1])
exten => s,n,EndWhile()

exten => s,n,PlayBack(IERI-IVR/materias/${ARG1})
exten => s,n,Set(CONTCAL=1)
exten => s,n,GotoIf($["${CONTCAL}" > "${NUMCAL[${CONTMAT}]}"]?${n}+8)
exten => s,n,Set(CALIF=${CALIF${CONTCAL}[${CONTMAT}]})
exten => s,n,PlayBack(IERI-IVR/numeros/${IF($["${CALIF:1:1}" =
                                        "."]?${CALIF:0:1}:${CALIF:0:2})})
exten => s,n,PlayBack(IERI-IVR/numeros/punto)
exten => s,n,PlayBack(IERI-IVR/numeros/${IF($["${CALIF:1:1}" =
                                        "."]?${CALIF:2:1}:${CALIF:3:1})})
exten => s,n,Wait(1)
exten => s,n,Set(CONTCAL=$[${CONTCAL}+1]) ; siguiente calificacion
exten => s,n,Goto(${n}-7)
exten => s,n(end),MacroExit()


; Macro para consultar al usuario si desea que se le repita
; la informacion proporcionada (ivr voz)
[macro-menuRepInfoVoz]
; ${ARG1} - etiqueta de la prioridad desde donde repetir la informacion

exten => s,1,SpeechLoadGrammar(MenuRepetir|/etc/asterisk/grammars/
                 MenuRepetir${IF($["${MACRO_EXTEN}" = "d"]?Notas)}.grxml)
exten => s,n,SpeechActivateGrammar(MenuRepetir)
exten => s,n(start),SpeechStart()
exten => s,n,SpeechBackground(IERI-IVR/prompts/
                   menu_voz_rep${IF($["${MACRO_EXTEN}" = "d"]?_notas)},5)
exten => s,n,Macro(rcvRespuestaMenu,start)
exten => s,n,SpeechDeactivateGrammar(MenuRepetir)
exten => s,n,SpeechUnloadGrammar(MenuRepetir)
exten => s,n,GotoIf($["${RESPUESTA}" = "repetir"]?
                                  ${MACRO_CONTEXT},${MACRO_EXTEN},${ARG1})
exten => s,n,GotoIf($["${RESPUESTA}" = "volver"]?
                                           ${MACRO_CONTEXT},d,getan)
exten => s,n,GotoIf($["${RESPUESTA}" = "principal"]?
                                           ${MACRO_CONTEXT},c,getan)
exten => s,n(end),MacroExit()


; Macro para consultar al usuario si desea que se le repita
; la informacion proporcionada (ivr dtmf)
[macro-menuRepInfoDTMF]
; ${ARG1} - etiqueta de la prioridad desde donde repetir la informacion

exten => s,1,Background(IERI-IVR/prompts/
         menu_dtmf_rep${IF($["${MACRO_CONTEXT}" = "IERI-IVR-DTMF-notas"]?
                                      _notas)},m,,macro-menuRepInfoDTMF)
exten => s,n,Goto(t,1)
exten => 1,1,Goto(${MACRO_CONTEXT},${MACRO_EXTEN},
                     ${IF($["${MACRO_CONTEXT}" = "IERI-IVR-DTMF-notas"]?
                                       $[${MACRO_PRIORITY}-1]:${ARG1})})
exten => 2,1,GotoIf($["${MACRO_CONTEXT}" = "IERI-IVR-DTMF-notas"]?
```

```
                      IERI-IVR-DTMF-notas,s,mnotas:IERI-IVR-DTMF,s,mdtmf)
exten => 3,1,GotoIf($["${MACRO_CONTEXT}" = "IERI-IVR-DTMF-notas"]?
                                              IERI-IVR-DTMF,s,mdtmf:i,1)
exten => _[4-90],1,Goto(i,1)

exten => i,1,Playback(IERI-IVR/prompts/op_no_valida)
exten => i,n,Goto(s,1)  ; Cuando se ha digitado una opcion no valida

exten => t,1,Playback(IERI-IVR/prompts/op_no_digitada)
exten => t,n,Goto(s,1)  ; Cuando no se ha digitado ninguna opcion


; Macro para verificar un numero maximo de intentos al ingresar el
; numero unico (ivr voz)
[macro-verificarMaxIntentos]
; Sin argumentos

exten => s,1,Set(CONTADOR=$[${CONTADOR}+1])
exten => s,n,GotoIf($["${CONTADOR}" < "3"]?end)
exten => s,n,PlayBack(IERI-IVR/prompts/ayuda)
exten => s,n,Set(CONTADOR=0)
exten => s,n(end),MacroExit()


; Macro para verificar si las consultas a la bdd son exitosas
[macro-verificarConexionBDD]
; Sin parametros

exten => s,1,GotoIf($["${ESTBDD}" != "error"]?end)
exten => s,n,PlayBack(IERI-IVR/prompts/fallo_conexion)
exten => s,n,Playback(IERI-IVR/prompts/despedida)
exten => s,n,Wait(1)
exten => s,n,HangUp()
exten => s,n(end),MacroExit()
```

A.2.2 /etc/asterisk/modules.conf

```
; Archivo de configuracion de los módulos que se cargan al
; arrancar Asterisk

[modules]

Autoload = yes
load   => res_speech.so
noload => pbx_gtkconsole.so
noload => pbx_kdeconsole.so
noload => app_intercom.so
noload => chan_modem.so
noload => chan_modem_aopen.so
noload => chan_modem_bestdata.so
noload => chan_modem_i4l.so
load   => res_musiconhold.so
noload => chan_alsa.so
```

A.2.3 /etc/asterisk/zapata.conf

```
; Archivo de configuracion para la interfaz con Zaptel

[channels]

language=es
context=IERI-IVR
signalling=fxs_ks
usecallerid=yes
hidecallerid=no
callwaiting=no
echocancel=yes
rxgain=0.0
txgain=0.0
group=1
busydetect=yes
busycount=2
channel => 1-4
```

A.3 LUMENVOX

A.3.1 /etc/asterisk/lumenvox.conf

```
; Configuracion del motor de reconocimiento

[general]

servers=127.0.0.1
save_sound_files=no


[default]

vad_bargein_level=40
vad_eos_delay=2000
vad_noise_floor=200
vad_wind_back=255
vad_burst_threshold=100
end_of_speech_timeout=8000
use_oov_filter=no
```

A.3.2 /opt/lumenvox/engine_7.5/bin/license_client.conf

```
# Direccion del servidor de licencias

ip_address=127.0.0.1
port_number=7569
```

A.4 GNU/LINUX

A.4.1 /usr/sbin/safe_asterisk

```
#!/bin/sh
CLIARGS="$*"                # Grab any args passed to safe_asterisk
TTY=9                       # TTY (if you want one) for Asterisk to run on
CONSOLE=yes                 # Whether or not you want a console
#NOTIFY=ben@alkaloid.net    # Who to notify about crashes
MACHINE=`hostname`          # To specify which machine has crashed when
getting the mail
DUMPDROP=/tmp
ASTSBINDIR=/usr/sbin
ASTPIDFILE=/var/run/asterisk.pid

#
# Don't fork when running "safely"
#
ASTARGS=""
if [ "$TTY" != "" ]; then
        if [ -c /dev/tty${TTY} ]; then
                TTY=tty${TTY}
        elif [ -c /dev/vc/${TTY} ]; then
                TTY=vc/${TTY}
        else
                echo "Cannot find your TTY (${TTY})" >&2
                exit 1
        fi
        ASTARGS="${ASTARGS} -vvvg"
        if [ "$CONSOLE" != "no" ]; then
                ASTARGS="${ASTARGS} -c"
        fi
fi
if [ ! -w ${DUMPDROP} ]; then
        echo "Cannot write to ${DUMPDROP}" >&2
        exit 1
fi


#
# Let Asterisk dump core
#
ulimit -c unlimited

#
# Don't die if stdout/stderr can't be written to
#
trap '' SIGPIPE

#
# Run scripts to set any environment variables or do any other
# system-specific setup needed
#

if [ -d /etc/asterisk/startup.d ]; then
        for script in /etc/asterisk/startup.d/*.sh; do
                if [ -x ${script} ]; then
```

```
                        source ${script}
                fi
        done
fi

run_asterisk()
{
        while :; do

                if [ "$TTY" != "" ]; then
                        cd /tmp
                        stty sane < /dev/${TTY}

                        export LV_LICENSE="/opt/lumenvox/\
                                                licenseserver_7.5/bin"
                        export LVBIN=/opt/lumenvox/engine_7.5/bin
                        export LVLIB=/opt/lumenvox/engine_7.5/lib
                        export LVINCLUDE=/opt/lumenvox/engine_7.5/include
                        export LVLANG=/opt/lumenvox/engine_7.5/Lang
                        export LVRESPONSE=/opt/lumenvox/engine_7.5/Lang
                        export LD_LIBRARY_PATH=$LVLIB:$LD_LIBRARY_PATH
                        export LD_RUN_PATH=$LVLIB:$LD_RUN_PATH

                        ${ASTSBINDIR}/asterisk -f ${CLIARGS} ${ASTARGS}
                                              >& /dev/${TTY} < /dev/${TTY}
                else
                        cd /tmp

                        export LV_LICENSE="/opt/lumenvox/\
                                                licenseserver_7.5/bin"
                        export LVBIN=/opt/lumenvox/engine_7.5/bin
                        export LVLIB=/opt/lumenvox/engine_7.5/lib
                        export LVINCLUDE=/opt/lumenvox/engine_7.5/include
                        export LVLANG=/opt/lumenvox/engine_7.5/Lang
                        export LVRESPONSE=/opt/lumenvox/engine_7.5/Lang
                        export LD_LIBRARY_PATH=$LVLIB:$LD_LIBRARY_PATH
                        export LD_RUN_PATH=$LVLIB:$LD_RUN_PATH

                        ${ASTSBINDIR}/asterisk -f ${CLIARGS} ${ASTARGS}
                fi
                EXITSTATUS=$?
           echo "Asterisk ended with exit status $EXITSTATUS"
                if [ "$EXITSTATUS" = "0" ]; then
                        # Properly shutdown....
                        echo "Asterisk shutdown normally."
                        exit 0
                elif [ $EXITSTATUS -gt 128 ]; then
                        let EXITSIGNAL=EXITSTATUS-128
                        echo "Asterisk exited on signal $EXITSIGNAL."
                        if [ "$NOTIFY" != "" ]; then
                                echo "Asterisk on $MACHINE exited on \
                                signal $EXITSIGNAL.  Might want to   \
                                                  take a peek." | \
                                mail -s "Asterisk Died" $NOTIFY
                        fi

                        PID=`cat ${ASTPIDFILE}`
                        if [ -f /tmp/core.${PID} ]; then
                                mv /tmp/core.${PID} ${DUMPDROP}/core\
                                           .`hostname`-`date -Iseconds` &
```

```
                        elif [ -f /tmp/core ]; then
                                mv /tmp/core ${DUMPDROP}/core\
                                        .`hostname`-`date -Iseconds` &
                        fi
                else
                        if [ "${EXITSTATUS}" = "0" ]; then
                                echo "Asterisk ended normally. \
                                                        Aborting."
                                exit 0
                        else
                                echo "Asterisk died with code \
                                                $EXITSTATUS."

                                PID=`cat ${ASTPIDFILE}`
                                if [ -f /tmp/core.${PID} ]; then
                                        mv /tmp/core.${PID} ${DUMPDROP}\
                                      /core.`hostname`-`date -Iseconds` &
                                elif [ -f /tmp/core ]; then
                                        mv /tmp/core ${DUMPDROP}\
                                      /core.`hostname`-`date -Iseconds` &
                                fi
                        fi
                fi
                echo "Automatically restarting Asterisk."
                sleep 4
        done
}

run_asterisk &
```

A.4.2 /etc/rc.local

```
# Este script es ejecutado después de todos los scripts de inicio

touch /var/lock/subsys/local
/usr/src/jdk1.6.0/bin/java -cp /usr/local/agi-java/jtds-1.2.1.jar:
           /usr/local/agi-java:/usr/local/agi-java/asterisk-java-0.3.jar:
                           org.asteriskjava.fastagi.DefaultAgiServer &
```

ANEXO B: SENTENCIAS DE COMANDOS SQL

B.1 CREACIÓN DE BASE DE DATOS “PRAPRO” PARA MANEJAR LOS SERVICIOS ADICIONALES

```
USE master
GO

IF EXISTS (SELECT * FROM sysdatabases WHERE NAME = 'PRAPRO')
      DROP DATABASE PRAPRO

CREATE DATABASE PRAPRO
GO

USE PRAPRO
GO


CREATE TABLE ESTUDIANTES (
      codest varchar(9) NOT NULL,
      nomest varchar(50) NOT NULL,
      totcred int NULL,
      mattes bit NULL
      constraint pk_estudiantes primary key(codest)
)
GO


CREATE TABLE PRACTICAS (
      pracid int IDENTITY(1, 1) NOT NULL,
      codest varchar(9) NOT NULL,
      nomemp varchar(30) NOT NULL,
      numhor int NOT NULL,
      fecrec smalldatetime NOT NULL,
      fecrev smalldatetime NOT NULL,
      estcer varchar(2000) NOT NULL
      constraint pk_practicas primary key(pracid),
      constraint fk_estpra foreign key(codest) references ESTUDIANTES
)
GO


CREATE TABLE PROYECTOS (
      proyid int IDENTITY(1, 1) NOT NULL,
      codest varchar (9) NOT NULL,
      templa varchar (300) NOT NULL,
      nomdir varchar (30) NOT NULL,
      fecrec smalldatetime NOT NULL,
      fecrev smalldatetime NOT NULL,
      estpla varchar(2000) NOT NULL
      constraint pk_proyectos primary key(proyid),
      constraint fk_estpro foreign key(codest) references ESTUDIANTES
)
GO
```

```
-- Desencadenadores para verificar el correcto ingreso, actualización
-- y eliminación de registros de las tablas


CREATE TRIGGER tg_insEstudiante
ON ESTUDIANTES
FOR INSERT, UPDATE
AS
      IF @@rowcount=0
            ROLLBACK TRANSACTION
      ELSE
            PRINT 'Inserción de estudiante, correcto'
GO


CREATE TRIGGER tg_insPractica
ON PRACTICAS
FOR INSERT, UPDATE
AS
      IF @@rowcount=0
            ROLLBACK TRANSACTION
      ELSE
            PRINT 'Inserción de certificado de prácticas
                  preprofesionales, correcto'
GO


CREATE TRIGGER tg_insProyecto
ON PROYECTOS
FOR INSERT, UPDATE
AS
      IF @@rowcount=0
            ROLLBACK TRANSACTION
      ELSE
            PRINT 'Inserción de plan de proyecto de titulación,
                  correcto'
GO


CREATE TRIGGER tg_delEstudiante
ON ESTUDIANTES
FOR DELETE
AS
      IF @@rowcount=0
            ROLLBACK TRANSACTION
      ELSE
            PRINT 'Eliminación de estudiante, correcto'
GO


CREATE TRIGGER tg_delPractica
ON PRACTICAS
FOR DELETE
AS
      IF @@rowcount=0
            ROLLBACK TRANSACTION
      ELSE
            PRINT 'Eliminación de certificado de prácticas
                  preprofesionales, correcto'
GO
```

```
CREATE TRIGGER tg_delProyecto
ON PROYECTOS
FOR DELETE
AS
      IF @@rowcount=0
            ROLLBACK TRANSACTION
      ELSE
            PRINT 'Eliminación de plan de proyecto de titulación,
                   correcto'
GO
```

B.2 CREACIÓN DE PROCEDIMIENTOS ALMACENADOS EN EL SAE PARA LA CONSULTA DE NOTAS

```
USE SAEIERI

IF EXISTS (SELECT * FROM sysobjects WHERE id =
           object_id (N'sp_reqNombre') and
           OBJECTPROPERTY(id, N'IsProcedure')=1)
      DROP PROCEDURE sp_reqNombre
GO

CREATE PROCEDURE  sp_reqNombre
      @numUni varchar(9)
AS
      SELECT estatue, nomest, sexoes FROM SAEESTUD
      WHERE codest = @numUni
GO


IF EXISTS (SELECT * FROM sysobjects WHERE id =
           object_id(N'sp_reqNotas') and
           OBJECTPROPERTY(id, N'IsProcedure')=1)
      DROP PROCEDURE sp_reqNotas
GO

CREATE PROCEDURE  sp_reqNotas
      @numUni varchar(9)
AS
      SELECT codmat, numcal, calif1, calif2, calif3, calif4, calif5,
               sumato, aprueb FROM SAECALIF
            WHERE CODEST = @numUni
GO


IF EXISTS (SELECT * FROM sysobjects WHERE id =
           object_id(N'sp_reqEstudiantes') and
           OBJECTPROPERTY(id, N'IsProcedure')=1)
      DROP PROCEDURE sp_reqEstudiantes
GO

CREATE PROCEDURE sp_reqEstudiantes
      @totCred int
AS
      SELECT codest, nomest, totcred FROM SAEESTUD
```

```
      WHERE estatue = 'MATRICULADO' AND totcred >= @totCred
GO

IF EXISTS (SELECT * FROM sysobjects WHERE id =
           object_id(N'sp_reqTesis')
           and OBJECTPROPERTY(id, N'IsProcedure')=1)
      DROP PROCEDURE sp_reqTesis
GO

CREATE PROCEDURE sp_reqTesis
      @numUni varchar(9)
AS
      SELECT count(*) FROM SAECALIF
      WHERE codest = @numuni AND codmat = 'IRO920'
GO
```

ANEXO C: ARCHIVOS DE AUDIO

A continuación se tiene una lista de los archivos de audio que se utilizan en el IVR de este proyecto, todos han sido grabados en formato WAV a 8 KHz, con señal de un solo canal (mono).

C.1 ARCHIVOS QUE FORMAN EL DIALOGO DEL IVR

La tabla muestra los archivos de audio que deben encontrarse en el directorio */var/lib/asterisk/sounds/IERI-IVR/prompts*, la primera columna es el nombre del archivo (sin la extensión .wav) y la segunda el contenido que debe tener el mismo.

Ayuda	Ayuda del sistema, si su número único consta de siete dígitos puede decirlo en combinaciones de unidades y decenas, si consta de nueve dígitos debe empezar diciendo el año y los cinco últimos dígitos en combinaciones de unidades y decenas, por ejemplo para siete dígitos, cero, dos, diez, cero, diecisiete, o para nueve dígitos, dos mil, cinco, treinta y uno, cuatro, quince. También puede utilizar el teclado de su teléfono y marcar su número único
Bienvenida	Bienvenida señorita
Bienvenido	Bienvenido señor
Despedida	Gracias por usar este servicio, hasta pronto
diga_digite_nu	Por favor, diga o digite su número único, también puede decir ayuda si necesita información acerca de cómo utilizar este sistema
diga_si_no	Diga si o no
digite_nu	Por favor, digite su número único
fallo_conexion	Lo siento, falló la conexión con la base de datos
Hola	Hola, éste es el servicio estudiantil de la Escuela Politécnica Nacional
menu_dtmf	Por favor digite uno para consultar sus notas, dos para certificados de prácticas preprofesionales o tres para planes de proyectos de titulación
menu_dtmf_notas	Digite uno para conocer sus notas del primer bimestre, dos para el segundo, tres para supletorio o cuatro para conocer todas sus notas
menu_dtmf_rep	Si desea volver a escuchar esta información presione uno o dos para regresar al menú principal
menu_dtmf_rep_notas	Si desea volver a escuchar esta información presione uno, dos para regresar al menú de notas o tres para ir al menú principal
menu_voz	Por favor diga que desea consultar: notas, certificados de prácticas preprofesionales o planes de proyectos de titulación
menu_voz_notas	Diga el nombre de la materia que desea consultar, primer bimestre,

	segundo bimestre, supletorio o todas si desea consultar todas sus notas
menu_voz_rep	Diga repetir para volver a escuchar esta información o menú principal para utilizar los servicios adicionales
menu_voz_rep_notas	Diga repetir si desea volver a escuchar ésta información, volver para consultar otra materia o bimestre o menú principal para utilizar los servicios adicionales
ningun_proyecto	Lo siento, usted no tiene registrado ningún plan de proyecto de titulación o no ha sido revisado
Ninguna_nota	Lo siento, usted no tiene registrada ninguna nota
Ninguna_practica	Lo siento, usted no tiene registrado ningún certificado de practicas profesionales o no ha sido revisado
no_disponible	Lo siento, el servicio no está disponible por el momento, falló la conexión a la base de datos
no_entendi	Lo siento, no le entendí
no_escuche	Lo siento, no le escuché
notas_1bim	Sus notas del primer bimestre son:
notas_2bim	Sus notas del segundo bimestre son:
notas_3bim	Sus notas de los exámenes supletorios son:
notas_todas	Sus notas son:
nu_no_coincide	Lo siento, el número único que ingresó no coincide con ningún registro en la base de datos, por favor inténtelo nuevamente
nu_no_valido	Lo siento, el número único que ingresó no es válido
op_no_digitada	Usted no ha digitado ninguna opción
op_no_valida	Lo siento, la opción que digitó no es válida
rev_estado	Ha sido revisado y su estado es
rev_practicas	Su certificado de prácticas preprofesionales en la empresa
rev_proyectos	Su plan de proyecto de titulación de tema
su_nu_es	Su número único es
su_respuesta	Su respuesta fue
Supletorio	Examen supletorio

C.2 ARCHIVOS PARA REPRODUCIR NÚMEROS

En el directorio /*var*/*lib*/*asterisk*/*sounds*/*IERI-IVR*/*numeros* se deben tener archivos de audio correspondientes a los números desde 0 hasta el 99, de modo que el nombre del archivo corresponda al mismo número seguido de la extensión .wav, por ejemplo:

1	uno
10	diez
32	treinta y dos
2000	dos mil
punto	punto

En la columna de la derecha se indica el contenido del audio. Los dos últimos corresponden a archivos de audio adicionales que deben constar en el mismo directorio. El último específicamente, sirve para decir las notas a los estudiantes y el anterior para pedir la confirmación al usuario cuando el número único consta de nueve dígitos.

C.3 ARCHIVOS PARA REPRODUCIR LAS MATERIAS

Para proporcionar las calificaciones a los estudiantes se necesita archivos de audio que digan el nombre de cada materia, el directorio por defecto para estos archivos es *var*/*lib*/*asterisk*/*sounds*/*IERI-IVR*/*materias* y el nombre de cada uno corresponde al código de la materia seguido de la extensión .wav. En la tabla siguiente se muestra cada una de las materias existentes al momento del desarrollo de este proyecto.

IRO116	Cálculo
IRO124	Algebra Lineal
IRO134	Química General
IRO143	Tecnologías de la Información
IRO162	Ecología y Medio Ambiente
IRO176	Física General I
IRO214	Ecuaciones Diferenciales
IRO224	Análisis Vectorial
IRO231	Laboratorio de Tecnología Eléctrica
IRO232	Tecnología Eléctrica
IRO243	Programación
IRO252	Expresión Oral Y Escrita
IRO274	Física General II

IRO314	Matemáticas Avanzadas
IRO332	Laboratorio de Circuitos Eléctricos I
IRO335	Análisis De Circuitos Eléctricos I
IRO345	Sistemas Operativos
IRO363	Probabilidad Y Estadística
IRO374	Física Moderna
IRO414	Análisis de Señales y Sistemas
IRO422	Laboratorio de Circuitos Eléctricos II
IRO424	Análisis de Circuitos Eléctricos II
IRO432	Laboratorio de Dispositivos Electrónicos
IRO435	Dispositivos Electrónicos
IRO444	Bases de Datos
IRO474	Teoría Electromagnética
IRO522	Laboratorio de Sistemas Digitales
IRO524	Sistemas Digitales
IRO532	Laboratorio De Circuitos Electrónicos
IRO534	Circuitos Electrónicos
IRO543	Programación Orientada a Objetos
IRO552	Laboratorio de Teoría de Comunicaciones
IRO563	Teoría de la Información Y Codificación
IRO575	Sistemas de Comunicación Radiantes
IRO613	Sistemas de Cableado Estructurado
IRO622	Laboratorio de Sistemas Microprocesados
IRO623	Sistemas Microprocesados
IRO644	Programación con Herramientas Visuales
IRO654	Redes de Área Local
IRO683	Marco Regulatorio de Servicios de Telecomunicaciones
IRO713	Ingeniería Financiera
IRO735	Aplicaciones Distribuidas
IRO744	Introducción a La Multimedia
IRO754	Redes de Área Extendida
IRO763	Evaluación de Redes
IRO774	Comunicaciones Inalámbricas
IRO783	Preparación, Gestión y Evaluación de Proyectos
IRO813	Administración General
IRO823	Comercialización
IRO843	Redes e Intranet
IRO863	Redes TCP/IP
IRO864	Administración y Gestión de Redes
IRO874	Seguridad en Redes

IRO920	Proyecto de Titilación
IRC573	Herramientas de Generacion Multimedia
IRC585	Sistemas de Transmisión
IRC633	Televisión
IRC643	Procesamiento Digital de Señales
IRC674	Telefonía
IRC754	Redes de Área Local Inalámbricas
IRC823	Interfaces para Microcomputadores
IRC873	Comunicaciones Ópticas
IRC883	Comunicaciones Satelitales
IERI01	Teoría de Comunicaciones
HSE112	Realidad Socioeconómica del Ecuador
HSE132	Ética Profesional
HSE142	La Ciencia y la Tecnología
HSE212	Desafíos del Mundo Actual
HSE232	Legislación Laboral
HSE282	Teoría y Práctica de la Realización Audiovisual
HSE312	Apreciación Cinematográfica
HSE322	Contacto con la Música Universal
HSE332	Introducción al Arte
HSE352	Sexualidad Humana
HSE372	Psicología de la Personalidad
HSE382	Derechos Humanos Y Ciudadanía

C.4 ADICIONALES

Además, se necesitan archivos adicionales para cuando se pide confirmación al usuario acerca de alguna respuesta que ha proporcionado pero que el sistema no tiene la certeza necesaria para tomarla como verdadera. En el directorio *var*/*lib*/*asterisk*/*sounds*/*IERI-IVR*/*respuestas* deben constar los siguientes archivos.

1_bimestre	Primer bimestre?
2_bimestre	Segundo bimestre?
notas	Consulta de notas?
practicas	Certificados de prácticas preprofesionales?
principal	Menú principal?
proyectos	Planes de proyectos de titulación?
repetir	Repetir?
supletorio	Supletorio?
todas notas	Todas las notas?
volver	Volver?

ANEXO D: ARCHIVOS DE GRAMÁTICA

D.1 /etc/asterisk/grammars/NumeroUnico.grxml

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO" version="1.0"
         root="NumeroUnico"
         mode="voice"
         tag-format="lumenvox/1.0">


<!-- Regla principal: Numero único -->

<rule id="NumeroUnico" scope="public">

<tag> $=""; </tag>

<one-of>

<!-- Numero único de 7 digitos -->

<item> <one-of>

<!-- 1 dígito, 1 decena, 1 dígito, 1 decena, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_x12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 dígito, 1 decena, 2 dígitos, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_x12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 dígito, 1 decena, 4 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
```

```
      <item repeat="1"> <ruleref uri="#decena_x12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="4"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 dígito, 2 decenas, 2 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_x12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 dígito, 3 decenas -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_x12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 digitos, 1 decena, 1 dígito, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 dígitos, 1 decena, 3 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="3"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 dígitos, 2 decenas, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
```

```
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 3 dígitos, 1 decena, 2 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 3 dígitos, 2 decenas -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 4 dígitos, 1 decena, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 5 dígitos, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
```

```
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 7 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_09"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="4"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 decena, 1 dígito, 1 decena, 2 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 decena, 1 dígito, 2 decenas -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 decena, 2 dígitos, 1 decena, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 decena, 3 dígitos, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
```

```
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 decena, 5 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="4"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 decenas, 1 dígito, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 decenas, 3 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="3"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 3 decenas, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#decena_9x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

</one-of> </item>


<!-- Numero único de 9 dígitos -->

<item>

<!-- Empieza con el año de inscripcion (4 dígitos) -->

<item> <ruleref uri="#anio"/> <tag> $+=$$; </tag> </item>
<one-of>
```

```
<!-- Combinaciones de los 5 dígitos restantes -->

<!-- 1 dígito, 1 decena, 2 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 1 dígito, 2 decenas -->
<item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 dígitos, 1 decena, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 3 dígitos, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="2"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 5 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#digito_12"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="4"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      </item>

<!-- 1 decena, 1 dígito, 1 decena -->
<item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
</item>
```

```
<!-- 1 decena, 3 dígitos -->
<item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="3"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

<!-- 2 decenas, 1 dígito -->
<item>
      <item repeat="1"> <ruleref uri="#decena_12x"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#decena"/>
            <tag> $+=$$; </tag> </item>
      <item repeat="1"> <ruleref uri="#digito"/>
            <tag> $+=$$; </tag> </item>
</item>

</one-of> </item>

<item> ayuda <tag> $="ayuda"; </tag> </item> </one-of>

</rule>


<!-- SUBREGLAS USADAS EN LAS COMBINACIONES DEL NÚMERO ÚNICO-->


<!-- Subregla: Dígitos (solamente 1 y 2) -->

<rule id="digito_12" scope="public">

      <one-of xml:lang="es-CO">
            <item> uno     <tag> $="1_"; </tag> </item>
            <item> dos     <tag> $="2_"; </tag> </item>
      </one-of>

</rule>


<!-- Subregla: Dígitos (solamente 0 y 9) -->

<rule id="digito_09" scope="public">

      <one-of xml:lang="es-CO">
            <item> cero     <tag> $="0_"; </tag> </item>
            <item> nueve    <tag> $="9_"; </tag> </item>
      </one-of>

</rule>


<!-- Subregla: Dígitos (desde el 0 hasta el 9) -->

<rule id="digito" scope="public">

      <one-of xml:lang="es-CO">
            <item> cero       <tag> $="0_"; </tag> </item>
            <item> uno        <tag> $="1_"; </tag> </item>
            <item> dos        <tag> $="2_"; </tag> </item>
```

```
            <item> tres       <tag> $="3_"; </tag> </item>
            <item> cuatro     <tag> $="4_"; </tag> </item>
            <item> cinco      <tag> $="5_"; </tag> </item>
            <item> seis       <tag> $="6_"; </tag> </item>
            <item> siete      <tag> $="7_"; </tag> </item>
            <item> ocho       <tag> $="8_"; </tag> </item>
            <item> nueve      <tag> $="9_"; </tag> </item>
      </one-of>

</rule>
```

<!-- Subregla: Decenas (solamente terminadas en 1 y 2) -->

```
<rule id="decena_x12" scope="public">

      <one-of>
            <item> once             <tag> $="11_"; </tag> </item>
            <item> doce             <tag> $="12_"; </tag> </item>
            <item> veintiuno        <tag> $="21_"; </tag> </item>
            <item> veintidos        <tag> $="22_"; </tag> </item>
            <item> treintiuno       <tag> $="31_"; </tag> </item>
            <item> treintaiuno      <tag> $="31_"; </tag> </item>
            <item> trentiuno        <tag> $="31_"; </tag> </item>
            <item> trentaiuno       <tag> $="31_"; </tag> </item>
            <item> treintidos       <tag> $="32_"; </tag> </item>
            <item> treintaidos      <tag> $="32_"; </tag> </item>
            <item> trentidos        <tag> $="32_"; </tag> </item>
            <item> trentaidos       <tag> $="32_"; </tag> </item>
            <item> cuarentiuno      <tag> $="41_"; </tag> </item>
            <item> cuarentaiuno     <tag> $="41_"; </tag> </item>
            <item> cuarentidos      <tag> $="42_"; </tag> </item>
            <item> cuarentaidos     <tag> $="42_"; </tag> </item>
            <item> cincuentiuno     <tag> $="51_"; </tag> </item>
            <item> cincuentaiuno    <tag> $="51_"; </tag> </item>
            <item> cincuentidos     <tag> $="52_"; </tag> </item>
            <item> cincuentaidos    <tag> $="52_"; </tag> </item>
            <item> sesentiuno       <tag> $="61_"; </tag> </item>
            <item> sesentaiuno      <tag> $="61_"; </tag> </item>
            <item> sesentidos       <tag> $="62_"; </tag> </item>
            <item> sesentaidos      <tag> $="62_"; </tag> </item>
            <item> setentiuno       <tag> $="71_"; </tag> </item>
            <item> setentaiuno      <tag> $="71_"; </tag> </item>
            <item> setentidos       <tag> $="72_"; </tag> </item>
            <item> setentaidos      <tag> $="72_"; </tag> </item>
            <item> ochentiuno       <tag> $="81_"; </tag> </item>
            <item> ochentaiuno      <tag> $="81_"; </tag> </item>
            <item> ochentidos       <tag> $="82_"; </tag> </item>
            <item> ochentaidos      <tag> $="82_"; </tag> </item>
            <item> noventiuno       <tag> $="91_"; </tag> </item>
            <item> noventaiuno      <tag> $="91_"; </tag> </item>
            <item> noventidos       <tag> $="92_"; </tag> </item>
            <item> noventaidos      <tag> $="92_"; </tag> </item>
      </one-of>
</rule>
```

<!-- Subregla: Decenas (solamente las que empiezan con 1 y 2) -->

```
<rule id="decena_12x" scope="public">
```

```
      <one-of>
            <item> diez             <tag> $="10_"; </tag> </item>
            <item> once             <tag> $="11_"; </tag> </item>
            <item> doce             <tag> $="12_"; </tag> </item>
            <item> trece            <tag> $="13_"; </tag> </item>
            <item> catorce          <tag> $="14_"; </tag> </item>
            <item> quince           <tag> $="15_"; </tag> </item>
            <item> dieciseis        <tag> $="16_"; </tag> </item>
            <item> diecisiete       <tag> $="17_"; </tag> </item>
            <item> dieciocho        <tag> $="18_"; </tag> </item>
            <item> diecinueve       <tag> $="19_"; </tag> </item>

            <item> veinte           <tag> $="20_"; </tag> </item>
            <item> veintiuno        <tag> $="21_"; </tag> </item>
            <item> veintidos        <tag> $="22_"; </tag> </item>
            <item> veintitres       <tag> $="23_"; </tag> </item>
            <item> veinticuatro     <tag> $="24_"; </tag> </item>
            <item> veinticinco      <tag> $="25_"; </tag> </item>
            <item> veintiseis       <tag> $="26_"; </tag> </item>
            <item> veintisiete      <tag> $="27_"; </tag> </item>
            <item> veintiocho       <tag> $="28_"; </tag> </item>
            <item> veintinueve      <tag> $="29_"; </tag> </item>
        </one-of>

</rule>


<!-- Subregla: Decenas (solamente las que empiezan con 1 y 2) -->

<rule id="decena_9x" scope="public">

        <one-of>
                <item> noventa          <tag> $="90_"; </tag> </item>
                <item> noventiuno       <tag> $="91_"; </tag> </item>
                <item> noventaiuno      <tag> $="91_"; </tag> </item>
                <item> noventidos       <tag> $="92_"; </tag> </item>
                <item> noventaidos      <tag> $="92_"; </tag> </item>
                <item> noventitres      <tag> $="93_"; </tag> </item>
                <item> noventaitres     <tag> $="93_"; </tag> </item>
                <item> noventicuatro    <tag> $="94_"; </tag> </item>
                <item> noventaicuatro   <tag> $="94_"; </tag> </item>
                <item> noventicinco     <tag> $="95_"; </tag> </item>
                <item> noventaicinco    <tag> $="95_"; </tag> </item>
                <item> noventiseis      <tag> $="96_"; </tag> </item>
                <item> noventaiseis     <tag> $="96_"; </tag> </item>
                <item> noventisiete     <tag> $="97_"; </tag> </item>
                <item> noventaisiete    <tag> $="97_"; </tag> </item>
                <item> noventiocho      <tag> $="98_"; </tag> </item>
                <item> noventaiocho     <tag> $="98_"; </tag> </item>
                <item> noventinueve     <tag> $="99_"; </tag> </item>
                <item> noventainueve    <tag> $="99_"; </tag> </item>
        </one-of>

</rule>


<!-- Subregla: Decenas (desde el 10 hasta 99) -->

<rule id="decena" scope="public">
```

```
<one-of>
      <item> diez             <tag> $="10_"; </tag> </item>
      <item> once             <tag> $="11_"; </tag> </item>
      <item> doce             <tag> $="12_"; </tag> </item>
      <item> trece            <tag> $="13_"; </tag> </item>
      <item> catorce          <tag> $="14_"; </tag> </item>
      <item> quince           <tag> $="15_"; </tag> </item>
      <item> dieciseis        <tag> $="16_"; </tag> </item>
      <item> diecisiete       <tag> $="17_"; </tag> </item>
      <item> dieciocho        <tag> $="18_"; </tag> </item>
      <item> diecinueve       <tag> $="19_"; </tag> </item>

      <item> veinte           <tag> $="20_"; </tag> </item>
      <item> veintiuno        <tag> $="21_"; </tag> </item>
      <item> veintidos        <tag> $="22_"; </tag> </item>
      <item> veintitres       <tag> $="23_"; </tag> </item>
      <item> veinticuatro     <tag> $="24_"; </tag> </item>
      <item> veinticinco      <tag> $="25_"; </tag> </item>
      <item> veintiseis       <tag> $="26_"; </tag> </item>
      <item> veintisiete      <tag> $="27_"; </tag> </item>
      <item> veintiocho       <tag> $="28_"; </tag> </item>
      <item> veintinueve      <tag> $="29_"; </tag> </item>

      <item> treinta          <tag> $="30_"; </tag> </item>
      <item> treintiuno       <tag> $="31_"; </tag> </item>
      <item> treintaiuno      <tag> $="31_"; </tag> </item>
      <item> trentiuno        <tag> $="31_"; </tag> </item>
      <item> trentaiuno       <tag> $="31_"; </tag> </item>
      <item> treintidos       <tag> $="32_"; </tag> </item>
      <item> treintaidos      <tag> $="32_"; </tag> </item>
      <item> trentidos        <tag> $="32_"; </tag> </item>
      <item> trentaidos       <tag> $="32_"; </tag> </item>
      <item> treintitres      <tag> $="33_"; </tag> </item>
      <item> treintaitres     <tag> $="33_"; </tag> </item>
      <item> trentitres       <tag> $="33_"; </tag> </item>
      <item> trentaitres      <tag> $="33_"; </tag> </item>
      <item> treinticuatro    <tag> $="34_"; </tag> </item>
      <item> treintaicuatro   <tag> $="34_"; </tag> </item>
      <item> trenticuatro     <tag> $="34_"; </tag> </item>
      <item> trentaicuatro    <tag> $="34_"; </tag> </item>
      <item> treinticinco     <tag> $="35_"; </tag> </item>
      <item> treintaicinco    <tag> $="35_"; </tag> </item>
      <item> trenticinco      <tag> $="35_"; </tag> </item>
      <item> trentaicinco     <tag> $="35_"; </tag> </item>
      <item> treintiseis      <tag> $="36_"; </tag> </item>
      <item> treintaiseis     <tag> $="36_"; </tag> </item>
      <item> trentiseis       <tag> $="36_"; </tag> </item>
      <item> trentaiseis      <tag> $="36_"; </tag> </item>
      <item> treintisiete     <tag> $="37_"; </tag> </item>
      <item> treintaisiete    <tag> $="37_"; </tag> </item>
      <item> trentisiete      <tag> $="37_"; </tag> </item>
      <item> trentaisiete     <tag> $="37_"; </tag> </item>
      <item> treintiocho      <tag> $="38_"; </tag> </item>
      <item> treintaiocho     <tag> $="38_"; </tag> </item>
      <item> trentiocho       <tag> $="38_"; </tag> </item>
      <item> trentaiocho      <tag> $="38_"; </tag> </item>
      <item> treintinueve     <tag> $="39_"; </tag> </item>
      <item> treintainueve    <tag> $="39_"; </tag> </item>
      <item> trentinueve      <tag> $="39_"; </tag> </item>
      <item> trentainueve     <tag> $="39_"; </tag> </item>
```

```
<item> cuarentiuno      <tag> $="41_"; </tag> </item>
<item> cuarentaiuno     <tag> $="41_"; </tag> </item>
<item> cuarentidos      <tag> $="42_"; </tag> </item>
<item> cuarentaidos     <tag> $="42_"; </tag> </item>
<item> cuarentitres     <tag> $="43_"; </tag> </item>
<item> cuarentaitres    <tag> $="43_"; </tag> </item>
<item> cuarenticuatro   <tag> $="44_"; </tag> </item>
<item> cuarentaycuatro  <tag> $="44_"; </tag> </item>
<item> cuarenticinco    <tag> $="45_"; </tag> </item>
<item> cuarentaicinco   <tag> $="45_"; </tag> </item>
<item> cuarentiseis     <tag> $="46_"; </tag> </item>
<item> cuarentaiseis    <tag> $="46_"; </tag> </item>
<item> cuarentisiete    <tag> $="47_"; </tag> </item>
<item> cuarentaisiete   <tag> $="47_"; </tag> </item>
<item> cuarentiocho     <tag> $="48_"; </tag> </item>
<item> cuarentaiocho    <tag> $="48_"; </tag> </item>
<item> cuarentinueve    <tag> $="49_"; </tag> </item>
<item> cuarentainueve   <tag> $="49_"; </tag> </item>
<item> cuarenta         <tag> $="40_"; </tag> </item>

<item> cincuenta        <tag> $="50_"; </tag> </item>
<item> cincuentiuno     <tag> $="51_"; </tag> </item>
<item> cincuentaiuno    <tag> $="51_"; </tag> </item>
<item> cincuentidos     <tag> $="52_"; </tag> </item>
<item> cincuentaidos    <tag> $="52_"; </tag> </item>
<item> cincuentitres    <tag> $="53_"; </tag> </item>
<item> cincuentaitres   <tag> $="53_"; </tag> </item>
<item> cincuenticuatro  <tag> $="54_"; </tag> </item>
<item> cincuentaicuatro <tag> $="54_"; </tag> </item>
<item> cincuenticinco   <tag> $="55_"; </tag> </item>
<item> cincuentaicinco  <tag> $="55_"; </tag> </item>
<item> cincuentiseis    <tag> $="56_"; </tag> </item>
<item> cincuentaiseis   <tag> $="56_"; </tag> </item>
<item> cincuentisiete   <tag> $="57_"; </tag> </item>
<item> cincuentaisiete  <tag> $="57_"; </tag> </item>
<item> cincuentiocho    <tag> $="58_"; </tag> </item>
<item> cincuentaiocho   <tag> $="58_"; </tag> </item>
<item> cincuentinueve   <tag> $="59_"; </tag> </item>
<item> cincuentainueve  <tag> $="59_"; </tag> </item>

<item> sesenta          <tag> $="60_"; </tag> </item>
<item> sesentiuno       <tag> $="61_"; </tag> </item>
<item> sesentaiuno      <tag> $="61_"; </tag> </item>
<item> sesentidos       <tag> $="62_"; </tag> </item>
<item> sesentaidos      <tag> $="62_"; </tag> </item>
<item> sesentitres      <tag> $="63_"; </tag> </item>
<item> sesentaitres     <tag> $="63_"; </tag> </item>
<item> sesenticuatro    <tag> $="64_"; </tag> </item>
<item> sesentaicuatro   <tag> $="64_"; </tag> </item>
<item> sesenticinco     <tag> $="65_"; </tag> </item>
<item> sesentaicinco    <tag> $="65_"; </tag> </item>
<item> sesentiseis      <tag> $="66_"; </tag> </item>
<item> sesentaiseis     <tag> $="66_"; </tag> </item>
<item> sesentisiete     <tag> $="67_"; </tag> </item>
<item> sesentaisiete    <tag> $="67_"; </tag> </item>
<item> sesentiocho      <tag> $="68_"; </tag> </item>
<item> sesentaiocho     <tag> $="68_"; </tag> </item>
<item> sesentinueve     <tag> $="69_"; </tag> </item>
<item> sesentainueve    <tag> $="69_"; </tag> </item>
```

```
<item> setenta          <tag> $="70_"; </tag> </item>
<item> setentiuno       <tag> $="71_"; </tag> </item>
<item> setentaiuno      <tag> $="71_"; </tag> </item>
<item> setentidos       <tag> $="72_"; </tag> </item>
<item> setentaidos      <tag> $="72_"; </tag> </item>
<item> setentitres      <tag> $="73_"; </tag> </item>
<item> setentaitres     <tag> $="73_"; </tag> </item>
<item> setenticuatro    <tag> $="74_"; </tag> </item>
<item> setentaicuatro   <tag> $="74_"; </tag> </item>
<item> setenticinco     <tag> $="75_"; </tag> </item>
<item> setentaicinco    <tag> $="75_"; </tag> </item>
<item> setentiseis      <tag> $="76_"; </tag> </item>
<item> setentaiseis     <tag> $="76_"; </tag> </item>
<item> setentisiete     <tag> $="77_"; </tag> </item>
<item> setentaisiete    <tag> $="77_"; </tag> </item>
<item> setentiocho      <tag> $="78_"; </tag> </item>
<item> setentaiocho     <tag> $="78_"; </tag> </item>
<item> setentinueve     <tag> $="79_"; </tag> </item>
<item> setentainueve    <tag> $="79_"; </tag> </item>

<item> ochenta          <tag> $="80_"; </tag> </item>
<item> ochentiuno       <tag> $="81_"; </tag> </item>
<item> ochentaiuno      <tag> $="81_"; </tag> </item>
<item> ochentidos       <tag> $="82_"; </tag> </item>
<item> ochentaidos      <tag> $="82_"; </tag> </item>
<item> ochentitres      <tag> $="83_"; </tag> </item>
<item> ochentaitres     <tag> $="83_"; </tag> </item>
<item> ochenticuatro    <tag> $="84_"; </tag> </item>
<item> ochentaicuatro   <tag> $="84_"; </tag> </item>
<item> ochenticinco     <tag> $="85_"; </tag> </item>
<item> ochentaicinco    <tag> $="85_"; </tag> </item>
<item> ochentiseis      <tag> $="86_"; </tag> </item>
<item> ochentaiseis     <tag> $="86_"; </tag> </item>
<item> ochentisiete     <tag> $="87_"; </tag> </item>
<item> ochentaisiete    <tag> $="87_"; </tag> </item>
<item> ochentiocho      <tag> $="88_"; </tag> </item>
<item> ochentaiocho     <tag> $="88_"; </tag> </item>
<item> ochentinueve     <tag> $="89_"; </tag> </item>
<item> ochentainueve    <tag> $="89_"; </tag> </item>

<item> noventa          <tag> $="90_"; </tag> </item>
<item> noventiuno       <tag> $="91_"; </tag> </item>
<item> noventaiuno      <tag> $="91_"; </tag> </item>
<item> noventidos       <tag> $="92_"; </tag> </item>
<item> noventaidos      <tag> $="92_"; </tag> </item>
<item> noventitres      <tag> $="93_"; </tag> </item>
<item> noventaitres     <tag> $="93_"; </tag> </item>
<item> noventicuatro    <tag> $="94_"; </tag> </item>
<item> noventaicuatro   <tag> $="94_"; </tag> </item>
<item> noventicinco     <tag> $="95_"; </tag> </item>
<item> noventaicinco    <tag> $="95_"; </tag> </item>
<item> noventiseis      <tag> $="96_"; </tag> </item>
<item> noventaiseis     <tag> $="96_"; </tag> </item>
<item> noventisiete     <tag> $="97_"; </tag> </item>
<item> noventaisiete    <tag> $="97_"; </tag> </item>
<item> noventiocho      <tag> $="98_"; </tag> </item>
<item> noventaiocho     <tag> $="98_"; </tag> </item>
<item> noventinueve     <tag> $="99_"; </tag> </item>
<item> noventainueve    <tag> $="99_"; </tag> </item>
```

```
      </one-of>

</rule>


<!-- Subregla: Año (año con el que empieza el numero único) -->

<rule id="anio" scope="public">

      <item > dos mil </item>

      <one-of>
            <item> tres     <tag> $="2003_"; </tag> </item>
            <item> cuatro   <tag> $="2004_"; </tag> </item>
            <item> cinco    <tag> $="2005_"; </tag> </item>
            <item> seis     <tag> $="2006_"; </tag> </item>
            <item> siete    <tag> $="2007_"; </tag> </item>
            <item> ocho     <tag> $="2008_"; </tag> </item>
            <item> nueve    <tag> $="2009_"; </tag> </item>
            <item> diez     <tag> $="2010_"; </tag> </item>
            <item> once     <tag> $="2011_"; </tag> </item>
            <item> doce     <tag> $="2012_"; </tag> </item>
      </one-of>

</rule>

</grammar>
```

D.2 /etc/asterisk/grammars/NumeroUnicoDTMF.grxml

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         version="1.0"
         root="NumeroUnico"
         mode="dtmf">


<rule id="NumeroUnico" scope="public">

      <one-of>
            <item repeat="7"> <ruleref uri="#digito"/> </item>
            <item repeat="9"> <ruleref uri="#digito"/> </item>
      </one-of>

</rule>

<!-- Subregla: Dígito (0 al 9) -->

<rule id="digito" scope="public">
      <one-of>
            <item> 0 </item>
            <item> 1 </item>
            <item> 2 </item>
            <item> 3 </item>
```

```
            <item> 4 </item>
            <item> 5 </item>
            <item> 6 </item>
            <item> 7 </item>
            <item> 8 </item>
            <item> 9 </item>
      </one-of>

</rule>

</grammar>
```

D.3 /etc/asterisk/grammars/MenuIVR.grxml

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO" version="1.0"
         root="Menu"
         mode="voice"
         tag-format="lumenvox/1.0">


<!-- Regla principal: Menú -->

<rule id="menu" scope="public">

      <one-of>
            <item> notas              <tag> $="notas";     </tag> </item>
            <item> consulta de notas  <tag> $="notas";     </tag> </item>
            <item> consultar notas    <tag> $="notas";     </tag> </item>
            <item> prácticas          <tag> $="practicas"; </tag> </item>
            <item> prácticas preprofesionales
                                      <tag> $="practicas"; </tag> </item>
            <item> consultar prácticas
                                      <tag> $="practicas"; </tag> </item>
            <item> consulta de prácticas preprofesionales
                                      <tag> $="practicas"; </tag> </item>
            <item> proyectos          <tag> $="proyectos"; </tag> </item>
            <item> proyectos de titulación
                                      <tag> $="proyectos"; </tag> </item>
            <item> consultar proyectos de titulación
                                      <tag> $="proyectos"; </tag> </item>
            <item> tesis              <tag> $="proyectos"; </tag> </item>
      </one-of>

   </rule>

</grammar>
```

D.4 /etc/asterisk/grammars/MenuRepetir.grxml

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO" version="1.0"
         root="Menu"
         mode="voice"
         tag-format="lumenvox/1.0">

<!-- Regla principal: Menú -->

<rule id="Menu" scope="public">

      <one-of>
            <item> repetir <tag> $="repetir"; </tag>  </item>
            <item> menu principal <tag> $="principal"; </tag>  </item>
      </one-of>

</rule>

</grammar>
```

D.5 /etc/asterisk/grammars/MenuRepetirNotas.grxml

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO" version="1.0"
         root="Menu"
         mode="voice"
         tag-format="lumenvox/1.0">


<!-- Regla principal: Menú -->

<rule id="Menu" scope="public">

      <one-of>
            <item> repetir <tag> $="repetir"; </tag>  </item>
            <item> volver <tag> $="volver"; </tag>  </item>
            <item> menu principal <tag> $="principal"; </tag>  </item>
      </one-of>

</rule>

</grammar>
```

D.6 /etc/asterisk/grammars/SiNo.grxml

```
<?xml version="1.0"?>

<grammar xmlns="http://www.w3.org/2001/06/grammar"
         xml:lang="es-CO"
         version="1.0"
         root="respuesta"
         mode="voice"
         tag-format="lumenvox/1.0">

<!-- Regla principal: Respuesta -->

<rule id="respuesta" scope="public">

      <one-of>
            <item> si <tag> $='si' </tag> </item>
            <item> no <tag> $='no' </tag> </item>
      </one-of>

</rule>

</grammar>
```

ANEXO E: SCRIPTS DE LA APLICACIÓN

E.1 SCRIPT PARA LA GENERCIÓN DE ARCHIVOS DE GRAMÁTICA EN EL MENÚ DE NOTAS

El siguiente script debe ejecutarse una vez concluidas las matrículas extraordinarias cada semestre, se localiza en */usr/local/agi-bin/*. Los parámetros que froman el URL de la conexión a la base de datos son solamente referenciales.

```
import java.sql.*;
import java.io.*;

public class XMLCreateGrammars
{

public static void main(String[] args)
{
      // URL de la direccion de la Base de Datos del SAE
      String urlDB = "jdbc:jtds:sqlserver://192.168.50.109:1433"+
                     "/SAEIERI;user=afuentes;password=andpass;TDS=7.0";

      // Objetos JDBC
      Connection con = null;
      Statement stmt1 = null;
      ResultSet rs1 = null;

      try { // Se establece la conexión

      Class.forName("net.sourceforge.jtds.jdbc.Driver");
      con = DriverManager.getConnection(urlDB);

      // Se consulta la lista de estudiantes matriulados en el SAE
      String SQL1 = "EXEC sp_reqEstudiantes 0";
      stmt1 = con.createStatement();
      rs1 = stmt1.executeQuery(SQL1);

      // Se crea un archivo de gramatica por estudiante
      while (rs1.next()) {

      FileOutputStream out = new FileOutputStream(
      "/etc/asterisk/grammars/materias/" + rs1.getString(1) + ".grxml");
      PrintStream ps = new PrintStream(out);

      ps.print("<?xml version=\"1.0\"?>\n\n<grammar xmlns=\" +
         "http://www.w3.org/2001/06/grammar\" \n\t xml:lang=\"es-CO\" ");
      ps.print("\n\t version=\"1.0\" \n\t root=\"Materias\" \n\t mode=\"
         "voice\" \n\t tag-format=\"lumenvox/1.0\">\n\n");
      ps.print("<rule id=\"Materias\" scope=\"public\">\n\n\t" +
         "<one-of>\n\n");
```

```
      // Se consulta las materias por estudiante
      String SQL2 = "EXECUTE sp_reqNotas '" + rs1.getString(1) + "'";
      Statement stmt2 = con.createStatement();
      ResultSet rs2 = stmt2.executeQuery(SQL2);

      while (rs2.next())
            ps.print(nombresMateria(rs2.getString(1)));

      if (rs2 != null) try { rs2.close(); } catch(Exception e) {}
      if (stmt2 != null) try { stmt2.close(); } catch(Exception e) {}

      ps.print("\n\t\t");     // Adicionales en todos los archivos
      ps.print("<item> primer bimestre <tag> $=\"1_bimestre\" +
               "</tag> </item> \n\t\t");
      ps.print("<item> segundo bimestre <tag> $=\"2_bimestre\" +
               "</tag> </item> \n\t\t");
      ps.print("<item> supletorio <tag> $=\"supletorio\" +
               "</tag> </item> \n\t\t");
      ps.print("<item> todas <tag> $=\"todasnotas\" +
               "</tag> </item> \n\n\t");
      ps.print("</one-of>\n\n</rule>\n\n</grammar>");
      ps.close(); }
      }

      catch (Exception e) {
         e.printStackTrace();
      }

      finally {
            if (rs1 != null) try {rs1.close();} catch(Exception e){}
            if (stmt1 != null) try {stmt1.close();} catch(Exception e){}
            if (con != null) try {con.close();} catch(Exception e){}
      }
}

// Metodo para discriminar la materia segun el código

static String nombresMateria(String codmat)
{
      String strMat = "\n\t\t";
      String tag = "<tag> $=\"" + codmat + "\" +
                   "</tag> </item>\n\t\t";

      if(codmat.equals("IRO116")){
            strMat += "<item> Calculo " + tag;
      }else
      if(codmat.equals("IRO124")){
            strMat += "<item> Algebra Lineal " + tag;
      }else
      if(codmat.equals("IRO134")){
            strMat += "<item> Quimica General " + tag;
            strMat += "<item> Quimica " + tag;
      }else
      if(codmat.equals("IRO143")){
            strMat += "<item> Tecnologias de la Informacion " + tag;
      }else
      if(codmat.equals("IRO162")){
            strMat += "<item> Ecologia y Medio Ambiente " + tag;
            strMat += "<item> Ecologia " + tag;
      }else
```

```
if(codmat.equals("IRO176")){
      strMat += "<item> Fisica General Uno " + tag;
      strMat += "<item> Fisica Uno " + tag;
}else
if(codmat.equals("IRO214")){
      strMat += "<item> Ecuaciones Diferenciales " + tag;
}else
if(codmat.equals("IRO224")){
      strMat += "<item> Analisis Vectorial " + tag;
}else
if(codmat.equals("IRO231")){
      strMat += "<item> Laboratorio de Tecnologia Electrica " +
                 tag;
}else
if(codmat.equals("IRO232")){
      strMat += "<item> Tecnologia Electrica " + tag;
}else
if(codmat.equals("IRO243")){
      strMat += "<item> Programacion " + tag;
}else
if(codmat.equals("IRO252")){
      strMat += "<item> Expresion Oral y Escrita " + tag;
}else
if(codmat.equals("IRO274")){
      strMat += "<item> Fisica General Dos " + tag;
      strMat += "<item> Fisica Dos " + tag;
}else
if(codmat.equals("IRO314")){
      strMat += "<item> Matematicas Avanzadas " + tag;
}else
if(codmat.equals("IRO332")){
      strMat += "<item> Laboratorio de Circuitos Electricos Uno " +
                 tag;
      strMat += "<item> Laboratorio de Circuitos Uno " + tag;
}else
if(codmat.equals("IRO335")){
      strMat += "<item> Analisis de Circuitos Electricos Uno " +
                 tag;
      strMat += "<item> Analisis de Circuitos Uno " + tag;
      strMat += "<item> Circuitos Electricos Uno " + tag;
}else
if(codmat.equals("IRO345")){
      strMat += "<item> Sistemas Operativos " + tag;
}else
if(codmat.equals("IRO363")){
      strMat += "<item> Probabilidad y Estadistica " + tag;
      strMat += "<item> Probabilidad " + tag;
}else
if(codmat.equals("IRO374")){
      strMat += "<item> Fisica Moderna " + tag;
}else
if(codmat.equals("IRO414")){
      strMat += "<item> Analisis de Seniales y Sistemas " + tag;
      strMat += "<item> Seniales y Sistemas  " + tag;
}else

if(codmat.equals("IRO422")){
      strMat += "<item> Laboratorio de Circuitos Electricos Dos " +
                 tag;
      strMat += "<item> Laboratorio de Circuitos Dos " + tag;
```

```
}else
if(codmat.equals("IRO424")){
      strMat += "<item> Analisis de Circuitos Electricos Dos " +
                tag;
      strMat += "<item> Analisis de Circuitos Dos " + tag;
      strMat += "<item> Circuitos Electricos Dos " + tag;
}else
if(codmat.equals("IRO432")){
      strMat += "<item> Laboratorio de Dispositivos Electronicos "
                + tag;
}else
if(codmat.equals("IRO435")){
      strMat += "<item> Dispositivos Electronicos " + tag;
}else
if(codmat.equals("IRO444")){
      strMat += "<item> Bases de Datos " + tag;
}else
if(codmat.equals("IRO474")){
      strMat += "<item> Teoria Electromagnetica " + tag;
}else
if(codmat.equals("IRO522")){
      strMat += "<item> Laboratorio de Sistemas Digitales " + tag;
      strMat += "<item> Laboratorio de Digitales " + tag;
}else
if(codmat.equals("IRO524")){
      strMat += "<item> Sistemas Digitales " + tag;
      strMat += "<item> Digitales " + tag;
}else
if(codmat.equals("IRO532")){
      strMat += "<item> Laboratorio de Circuitos Electronicos " +
                tag;
}else
if(codmat.equals("IRO534")){
      strMat += "<item> Circuitos Electronicos " + tag;
}else
if(codmat.equals("IRO543")){
      strMat += "<item> Programacion Orientada a Objetos " + tag;
      strMat += "<item> POO " + tag;
}else
if(codmat.equals("IRO552")){
      strMat += "<item> Laboratorio de Teoria de Comunicaciones " +
                tag;
      strMat += "<item> Laboratorio de Teoria " + tag;
}else
if(codmat.equals("IRO563")){
      strMat += "<item> Teoria de la Informacion y Codificacion " +
                tag;
      strMat += "<item> TIC " + tag;
}else
if(codmat.equals("IRO575")){
      strMat += "<item> Sistemas de Comunicaciones Radiantes " +
                tag;
      strMat += "<item> Radiantes " + tag;
}else
if(codmat.equals("IRO613")){
      strMat += "<item> Sistemas de Cableado Estructurado " + tag;
      strMat += "<item> Cableado Estructurado " + tag;
      strMat += "<item> Cableado " + tag;
}else
if(codmat.equals("IRO622")){
```

```
      strMat += "<item> Laboratorio de Sistemas Microprocesados " +
                 tag;
      strMat += "<item> Laboratorio de Micros " + tag;
}else
if(codmat.equals("IRO623")){
      strMat += "<item> Sistemas Microprocesados " + tag;
      strMat += "<item> Microprocesados " + tag;
      strMat += "<item> Micros " + tag;
}else
if(codmat.equals("IRO644")){
      strMat += "<item> Programacion con Herramientas Visuales " +
                 tag;
      strMat += "<item> PE ACHE VE " + tag;
}else
if(codmat.equals("IRO654")){
      strMat += "<item> Redes de Area Local " + tag;
      strMat += "<item> LAN " + tag;
}else
if(codmat.equals("IRO683")){
      strMat += "<item> Marco Regulatorio de Servicios de " +
                 Telecomunicaciones " + tag;
      strMat += "<item> Marco Regulatorio " + tag;
}else
if(codmat.equals("IRO713")){
      strMat += "<item> Ingenieria Financiera " + tag;
      strMat += "<item> Financiera " + tag;
}else
if(codmat.equals("IRO735")){
      strMat += "<item> Aplicaciones Distribuidas " + tag;
      strMat += "<item> Distribuidas " + tag;
}else
if(codmat.equals("IRO744")){
      strMat += "<item> Introduccion a la Multimedia " + tag;
      strMat += "<item> Multimedia " + tag;
}else
if(codmat.equals("IRO754")){
      strMat += "<item> Redes de Area Extendida " + tag;
      strMat += "<item> WAN " + tag;
}else
if(codmat.equals("IRO763")){
      strMat += "<item> Evaluacion de Redes " + tag;
      strMat += "<item> Evaluacion " + tag;
}else
if(codmat.equals("IRO774")){
      strMat += "<item> Comunicaciones Inalambricas " + tag;
      strMat += "<item> Inalambricas " + tag;
}else
if(codmat.equals("IRO783")){
      strMat += "<item> Preparacion Gestion y Evaluacion de " +
                 Proyectos " + tag;
      strMat += "<item> Proyectos " + tag;
          }else
if(codmat.equals("IRO813")){
      strMat += "<item> Administracion General " + tag;
      strMat += "<item> Administracion " + tag;
}else
if(codmat.equals("IRO823")){
      strMat += "<item> Comercializacion " + tag;
}else
if(codmat.equals("IRO843")){
```

```
      strMat += "<item> Redes e Intranet " + tag;
      strMat += "<item> Intranet " + tag;
}else
if(codmat.equals("IRO863")){
      strMat += "<item> Redes TECEPE IPE " + tag;
      strMat += "<item> TECEPE IPE " + tag;
}else
if(codmat.equals("IRO864")){
      strMat += "<item> Administracion y Gestion de Redes " + tag;
      strMat += "<item> Gestion de Redes " + tag;
}else
if(codmat.equals("IRO874")){
      strMat += "<item> Seguridad en Redes " + tag;
      strMat += "<item> Seguridad " + tag;
}else
if(codmat.equals("IRO920")){
      //Esta información no es proporcionada por el sistema
      //strMat += "<item> Proyecto de Titulacion " + tag;
}else
if(codmat.equals("IERI01")){
      strMat += "<item> Teoria de Comunicaciones " + tag;
}else
if(codmat.equals("IRC573")){
      strMat += "<item> Herramientas de Generacion Multimedia " +
                tag;
      strMat += "<item> Herramientas Multimedia " + tag;
}else
if(codmat.equals("IRC585")){
      strMat += "<item> Sistemas de Transmision " + tag;
}else
if(codmat.equals("IRC633")){
      strMat += "<item> Television " + tag;
}else
if(codmat.equals("IRC643")){
      strMat += "<item> Procesamiento Digital de Seniales " + tag;
}else
if(codmat.equals("IRC674")){
      strMat += "<item> Telefonia " + tag;
}else
if(codmat.equals("IRC754")){
      strMat += "<item> Redes de Area Local Inalambricas " + tag;
      strMat += "<item> Vi LAN " + tag;
}else
if(codmat.equals("IRC823")){
      strMat += "<item> Interfaces para Microcomputadoras " + tag;
}else
if(codmat.equals("IRC873")){
      strMat += "<item> Comunicaciones Opticas " + tag;
      strMat += "<item> Opticas " + tag;
}else
if(codmat.equals("IRC883")){
      strMat += "<item> Comunicaciones Satelitales " + tag;
      strMat += "<item> Satelitales " + tag;
}else
if(codmat.equals("HSE112")){
      strMat += "<item> Realidad Socioeconomica del Ecuador " +
                tag;
      strMat += "<item> Realidad " + tag;
}else
if(codmat.equals("HSE132")){
```

```
            strMat += "<item> Etica Profesional " + tag;
            strMat += "<item> Etica " + tag;
      }else
      if(codmat.equals("HSE142")){
            strMat += "<item> La Ciencia y la Tecnologia en Relac " +
                        tag;
      }else
      if(codmat.equals("HSE212")){
            strMat += "<item> Desafios del Mundo Actual " + tag;
            strMat += "<item> Desafios " + tag;
      }else
      if(codmat.equals("HSE232")){
            strMat += "<item> Legislacion Laboral " + tag;
            strMat += "<item> Legislacion " + tag;
      }else
      if(codmat.equals("HSE282")){
            strMat += "<item> Teoria y Practica de la Realizacion " +
                        Audiovisual " + tag;
      }else
      if(codmat.equals("HSE312")){
            strMat += "<item> Apreciacion Cinematografica " + tag;
      }else
      if(codmat.equals("HSE322")){
            strMat += "<item> Contacto con la Musica Universal " + tag;
            strMat += "<item> Contacto con la Musica " + tag;
      }else
      if(codmat.equals("HSE332")){
            strMat += "<item> Introduccion al Arte " + tag;
            strMat += "<item> Arte " + tag;
      }else
      if(codmat.equals("HSE352")){
            strMat += "<item> Sexualidad Humana " + tag;
            strMat += "<item> Sexualidad " + tag;
      }else
      if(codmat.equals("HSE372")){
            strMat += "<item> Psicoligia de la Personalidad " + tag;
            strMat += "<item> Psicologia " + tag;
      }else
      if(codmat.equals("HSE382")){
            strMat += "<item> Derechos Humanos y Ciudadania " + tag;
            strMat += "<item> Derechos Humanos " + tag;
      }else
            System.out.println("... Error: materia " + codmat +
                              " no encontrada");
      return strMat;
      }
}
```

E.2 SCRIPTS PARA CONSULTAS A LA BASE DE DATOS

E.2.1/usr/local/agi-java/SQLReqNombre.java

```
import org.asteriskjava.fastagi.*;
```

```
import java.sql.*;

public class SQLReqNombre extends BaseAgiScript
{
      public void service(AgiRequest request, AgiChannel channel)
            throws AgiException
      {
            String numUnico = request.getParameter("nu");
            numUnico = numUnico.replaceAll("_", "");
                                          // ej. 0_2_10_0_17_ a 0210017
            channel.setVariable("NUMUNI", numUnico);

            // URL de la direccion de la Base de Datos del SAE
            String urlDB = "jdbc:jtds:sqlserver://192.168.50.109:1433"+
                        "/SAEIERI;user=afuentes;password=andpass;TDS=7.0";

            // Objetos JDBC
            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;

            try {

            // Se establece la conexion
            Class.forName("net.sourceforge.jtds.jdbc.Driver");
            con = DriverManager.getConnection(urlDB);

            // Se ejecuta el procedimiento almacenado
            String SQL = "EXECUTE sp_reqNombre '" + numUnico + "'";
            stmt = con.createStatement();
            rs = stmt.executeQuery(SQL);

            // Guardamos los valores en las variables de Asterisk
            while (rs.next()) {
                  channel.setVariable("ESTMAT", rs.getString(1));
                  channel.setVariable("NOMEST", rs.getString(2));
                  channel.setVariable("SEXOES", rs.getString(3));
                  channel.setVariable("ESTBDD", "ok");
                  }
            }

            catch (Exception e) {
                  e.printStackTrace();
                  channel.setVariable("ESTBDD", "error");
            }

            finally {
                  if(rs != null) try{rs.close();} catch(Exception e){}
                  if(stmt != null) try{stmt.close();}catch(Exception e){}
                  if(con != null) try{ con.close();} catch(Exception e){}
            }
      }
}
```

E.2.2/usr/local/agi-java/SQLReqNotas.java

```
import org.asteriskjava.fastagi.*;

import java.sql.*;

public class SQLReqNotas extends BaseAgiScript
{
      public void service(AgiRequest request, AgiChannel channel)
            throws AgiException
      {
            String numUnico = request.getParameter("nu");

            // URL de la direccion de la Base de Datos del SAE
            String urlDB = "jdbc:jtds:sqlserver://192.168.50.109:1433"+
                        "/SAEIERI;user=afuentes;password=andpass;TDS=7.0";

            // Objetos JDBC
            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;

            try {

            // Se establece la conexion
            Class.forName("net.sourceforge.jtds.jdbc.Driver");
            con = DriverManager.getConnection(urlDB);

            // Se ejecuta el procedimiento almacenado
            String SQL = "EXECUTE sp_reqNotas '" + numUnico + "'";
            stmt = con.createStatement();
            rs = stmt.executeQuery(SQL);

            int numMaterias = 0;
            int numCalifTotales = 0;
            int numCalif1Bim = 0;
            int numCalif2Bim = 0;
            int numCalifSupl = 0;

            // Guardamos los valores en las variables de Asterisk
            while (rs.next()) {
                  numMaterias++;
                  channel.setVariable("CODMAT[" + numMaterias + "]",
                                                      rs.getString(1));
                  channel.setVariable("NUMCAL[" + numMaterias + "]",
                                                      rs.getString(2));
                  channel.setVariable("CALIF1[" + numMaterias + "]",
                                                      rs.getString(3));
                  channel.setVariable("CALIF2[" + numMaterias + "]",
                                                      rs.getString(4));
                  channel.setVariable("CALIF3[" + numMaterias + "]",
                                                      rs.getString(5));
                  channel.setVariable("SUMATO[" + numMaterias + "]",
                                                      rs.getString(8));
                  channel.setVariable("APRUEB[" + numMaterias + "]",
                                                      rs.getString(9));

                  numCalifTotales += rs.getInt(2); // calif. totales
```

```
                  if(rs.getInt(2) == 3) { // 1 bim, 2 bim y supletorio
                        numCalif1Bim++;
                        numCalif2Bim++;
                        numCalifSupl++;
                  } else

                  if (rs.getInt(2) == 2) { // 1 bim y 2 bim
                        numCalif1Bim++;
                        numCalif2Bim++;
                  } else

                  if (rs.getInt(2) == 1) { // 1 bim
                        numCalif1Bim++;
                  } else ;
            }

      channel.setVariable("NUMMAT", String.valueOf(numMaterias));
      channel.setVariable("TOTCAL", String.valueOf(numCalifTotales));
      channel.setVariable("TOT1BI", String.valueOf(numCalif1Bim));
      channel.setVariable("TOT2BI", String.valueOf(numCalif2Bim));
      channel.setVariable("TOT3BI", String.valueOf(numCalifSupl));
      channel.setVariable("ESTBDD", "ok"); // Macro verificarConexionBDD

      }

      catch (Exception e) {
            e.printStackTrace();
            channel.setVariable("ESTBDD", "error");
      }

      finally {
            if(rs != null) try {rs.close();} catch(Exception e){}
            if(stmt != null) try {stmt.close();} catch(Exception e){}
            if(con != null) try {con.close();} catch(Exception e){}
                }
        }
}
```

E.2.3/usr/local/agi-java/SQLReqPracticas.java

```
import org.asteriskjava.fastagi.*;

import java.sql.*;

public class SQLReqPracticas extends BaseAgiScript
{
      public void service(AgiRequest request, AgiChannel channel)
            throws AgiException
      {
            String numUnico = request.getParameter("nu");

            // URL de la direccion de la Base de Datos PRAPRO
            String urlDB = "jdbc:jtds:sqlserver://192.168.50.109:1433"+
                        "/PRAPRO;user=afuentes;password=andpass;TDS=7.0";
```

```
            // Objetos JDBC
            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;

            try {

            // Se establece la conexión
            Class.forName("net.sourceforge.jtds.jdbc.Driver");
            con = DriverManager.getConnection(urlDB);

            // Se crea la consulta
            String SQL = "select estcer, nomemp from PRACTICAS where
                         codest = '" + numUnico + "'";
            stmt = con.createStatement();
            rs = stmt.executeQuery(SQL);

            // Guardamos los valores en las variables de Asterisk
            while (rs.next()) {
                  channel.setVariable("ESTCER", rs.getString(1));
                  channel.setVariable("NOMEMP", rs.getString(2));
                  channel.setVariable("ESTBDD", "ok");
                  }
            }

            catch (Exception e) {
                  e.printStackTrace();
                  channel.setVariable("ESTBDD", "error");
            }

            finally {
                if(rs != null) try {rs.close();} catch(Exception e){}
                if(stmt != null) try {stmt.close();} catch(Exception e){}
                if(con != null) try {con.close();} catch(Exception e){}
            }
      }
}
```

E.2.4/usr/local/agi-java/SQLReqProyectos.java

```
import org.asteriskjava.fastagi.*;

import java.sql.*;

public class SQLReqProyectos extends BaseAgiScript
{
      public void service(AgiRequest request, AgiChannel channel)
            throws AgiException
      {
            String numUnico = request.getParameter("nu");

            // URL de la direccion de la Base de Datos PRAPRO
            String urlDB = "jdbc:jtds:sqlserver://192.168.50.109:1433"+
                        "/PRAPRO;user=afuentes;password=andpass;TDS=7.0";

            // Objetos JDBC
```

```
            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;

            try {

            // Se establece la conexion
            Class.forName("net.sourceforge.jtds.jdbc.Driver");
            con = DriverManager.getConnection(urlDB);

            // Se crea la consulta
            String SQL = "select estpla, templa from PROYECTOS where
                         codest = '" + numUnico + "'";
            stmt = con.createStatement();
            rs = stmt.executeQuery(SQL);

            // Guardamos los valores en las variables de asterisk
            while (rs.next()) {
                  channel.setVariable("ESTPLA", rs.getString(1));
                  channel.setVariable("TEMPLA", rs.getString(2));
                  channel.setVariable("ESTBDD", "ok");
                  }
            }

            catch (Exception e) {
                  e.printStackTrace();
                  channel.setVariable("ESTBDD", "error");
            }

            finally {
                if(rs != null) try {rs.close();} catch(Exception e){}
                if(stmt != null) try {stmt.close();} catch(Exception e){}
                if(con != null) try {con.close();} catch(Exception e){}
            }
      }
}
```

ANEXO F: APLICACIÓN PARA EL MANEJO DE LA BASE DE DATOS PRAPRO

El acceso a la base de datos del SAE es restringido por parte de la UGI y la secretaria encargada de ingresar la información respecto a planes de proyectos de titulación y certificados de prácticas preprofesionales obviamente no podrá interactuar directamente con la base. Por esta razón ha sido necesario crear una aplicación que facilite el ingreso de la información y se la ha programado en lenguaje C++ en Visual Studio 7.1 con librerías estáticas. Aunque la aplicación ayuda al funcionamiento del sistema se ha optado por describirla aquí porque no forma parte del IVR como tal.

La aplicación se la ha denominado *"ierivox"* y comprende de una ventana principal en donde se encuentran dos tablas, la primera para almacenar los certificados de prácticas preprofesionales y la segunda para los planes de proyectos de titulación.

La información que se muestra es principalmente el número único y el nombre del estudiante, las fechas de recepción, revisión y estado del documento, la empresa y número de horas de las prácticas preprofesionales o el tema y director del plan de proyecto de titulación, esto se puede apreciar en la *figura F.1*.

En cada tabla existen dos botones que corresponden a *Agregar/Modificar* y *Eliminar*. El primero permite insertar un registro adicional de un estudiante en la tabla o editar el contenido de alguno si éste ya existe, y el segundo elimina el registro de un estudiante que previamente debió ser seleccionado en la tabla. En las *figuras F.3* y *F.4* se puede apreciar el cuadro de diálogo que se abre al pulsar el botón *Agregar/Modificar*, para certificados de prácticas preprofesionales y planes de proyectos de titulación respectivamente.

Para actualizar la lista de estudiantes del SAE de la carrera se lo hace desde el menú archivo, hay que tomar en cuenta que esto solo se debería hacer una vez por semestre, cuando hayan pasado las matrículas extraordinarias.

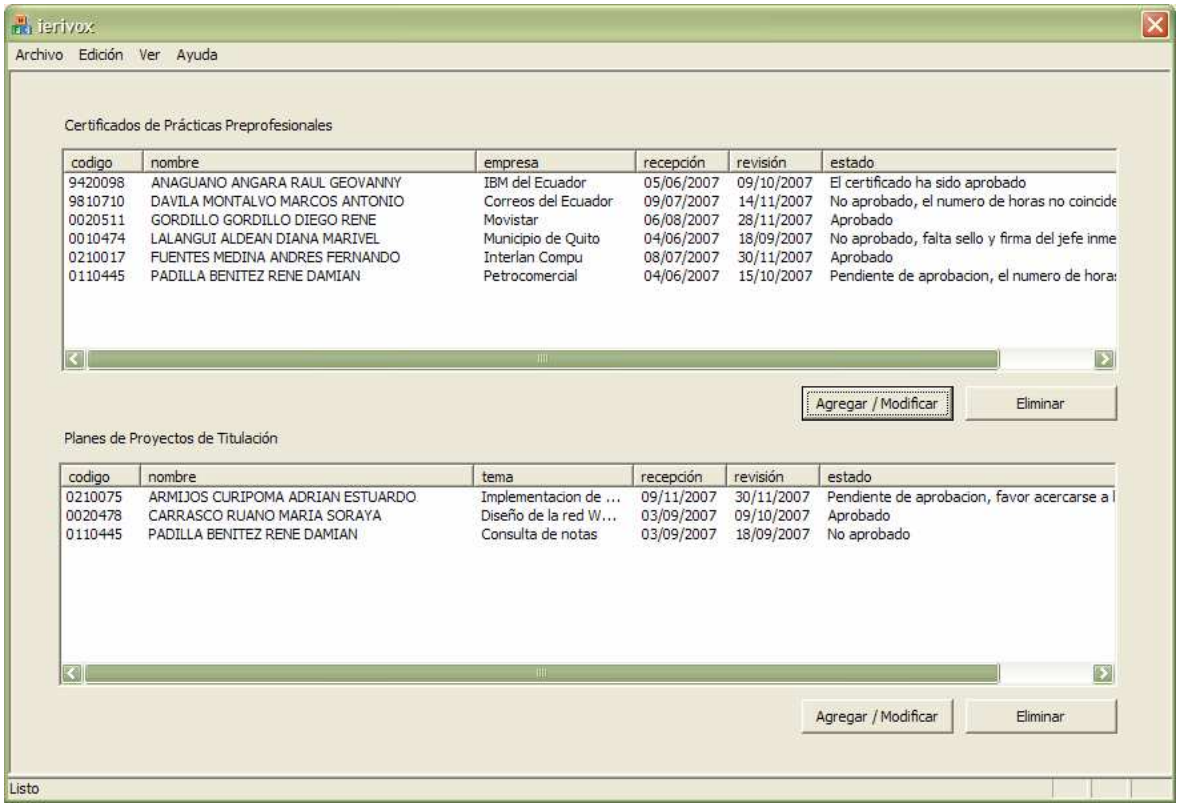

figura F.1: Ventana principal de IERIVOX

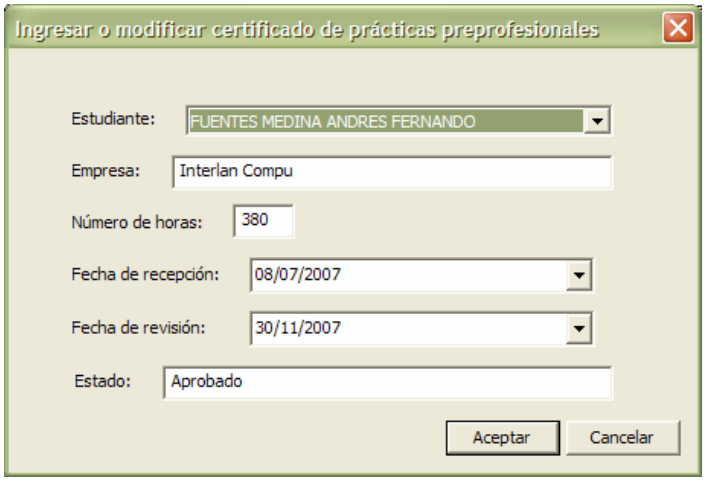

figura F.2: Insertar cerificado de prácticas preprofesionales

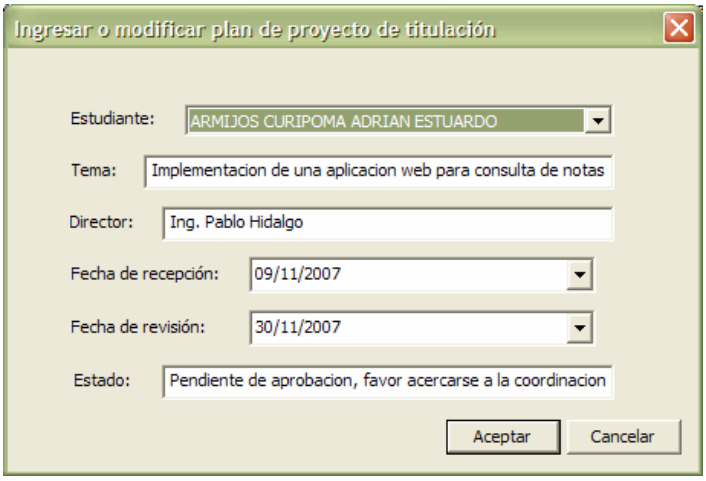

figura F.3: Insertar plan de proyecto de titulación

ANEXO G: DATOS TABULADOS DE LAS PRUEBAS REALIZADAS

# usuario	*primera llamada*			*Segunda llamada*		
	segundos	*asr*	*at*	*Segundos*	*asr*	*At*
Consulta de notas						
1	68	1	si	73	1	Si
2	64	1	si	51	1	Si
3	84	2	si	67	1	Si
4	52	1	si	88	2	Si
5	89	1	si	76	1	Si
6	103	5	si	66	1	Si
7	49	1	si	85	1	Si
8	94	3	si	90	2	Si
9	67	1	si	55	1	Si
10	77	1	si	69	1	Si
Promedio	***74,70***			***72,00***		
Consulta de certificados de prácticas preprofesionales						
1	43	1	si	54	2	Si
3	69	3	si	43	1	Si
3	55	1	si	39	1	Si
4	42	1	si	44	1	Si
5	43	1	si	38	1	Si
6	39	1	si	43	2	Si
7	86	4	si	36	1	Si
8	45	1	si	52	2	Si
9	58	2	si	41	1	Si
10	33	1	si	43	1	Si
Promedio	***51,30***			***43,30***		
Consulta de planes de proyectos de titulación						
1	40	1	si	39	1	Si
2	33	1	si	44	1	Si
3	39	1	si	49	2	Si
4	96	5	si	37	1	Si
5	45	1	si	76	3	Si
6	39	1	si	42	1	Si
7	45	1	si	87	4	Si
8	53	2	si	38	1	Si
9	38	1	si	44	1	Si
10	42	1	si	44	1	Si
Promedio	***47,00***			***50,00***		
Consulta de notas y certificados de prácticas preprofesionales						
1	79	1	si	84	1	Si

2	113	2	si	88	1	Si
3	69	1	si	72	1	Si
4	118	3	si	94	2	Si
5	111	1	si	113	1	Si
6	98	2	si	107	2	Si
7	76	1	si	126	3	Si
8	77	1	si	84	1	Si
9	110	2	si	98	1	Si
10	79	1	si	127	1	Si
Promedio	***93,00***			***99,30***		

Consulta de notas y planes de proyectos de titulación						
1	78	1	si	87	1	Si
2	109	1	si	95	1	Si
3	135	2	si	128	3	Si
4	96	1	si	88	1	Si
5	97	1	si	94	1	Si
6	87	1	si	119	2	Si
7	104	2	si	107	1	Si
8	89	1	si	98	1	Si
9	77	1	si	110	1	Si
10	96	1	si	120	2	Si
Promedio	***96,80***			***104,60***		

Consulta de certificados de prácticas preprofesionales y planes de proyectos de titulación						
1	78	3	si	68	1	Si
2	65	1	si	79	4	Si
3	72	1	si	67	1	Si
4	94	1	si	72	1	Si
5	69	3	si	64	1	Si
6	72	1	si	88	2	Si
7	72	1	si	62	1	Si
8	78	2	si	54	1	Si
9	65	1	si	71	1	Si
10	69	1	si	64	1	Si
Promedio	***73,40***			***68,90***		

Consulta de los tres servicios						
1	111	1	si	122	1	Si
2	129	2	si	132	2	Si
3	142	1	si	176	4	Si
4	102	1	si	129	1	Si
5	115	1	si	142	5	Si
6	144	4	si	109	1	Si
7	134	1	si	110	1	Si
8	103	1	si	119	3	Si
9	127	2	si	106	1	Si
10	123	1	si	123	2	Si
Promedio	***123,00***			***126,80***		

ANEXO H: MANUAL DE MIGRACIÓN DEL SISTEMA A OTRAS CARRERAS

Aunque el IVR de este proyecto fue creado específicamente para la carrera de Electrónica y Redes de Información es completamente factible instalarlo en otra carrera, en este anexo se detallan los pasos necesarios.

La instalación del software es similar, de igual manera los archivos de configuración de Asterisk y *Lumenvox*. Lo que se debe cambiar es la dirección en donde se encuentra la base de datos de los estudiantes. Los archivos de gramática junto con el análisis del número único han sido programados considerando un estudiante en general de la Escuela Politécnica Nacional, razón por la cual tampoco se necesita hacer algún tipo modificación, a excepción de los archivos de gramática por estudiante (*/etc/asterisk/gramar/materias*).

Por lo tanto, el primer paso es crear los procedimientos almacenados en el SAE de la carrera en la que se vaya a instalar el sistema, para esto se deben ejecutar los dos *scripts* de sentencias de comandos SQL del *Anexo B*, y crear un usuario con permisos de ejecución de los procedimientos almacenados y permisos de lectura, escritura y ejecución para la base de datos *PRAPRO*, de la misma manera que se lo hizo en la carrera de Redes de Información. Una vez concluido este paso ya tenemos la conexión a la base de datos.

El siguiente paso es crear los archivos de gramática por estudiante, para esto se necesita editar el archivo */usr/local/agi-java/XMLCreateGrammars.java* que se indicó en el *Anexo E.1* y agregarle las materias adicionales que se dicten en dicha carrera y no en la de Electrónica y Redes de Información, por ejemplo:

```
if(codmat.equals("HSE322")){
      strMat += "<item> Contacto con la Musica Universal " + tag;
      strMat += "<item> Contacto con la Musica " + tag;
}else
```

El código anterior se lo inserta antes de la sentencia *else* final en la función *nombresMateria*. También se debe direccionar la consulta a la base de datos correspondiente, esto se lo hace al inicio del archivo en donde se indica el URL de conexión, se debe cambiar la dirección IP, el nombre de la base de datos, el nombre del usuario y la contraseña, esto se lo hace exactamente en la líneas:

```
// URL de la dirección de la Base de Datos del SAE
String urlDB = "jdbc:jtds:sqlserver://192.168.50.109:1433"+
               "/SAEIERI;user=afuentes;password=andpass;TDS=7.0";
```

Se guardan los cambios en el archivo y se lo compila:

```
# cd /usr/local/agi-java
# javac XMLCreateGrammars.java
```

En los 4 *scripts* del *Anexo E.2* se debe también redireccionar a la nueva base de datos y luego compilarlos de la misma manera como se indicó anteriormente.

Se debe crear también, archivos de audio por cada materia nueva y guardarlos en el directorio (*/var/lib/asterisk/sounds/IERI-IVR/materias*) de la misma manera que se indicó en el *Anexo C.3*.

Esto es todo lo que hay que cambiar, es recomendable reiniciar el sistema antes de ponerlo en producción.

ANEXO I: TABLA DE ERLANG B

N	GOS										N
	0.007	**0.008**	**0.009**	**0.01**	**0.02**	**0.03**	**0.05**	**0.1**	**0.2**	**0.4**	
1	.00705	.00806	.00908	.01010	.02041	.03093	.05263	.11111	.25000	.66667	**1**
2	.12600	.13532	.14416	.15259	.22347	.28155	.38132	.59543	1.0000	2.0000	**2**
3	.39664	.41757	.43711	.45549	.60221	.71513	.89940	1.2708	1.9299	3.4798	**3**
4	.77729	.81029	.84085	.86942	1.0923	1.2589	1.5246	2.0454	2.9452	5.0210	**4**
5	1.2362	1.2810	1.3223	1.3608	1.6571	1.8752	2.2185	2.8811	4.0104	6.5955	**5**
6	1.7531	1.8093	1.8610	1.9090	2.2759	2.5431	2.9603	3.7584	5.1086	8.1907	**6**
7	2.3149	2.3820	2.4437	2.5009	2.9354	3.2497	3.7378	4.6662	6.2302	9.7998	**7**
8	2.9125	2.9902	3.0615	3.1276	3.6271	3.9865	4.5430	5.5971	7.3692	11.419	**8**
9	3.5395	3.6274	3.7080	3.7825	4.3447	4.7479	5.3702	6.5464	8.5217	13.045	**9**
10	4.1911	4.2889	4.3784	4.4612	5.0840	5.5294	6.2157	7.5106	9.6850	14.677	**10**
11	4.8637	4.9709	5.0691	5.1599	5.8415	6.3280	7.0764	8.4871	10.857	16.314	**11**
12	5.5543	5.6708	5.7774	5.8760	6.6147	7.1410	7.9501	9.4740	12.036	17.954	**12**
13	6.2607	6.3863	6.5011	6.6072	7.4015	7.9667	8.8349	10.470	13.222	19.598	**13**
14	6.9811	7.1155	7.2382	7.3517	8.2003	8.8035	9.7295	11.473	14.413	21.243	**14**
15	7.7139	7.8568	7.9874	8.1080	9.0096	9.6500	10.633	12.484	15.608	22.891	**15**
16	8.4579	8.6092	8.7474	8.8750	9.8284	10.505	11.544	13.500	16.807	24.541	**16**
17	9.2119	9.3714	9.5171	9.6516	10.656	11.368	12.461	14.522	18.010	26.192	**17**
18	9.9751	10.143	10.296	10.437	11.491	12.238	13.385	15.548	19.216	27.844	**18**
19	10.747	10.922	11.082	11.230	12.333	13.115	14.315	16.579	20.424	29.498	**19**
20	11.526	11.709	11.876	12.031	13.182	13.997	15.249	17.613	21.635	31.152	**20**
21	12.312	12.503	12.677	12.838	14.036	14.885	16.189	18.651	22.848	32.808	**21**
22	13.105	13.303	13.484	13.651	14.896	15.778	17.132	19.692	24.064	34.464	**22**
23	13.904	14.110	14.297	14.470	15.761	16.675	18.080	20.737	25.281	36.121	**23**
24	14.709	14.922	15.116	15.295	16.631	17.577	19.031	21.784	26.499	37.779	**24**
25	15.519	15.739	15.939	16.125	17.505	18.483	19.985	22.833	27.720	39.437	**25**
26	16.334	16.561	16.768	16.959	18.383	19.392	20.943	23.885	28.941	41.096	**26**
27	17.153	17.387	17.601	17.797	19.265	20.305	21.904	24.939	30.164	42.755	**27**
28	17.977	18.218	18.438	18.640	20.150	21.221	22.867	25.995	31.388	44.414	**28**
29	18.805	19.053	19.279	19.487	21.039	22.140	23.833	27.053	32.614	46.074	**29**
30	19.637	19.891	20.123	20.337	21.932	23.062	24.802	28.113	33.840	47.735	**30**