# Fachhochschule Braunschweig/Wolfenbüttel
# University of Applied Sciences

## Proyecto Final de Carrera
## Editor de texto para el interfaz gráfico de AskoziaPBX

Adriana Arroyo García

Matrikel Nummer: 1008493
NIA: 100055633

Mentor : Prof. Diedrich Wermser
Supervisor: Michael Iedema

Julio 2009

Certifico que, excepto referencia expecífica, el trabajo descrito en este proyecto es original. Tampoco el proyecto en si, ni una parte de él, ha sido presentado con anterioridad en ninguna Universidad.

La autora.

# Índice de contenido

# Índice de figuras

# i Abstracto

Desde que en en los 80 tuviera lugar el boom tecnológico, sus avances han sido aplicados en muchos campos, incluidos las comunicaciones. Mas específicamente, la red de telefonía, que hasta hace poco era analógica, ha evolucionado recientemente a digital.
De este contexto se desprende que las necesidades comunicativas han cambiado. Hasta hace unos años, la mayoría del tráfico era tráfico de voz; sin embargo hoy dicho porcentaje es mínimo, habiendo una creciente demanda de tráfico de datos, gracias a la gran evolución de Internet.
Aquí los datos se convierten en una prioridad, así como su tecnología y sus protocolos. Por esto se empezó a pensar como tratar ambos tráficos y unirlos en uno solo. De aquí viene la idea de trasmitir voz sobre tecnología IP. (VoIP)
A finales de los 90 la tecnología había avanzado lo suficiente como para crear soluciones que permitieran a un ordenador comportarse como una centralita telefónica (PBX), y pocos años después, empezaron a ser ofrecidos servicios extensivos a los usuarios de VoIP.
Por lo tanto, intentamos facilitar el uso de dichas PBX virtuales, creando interfaces gráficas amigables, que no requieran un gran conocimiento sobre como configurar un sistema telefónico.
AskoziaPBX es un proyecto cuyos objetivos son acercar la tecnología VoIP a usuarios medios. Aquí es donde mi proyecto encaja. Con este interfaz gráfico, se dará a posibilidad de mostrar el archivo de configuración del sistema, y poder llevar a cabo los cambios deseados.
En otras palabras, el proyecto esta basado en el desarrollo e integración de un editor empotrado en el interfaz gráfico, para gestionar la configuración de la PBX, de forma que se puedan realizar cambios de forma dinámica. Se explicará en detalle el proceso de integración y reducción del código requerido para convertir mi proyecto en un paquete válido para AskoziaPBX.

## ii Agradecimientos

Agradezco a mi tutor de proyecto, Dr. Diedrich Wermser, que se ocupó de todo a mi llegada y me presentó al equipo del laboratorio. Sus consejos y correcciones han sido cruciales para el buen desarrollo del proyecto. Me gustaría también agradecer a Michael Iedema, líder del proyecto AskoziaPBX, que definió mi proyecto, aprobó la parte técnica y corrigió las versiones finales de la documentación. De la misma forma, doy las gracias a mi tutora en España, Florina Almenárez, quien estuvo siempre disponible para resolver mis dudas.
Gracias también al equipo del laboratorio IKT al completo, especialmente a Akif Dinç y Mathias Krüger, quienes me facilitaron mucho el desarrollo de mi proyecto y ayudaron a adaptarme al equipo a base de café.
No puedo olvidar a mi amiga Elisa Martín-Caro, ya que sin ella, probablemente no sería quien soy ahora, y mi estancia en Alemania no habría sido la misma.
Finalmente quiero agradecer a mi familia y mis amigos que, aunque no todos estuvieron allí siempre me apoyaron y alentaron.

# 1. Introducción

## *1.1 Motivación*

Durante mi estancia en la Universidad, siempre he estado interesada en cómo facilitar y mejorar las cosas a través de la programación. Hoy en día muchas cosas pueden ser conectadas a la red y ser gestionadas desde un ordenador. Esta teoría puede también ser aplicada a la evolución de las centralitas telefónicas (PBX).
Como sabemos, una PBX es una centralita de intercambio telefónico, que sirve a una determinada area. La principal ventaja de estas centrales reside el ahorro en las llamadas internas. Cuando las PBX ganaron popularidad, se comenzaron a ofrecer servicios que no estaban disponibles hasta ahora, como los contestadores, la rellamada, o la identificación de llamada.
En los 90 tuvieron lugar avances significativos en nuevos tipos de PBX. Debido a la creciente demanda de tráfico de datos, las compañías necesitaban redes de conmutación de paquetes para este tráfico, y era una buena idea utilizar esta misma red para cursar las llamadas telefónicas. Además la disponibilidad de Internet como sistema global, hizo de las comunicaciones por conmutación de paquetes una opción muy atractiva. Todos estos factores promovieron el desarrollo de las PBX VoIP. (Técnicamente nada estaba siendo intercambiado (exchanged) pero el término PBX se siguió utilizando por aceptación popular.
Mas específicamente, en 1999 fue distribuida la especificación de SIP (Session Initiation Protocol, RFC 2543), y Mark Spencer, de Digium creó la primera PBX utilizando SIP en código abierto (Asterisk). Cinco años después, en 2004, comenzaron a proliferar los proveedores de servicios comerciales VoIP, hasta hoy.
Asterisk, al igual que cualquier PBX, permite a un numero de teléfonos agregados realizar llamadas entre si, y conectarse también a otros servicios telefónicos como PSTN, RDSI, o otro proveedor VoIP. Originalmente fue diseñado para el sistema operativo Linux, sin embargo en la actualidad podemos encontrar versiones de Asterisk funcionando sobre OpenBSD, FreeBSD, MacOS X, Sun Solaris y Microsoft Windows, aunque la plataforma mejor soportada sigue siendo la originaria. Está diseñado para permitir diferentes tipos de hardware, middleware y software para telefonía IP, y par interactuar entre ellos de forma consistente. Proporciona múltiples niveles, gestionando tanto TDM como los paquetes de voz en los niveles inferiores, mientras que a su vez ofrece una plataforma altamente flexible para la PBX y otras aplicaciones telefónicas como IVR.
Asterisk también puede realizar puentes y traducir diferentes tipos de protocolos VoIP como SIP, MGCP, ó H.323. Al mismo tiempo, proporciona una plataforma de servidor completamente equipada para el marcado predictivo, IVR personalizado o conferencias, entre otros. Asterisk también permite a los desarrolladores modelar nuevos sistemas de telefonía de forma eficiente, o migrar gradualmente las sistemas existentes a la nueva tecnología.
En este contexto se desarrolla el proyecto Askozia, mas específicamente AskoziaPBX, diseñado y desarrollado por el IKT (Institut für Kommunikations Systeme und Technologien).
Con todas estas utilidades es fácil de imaginar lo complejo que puede resultar esta tecnología para un usuario inexperto. Por esta razón, AskoziaPBX pretende poner todo el poder de Asterisk a disposición del usuario medio, a través de un interfaz de usuario amigable. Una PBX empotrada en un PC que simplifica también las actualizaciones, los backups y el provisionamiento.

## *1.2 Objetivos y tareas*

Con el fin de facilitar la realización de cambios en los archivos de configuración del sistema, el equipo de desarrollo tuvo la idea de desarrollar un editor web que, empotrado en el interfaz gráfico de la PBX, permita la edición y guardado dinámico de los archivos. Esto evita tener que actualizar el sistema al completo con cada cambio.
Una vez que el objetivo estuvo definido, se empezó a pensar en como clarificar los archivos y hacerlos mas comprensibles. Por esto, decidimos implementar una solución que resalte la sintaxis de los archivos de configuración de Asterisk.

# 2. Estado del arte

## *2.1 Introducción a las comunicaciones VoIP*

Después de esta introducción, vamos a centrar la explicación en las tecnologías VoIP (Voice over Internet Protocol). Desde que en 1999 Mark Spencer creara Asterisk, el uso de las comunicaciones VoIP creció rápidamente, pero, ¿por qué?
La telefonía IP tiene muchos beneficios. Como hemos dicho en la introducción, el mas importante sería la capacidad de proporcionar comunicaciones de calidad, ademas de incrementar la productividad. La telefonía IP mejora las comunicaciones ofreciendo una manera eficiente y efectiva de comunicarse, escalable y fácil de gestionar. También ofrece ahorros de coste significativos, ya que utiliza la red IP para cursar la voz. A medida que las tecnologías avanzaron, la telefonía IP se convirtió en la alternativa que ofrecía mejor relación calidad-precio.
Usando esta tecnología, una compañía puede abrir una nueva sede y configurar teléfonos con un coste y tiempo de instalación mínimos. De esta forma las compañías optimizan su expansión en el mercado global.
De la misma forma que las redes tradicionales, usamos una PBX que gestiona el tráfico de voz y datos. Son las IP-PBX, un sistema telefónico comercial diseñado para cursar voz o video sobre una red de datos e interoperar con normalidad con la PSTN (Public Switched Telephone Network, en España, POTS). Dichas PBXs pueden existir como un objeto hardware, o virtualmente, como sistema software instalado en un ordenador. AskoziaPBX puede ser encontrado en ambos formatos.

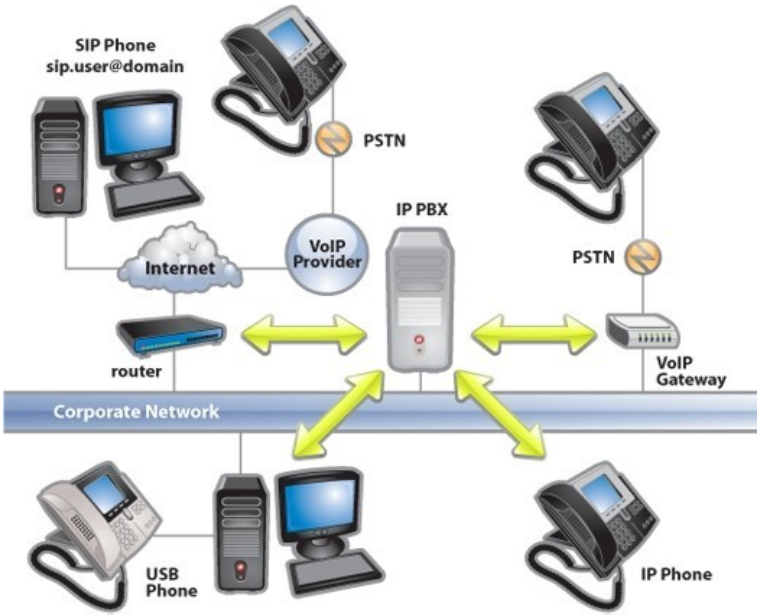


*Figura 1: Uso de una IP-PBX*

Como podemos ver en el dibujo, los gateways VoIP pueden combinarse con la funcionalidad tradicional de una PBX, permitiendo utilizar su intranet, disfrutando de los beneficios de una red única para voz y datos, o por el contrario, puede ser usada únicamente como sistema puro IP, el cual en al mayoría de los casos reducirá ampliamente los costes, ofrecerá movilidad, y aumentará la redundancia y por tanto la robustez del sistema.

## *2.2 El proyecto Askozia*

De acuerdo con estos términos, el equipo de Askozia pensó en una solución que simplificara toda esa complejidad al usuario medio, que pudiera ser también utilizada como una PBX corriente, con valores lógicos por defecto para facilitar la configuración (en otro caso el usuario debería conocer cada opción y como configurarla). Por eso, el sistema al completo es configurable a través de un interfaz web. En caso de que el usuario quiera hacer cambios manuales, o comprobar los ficheros de configuración, deberán revisar un único fichero XML.
El interfaz usa terminología común, e incluye únicamente lo necesario ya que, como hemos dicho, Asterisk puede ser muy complejo. Además, pensando en la arquitectura y futuros desarrollos, AskoziaPBX posee un sistema de paquetes simple, donde resulta fácil añadir funcionalidades al firmware existente.

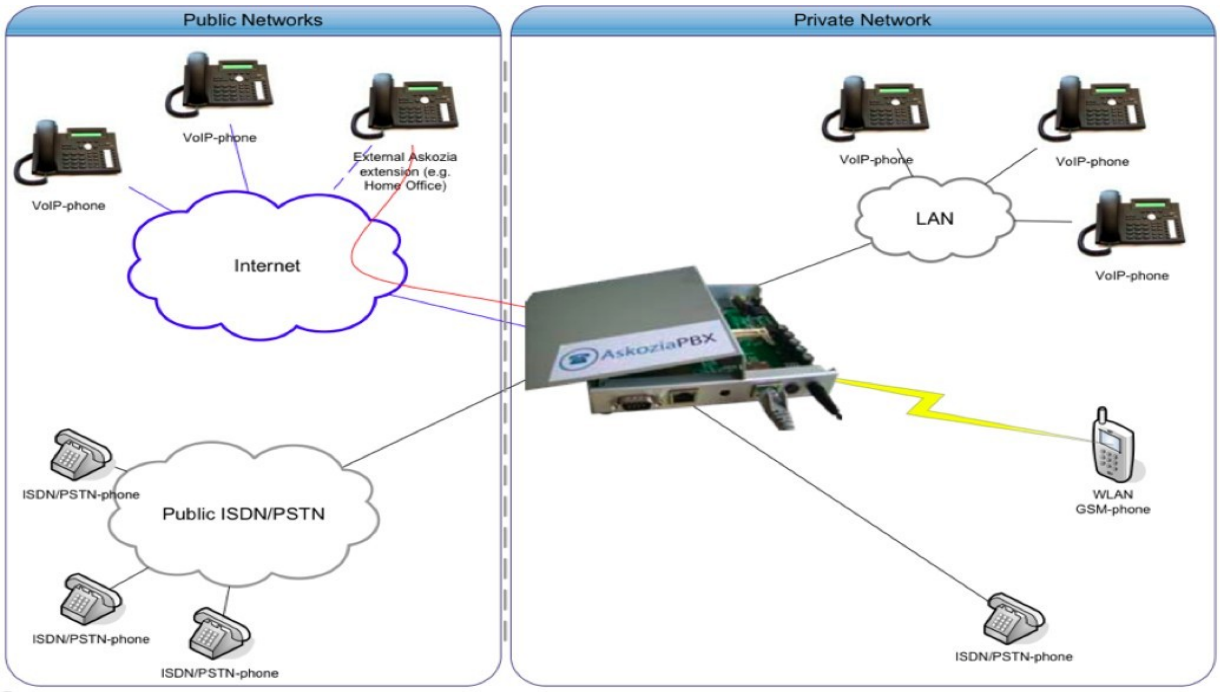


*Figura 2: Posible uso de AskoziaPBX*

### 2.2.1 El interfaz gráfico

Una de las mayores ventajas de la simplicidad de Askozia es que puede ser gestionada completamente a través de un interfaz gráfico amigable e intuitivo, con explicaciones, como se muestra a continuación.


*Figura 3: System Information en AskoziaPBX GUI*

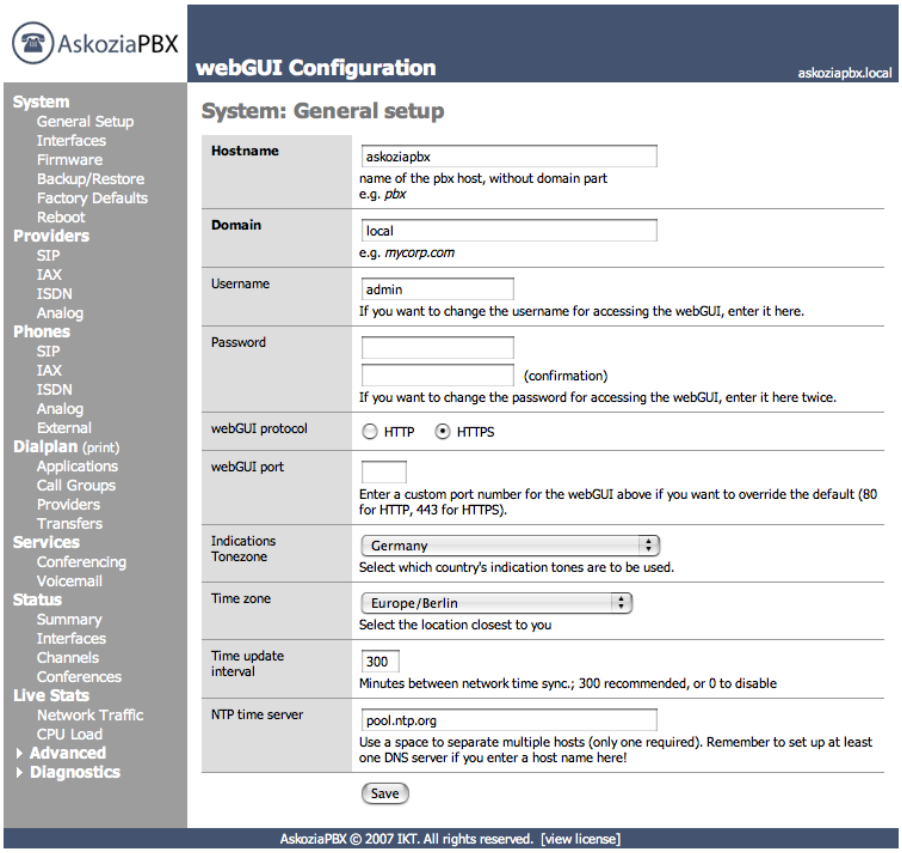


*Figura 4: General Setup en AskoziaPBX GUI*

Nuestro objetivo es que haya varias posibilidades para variar la configuración de la PBX. Hasta ahora, los cambios se podían realizar a través del interfaz gráfico o bien descargando el archivo de configuración, modificándolo, resubiendo el archivo y reiniciando el sistema. Esta ultima posibilidad puede ser un poco pesada cuando queremos hacer varios cambios y comprobar sus efectos, ya que puede tomar mas tiempo del deseado.
Una vez explicado el entorno en el que trabajamos, podemos aclarar donde encaja nuestro proyecto. Como hemos explicado, si el usuario quisiera realizar cambios de forma manual deberá buscar el archivo específico en el árbol de directorios. Se pretende que estos ficheros sean accesibles directamente, y dar la posibilidad de mostrarlo, editarlo, y guardar los cambios o descartarlos. Resumiendo, implementaremos un editor empotrado en el interfaz web con las capacidades enunciadas arriba.

### *2.3 Editores web*

Existe un amplio rango de posibilidades en lo referente a la edición de texto web, pero buscábamos algo compacto, con pocos archivos, con el fin de empaquetarlo fácilmente y que encaje lo mejor posible en nuestro sistema empotrado. Finalmente, y después de considerar varias opciones, como FCKEditor, decidimos usar Codemirror, programado en Javascript.
Elegimos el editor de acuerdo con esta tabla de características:

| | Size | Integration | Highlighting oriented | Document oriented | Code oriented |
|---|---|---|---|---|---|
| CodeMirror | 288 KB | Web oriented | ✓ | ✗ | ✓ |
| FCKEditor | 5'2 MB | Web oriented | ✗ | ✓ | ✗ |

*Figura 5: Comparativa entre Codemirror y FCKEditor*

Esta librería puede ser utilizada para crear un interfaz editor relativamente agradable para lenguajes de programación, HTML, marcado y similares. Además, si implementamos un parser para el lenguaje que utilicemos, el código aparecerá coloreado según le indiquemos y también lo indentará. Este es un ejemplo de uso de FCKEditor:

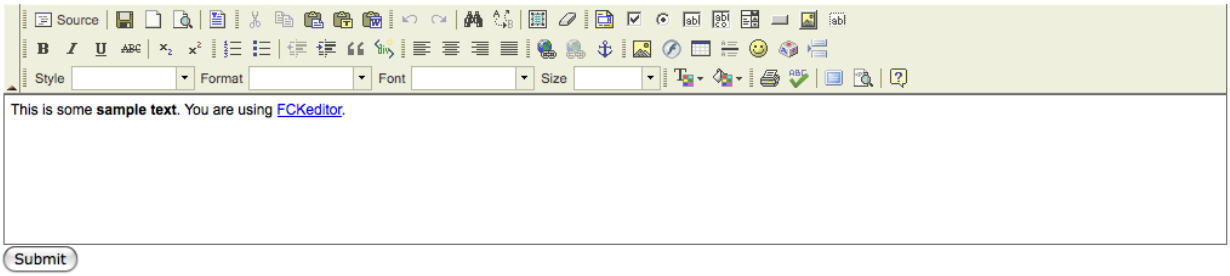


*Figura 6: FCKEditor*

Y aquí tenemos algunos ejemplos reales de uso de la librería Codemirror:

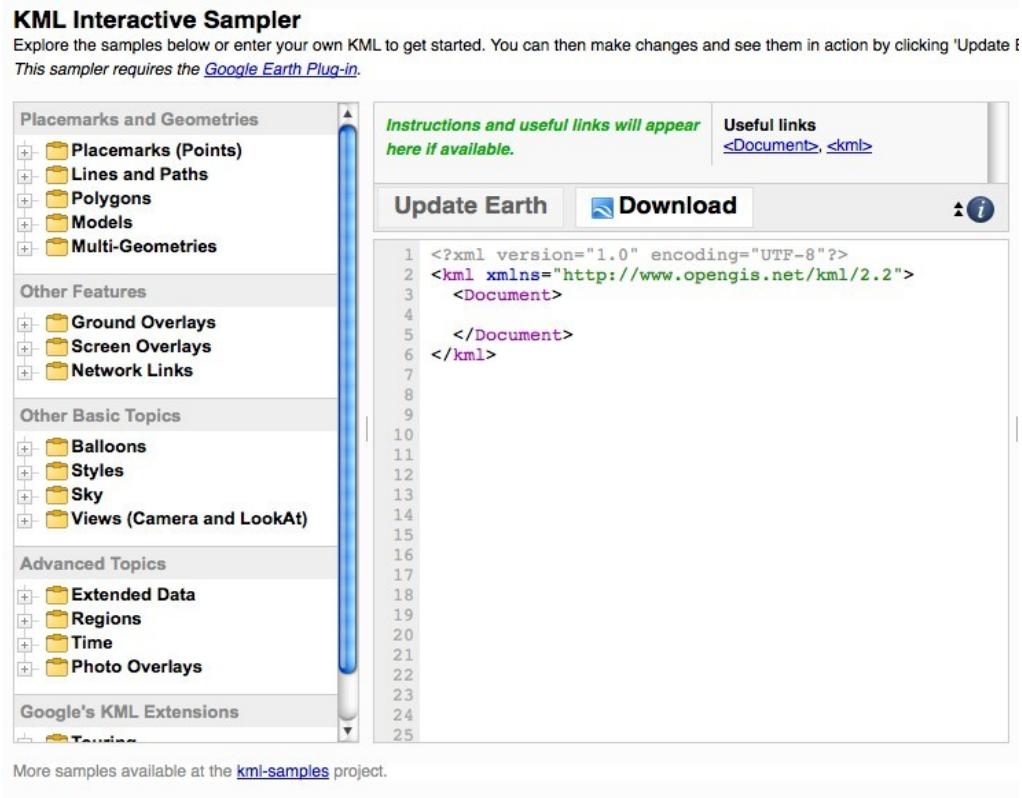


*Figura 7: Google KLM Interactive Sampler*

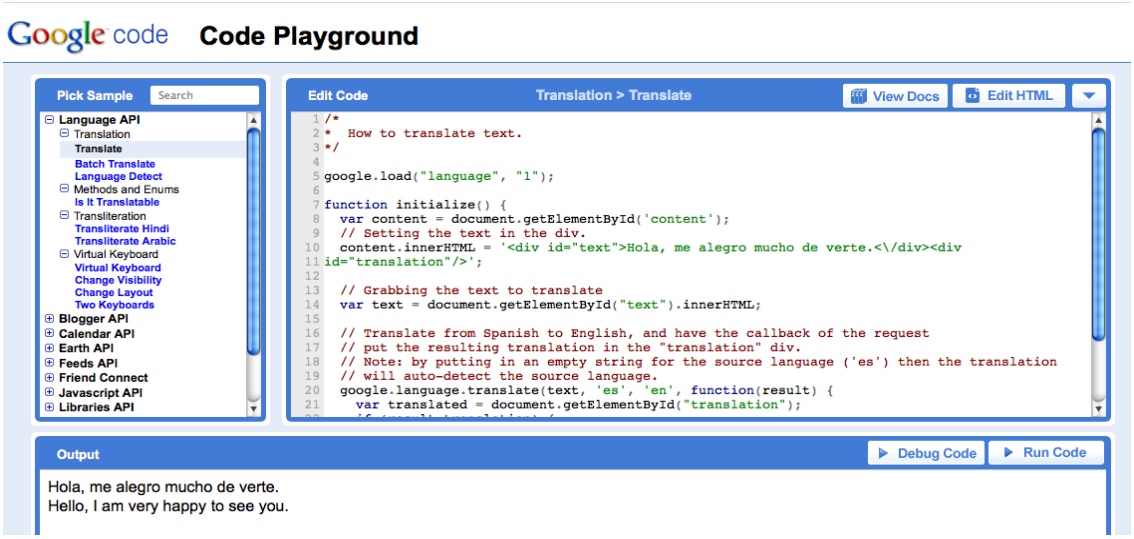


*Figura 8: Google Code Playground*

Además Codemirror funciona con los principales exploradores de internet: Firefox 1.5, Internet Explorer 6, Safari 3, Opera 9.52 (o posteriores), y Chrome.

## 3. Descripción general

En las siguientes páginas será expuesta la descripción general del proyecto. Se mostrará cómo funciona Codemirror, cómo se adaptó al código de AskoziaPBX, y finalmente, detallaremos cómo fue escrito el parser que nos ayuda a resaltar y comprender mejor la sintaxis de los archivos de configuración de Asterisk.

### *3.1 ¿Cómo funciona Codemirror?*

Para usar Codemirror en un documento, debemos añadir un tag <script> en la cabecera de nuestra página, que cargue codemirror.js. Esto añade dos objetos al entorno de desarrollo, Codemirror y CodemirrorConfig. El primero es el interfaz para el editor y el segundo puede ser utilizado para configurarlo.

```
<script src="js/codemirror.js" type="text/javascript"></script>
```

*Figura 9: Incluimos el script Codemirror*

Para añadir el editor a un documento se debe elegir un lugar, un parser, y una hoja de estilo que se le aplique. Existen muchas opciones para colocarlo, como sustituyendo un nodo DOM, o creándolo como un nodo DOM por si mismo. Tras considerarlo, nos dimos cuenta de que necesitábamos mostrar un pedazo de código que debía ser editable, por lo que pensamos en un textarea. Además, otra de las opciones de colocación de Codemirror es Codemirror.fromTextArea, que dado un nodo textarea, esconde dicho textarea y lo sustituye por el frame editor Codemirror. Si el textarea es parte de un formulario no habrá ningún problema y será registrado como tal, por lo que los datos pueden ser enviados normalmente. Esta parte soporta nuestros requerimientos, ya que necesitaremos enviar el contenido del textarea a un script php que efectué los cambios pertinentes en la configuración.
Por ejemplo en nuestro caso, para añadir un editor de archivos de configuración Asterisk en el cuerpo del documento:

```
<script type="text/javascript">
  var editor = CodeMirror.fromTextArea('code', {
    height: "350px",
    parserfile: [ "parsea.js"],
    stylesheet: ["../../css/acolors.css"],
    path: "../../js/",
    parserConfig: {
        indentUnit: 4
    },
    continuousScanning: 500,

  });
</script>
```

*Figura 10: Añadimos el editor a nuestra web*

El primer argumento para el constructor de Codemirror puede ser un nodo DOM, en cuyo caso el editor queda pendido de ese nodo, o una función, que será llamada con el nodo IFRAME como argumento, y de la que se espera que sitúe ese nodo en algún lugar en el documento.
El segundo argumento (opcional) es un objeto que especifica opciones. CodemirrorConfig posee un conjunto de opciones por defecto, pero cada instancia del editor puede tener características específicas que sobrescriban las dadas. En nuestro caso, especificamos que el editor debe usar dos

archivos específicos: el parser parsea.js, y la hoja de estilo para resaltar la sintaxis acolors.css.

La razón por la que el path al script tiene que ser configurado es que codemirror cargará además una serie de fichero que contienen utilidades que pueden ser aplicadas al editor. Para poder añadirlos y utilizarlos correctamente debe saber donde encontrarlos. Estos son los ficheros javascripts que son parte de Codemirror:

- codemirror.js : Interfaz principal, se ocupa de la configuración por defecto y de definir las frames del editor. Debe incluirse en el documento HTML.
- editor.js : Es el código que detecta el input de usuario, coloreándolo e indentándolo.
- util.js : Algunas funciones genéricas.
- undo.js : Implementa el historial de deshacer del editor.
- stringstream.js : Objeto que traslada el input de usuario al parser.
- select.js : Algunas utilidades de ayuda para trabajar con texto seleccionado y posiciones de cursor.
- tokenize.js : ayuda para la construcción de tokenizers.

### 3.2 Arquitectura software

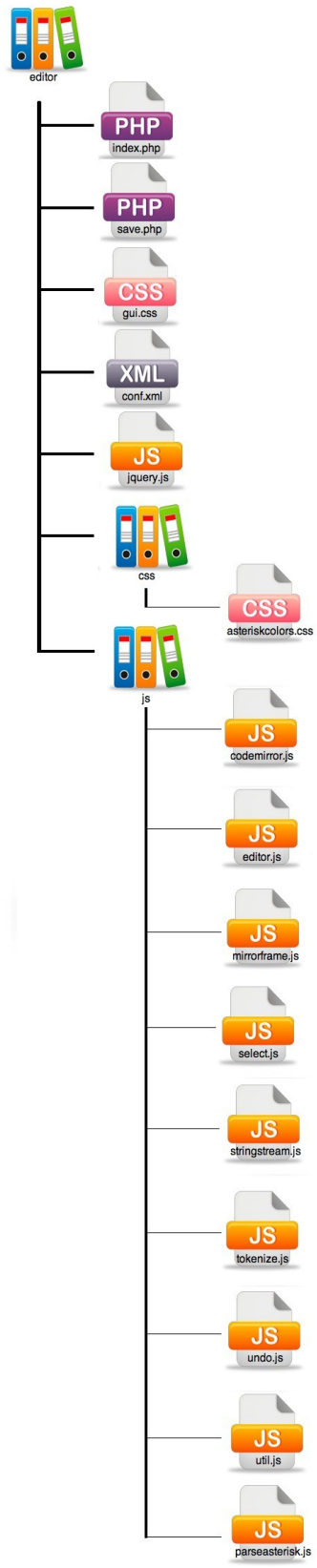


*Figura 11: Esquema de directorio*

Leyenda:

- editor: directorio principal.
  - index.php : pagina web del interfaz gráfico donde esta emplazado el editor.
  - save.php : script que salva los cambios del archivo
  - gui.css : hoja de estilo del interfaz gráfico de AskoziaPBX.
  - jquery.js
- css: directorio de hojas de estilo
  - asteriskcolors.css : colores para resaltar la sintaxis de los archivos de configuración de Asterisk.
- js : directorio javascript
  - codemirror.js : Interfaz principal, se ocupa de la configuración por defecto y de definir las frames del editor. Debe incluirse en el documento HTML.
  - editor.js : Es el código que detecta el input de usuario, coloreándolo e indentándolo.
  - util.js : Algunas funciones genéricas.
  - undo.js : Implementa el historial de deshacer del editor.
  - stringstream.js : Objeto que traslada el input de usuario al parser.
  - select.js : Algunas utilidades de ayuda para trabajar con texto seleccionado y posiciones de cursor.
  - tokenize.js : ayuda para la construcción de tokenizers.
  - parseasterisk.js : js que hace tokens y parsea los archivos de configuración de Asterisk, y los colorea de acuerdo a asteriskcolors.js

De acuerdo con la arquitectura de AskoziaPBX, vamos a configurar el código de nuestro editor en un archivo por separado, con otras funciones comunes, con el fin de que pueda ser añadido en cualquier página las veces que sea necesario fácilmente.

```
<?php
function open_editor($fileroute) {
    ?><span id="editor_wrapper">
        <form id="editor_form" action="save.php" method="post" enctype="multipart/form-data">
            <textarea id="code" name="code" cols="120" rows="30">
                        <?php echo htmlspecialchars(file_get_contents($fileroute));?>
            </textarea>
            </br>
            <input name="Save" type="submit" class="formbtn" id="save" value="Guardar configuración">
            <input name="Cancel" type="submit" class="formbtn" id="cancel" value="Cancelar cambios">
        </form>

        <script type="text/javascript">
          var editor = CodeMirror.fromTextArea('code', {
            height: "350px",
            parserfile: [ "parsea.js"],
            stylesheet: ["../../css/acolors.css"],
            path: "../../js/",
            lineNumbers: true,
            parserConfig: {
                indentUnit: 4
            },
            continuousScanning: 500,

          });
        </script>
        </span><?php
}

?>
```

*Figura 12: open_editor() en editor.inc*

Debemos también incluir este archivo en cada página que use el editor, como se indica a continuación:

```
<?php
include_once("./editor.inc");
```

*Figura 13: Incluimos editor.inc*

Como dijimos con anterioridad, este código puede constituir un archivo propio, o formar parte de un archivo común de funciones. La función incluye el código en la página, que llama a la función open_editor($filename), con la ruta del archivo a mostrar como argumento.
En el ejemplo arriba utilizamos la funcionalidad de Codemirror que transforma el contenido de un textarea en un editor. Podemos apreciar que se encuentra dentro de un formulario, que enviará el contenido, y el textarea, que se rellenará con el archivo que tenemos intención de visualizar ó modificar.
Finalmente incluye el script de Codemirror, que utiliza el textarea como parámetro(“code”), y también opciones de configuración como el tamaño del editor, el archivo parser ó el archivo css que contiene los colores que se usarán en el resaltado de la sintaxis, si se mostraran los números de linea, las unidades de indentación ó los mili segundos entre actualizaciones de resaltado.

```
<table width="100%" border="0" cellpadding="10" cellspacing="0">
    <tr>
        <td><p class="pgtitle">Editar <?php echo $filename ?> <a
            <div id="pghelp" class="pghelp" name="pghelp">The en

            <?php open_editor($filename);?>

        </td>
    </tr>
</table>
```

*Figura 14: ¿Como empotramos el editor?*

De esta forma, cunado entramos en el interfaz de usuario de AskoziaPBX, y nos dirigimos a la página donde se encuentra nuestro editor obtenemos:

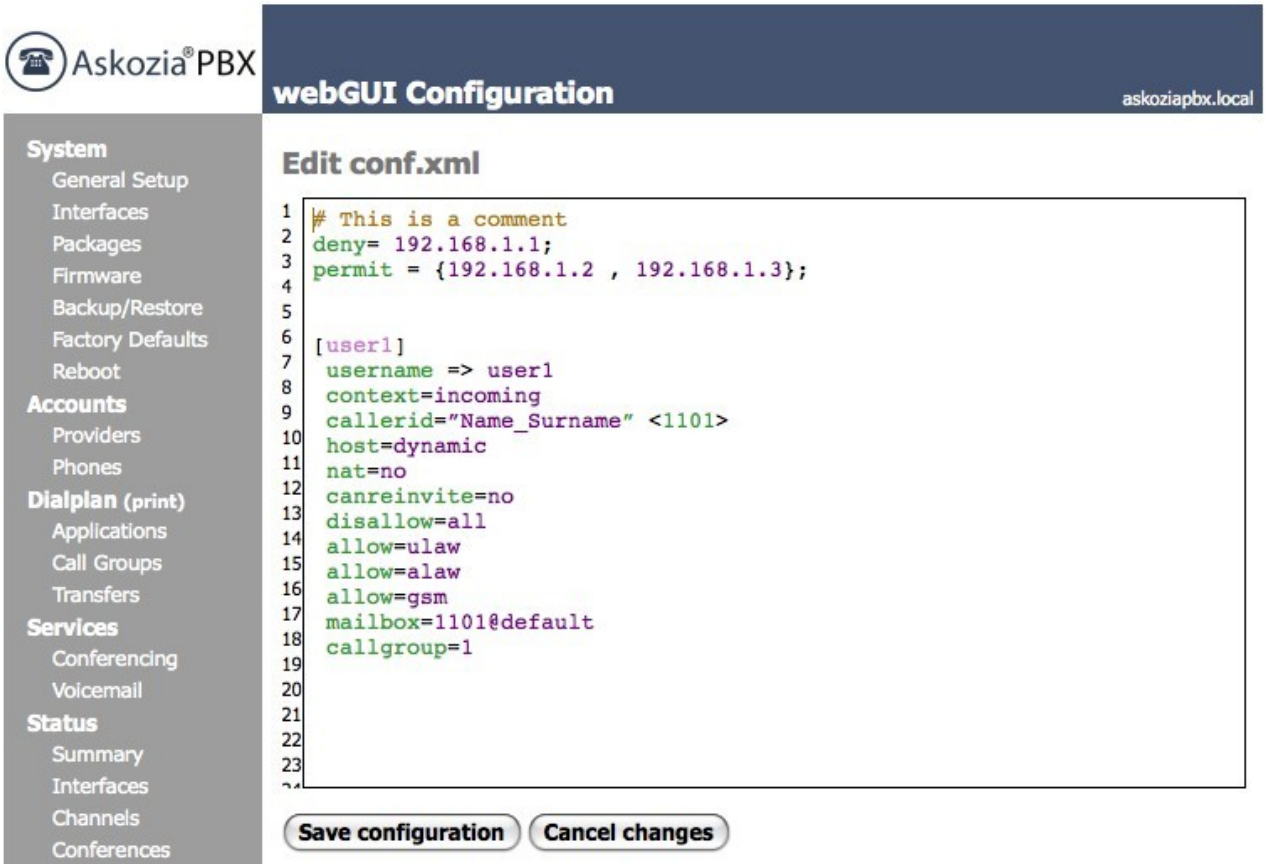


*Figura 15: Vista previa del editor*

## *3.3 Adaptación a sistemas empotrados con poca memoria disponible.*

Habiendo definido qué es lo que queremos que haga nuestro editor, es hora de borrar algunos archivos que no necesitaremos, y que pueden molestar ya que, como dijimos antes, buscamos una solución que encaje en un sistema empotrado.
El paquete Codemirror con el que comenzamos a trabajar, contiene funcionalidades y parsers para varios lenguajes como PHP, Perl, o JavaScript. Realmente no necesitamos ninguno de ellos. A pesar de que no son demasiado grandes, los borramos ya que son prescindibles para nuestra aplicación. El paquete original también contiene numerosas hojas de estilo css para dichos lenguajes. Sin embargo, estos archivos que hemos descartado serán muy útiles a la hora de diseñar nuestro propio parser y hoja de estilo, ya que nos sirven de guía y ejemplo.
Finalmente, nuestro paquete contará únicamente con los ficheros mostrados en la figura 11.

## *3.4 Resaltado*

Después de definir el objetivo principal del proyecto, y una vez que el editor se integró y funcionó perfectamente en la demostración, empezamos a clarificar los que aparecía en el textarea. El texto al completo aparecía en texto plano, con el mismo color de fuente, y la distinción entre partes del código no era inmediata.
Por esta razón, y gracias a que Codemirror nos permite crear nuestros propios parsers, con su css asociado y aplicarlos al texto, decidimos desarrollar un parser simple para los archivos de configuración de Asterisk.

### 3.4.1 El parser

Una vez que se ha explicado cómo funciona Codemirror, es fácil saber donde encajaría nuestro nuevo parser. Veremos y aclararemos cómo funciona nuestro parser para Asterisk y como es aplicado el css que diseñamos.
Primero definimos la sintaxis que pretendemos resaltar:

- Comentarios: # …
- Clasificadores: […], o (…), por ejemplo [user1]
- Valores clave: … = …

Puede parecer poco en un principio, pero la mayoría de los archivos de configuración de Asterisk están reducidos esencialmente a eso.

A continuación vemos en detalle el código explicado:

```
/*
Asterisk Parser
*/

var AParser = Editor.Parser = (function() {
  var tokenizeA = (function() {
    valueExpected = false;
    classExpected = false;
```

*Figura 16: Parser Asterisk. Tokenizer (1)*

Primero creamos el parser, y lo definimos como función, la cual implementará todos nuestros requerimientos a la hora de resaltar el texto. Dentro, nuestra primera tarea es dividir el código en tokens.
Entonces podremos distinguir si estamos esperando un valor o un clasificador.

Después se suceden las funciones de parseo:

```
function normal(source, setState) {
  var ch = source.next();

  if (ch == "=") {
  valueExpected = true;
    return "A-compare";
  }
  else if (ch == "#") {
    source.nextWhile(matcher(/\w/));
    setState(inComment)
    return null;
  }
  else if (/[+*\/]/.test(ch)) {
    return "A-select-op";
  }
  else if (/[,;{}:<>]/.test(ch)) {
    return "A-punctuation";
  }
  else if (ch == "[" || ch == "(") {
    classExpected = true;
    return "A-punctuation";
  }
  else if (ch == "]" || ch == ")") {
    classExpected = false;
    return "A-punctuation";
  }
  else if (ch == "\"" || ch == "'") {
    source.nextWhile(matcher(/[\\\w]/));
    setState(inString);
    return null;
  }else{
    source.nextWhile(matcher(/[\w\\\-_.@]/));
    if(valueExpected){
        valueExpected = false;
        return "A-value";
    }
    else if(classExpected){
        classExpected = false;
        return "A-clasif";
    }else
        return "A-identifier";
  }
}
```

*Figura 17: Parser Asterisk. Tokenizer (2)*

Esta es la función de parseo del estado normal. El parser la llamará a no ser que se encuentre un comentario o un string. Dividirá el código en tokens de acuerdo con su funcionalidad y devolverá una marca que identificará el tipo de token encontrado, como podemos ver en la figura 17.

```
function inComment(source, setState) {
  while (!source.endOfLine()) {
    var ch = source.next();
    }
      setState(normal);

  return "A-comment";
}


function inString(quote) {
  return function(source, setState) {
    var escaped = false;
    while (!source.endOfLine()) {
      var ch = source.next();
      if (ch == quote && !escaped)
        break;
      escaped = !escaped && ch == "\\";
    }
    if (!escaped)
      setState(normal);
    return "A-string";
  };
}
```

*Figura 18: Parser Asterisk. Tokenizer (3)*

Tendremos varias funciones específicas para tratar los tokens especiales, como comentarios o strings.

```
return function(source, startState) {
  return tokenizer(source, startState || normal);
};
```

*Figura 19: Parser Asterisk. Fin de la función tokenizer*

Cuando no se ha encontrado nada especial, entonces devolvemos el estado normal, y normal ejecuta de nuevo.

Ahora vamos a parsear los tokens de acuerdo con los identificadores que hemos estado devolviendo:

```
function parseA(source, basecolumn) {
  basecolumn = basecolumn || 0;
  var tokens = tokenizeA(source);
  var inBraces = false, inRule = false;
```

*Figura 20: Parser Asterisk. Inicio del parseo*

Primero comprobamos la columna base (para la indentación) y obtenemos los tokens hechos arriba.

Los objetos token representan una porción de código significante, del texto que se edita. Dicho objeto debe tener un valor y una propiedad en al hoja de estilo css para que pueda ser coloreado. Puede ser cualquier cosa excepto un espacio en blanco al principio de linea, que debe ser llamado “whitespace” y tener una clase con su nombre en la hoja de estilo. El editor debe ser capaz de reconocerlos cuando indenta las lineas. Además de eso, cada carácter de nueva linea debe contar con su propio token, ya que de esta forma sabremos cual es el nivel optimo de indentación para esa linea.

```
var iter = {
  next: function() {
    var token = tokens.next(), style = token.style, content = token.content;

    if (style == "A-identifier" && inRule)
      token.style = "A-value";

    if (content == "\n")
      token.indentation = indentA(inBraces, inRule, basecolumn);

    if (content == "{")
      inBraces = true;
    else if (content == "}")
      inBraces = inRule = false;
    else if (inBraces && content == ";")
      inRule = false;
   else if (inBraces && style != "A-comment" && style != "whitespace")
      inRule = true;

    return token;
  },
```

*Figura 21: Parser Asterisk. Iterador*

Este iterador debe también tener un método de copia. Este método, llamado sin argumentos, devuelve una función que representa el estado actual del parser.

```
  copy: function() {
    var _inBraces = inBraces, _inRule = inRule, _tokenState = tokens.state;
    return function(source) {
      tokens = tokenizeA(source, _tokenState);
      inBraces = _inBraces;
      inRule = _inRule;
      return iter;
    };
  }
};
return iter;
}
```

*Figura 22: Parser Asterisk. Método copia*

Cuando es llamada esta función de estado con un string como argumento, devuelve un objeto parser que continua parseando usando el antiguo estado y el nuevo string introducido para parsear. Puede asumirse que solo un parser puede estar actuando simultáneamente.

```
  return {make: parseA, electricChars: "}"};
})();
```

*Figura 23: Parser Asterisk. Make*

Cuando el método make es invocado, devuelve un iterador MochiKit-style; objeto con un método next() que parará la iteración cuando esta se encuentre en su final.

Un parser está implementado por uno o más archivos que, cuando se cargan, añaden un objeto Parser al objeto editor definido por editor.js. Este objeto debe implementar el siguiente interfaz:

- make (stream): función que, dedo un string, crea un parser. El comportamiento de dicho parser queda explicado arriba.
- ElectricChars: string opcional que contiene los caracteres que, cuando se encuentran, provocan la indentación de la linea. Por ejemplo {}.

- configure(object): función opcional que puede ser usada para configurar el parser, si existe y el editor tiene la opción de parserConfig, será invocado con el valor de dicha opción.

### 3.4.2 El archivo css

A continuación vemos como esta compuesto el archivo css:

```
.editbox {
  margin: .4em;
  padding: 0;
  font-family: monospace;
  font-size: 10pt;
  color: black;
}
pre.code, .editbox {
  color: #666666;
}
.editbox p {
  margin: 0;
}
span.A-value {
  color: #770088;
}
span.A-identifier {
  color: 339933;
}
span.A-colorcode {
  color: #004499;
}
span.A-comment {
  color: #AA7700;
}
span.A-string {
  color: #AA2222;
}
span.A-select-op {
    color: black;
}
span.A-punctuation{
    color: black;
}
span.A-compare{
    color: black;
}
span.A-clasif {
  color: #CC77CC;
}
```

*Figura 24: Hoja de estilo para el resaltado del código Asterisk*

Los colores pueden ser fácilmente modificados cambiando este archivo.

### *3.5 Unión con el código existente en AskoziaPBX*

Como dijimos en capítulos anteriores, necesitábamos empaquetar nuestro código de forma que encajara bien en un sistema empotrado, al que se le presupone poca memoria. Por esa razón, prescindimos de muchos archivos del paquete original de Codemirror que no eran necesarios, como los parsers de Javascript o PHP. Dejamos los archivos que contienen las funciones comunes, como undo.js y util.js. Aunque en principio la mayoría de la funcionalidad que ofrecen no se usa, decidimos incluirlos pensando en una posible expansión y mejora del código.
El esquema final aparece en la figura 11.

## 4. Mejora de AskoziaPBX GUI. Resultados finales

Realmente AskoziaPBX ya tenía un editor que abría archivos a través de su ruta en el sistema, y también editaba y guardaba los cambios. A pesar de eso, este editor producía errores cuando se abrían archivos php ( llaves mal cerradas producían desorden en todo el documento)

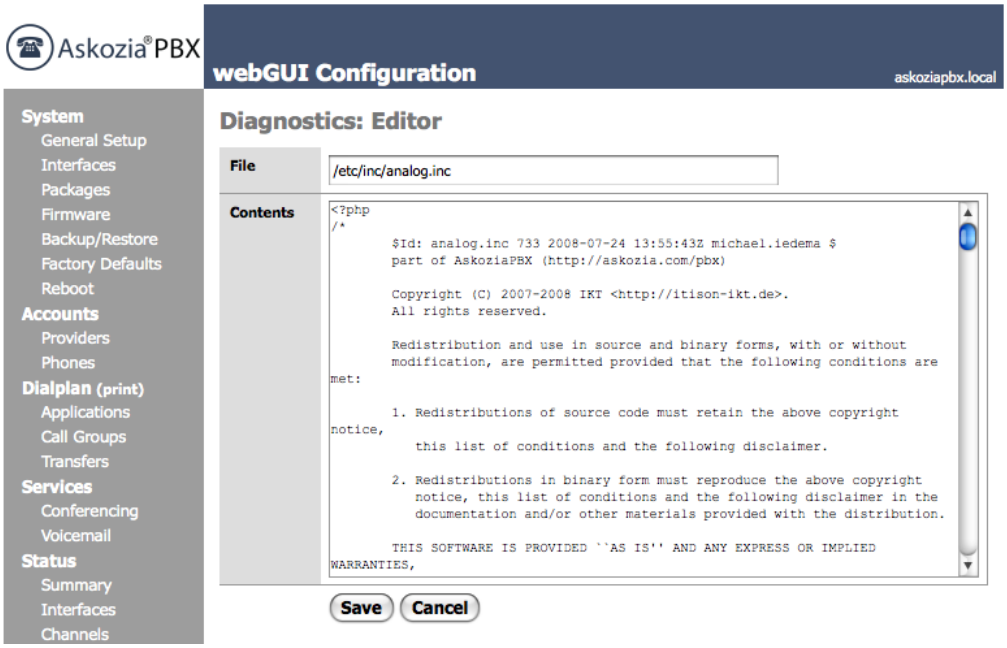


*Figura 25: Antiguo editor funcionando correctamente*

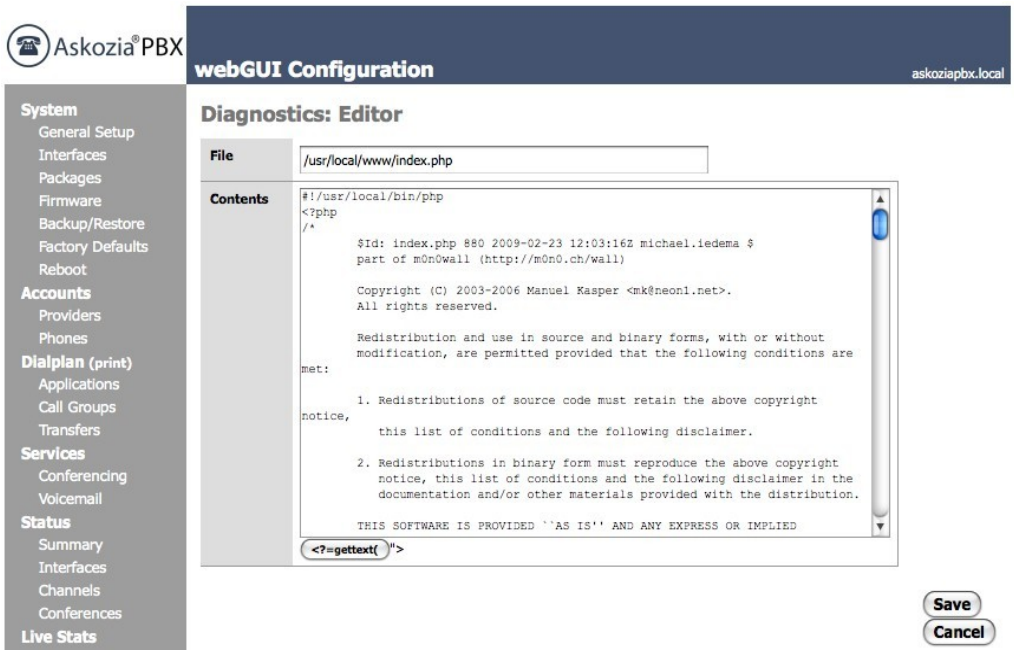


*Figura 26: Antiguo editor funcionando mal*

Como podemos ver en las figuras 25 y 26, el editor lleva a cabo su tarea de forma irregular cuando se abren archivos php. Y si esto ocurre la pagina que contiene el editor se desordena por completo. El nuevo editor es robusto en cuando a problemas de anidamiento, ya que deriva la cuestión a Codemirror.

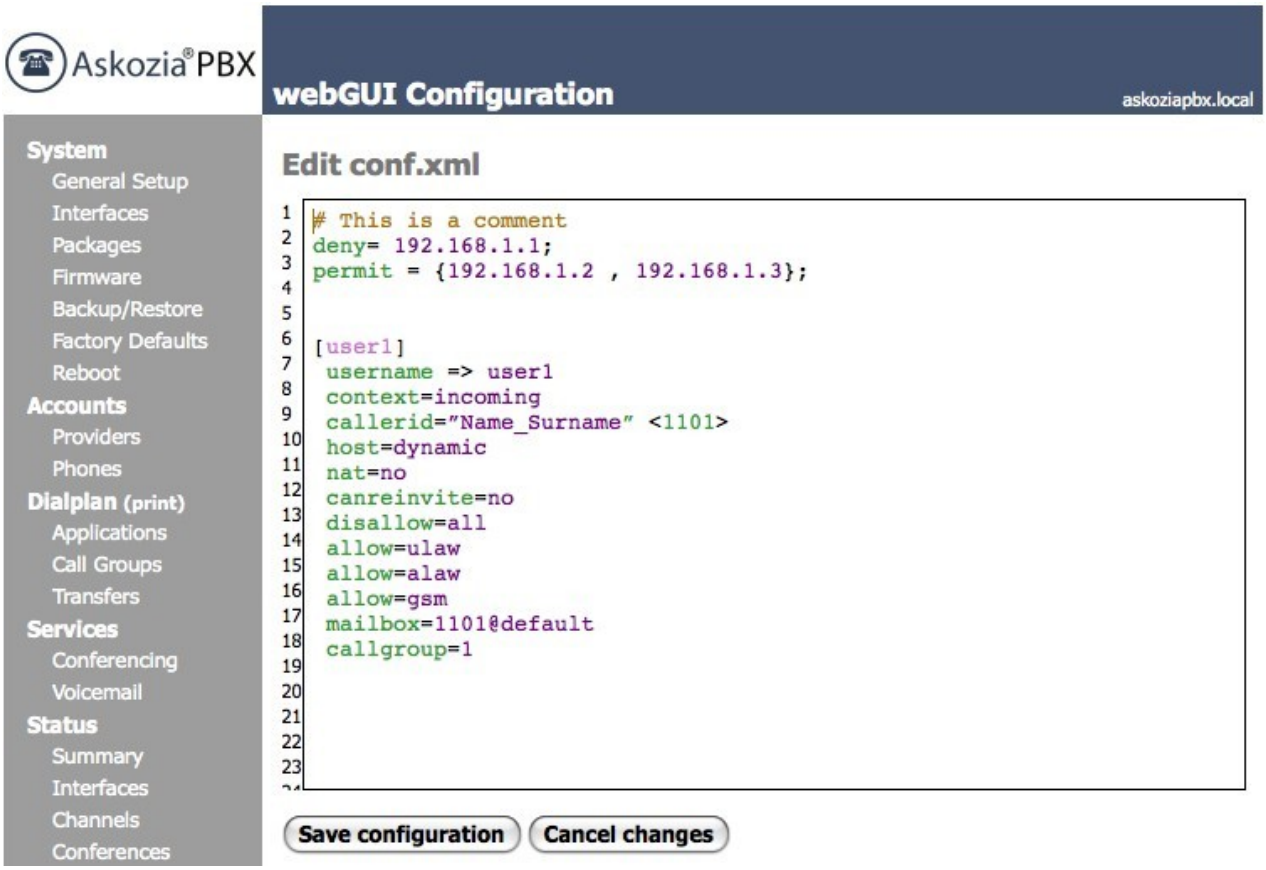

*Figura 27: Nuevo editor*

Codemirror trata el contenido del textarea como texto plano en un principio ( después lo coloreará y lo indentará de acuerdo con el parser y la hoja de estilo dadas). Esta es la razón por la cual no se preocupa de las llaves, y conserva la página en perfecto estado.
Este proyecto será incluido en la versión AskoziaPBX 2.0
https://wush.net/trac/askozia/ticket/19#comment:1

## 5. Conclusiones y trabajo futuro

Como conclusión, podemos resumir que el editor hará que la modificación de los archivos de configuración sea mucho mas fácil, flexible y visual, mediante el uso de Codemirror, herramienta open source ampliamente soportada y utilizada por numerosas aplicaciones como Google Code Playground.

Encontré dificultad en la programación del parser para archivos de configuración de Asterisk, ya que no me había enfrentado nunca a su sintaxis, y supuso un esfuerzo extra también, el hecho de comprender los parsers de ejemplo, ya que eran bastante mas complicados de los que yo necesitaba hacer.

En el aspecto personal, durante mi estancia en Wolfenbüttel, he aprendido cómo trabajar y relacionarme en un equipo internacional, y ha sido una gran experiencia para mi. Por otro lado, en el aspecto técnico, aprendí a implementar código acorde con lo ya hecho, y encajarlo en un proyecto funcionando. Además mejoré notablemente mis destrezas con Javascript, PHP, e interfaces web.

El editor puede completarse y mejorarse de muchas maneras. En este momento sólo están implementadas unas pocas opciones, mostrar y editar un fichero, y guardar o descartar los cambios. Codemirror tiene un archivo extra llamado undo.js que ha sido incluido en el paquete de acuerdo a esta posibilidad de expansión en futuras versiones, ya que ofrece la posibilidad de deshacer el último cambio hecho en el archivo. Como trabajo futuro puede ser implementada esta funcionalidad, ya que encaja en el proyecto, dando la posibilidad de descartar los cambios de forma individual. En este momento, una vez que se acometen los cambios, no es posible deshacerlos.

Concluimos afirmando que este proyecto no es crucial para el funcionamiento del sistema, pero supone una gran ayuda para aquellos que lo usan regularmente e intenta mejorar y hacer mas confortable el uso de AskoziaPBX.

## 6. Referencias


[1] http://marijn.haverbeke.nl/codemirror/ : CodeMirror In-browser code editing.

[2] http://marijn.haverbeke.nl/codemirror/manual.html : CodeMirror manual.

[3] http://groups.google.com/group/codemirror?pli=1 : Group/mailing list for the CodeMirror in-browser code editor.

[4] http://kml-samples.googlecode.com/svn/trunk/interactive/index.html : Google KLM interactive sampler.

[5] http://code.google.com/apis/ajax/playground/ : Google Code Playground.

[6] http://www.askozia.com/ : Askozia Project.

[7] http://www.askozia.com/pbx/screenshots/ : Askozia Screenshots.

[8] http://www.en.voipforo.com/asterisk/asterisk-introduction.php : Introduction to Asterisk.

[9] www.asteriskportal.nl/downloads/asterisk-non-technical-review.pdf : Non-technical review of Asterisk.

[10] http://en.wikipedia.org/wiki/VoIP : VoIP technology.

[11] http://www.premcom.com/category/4/IP_Telephony/page/86/Why_IP_Telephony_.aspx : Why IP telephony?

[12] http://en.wikipedia.org/wiki/IP_PBX : IP PBX definition.

[13] http://www.php.net/manual/es/ : PHP manual.

[14] D.Sklar, A.Trachtenberg. PHP 5 Cookbook. O'Reilly. ISBN 3897214091. 2002.

[15] D.Flanagan . JavaScript, The definitive Guide. O'Reilly. ISBN 9783897214910. 2006.

[16] http://dryicons.com/images/icon_sets/coquette_part_5_icons_set/png/128x128 : icons for the scheme.

[17] http://voip.megawan.com.ar/doku.php/asterisk_configuracion_plantillas : Asterisk configuration examples.

[18] D.Gomillon, B.Dempster. Building Telephony Systems with Asterisk. Packt Publishing Ltd. ISBN 1904811159. September 2005.

[19] www.regular-expressions.info/ : Regular expressions manual and info.

[20] http://regexlib.com/ : Regular expressions library.

[21] http://141.41.40.62/diag_editor.php : Old Editor ( Askozia example installation)