FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

IPBrick.GT – Módulo gráfico para gestão do dialplan

Tiago Dias

VERSÃO FINAL

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Orientador: Professor João Neves

Co-Orientador: Eng. Miguel Ramalhão

23 de Julho de 2018

Resumo

O “dialplan” é o elemento central num sistema de comunicações que tenha como base o serviço de voz sobre IP (*Internet Protocol*). Surge como uma combinação de funções e instruções que definem os fluxos de entrada e saída de chamadas de voz na rede de comunicações de uma organização.

O serviço VoIP na IPBrick, no qual se insere o dialplan, é implementado pelos serviços Kamailio e Asterisk e possui inúmeras funções disponíveis para configuração: filas de espera, escalonadores, conferências, grupos e sequências de atendimento, entre muitos outros. Como tal, é um serviço de elevada complexidade dada a variedade e quantidade de combinações e caminhos que uma determinada chamada de voz pode seguir.

Este trabalho de dissertação tem a finalidade de dotar a interface de administração do sistema operativo IPBrick OS de um módulo capaz de criar uma representação gráfica do dialpan configurado (blocos de funções), respetivas interações e com a possibilidade do administrador IPBrick realizar alterações *on-demand (drag&drop)*.

Abstract

The “dialplan” is the core element of a communications system based on voice over IP services. It is seen as a combination of functions and instructions that define inbound and outbound routes of voice calls in organization’s network.

IPBrickOS is an operating system that includes all kinds of network configurations, between them the VoIP service. It is this service, where the dialplan is included, that is implemented by Asterisk and Kamalio. The functions that are able to be configured are call queues, schedulers, conferences, call groups, attendance sequences and others. As such, it is a service with a lot of complexity given the amount and variety of combinations and flows that a single voice call can have.

Thus, the main goal of this dissertation is to endow the current IPBrick admnistration operating system interface with a module capable of making the internal call flows management and their graphical representation. Also, it has to give the administrator the possibility to make changes on demand with a drag and drop mechanism.

Agradecimentos

Agradeço ao Professor João Neves, por me ter orientado sempre no melhor caminho e por todos os conhecimentos que me transmitiu.

Agradeço aos meus pais e ao meu padrinho, para além de todo o apoio inquestionável, sempre me proporcionaram tudo para chegar até aqui.

Ao Vítor Fernandes e ao Pedro Rodrigues, que sempre estiveram presentes e foram dois pilares importantes nesta longa caminhada.

Ao Eng. Vítor Vidal da IPBrick, por nunca me ter negado ajuda, pelas indicações que fizeram toda a diferença e por tudo o que me ensinou. Agradeço também ao Eng. Miguel Ramalhão da IPBrick, por me ter dado a oportunidade de trabalhar neste trabalho de dissertação.

Tiago Dias

"If you can't explain it simply, you don't understand it well enough."

Albert Einstein

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas, Siglas e Acrónimos

ACK	*Acknowledgement*
AJAX	*Asynchronous Javascript and XML*
API	*Application Programming Interface*
CDR	*Call Detail Record*
CEL	*Call Event Log*
CIDR	*Caller ID Restrictions*
CLI	*Command-line Interface*
CQ	*Call Queue*
CSRF	*Cross-site Request Forgery*
CSS	*Cascading Style Sheets*
DA	*Direct Access*
DNS	*Domain Name System*
FTP	*File Transfer Protocol*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IAX	*Inter Asterisk eXchange*
ID	*Identifier*
IP	*Internet Protocol*
IVR	*Interactive Voice Response*
JS	*JavaScript*
JSON	*JavaScript Object Notation*
LDAP	*Lightweight Directory Access Protocol*
MAC	*Media Access Control*
MD5	*Message-Digest algorithm 5*
NAT	*Network Address Translation*
OWASP	*Open Web Application Security Project*
OS	*Operating System*
PBX	*Private Branch Exchange*
PHP	*PHP: Hypertext Preprocessor*
PIN	*Postal Index Number*
PSTN	*Public Switched Telephone Network*
RTCP	*Real-Time Transport Control Protocol*
SIP	*Session Initiation Protocol*
RFC	*Request for Comments*
RTP	*Real-Time Transport Protocol*
SMS	*Short Message Service*
SQL	*Structured Query Language*
SQLi	*Structured Query Language Injection*

SVG	*Scalable Vector Graphics*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UA	*User Agent*
UDP	*User Datagram Protocol*
UI	*User Interface*
UCoIP	*Unified Communications Over IP*
URL	*Uniform Resource Locators*
VoIP	*Voice Over Internet Protocol*
VPN	*Virtual Private Network*
XML	*Extensible Markup Language*
XSS	*Cross-site Scripting*

Capítulo 1

Introdução

1.1 Contexto

A IPBrick.GT é uma solução de comunicações integrada na empresa IPBrick SA, pioneira na aplicação do conceito UCoIP *(Unified Communications over IP)*. Este conceito é definido por uma integração dos vários serviços de comunicação existentes (Voz, Vídeo, *Instant Messaging*, Mail e Web) suportados por um único domínio, sendo o seu principal objetivo reduzir os endereços associados a uma determinada pessoa, convergindo apenas para o seu endereço de email corporativo [1, 2].

A IPBrick SA dispõe de uma plataforma na qual implementa os seus vários serviços e soluções. O IPBrick OS, baseado em Debian, é o sistema operativo comercializado pela empresa e apresenta soluções para comunicações empresariais (Comunicações Unificadas sobre IP, Correio Eletrónico e Ferramentas Colaborativas, Gestão de Documentos e Processos e Rede Social Corporativa). Este sistema, é dotado de uma interface de gestão administrativa, a qual permite ao administrador da rede realizar as configurações necessárias de segurança, organização e gestão de utilizadores, serviços e documentos.

O serviço VoIP da empresa é implementado pelas plataformas Kamailio e Asterisk e tem um papel de destaque na interface de gestão administrativa. Este serviço é exigente, uma vez que, possui inúmeras funções disponíveis para configuração: números de entrada, extensões, *Interactive Voice Response* (IVR), *call queues*, escalonadores, classes de acesso, redirecionamentos, sequências e grupos de atendimento, entre muitos outros. A configuração destas funções, assim como as interações entre as mesmas, dependem da forma como é feita a gestão do dialplan.

O "dialplan"(daqui para a frente referido sempre como dialplan) é um elemento central de uma infraestrutura capaz de realizar chamadas de voz através da Internet, dado que é o elemento que dita como são tratadas as rotas de entrada e saída das chamadas de voz do sistema. Deste modo, o dialplan é um módulo essencial neste serviço porque é responsável por instruir o agente de processamento de chamadas de como devem ser feitas as rotas. Neste caso particular, deve permitir ao administrador criar uma lista de instruções que o Asterisk irá seguir.

"The dialplan is truly the heart of any Asterisk system, as it defines how Asterisk handles inbound and outbound calls. In a nutshell, it consists of a list of instructions or steps that Asterisk will follow." [3, chapter 6]

1.2 Motivação

Este tema proposto pela IPBrick SA surge no âmbito de uma necessidade da empresa de organizar e gerir de forma mais fácil o dialplan associado ao seu serviço de voz sobre Internet, sendo que, a resolução deste problema será inserida numa nova interface de administração.

Como é do interesse da empresa atualizar o seu sistema operativo, baseando-se na mais recente versão estável do Debian (*Debian Stretch* [4]), o módulo de gestão gráfico para o dialplan será estruturado de forma a não introduzir perturbações a nível de compatibilidade com qualquer sistema.

O dialplan, é um elemento chave no sistema de Comunicações Unificadas, podendo ser extremamente abrangente e complexo, no entanto, a sua gestão gráfica deverá garantir ao administrador que todas as decisões de estabelecimento de conexões que são efetuadas, apresentam-se corretas e bem definidas. Deste modo, a conceção do módulo, deve ser orientada, fundamentalmente, a disponibilizar ao administrador as seguintes capacidades: primeiramente uma leitura rápida e percetível das configurações (*call flows*) já feitas, depois, intuição e facilidade em fazer as configurações necessárias, num cenário simples e, sem esquecer, a validação e controlo de erros na aplicação das configurações.

Assim sendo, juntamente com o conhecimento prático das funcionalidades e configurações que advêm da implementação do serviço de voz da IPBrick, será requerida a aplicação de *frameworks* atuais que possam ser vantajosas no desenvolvimento desta solução.

1.3 Objetivos

Atualmente, o dialplan existente na interface de administração do IPBrick OS não dispõe de uma representação gráfica que proporcione uma visão clara das configurações existentes nem que tenha a capacidade de as alterar. A solução que existe é baseada em tabelas, ou seja, não é de fácil interpretação e manutenção. Portanto, o objetivo principal será dotar esta interface de uma ferramenta gráfica capaz de providenciar ao administrador uma maneira percetível e intuitiva de leitura e configuração do dialplan.

A implementação deste módulo deverá ser plenamente compatível com arquitetura do sistema atual da IPBrick, no entanto, o seu *design* será projetado de forma independente do atual *design* da interface de gestão administrativa. Será uma solução baseada na metodologia *drag and drop*, isto é, para além de ser uma representação gráfica do que já se encontra configurado com as respetivas interações, terá a possibilidade de interligar entre si blocos de funções, a partir dos quais o administrador poderá facilmente adicionar ou remover possíveis configurações *on demand*.

O modulo gráfico será realizado de modo a ser acedido a partir da interface web de administração do IPBrick OS, deste modo, além da sua conceção, será feita uma análise da componente de segurança do dialplan, por forma a garantir integridade dos dados.

1.4 Estrutura do Documento

Para além da introdução, este documento esta organizado em 5 capítulos.

No capítulo atual encontra-se descrito o contexto, a motivação e os objetivos.

No capítulo 2 é apresentada a revisão bibliográfica e trabalhos relacionados.

No capítulo 3 é apresentada a caracterização e definição do problema, assim como a solução proposta e quais as tecnologias envolvidas.

No capítulo 4 é descrito todo o processo envolvido na implementação do projeto, estando dividido em duas partes, importação e exportação de configurações.

No capítulo 5 são apresentadas conclusões acerca deste trabalho de dissertação e o trabalho futuro a desenvolver.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é apresentada a revisão bibliográfica e trabalhos relacionados.

2.1 Protocolos

2.1.1 *Voice Over Internet Protocol*

VoIP (*Voice Over Internet Protocol*) é um protocolo definido como um conjunto de tecnologias e componentes que permitem o estabelecimento de uma ligação através da Internet, para comunicação de voz ou sessões de multimédia [5]. A implementação deste sistema tem como base protocolos específicos para sinalização e transporte. Os protocolos de sinalização servem para estabelecer, modificar ou terminar uma sessão de comunicação. Os protocolos de transporte servem para fazer a entrega dos pacotes que contêm dados, após a sessão ser estabelecida corretamente. Os protocolos mais utilizados para sinalização são o SIP (*Session Initiation Protocol*) e o IAX (*Inter Asterisk eXchange*). Para o transporte são utilizados os protocolos RTP (*Real-Time Protocol*) e RTCP (*Real-time Transport Control Protocol*) [6].

2.1.2 *Session Initiation Protocol*

SIP *Session Initiation Protocol* é um protocolo da camada de aplicação responsável por estabelecer, modificar e terminar sessões de troca de dados entre UA's (*user agents*) [7].

Numa rede VoIP, este protocolo é usado para iniciar uma chamada via Internet. Um UA pode ser um cliente ou um servidor, sendo que, o cliente é responsável por iniciar os pedidos, o servidor faz o processamento dos mesmos. Ou seja, este protocolo utiliza o modelo *request and reply* para iniciar as sessões de comunicação.

Os servidores SIP são aplicações que recebem notificações e executam operações em conformidade com a solicitação efetuada. Existem três tipos de servidores SIP: *proxy SIP, redirect SIP e register SIP*. O servidor *proxy* faz o encaminhamento entre o domínio de origem e o domínio de destino quando recebe um pedido. O servidor de redirecionamento não encaminha pedidos SIP, apenas responde ao UA de origem qual é o endereço do UA de destino ou qual o endereço do

servidor mais próximo. O servidor de registo aceita pedidos de registo de utilizadores e contém dados sobre a localização dos UA [8].

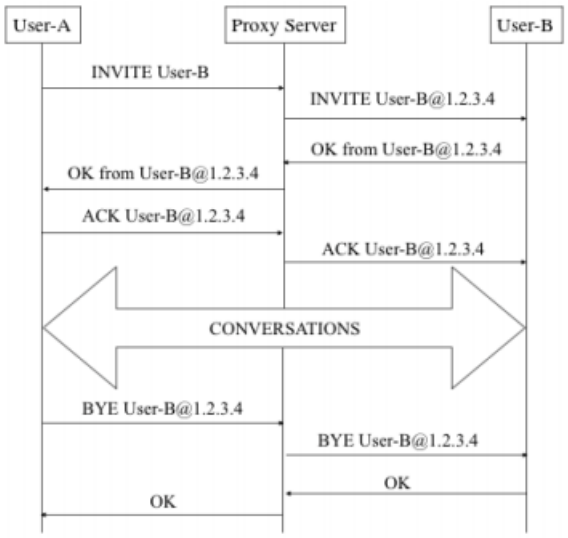


Figura 2.1: *SIP Call Setup*

A figura 2.1 [6] ilustra um exemplo de estabelecimento de uma sessão utilizando este protocolo. Para uma sessão se iniciar é necessário que o cliente esteja registado um servidor VoIP, que serve como *gateway* para fazer o encaminhamento do pedido para o respetivo UA de destino. O utilizador A faz o pedido para estabelecer uma sessão com o utilizador B, começando por fazer o pedido através da mensagem de INVITE para o utilizador B através do servidor proxy. Um INVITE contém informação necessária para o servidor de proxy autenticar o pedido e fazer o seu encaminhamento. O utilizador B ao receber o INVITE responde com a mensagem de OK, confirmando que existem condições para se iniciar a sessão. Assim sendo, o utilizador B recebe a resposta e manda o ACK para confirmar que também se encontra em condições para a chamada. A sessão é estabelecida entre as duas entidades (UA's) e termina com a mensagem de BYE e posterior recessão de um OK [6].

2.1.3 *Inter Asterisk eXchange*

IAX é um protocolo desenvolvido pela Digium com o objetivo de estabelecer comunicação entre servidores Asterisk, por forma a combater algumas limitações impostas pelo SIP. O IAX é um protocolo mais flexível e mais facilmente configurável, visto que o seu código fonte é disponibilizado podendo ser alterado conforme as características do sistema [9].

IAX é um protocolo *all-in-one* da camada de aplicação que combina os serviços de sinalização e dados na mesma *stream* de comunicação. Ao utilizar um única *stream* de dados UDP (*User Datagram Protocol*), tipicamente, na porta 4569, tem a vantagem de simplificar a gestão de rede e de *firewall* uma vez que consegue eliminar a necessidade de uso de outros protocolos para contornar o NAT (*Network Address Translation*) [10].

2.1.4 *Real-Time Transport Protocol*

RTP (*Real-Time Transport Protocol*) é um protocolo definido pela RFC 3550, para transporte de informação ponto-a-ponto, usualmente no transporte de dados como áudio ou vídeo, através de uma rede. Não garante por si só reserva de recursos e qualidade de serviço em serviços de tempo real [11]. Este protocolo é conjugado com o RTP *Control Protocol* (RTCP), que faz a gestão de alocação de recursos na rede para garantir uma determinada qualidade de serviço.

2.1.5 *Hypertext Transfer Protocol*

HTTP (*Hypertext Transfer Protocol*) é um protocolo da camada de aplicação responsável por transmitir informação através da Internet [12]. Este protocolo designa-se como sendo um genérico e *stateless*, o que significa que não precisa necessariamente que o servidor HTTP guarde informação acerca dos pedidos, da sua duração ou mesmo acerca do utilizador. As aplicações web são capazes de guardar informação a partir de*cookies* ou mesmo usando determinadas variáveis escondidas e embutidas nos formulários.

Cookies HTTP *cookie* são um elemento chave do protocolo HTTP que a maior parte das aplicações *web* dependem. São frequentemente utilizadas como um veículo para explorar vulnerabilidades [13]. O mecanismo das *cookies* passa por um envio de ficheiro do servidor para o *browser* do utilizador, sendo que, o *browser* pode guardar e mandar de volta após novo pedido ao *browser*. Tipicamente, as *cookies* são usadas se dois pedidos forem feitos a partir do mesmo *browser*, desta forma, é possível, por exemplo, manter o *login* de um determinado utilizador [14]. A emissão da *cookie* é feita do lado do servidor pelo cabeçalho de resposta *Set-Cookie*, este cabeçalho inclui atributos opcionais que dependem de como o *browser* controla a *cookie*, entre eles:

- *Expires*: representa a validade da *cookie*.
- *Domain*: especifica o domínio para o qual a *cookie* é válida.
- *Path*: URL para o qual a *cookie* é valida
- *Secure*: define se a *cookie* é enviada apenas em HTTPS

HTTPS HTTPS é uma implementação na qual existe um camada adicional que encripta a comunicação, TLS (*Transport Layer Security*). A motivação para a criação desta adaptação passa pela proteção da integridade dos dados, ou seja, se se captar uma ligação que use HTTPS os dados estão encriptados.

O HTTPS é um protocolo que usa a porta TCP/IP 443 por definição e a comunicação é feita, entre cliente e servidor, via *request/response*. Os pedidos podem ser enviados via URL (*Uniform Resource Locators*), no qual se encontra o domínio e o *path* para a página a ser acedida. Os pedidos têm métodos associados de modo a desencadearem a ação desejada aquando de um pedido, os métodos mais utilizados são:

- GET: faz o *fetch* de um determinado recurso. A página recebe via URL ou por formulário a informação necessária para o servidor encontrar e retornar o recurso.
- POST: criação de um novo recurso, ou seja, este pedido tem informação (*payload*) específica para obter um novo recurso.

Para além deste métodos, outro elemento importante na estrutura deste protocolo são os *status codes*, que indicam como é que foi interpretado no servidor o pedido feito pelo cliente. Podem ser [15]:

- 2xx - Indicam que o pedido foi devidamente processado, havendo variações para o caso de o conteúdo estar ou não completo.
- 3xx - Indica que é necessário redirecionar o pedido.
- 4xx - Indica que o pedido está mal formulado ou que a página se encontra inválida.
- 5xx - Indica que houve uma falha no servidor durante o processamento do *request*

2.2 Infraestruturas

2.2.1 *Private Branch Exchange*

Num sistema de comunicações unificadas, um PBX (*Private Branch Exchange*) funciona como o sistema central de comutação para chamadas. O PBX é o responsável por lidar com o tráfego interno entre estações e atua como *gateway* para o exterior.

A um sistema VoIP encontram-se associados vários serviços para além de apenas chamadas de voz. É possível configurar serviços como *voicemail*, IVR (*Interactive Voice Response*), chamadas em conferência, classes de acesso, escalonadores, filas de espera, etc. Todas estas possíveis configurações podem ser efetuadas a partir de um IPBX, que é responsável por agregar os vários constituintes do sistema e fazer a interligação das comunicações utilizando o protocolo VoIP.

Um PBX tradicional é constituído por dois elementos chave: linhas e estações. As linhas, conectadas à rede telefónica pública, PSTN (*Public Switched Telephony Network*) são associadas em *trunks*, isto é, há uma conexão entre estações através da rede VoIP sem necessidade de um operador. Estações são simplesmente telefones ou outros dispositivos de *end-point* [3].

A missão principal do PBX é providenciar um acesso partilhado com recursos limitados, ou seja, em vez de termos uma linha separada para cada telefone, temos uma linha partilhada e configurada para todos.

Atualmente, a comunicação expandiu-se para vários tipos, como *email*, mensagens instantâneas, videoconferências, partilha de ficheiros e telefonia móvel, portanto, Comunicações Unificadas é um termo abrangente que descreve o processo de fusão destas tecnologias, integrando-as com os processos de negócio. As comunicações unificadas visam aumentar a eficiência ao mesmo tempo que simplificam a sua gestão [16].

2.2.2 SIP Proxy Server

Um SIP *proxy* é um elemento da estrutural da rede que tem como principal objetivo facilitar a comunicação entre endereços SIP. Essencialmente tem como função fazer o roteamento e direcionamento de chamadas, ou seja, encontrar o destino correto para uma determinada sessão [17]. A base desta ferramenta é a comunicação feita através do protocolo SIP, não tendo as funções como IVR ou *voicemail*

Em termos de segurança e balanceamento de carga, a utilização de um servidor SIP *proxy* a par com um IP PBX, é vantajoso. Um servidor SIP *proxy* consegue identificar a identidade e o crédito de um utilizador, tendo também a capacidade de evitar congestionamentos na rede.

2.3 IPBrick OS

IPBrick OS é um sistema operativo, baseado na distribuição Linux Debian, desenvolvido e distribuído pela empresa IPBrick SA. O sistema é dividido em três servidores capazes de gerir comunicações, documentos, processos e outro tipo de ferramentas colaborativas dentro de uma cooperação. Podemos dividir o IPBrick OS em diferentes servidores, os quais têm funções deliberadas diferentes mas que funcionam de forma dependente:

- Servidor de Intranet - Gestão de domínio, utilizadores e áreas de trabalho.
- Servidor de Comunicações Unificadas - Gestão de comunicações VoIP, *Instant Messaging*, FAX, SMS, E-mail e servidores Web
- Servidor de Segurança - Regras de *firewall* e configuração dos servidores de *proxy*, VPN,*email* e FTP.

Para aceder às várias possibilidades de configuração destes serviços, existe uma interface gráfica. Esta interface é disponibilizada a partir de um servidor HTTP sendo acessível pelo *browser*, através do IP privado dado à maquina onde o sistema se encontra instalado. O acesso à interface é controlado, ou seja, apenas o administrador tem acesso e consegue fazer as alterações necessárias e que mais convém ao cliente. Assim, todas as aplicações distribuídas pela IPBrick SA estão acessíveis, de alguma forma, através desta interface que contém toda a informação da infraestrutura de rede de um determinado cliente.

2.3.1 Apache

Apache HTTP *server* é um servidor HTTP, disponibilizado como um software *open-source* e multiplataforma (UNIX e Windows). O objetivo desta ferramenta é disponibilizar um servidor seguro, eficiente e extensível com a capacidade de ser compatível com os padrões atuais do protocolo HTTP [18].

A configuração deste servidor passa por definir a porta para na qual queremos aceder ao servidor, sendo possível definir a página web que é possível aceder a partir da criação de ficheiros

que assim o permitam. Por omissão a porta TCP/IP 80 é definida como a principal, já a porta 443 definida para usar o protocolo HTTP mas com camada de encriptação.

2.3.2 Aplicação Web

A aplicação *web* da IPBrick que serve como interface de gestão administrativa estabelece a comunicação segundo o protocolo HTTP que providencia resiliência e evita a criação de uma ligação para cada cliente que acede à aplicação. Possibilita a criação de túneis de comunicação ou se houver necessidade a criação de um *proxy*.

Uma interface de administração é a chave para uma infraestrutura, neste caso, para um sistema IPBrick, uma vez que, a gestão dos vários serviços de comunicação, das máquinas, dos utilizadores e dos servidores é feita através da mesma.

Apesar do acesso a esta página *web* ser apenas através de HTTPS, ou seja, usando TLS, a ideia de que a aplicação se encontra protegida não poderá ser totalmente assegurada, uma vez que, existem vários tipos de riscos associados que podem levar a que dados sejam comprometidos. O TLS apenas garante a integridade e confidencialidade dos dados que são transmitidos na ligação entre o cliente e o servidor *web*. Outro tipo de vulnerabilidades associadas às aplicações web como XSS, SQLi ou CSRF, não são evitadas apenas ao utilizarmos uma camada de segurança na ligação.

O facto de um utilizador, neste caso administrador, poder inserir e submeter determinados dados numa aplicação é a razão crucial para que existam riscos associados a uma aplicação web. Os parâmetros que são transmitidos entre a ligação cliente servidor podem ser alterados de tal forma que um determinado pedido provoque eventos inesperados na aplicação.

Associado à tecnologia HTML existe criação de formulários através de uma *tag*. Os formulários são usados no lado do cliente para que este proceda ao seu preenchimento, tendo a possibilidade de dar os *inputs* de forma correta ou então tentar injetar código malicioso. O método POST usado pelas *forms* é o *trigger* para que a informação seja enviada para o servidor. O pedido feito tem associado uma *cookie*, correspondente à sessão de um determinado cliente. Com o método GET, é possível obter informação, através de parâmetros que podem ser introduzidos no URL.

As vulnerabilidades que afetam a camada de aplicação revelam que, qualquer informação que o utilizador possa introduzir deve ser considerada perigosa [19]. Através desta premissa de não confiar no utilizador, uma aplicação irá ser desenvolvida de forma mais segura e robusta [20].

2.3.2.1 CSRF

Cross-Site Request Forgery é uma vulnerabilidade na qual o atacante se aproveita da autenticação de um utilizador para realizar ações indesejadas nessa mesma página web. Surge como um ataque no qual há alteração dos pedidos feitos, não havendo roubo de dados ao utilizador [21]. O impacto desta vulnerabilidade relaciona-se com o tipo de ação induzida pelo atacante e pelo tipo de privilégios do utilizador comprometido tem na aplicação em questão. As limitações são em conformidade com o tipo de exposição da aplicação comprometida.

Mitigação Segundo a OWASP [22], as recomendações gerais para defesa deste tipo de ataque passam pela verificação da origem do pedido através dos cabeçalhos e pela verificação do *CSRF token*.

Os cabeçalhos HTTP que permitem a verificação da integridade da ligação, são os cabeçalhos *Origin Header* e *Referer Header*. O cabeçalho de origem indica a fonte que fez o *fetch* dos dados [23]. O *Referer Header* contem o endereço da página *web* anterior a partir do qual foi solicitado o endereço da página web atual, os servidores podem também usar este cabeçalho com o objetivo de identificar de onde os utilizadores visitam o site para uso em dados estatísticos.

Assim que se tenha verificado a origem do pedido, é recomendado que haja uma segunda verificação. Esta segunda verificação envolve um mecanismo mais especifico que passa pela criação de *CSRF tokens*. Um *token CSRF* é único por sessão, tem um valor *random*, é gerado de modo criptográfico e é normalmente adicionado aos formulários HTML através da *tag* input do tipo *hidden* e com um nome, por exemplo, *"CSRFToken"* [22]. Este valor é expirado ao mesmo tempo que a sessão, deste modo, quando um pedido é efetuado por um utilizador, o servidor deve verificar a validade do *token* incluído no pedido e comparar com o *token* associado à sessão, se não corresponder o pedido deverá ser cancelado [22].

2.3.2.2 XSS

XSS, *Cross-Site Scripting* é um tipo de ataque no qual as aplicações web são comprometidas através da injeção de código malicioso *JavaScript*. Estes *scripts* podem aceder a *cookies*, tokens de sessão ou a outros informações confidenciais retidas pelo *browser* [24]. É considerada a vulnerabilidade mais crítica a nível da camada de aplicação. XSS pode ser de três tipos: *reflected*, *stored* ou *DOM-based*.

Reflected XSS Este tipo de XSS surge quando num determinado URL existe um campo de dados onde é introduzida uma mensagem que será mostrada na página, ou seja, o atacante pode submeter nesse campo uma mensagem que pretenda mostrar e a mesma será carregada. Qualquer utilizador que faça esse pedido, o resultado será refletido para o mesmo, isto é, o *payload* introduzido será entregue e executado por qualquer utilizador via *request and response* [13].

Stored XSS Neste caso, os *scripts* injetados são armazenados de forma permanente nos servidores de destino [24], pelo que o seu conteudo poderá afetar outros utilizadores que acedam à aplicação.

DOM-based XSS DOM-base *Cross-Site Scripting* é quando o ataque tem como alvo o *Document Object Model*.

Mitigação Segundo [25], a mitigação para este tipo de ataque, de um modo geral, passa por uma verificação evasiva da informação que é introduzida dos campos HTML. Assim sendo, para mitigar as vulnerabilidades de *stored XSS* e *reflected XSS*, o primeiro passa para por identificar as

instâncias de código da aplicação na qual o utilizador insere dados que vão ser usados para algum tipo de pedido. Desta forma, deve-se fazer o escape dos caracteres *(HTML encoding)*, tanto nos elementos, como nos atributos dos elementos da página de forma a prevenir a inserção de *scripts*, como é ilustrado na figura 2.2 [26] .

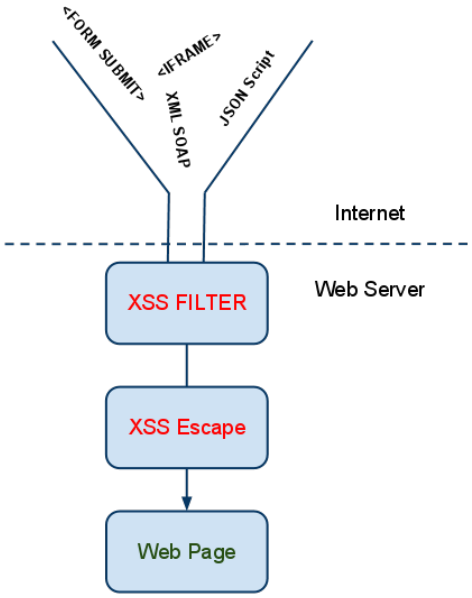


Figura 2.2: XSS Filtering/Escape

2.3.2.3 SQLi

SQL *injection* é um ataque no qual há a injeção de *queries* feitas a partir da aplicação. No caso de sucesso, as *queries* permitem obter informação relevante da base de dados e possivelmente, a modificação da mesma [27].

As falhas que permitem o sucesso deste tipo de ataque é quando o utilizador insere dados nos *inputs* que servem para gerar dinamicamente as *queries* à base de dados.[28].

Mitigação Para que seja evitado este tipo de ataque, é necessário que as *queries* não sejam feitas de forma dinâmica e, mais uma vez, prevenir que seja introduzido código malicioso nos *inputs*. As defesas primárias, tal como nas outras vulnerabilidades abordadas, passam validar os dados que os utilizadores possam introduzir na aplicação web.

No entanto, em termos de desenvolvimento da aplicação, é necessário usar parametrizações nas *queries*. As *queries* parametrizadas são definidas antecipadamente e determinados parâmetros dessa *query* são preparados para serem preenchidos com variáveis. Depois, o base de dados irá ser capaz de distinguir entre código SQL e informação proveniente do utilizador, desta forma, tentativas de injeção de código por SQL não chegarão a ser executados [20].

Em PHP, o uso de *PHP Data Objects (PDO)*, providencia métodos para uso e criação de *queries* parametrizadas e faz com que o código se torne abrangente, podendo ser adotado para comunicação com vários tipos de base de dados, assim como a leitura e interpretação do código PHP ficará mais intuitiva [20].

2.3.3 Asterisk

Asterisk é um IP PBX *open-source* que inclui todos os componentes necessários para construir um poderoso e escalável sistema de telefonia empresarial. Dispõe de recursos como: *voicemail, call queueing, interactive voice response e conference rooms* [16]. A comunicação entre servidores Asterisk pode ser feita utilizando o protocolo IAX ou através do protocolo SIP.

Esta ferramenta de uso gratuito, permite a implementação de uma central telefónica eficiente e completa, da forma mais adequada possível devido à sua flexibilidade no que diz respeito à adição ou não dos módulos que possui [9].

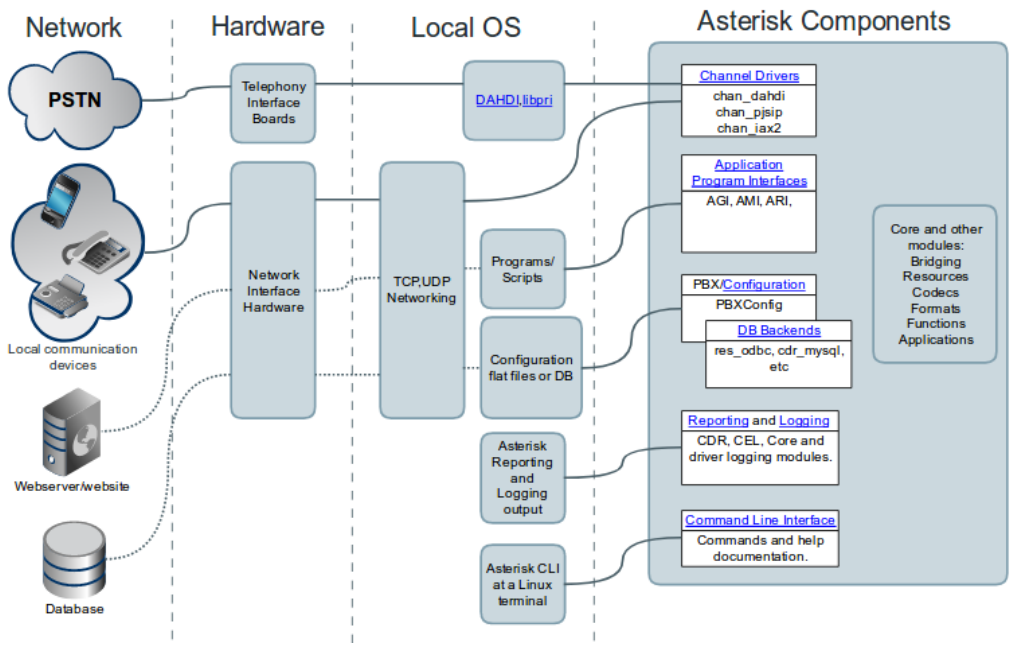


Figura 2.3: Arquitetura Asterisk

Como é ilustrado na figura 2.3 [29] o Asterisk pode ser usado como servidor de registo, no qual se encontram registados utilizadores SIP e telefones, físicos ou virtuais, que são identificados pelo seu endereço MAC ou por um nome identificativo. Esta ferramenta dispõe de uma interface que é acedida via linha de comandos, na qual é possível executar comandos que servem como instruções para o Asterisk, como por exemplo, recarregar configurações feitas no dialplan. Existem também vários tipos de módulos associados ao Asterisk, cada um deles providencia funcionalidades e capacidades especificas [30]:

- *Channel Drivers*: *drivers* usados para comunicar com dispositivos externos.

- *Dialplan Applications*: aplicações que providenciam funcionalidades de chamada ao sistema.
- *Dialplan Functions*: as funções são usadas para recolher, atribuir e manipular determinadas configurações de chamada, como por exemplo definir um *Caller* ID.
- *Resources*: recursos disponibilizados aos módulos como *music oh hold* ou *call parking*.
- *CODECs*: módulo para fazer codificação e descodifição de áudio ou vídeo.
- *File Format Drivers*: drivers usados para gravar ficheiros multimédia num determinado formato e converte-los para *streams* na rede.
- *Call Detail Record (CDR) Drivers*: *drivers* usados para gerar *logs* no disco ou na base de dados referentes às chamadas.
- *Call Event Log (CEL) Drivers*: *drivers* similares aos CDR mas gravam dados com mais detalhe.
- *Bridge Drivers*: *drivers* que dispõe de vários métodos para criar ligações entre participantes numa chamada.

Após a instalação do Asterisk, a sua configuração é feita através dos diferentes ficheiros de configuração localizados no diretório */etc/asterisk*.

Call Detail Records O Asterisk possui a capacidade de guardar informação acerca das chamadas efetuadas num ficheiro ou mesmo numa base de dados, sendo esta última a maneira mais eficiente de o usar. Esta informação das chamadas é referenciada como *Call Detail Records* e através desta informação gravada é possível monitorizar o sistema, procurando a presença de anomalias que possam ser prejudiciais [9].

Interactive Voice Response *Interactive voice response* (IVR) é um elemento que permite responder a um telefone de forma interativa, como a reprodução de um ficheiro ou mesmo fornecer informação da base de dados [9].

Voicemail O *voicemail* é uma maneira de guardar mensagens que é uma das *features* mais importantes que o Asterisk é capaz providenciar [3].

O *voicemail* é configurado a partir do ficheiro *voicemail.conf* onde são definidas as diferentes *voicemail* boxes [9]. É possível configurar o formato em que as mensagens de voz serão guardadas, ou mesmo, configurar o envio de uma notificação para *email* aquando da receção de mensagens.

Como forma de evitar a aglomeração de mensagens por ler, é possível definir o máximo de mensagens a guardar e ainda o tamanho de cada uma.

Music on Hold O Asterisk possui também um ficheiro de configuração que possibilita que seja introduzida uma música enquanto a chamada está em espera numa fila [9]. O ficheiro associado para possíveis configurações é *musiconhold.conf*, sendo que esta pode ser guardada em diferentes formatos.

Call Queues Como o nome indica, as filas de espera, refere-se às situações em que a chamada fica em espera, enquanto não é atendida. Acontece quando o número de chamadas feitas excede o número de chamadas que podem ser atendidas [31].

Para configurar as filas de espera, o ficheiro utilizado é o */queues.conf*, onde é possível dar nomes às filas de espera e selecionar a música ou o ficheiro para reproduzir [9].

Conference Rooms As salas de conferência são definidas e configuradas no ficheiro *meetme.conf* [9]. É possível proteger o acesso com algum tipo de *password* para controlar as entradas.

2.3.3.1 Asterisk Manager Interface

Asterisk Manager Interface (AMI) é um sistema de monitorização e gestão providenciado pelo Asterisk. Permite o controlo de eventos que ocorrem no sistema assim como a executar comandos. A configuração é feita a partir do ficheiro */etc/asterisk/manager.conf*, por *default* é a interface pode ser acedida via telnet para a porta 5038 ou por HTTP na porta 8088, mais uma vez por *default*. A partir desta ferramenta disponibilizada pelo Asterisk é possível fazer monitorização do dialplan, ou seja, o utilizador pode receber notificações quando alguma variável é mudada ou há a criação de novas *extensions* [3].

2.3.3.2 Asterisk Gateway Interface

Asterisk Gateway Interface (AGI) é uma interface criada para ser possível a manipulação do dialplan através do programa externo (via *pipes, stdin e stdout*). Basicamente aumenta o grau de liberdade no que toca à abordagem que pode ser feita à manipulação do dialplan, no entanto, tem o custo de aumentar a complexidade.

AGI permite ao Asterisk seguir instruções definidas num *script*, como por exemplo, ficar a espera de algum *input* ou fazer algum *handle* de chamadas. Ao permitir que o *script* seja executado através do dialplan, permite ao utilizador ter uma maior interatividade com o sistema, procurando fazer *debug* da aplicação [31].

A figura 2.4 [32] mostra a sintaxe necessária para executar o *script* via AGI. A linguagem do *script* tem que ter em consideração a performance do sistema, por exemplo, no caso de se invocar um *bash script*, este terá menos *overhead* em termos de processamento do que, por exemplo, se for em *JAVA*.

```
exten => _X., 1, AGI(script_name, param1, param2, param3)
```

Figura 2.4: Invocação de um *script* pelo dialplan

2.3.3.3 Asterisk Rest Interface

Asterisk Rest Interface (ARI) surgiu a partir do Asterisk 12, permite construir aplicações utilizando objetos primitivos do Asterisk. O estado dos objetos é controlado via *JSON* através de um WebSocket. Enquanto que AMI se relaciona com controlo e a AGI na execução de comandos, ARI surge como uma API assíncrona que permite a criação de aplicações através de uma interface [33].

2.3.4 Kamailio

Kamailio é uma ferramenta *open-source* de implementação de um servidor *proxy* SIP, é uma solução capaz de gerir dezenas de chamas por segundo, sendo viável em termos de escalonamento e introdução em ambientes corporativos.

Pode ser usado em várias plataformas VoIP, como por exemplo o Asterisk, agindo como um elemento que dá robustez ao sistema pois apresenta várias características que asseguram a integridade e autenticidade do sistema. Tem a capacidade de garantir comunicações seguras via TLS para VoIP, validação de autenticação e autorização, assim como suporte para vários sistemas de *backend* como PostgreSQL, MySQL ou Radius [34]. A figura 2.5, mostra a integração com o Asterisk.

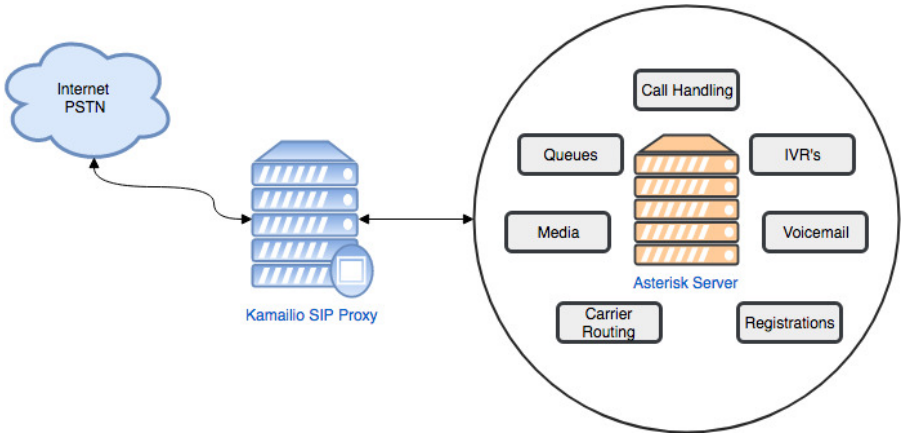


Figura 2.5: Kamailio & Asterisk - Esquema da ligação

2.3.5 Tecnologias Web *Front-end*

2.3.5.1 HTML

HTML (*HyperText Markup Language*) é uma linguagem de programação utilizada para a criação de páginas web, que pode ser lida e interpretada por qualquer browser. O formato dos ficheiros é .html, os quais devem ser escritos segundo normas definidas.

Estes documentos apresentam *tags*, que são interpretados como comandos e que vão alterar o que estiver inserido entre essas mesmas *tags*. O HTML descreve a estrutura da página web de forma semântica e está diretamente relacionada com a sua aparência.

2.3.5.2 JavaScript

JavaScript é uma linguagem de programação usada para gerar páginas web interativas. Tem como finalidade providenciar ao utilizador uma melhor experiência ao conseguir criar elementos interativos e animação na páginas web [35]. Esta linguagem permite melhorar a performance de uma aplicação web uma vez que várias tarefas podem ser geradas do lado do cliente, não sendo necessário comunicar com o servidor que pode por vez provocar latência. Normalmente, o *JavaScript* é utilizado para [13]:

- Validação de *user-input data* antes do envio do pedido para o servidor, para evitar erros e a integridade da informação.
- Dinamicamente modificar a interface em resposta a ações do utilizador
- Adquirir e modificar elementos da DOM (*Document Object Model*) dentro do *browser*

Document Object Model É uma representação abstrata de um documento HTML que permite que *scripts* do lado cliente acedam e modifiquem os elementos presentes na estrutura HTML. A aquisição dos elementos pode ser com base no seu ID ou classe. Esta manipulação é usada em aplicações baseadas em AJAX, descrito em 2.3.5.3;

2.3.5.3 AJAX

Asynchronous JavaScript And XML não é uma linguagem de programação mas sim a combinação de tecnologias como *JavaScript* e XML para promover a interatividade da página *web*. Utiliza solicitações assíncronas, permitindo que em determinadas situações, apesar de ser necessário a comunicação com o servidor, a página não tem necessariamente de voltar a carregar. Apesar do seu nome, os pedidos AJAX não se cingem apenas ao XML.

A tecnologia essencial desta linguagem é *XMLHttpRequest*, é através desta API que os *client-side scripts* fazem pedidos em *background*. Apesar do seu nome, é possível enviar e receber conteúdos.

2.3.5.4 XML

XML, *eXtensible Markup Language*, de acordo com [36] é uma linguagem de marcação desenvolvida com o objetivo de descrever informação, onde é o próprio programador que define as *tags* que usa para definir o conteúdo que deseja. Contrariamente, o HTML foca-se na representação da informação da página em vez da sua descrição. No entanto, estas duas linguagens complementam-se, uma vez que, a informação que é apresentada numa página web através de HTML, é guardada no formato XML. A leitura de um ficheiro XML, obtendo a informação necessária de acordo com a sua estrutura, denomina-se como XML *parsing*.

2.3.6 Tecnologias Web *Back-end*

2.3.6.1 PHP

O PHP é uma linguagem de programação para desenvolvimento de páginas *web* que tem a capacidade de ser integrável com HTML [37]. É uma linguagem extremamente modulada o que torna ideal para ser utilizada em servidores web. Pode ser utilizada por inúmeros sistemas operativos e tem características que fazem desta linguagem uma aposta já recorrente para implementação de páginas *web*.

2.3.6.2 PostgreSQL

PostgreSQL é um sistema que permite fazer a gestão de uma base de dados relacional. Uma base de dados serve para guardar e organizar informações acerca de utilizadores ou serviços que podem estar, ou não, relacionadas entre si. Toda esta informação está sobre a forma de tabelas, nas quais cada coluna é designada como atributo, ou seja, se tenho uma tabela onde guardo informação acerca de carros, uma das colunas poderá ser a cor.

As bases de dados, normalmente assumem dimensões consideráveis e como tal, é necessário que haja uma linguagem que permita aceder à informação que desejamos. A linguagem padrão das bases de dados relacionais é o SQL (*Structured Query Language*), ou seja, é possível fazer *queries* à base de dados para obter a informação de desejada.

2.4 Dialplan

O dialplan é o elemento fundamental de um sistema VoIP baseado em Asterisk. Todos os canais de comunicação que chegam ao sistema vão passar pelo dialplan, ou seja, este elemento será responsável pela gestão de fluxos de chamada, sendo esta gestão feita a partir de um ficheiro (*extension.conf*) que determina como é que os fluxos serão tratados. Este ficheiro é, de longe, o mais importante dos ficheiros de configuração do Asterisk, removê-lo é semelhante a remover sinalização numa intersecção entre várias vias de acesso, os carros querem tomar um rumo mas não há nada que os direcione [31]. Este ficheiro pode ser elaborado de três formas diferentes [3]:

1. Através do modelo mais tradicional, através da edição do ficheiro */etc/asterisk/extensions.conf*
2. Usando o AEL (Asterisk Extension Logic) */etc/asterisk/extensions.ael*
3. Através da linguagem de programação Lua através da edição do ficheiro */etc/asterisk/extensions.lua*

No sistema da IPBrick, a elaboração do dialplan é feita através da edição do ficheiro *extensions.conf*. O dialplan e o seu respetivo ficheiro de configuração, *extensions.conf*, têm ligação direta e dependente com outros ficheiros configuração, como por exemplo, *sip.conf* e *iax.conf*, relativas aos protocolos SIP e IAX respetivamente.

Existem três tipos de clientes SIP e IAX, podem ser do tipo *peer, user* ou *friend*. Como indicado na figura 2.6 [38], um *peer* apenas pode receber chamadas, um *user* pode fazer chamadas através do Asterisk mas não pode receber, um *friend* pode fazer e receber chamadas via PBX.

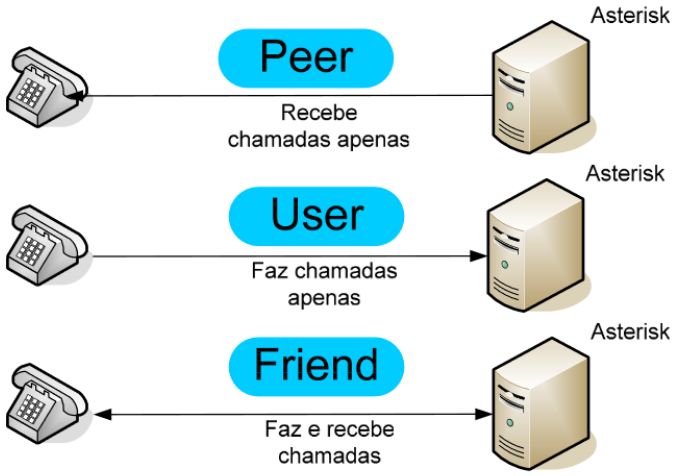


Figura 2.6: Tipos de clientes SIP/IAX

A lógica atribuída a cada chamada é definida pelo dialplan, ou seja, é necessário haver uma sincronia na configuração do dialplan com as dos protocolos relacionados com a comunicação. Assim, quando uma chamada chega ao sistema, é feita a correspondência do protocolo através do ficheiro de configuração respetivo, só depois é redirecionado para o dialplan, através do contexto definido. (Figura 2.7)

Este processo é na mesma definido para o exterior, ou seja, no caso de ser feita uma chamada para outro dialplan, o ficheiro de configuração do canal de comunicação, podendo ser SIP ou IAX (sip.conf e iax.conf, respetivamente), irá direcionar a chamada para o dialplan do destino, após esta ter passado pelo dialplan da origem.

O conceito de extensão é algo que deve estar presente na configuração de um sistema baseado em Asterisk. Na maior parte dos PBX's, uma extensão é um número para o qual se liga mas no Asterisk, uma extensão é um grupo de instruções definidas no dialplan [3]. Basicamente, uma extensão no Asterisk pode ser um número que liga para um determinado telefone ou serviço mas pode facilmente ser um nome, como por exemplo, *voicemail*, significando assim que poderá fazer correr uma determinada aplicação.

2.4.1 *Extensions.conf*

O dialplan contém instruções que o Asterisk irá seguir em resposta a determinados *triggers*, internos ou externos. As instruções são definidas no ficheiro de configuração *extensions.conf* com uma determinada sintaxe. Os elementos principais que constituem este ficheiro são: *contexts, extensions, priorities* e *applications*. [3]

2.4.1.1 Variáveis

Antes de entender os elementos presentes no ficheiro *extensions.conf*, é necessário perceber que a sintaxe para as configurações feitas neste ficheiro é simplificada com o uso de variáveis[39]. As variáveis podem ser globais, de canal ou de ambiente. As variáveis globais são configuradas na secção *[globals]* do ficheiro *extensions.conf*, para assim serem referenciadas sempre que necessário. As variáveis de canal são definidas para chamada em particular através do comando *Set(variable=x)* e são lidas usando a seguinte sintaxe *${variable}*.As variáveis de ambiente servem para aceder às variáveis do sistema Unix através do Asterisk [40].

2.4.1.2 *Contexts*

As diferentes secções definidas no dialplan são chamadas de *contexts*, garantindo assim a independência entre partes do dialplan com diferentes configurações associadas. Basicamente, um *context* é um grupo de *extensions*, cada *extension* deve existir dentro de um *context*. Desta forma, há um isolamento entre contextos, não havendo interação entre eles, possibilitando que haja certas ações que sejam só feitas por determinados *end-points*. Podem ser usado contextos para implementar diversos recursos, como por exemplo, permissões, rotas ou privacidade. Desta forma, é acrescentado um nível de segurança a este processo [9].

Existem dois contextos especiais: [general] e [globals]. O general tem algumas configurações que definem o comportamento do ficheiro, entre as quais, a variável *static* que define que o ficheiro será rescrito aquando de alguma alteração, a variável *writeprotect* que controla a possibilidade do ficheiro extensions.conf ser alterado a partir do CLI do Asterisk [31]. O contexto *globals*, serve para indicar as variáveis globais do ficheiro como será posteriormente explicado neste documento.

No entanto, há forma de incluir um *extensions* de um determinado *context* a interagir com outro usando a diretiva *include*. Através da figura 2.7 [3] podemos perceber o funcionamento e a necessidade da existência de um determinado contexto na configuração do Asterisk.

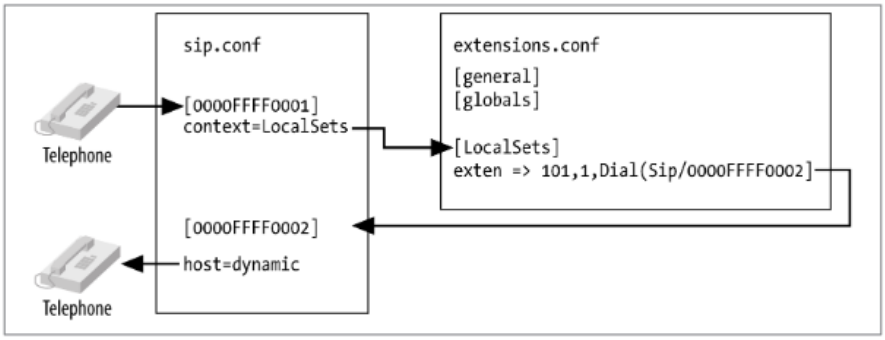


Figura 2.7: Relação entre os ficheiros de configuração e os contextos definidos no ficheiro *extensions.conf*

2.4.1.3 *Pattern matching*

É possível atribuir um padrão de ligação para determinadas extensões, isto é, associar uma letra a determinados dígitos e assim ser possível ligar para mais que uma extensão ou fazer *handling* de igual forma para ligações de determinadas extensões. Como exemplo, a figura 2.8, mostra que *extensions* entre 200 e 999 vão ouvir o som *"testmusic"*. A letra X refere-se a dígitos de 0 a 9, a letra Z de 1 a 9, e a letra N de 2 a 9 [3].

```
exten => _NXX,1,Playback(testmusic)
```

Figura 2.8: Exemplo de um padrão de correspondência

2.4.1.4 *Extensions*

Deste modo, quando o Asterisk, recebe uma chamada, de entrada ou saída, esta chamada irá pertencer a um determinado contexto. Este contexto está dependente do protocolo de comunicação em questão. Dentro de cada contexto, é possível definir quantas extensões o necessário. Desta forma, quando uma determinada extensão é desencadeada, o Asterisk segue os passos definidos para a mesma. Ou seja, são as extensões que definem o caminho que uma certa chamada irá ter. A sintaxe exigida para formular uma extensão é: *exten => name or number,priority,application()*. Desmistificando esta sintaxe, numa extensão é necessário definir o nome ou número da extensão, a prioridade e a aplicação que irá ser acionada.

2.4.1.5 Prioridade

A prioridade é um número sequencial, a partir de 1. Logicamente, para a mesma extensão as aplicações serão executadas de acordo com a sua prioridade. Atualmente, devido ao dimensionamento do dialplan, utiliza-se a prioridade *n* para várias extensões, que significa *next priority*.

2.4.1.6 Aplicações

As aplicações são as consequências visíveis do dialplan, isto é, cada aplicação do dialplan executa uma ação específica. Existem várias aplicações, sendo que algumas não necessitam de parâmetros, como *Answer()* ou *Hangup()*, outras precisam que sejam passados certos argumentos para que sejam devidamente executadas. A 2.9 [3], tem o exemplo do uso de aplicações no ficheiro *extensions.conf*.

As aplicações principais, mais utilizadas, são [9]:

- *Answer()*, para aceitar a chamada.
- *Playback(filename)*, para reproduzir algum ficheiro de som.
- *Hangup()* desligar a chamada.

```
[LocalSets] ; this is the context name
exten => 100,1,Dial(SIP/0000FFFF0001) ; Replace 0000FFFF0001 with your device name

exten => 101,1,Dial(SIP/0000FFFF0002) ; Replace 0000FFFF0002 with your device name

exten => 200,1,Answer()
   same => n,Playback(hello-world)
   same => n,Hangup()
```

Figura 2.9: Exemplo de código do ficheiro *extensions.conf*

- *Dial(technology/id, timeout, options)* faz com que um *end-point* sinalize a chamada.
- *Verbose* e *NoOp*, usadas para *debbuging*.
- *Goto(context, extension, priority)*: a chamada é direcionada para um *context*, extensão em específico, com uma prioridade associada tendo em conta determinada condição.
- *Voicemail(extension)* transfere a chamada para o *voicemail*.
- *Queue(queue_name | options)* coloca a chamada numa fila de espera, que deve ser configurada no ficheiro *queues.conf*.

Como exemplo, através do *sip.conf* é possível adicionar um utilizador do tipo *friend*, ou seja, é uma conexão que irá ser utilizada para fazer e receber telefonemas, partir e para o servidor local [31]. Na figura 2.10 [31], temos uma possível configuração.

```
[jrpbxuser]
type=friend
context=internal
username=jrpbxuser
secret=s3kr1tp@ss
mailbox=221
qualify=yes
host=dynamic
callerid="Joe Random PBX User" <3115550221>
dtmf=inband
```

Figura 2.10: Exemplo de código do ficheiro *sip.conf*

Answer(), Dial(), Hangup(): A aplicação Answer(), como o nome indica, serve para aceitar uma chamada. O seu primeiro parâmetro é um tempo (delay) em milisegundos, que serve para assegurar que o audio é processado no endpoint.

No ficheiro *extensions.conf*, é possível adicionar a extensão *exten => 221,1,Dial(SIP/jrpbxuser)*, na qual se faz corresponder a extensão 221 ao utilizador, com prioridade 1 e a chamada será através do protocolo SIP [31].

A aplicação serve para fazer uma chamada para outro dispositivo, como ilustra a figura 2.11. Como argumento, esta função recebe o tipo de tecnologia do canal de comunicação e o respetivo

nome que se encontra associado ao respetivo ficheiro de configuração. No seguinte exemplo será um dispositivo do tipo SIP. Esta aplicação também pode receber um determinado *timeout* como parâmetro, em segundos, que indica o tempo que a ligação estará activa enquanto nao for atendida e o parâmetro *options*, opcional, onde se indica opções como transferir a chamada ou desligar a chamada premindo um determinado dígito. Caso não seja atendida, é reproduzido o som *"notavailable"* e é desligada a chamada.

```
exten => 222,1, Dial(DIAL(SIP/jrpbxuser),10)
exten => 222,2,Playback(notavailable)
exten => 222,3, Hangup()
```

Figura 2.11: Exemplo de uso da aplicação *Dial()*

A figura 2.12 ilustra um exemplo simples de uso das aplicações Answer(), Playback() e Hangup(). A chamada para a extensão 222, irá ser atendida, depois irá tocar a musica *"testmusic"* e de seguida a chamada irá ser desligada.

```
exten => 222,1,Answer()
exten => 222,2,Playback(testmusic)
exten => 222,3, Hangup()
```

Figura 2.12: Exemplo de uso das aplicações *Answer(), Playback()* e *Hangup()*

Verbose() e NoOp() Estas aplicações são usadas para imprimir informação na linha de comandos. A aplicação Verbose(), recebe dois parâmetros, o primeiro caracteriza a verbosidade mínima para imprimir uma determinada mensagem, o segundo é a mensagem a imprimir[41].

A aplicação Noop(), significa *"No operation"*, basicamente tem a mesma finalidade que a aplicação Verbose(), no entanto, só imprime alguma coisa se a o nível de verbosidade for 3 ou superior.

Voicemail(): A aplicação do *voicemail* redireciona para a *mailbox* específica onde a mensagem será guardada. Um exemplo básico será, quando o utilizador não atende após um determinado *timeout*, a chamada é direcionada para o *voicemail* onde é possível, ao utilizador que originou a chamada, guardar a mensagem na mailbox do utilizador de destino. *VoiceMailMain()*, é outra aplicação relacionada, na qual é possível configurar uma mensagem de *voicemail*.

A configuração de uma *mailbox* tem a sintaxe ilustrada na figura 2.13 [3]. O parâmetro *mailbox* refere-se ao número correspondente da extensão associada, a *password* serve para aceder às mensagens gravadas nesta *mailbox*, o nome refere-se ao utilizador, assim como o *e-mail address.* O parâmetro *options* está relacionada com a *timezone* associada ao mesmo.

```
mailbox number => password,Nome,email address,options
```

Figura 2.13: Formato para configuração de uma *mailbox* no ficheiro *voicemail.conf*

Como exemplo, o *voicemail* pode ser configurado para o telemóvel e para o e-mail ao editar o respectivo ficheiro de configuração (*voicemail.conf*) com a seguinte linha de código: *221 => 90210, Joe Random PBXUSER, joe@example.net, joe@joecellphone.com* [31].

No ficheiro *extensions.conf*, as mensagens para serem guardadas no *voicemail*, é necessário passar dois argumentos à aplicação, conforme indicado na figura 2.14 [3]: a *mailbox* na qual a mensagem irá ficar e alguma opção relacionada, como por exemplo, marcar a mensagem como urgente [3]. No caso apresentado na figura 2.14, se a chamada, após 10 segundos não for atendida, será possível guardar uma mensagem no *voicemail* respetivo.

```
exten => 222,1,Dial(DIAL(SIP/jrpbxuser),10)
exten => 222,2,Voicemail(90210,option)
```

Figura 2.14: Formato para integração do *voicemail* no dialplan

Queue(): As filas de espera não são úteis, se não estiver garantido pelo sistema que as chamadas serão atendidas [3].

Para que os agentes respondam às chamadas feitas é necessário fazer uma gestão de recursos, podendo ser feita através do Asterisk CLI, através do ficheiro *queues.conf* ou dinamicamente através do dialplan (extensions.conf).

No grupo de suporte de uma cooperação, é normal que um agente não esteja constantemente com a sessão aberta para responder às chamadas portanto, ao usar o dialplan para usufruir de uma gestão dinâmica das filas é vantajoso. As aplicações que permitem fazer essa gestão são:

- *AddQueueMember()*
- *RemoveQueueMember()*
- *PauseQueueMember()*
- *UnpauseQueueMember()*

A diferença entre *pause/unpause* e *remove/add* é a possibilidade de permitir que um membro da *queue* esteja inativo ou ativo sem o remover da mesma [3].

Uma fila de espera, pode ser configurada de acordo com a figura 2.15 [31], onde definimos a estratégia para tocar em todos os *end-points* possíveis, com um determinado *timeout*, tempo em que o *end-point* era alertar da chamada e o *wrapuptime*, que é o tempo que o sistema tem de esperar para voltar a usar aquele *end-point* depois de este ter terminado uma sessão. O *periodic announce* define ficheiro de som que irá tocar enquanto a chamada ficar em espera, assim como o sua frequência. Finalmente, os membros da fila de espera.

No ficheiro *extensions.conf*, ficaria configurado da seguinte forma indicada na figura 2.16[31]:

Goto: Esta aplicação, recebe como parâmetros um determinado contexto, extensão e prioridade para direcionar uma chamada. No entanto também pode ser invocada apenas com os parâmetros

```
strategy=ringall
timeout=10
wrapuptime=30
periodic-announce = conglomocorp-your-call-is-important
periodic-announce-frequency=60
member=>SIP/10
member=>SIP/20
```

Figura 2.15: Exemplo de código do ficheiro *queues.conf*

```
[supportmenu]
exten => s,1,SetMusicOnHold(support)
exten => s,2,Playback(conglomocorp-welcome-to-support-queue)
exten => s,3,Queue(supportqueue)
```

Figura 2.16: Exemplo de integração de uma fila de espera no dialplan

extensão e prioridade. Uma variante é o *GotoIf()* que, consoante a uma determinada expressão seja verdade ou falsa, faz o encaminhamento.

```
exten => 123,1,NoOp()
    same => n,Answer()
    same => n,Set(COUNT=10)
    same => n(start),GotoIf($[${COUNT} > 0]?:goodbye)
    same => n,SayNumber(${COUNT})
    same => n,Set(COUNT=$[${COUNT} - 1])
    same => n,Goto(start)
    same => n(goodbye),Hangup()
```

Figura 2.17: Exemplo de uso da aplicação *GotoIf()*

Tendo em consideração que, quando estamos perante a mesma extensão, podemos usar o comando *same => priority,aplication* e a sintaxe das variáveis, a figura 2.17 [3] serve como um exemplo de uso da aplicação *GotoIf()*. Ao atribuir à variável *COUNT* o valor 10 e, de seguida, avaliar se *COUNT* é maior que 10, a direção será ou para a prioridade a seguir ou então para a prioridade com o rotulada como *goodbye*.

2.4.1.7 Funções

Dialplan permite o uso de funções úteis para dar mais robustez ao código, são funções inteligentes que permitem, por exemplo, calcular tamanho de strings, datas e tempo, assim como MD5 *checksums*. A sintaxe que é definida é: *FUNCTION_NAME(argument)*.

As funções do dialplan são similares às funções definidas nas várias linguagens de programação. São rotinas capazes de manusear dados de diferentes formas.

2.4.1.8 Macros

Macros são usados para evitar repetições no dialplan e auxiliam na alteração de configurações no mesmo [3].

Se, para um determinado utilizador, for definida uma série de instruções aquando de uma chamada para a sua extensão, que seja similar para outros utilizadores, é possível definir uma macro que contenha as instruções.

2.4.2 Integração com base de dados

A integração com a base de dados é uma etapa crucial no processo de construção de um sistema VoIP cooperativo baseado em Asterisk. O poder da base de dados é garantir a possibilidade de dinamicamente mudar as informações no dialplan [3]

A base de dados relacional utilizada na estrutura do IPBrick OS é o PostgreSQL e para que haja comunicação entre a base de dados e o asterisk é necessário criar uma camada de abstração para conseguir essa conexão. O *ODBC connector* permite que o Asterik comunique com a base de dados, em particular, com o PostgreSQL, a configuração para tal é feita no ficheiro */etc/odbinst.ini*. Para dar permissão aos módulos Asterisk para conectarem à base de dados, é editado o ficheiro */etc/asterisk/res_odbc.conf*. A figura 2.18 [3], mostra a relação entre os diferentes ficheiros de configuração acima referenciados.

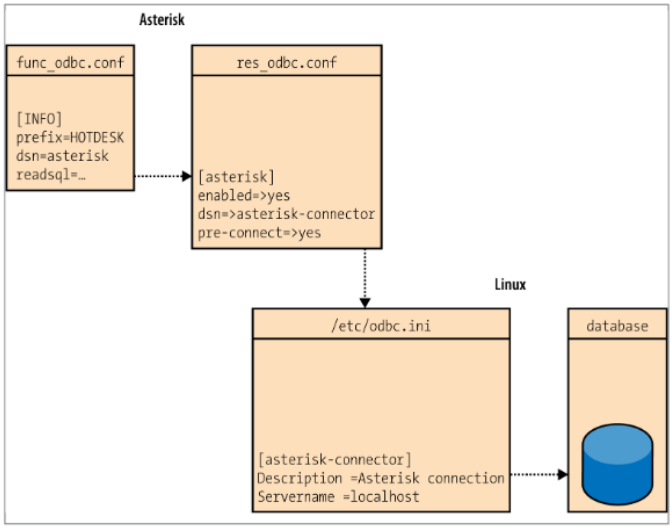


Figura 2.18: Relação entre os diferentes ficheiros de configuração que permitem a integração com a base de dados

2.5 Implementações gráficas de gestão do Dialplan

2.5.1 Visual PBX

Visual PBX (Figura 2.19 [42])é um inovadora *Graphic User Interface*, desenvolvida e distribuída pela empresa Apstel, com blocos de funções que têm a funcionalidade *drag and drop*. É uma solução que visa simplificar a configuração e usabilidade do Asterisk. Facilmente se consegue criar *IVR, conference rooms, call queues* e todas as outras funcionalidades dadas pelo Asterisk.

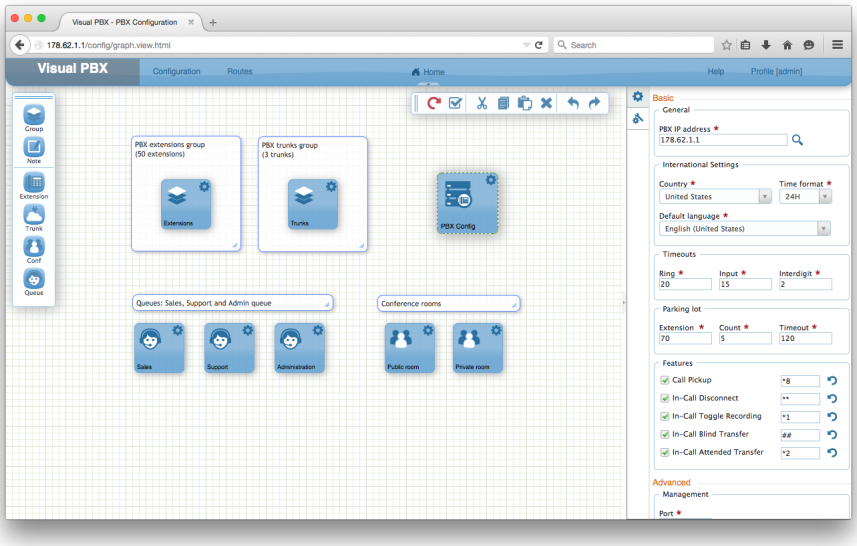


Figura 2.19: Interface VisualPBX

A instalação deste software é simples, através de uma imagem ISO, temos uma distribuição Linux com o Asterisk PBX e todos os *drivers* necessários.

2.5.1.1 Visual Dialplan

Visual Diaplan Professional (Figura 2.20 [43]) é também uma ferramenta distribuída pela Apstel com versão *trial* mas tem a diferença de ser integrado junto de um servidor Asterisk configurado e apenas é uma ferramenta usada para fazer o desenvolvimento e gestão do dialplan, na mesma, com modelação em *drag and drop*. Neste caso, é feita uma leitura as configurações do Asterisk assim como uma integração com uma base de dados relacional.

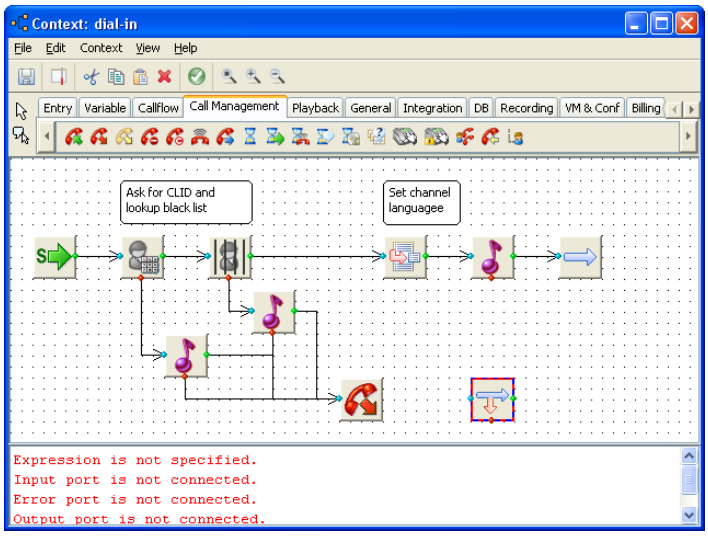


Figura 2.20: Interface Visual Dialplan

Esta solução é antiga face ao VisualPBX, apresenta um design pouco inovador e não é uma aplicação acessível a partir de uma página web, não sendo uma solução de todo viável para os objectivos da IPBrick SA.

2.5.2 Jive Dial-Plan Editor

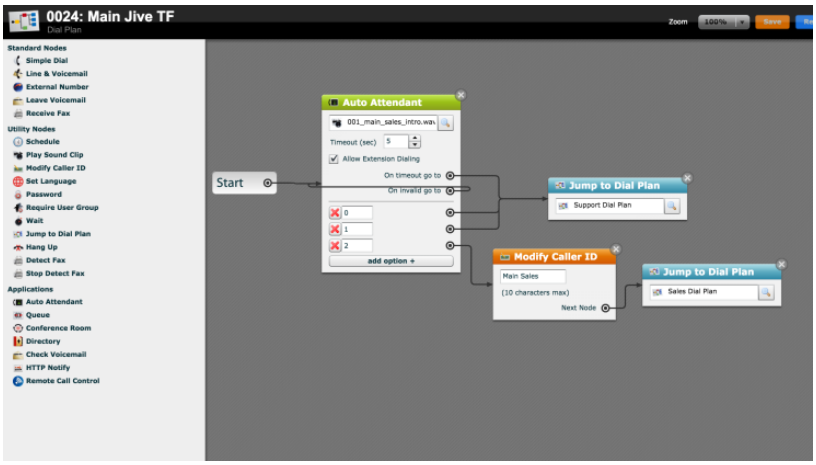


Figura 2.21: Interface Jive Dialplan

A Jive Communications é uma empresa que providencia soluções cooperativas para comunicações unificadas. Um dos seus produtos, é o *Dial Plan editor*, (Figura 2.21 [44]), uma ferramenta para mapear os *call flows* visualmente e configurá-los através da metodologia *drag and drop*.

Todas as configurações possíveis de se fazerem através do dial-plan são feitas através de blocos interligados, cada qual com a sua respetiva função. Esta solução visa simplificar e melhorar o processo de gestão do dialplan através de uma solução gráfica. No entanto, esta solução é paga e vem enquadrada com a distribuição do sistema VoIP da empresa.

2.6 Bibliotecas JavaScript

O módulo de gestão gráfico para o dialplan irá fazer parte da interface web de administração de um sistema IPBrick, deste modo, a sua implementação será feita tendo como princípio a compatibilidade entre este módulo e toda a arquitetura existente no sistema operativo. O fluxo de dados feito entre a interface, sistema e base de dados é feito usando essencialmente a linguagem web PHP, sendo que, do lado do cliente é usado código web (HTML e CSS) e *frameworks JavaScript* que são plenamente compatíveis.

Desta forma, o módulo de gestão para o dialplan deverá ser elaborado usando uma linguagem de programação web que permita a criação de interfaces interativas. A linguagem de programação JavaScript, a qual tem associada várias bibliotecas e frameworks, é a que permite um melhor desenvolvimento de interfaces de acordo com a premissa anteriormente referida.

No entanto, usar simplesmente código JavaScript nativo não seria viável para gerar uma página de gestão como esta, a qual deverá constituir um menu de blocos arrastáveis com propriedades

associadas, que permitam criar conexões entre si e que, de alguma forma, seja capaz de importar dados estruturalmente descritos e gerar uma representação gráfica a partir dos mesmos, assim como o processo inverso de obter informação acerca das interações realizadas. A juntar a esta necessidade, a biblioteca deve ser open-source, uma vez que, este módulo irá fazer parte de um produto comercial, o IPBrickOS, portanto, não deverá haver interferências com o licenciamentos. Deste modo, foram abordadas possíveis bibliotecas JS adequadas a esta necessidade.

2.6.0.1 jQuery UI

O *jQuery* UI é uma biblioteca permite criar *graphic user interfaces* devido aos *widgets* e interações associadas, entre elas, permite a criação de blocos arrastáveis e, com a integração de outros plugins, a capacidade de criar ligações entre eles. Apesar de todas as animações e interações que esta biblioteca permite criar, não dispõe de funcionalidades suficientes e acrescenta pouco ao que se consegue criar nativamente com *JavaScript*.

2.6.0.2 FlowChart jQuery

FlowChart jQuery surge como um *plugin*, *open-source*, que permite o desenho e edição de um *flowchart* através de blocos que têm parâmetros como *inputs*, *data* e *outputs*. Ou seja, permite que sejam criados ou removidos operadores em cada bloco assim como conexões entre as mesmas [45]. Desta forma, este *plugin* é uma forma de potencializar das funções *jQuery* e seria uma boa solução para o problema, no entanto, carece na falta de funcionalidades necessárias para o criação do módulo, nomeadamente a possibilidade de exportar ou importar informação estruturada.

2.6.0.3 GoJS

GoJS é uma biblioteca HTML/JavaScript que permite construir vários tipos de diagramas. Foi criada e está sob a licença da *Northwoods Software*. Esta API é rica em funcionalidades, permite que criação de diagramas interativos sendo a sua tecnologia compatível com diferentes *browsers* atuais e integrável com outras bibliotecas como *jQuery*. Tem a vantagem de não ser necessário a instalação de plugins adicionais, dependências ou outras bibliotecas, para além de ter uma documentação detalhada e vários exemplos que permitem reduzir o declive da curva de aprendizagem. Apesar de todas as vantagens referidas, esta biblioteca para ser utilizada na sua versão mais recente e para ser integrada num produto de ordem comercial é necessária a aquisição de uma licença paga. Ou seja, neste caso particular, com referido em 2.6, o módulo será integrado num produto comercial, IPBrick OS e como tal, a escolha deverá recair numa biblioteca *open-source*.

2.6.0.4 JointJS

JointJS é mais uma biblioteca que permite a criação de diagramas estáticos ou interativos na linguagem *JavaScript*. Apesar da sua potencialidade, exige a dependências que podiam entrar em

conflito com o sistema existente. Como anteriormente referido o *backend* já implementado na IPBrick é segundo a linguagem PHP, ou seja, a incorporação de tecnologias recentes que necessitem de uma infraestrutura de *backend* em *JavaScript* (como o *Node.js*) não é compatível com o sistema atual da IPBrick.

2.6.0.5 D3-node-editor

Em termos de *frontend* é de facto uma biblioteca que se destaca pela sua simplicidade, vanguardismo e usabilidade em termos de programação visual. É uma solução *open-source* que apresenta uma arquitetura simples e de fácil implementação, sem interferências no que diz respeito a possíveis dependências de *backend* com o sistema da IPBrick ou de licenciamento. No entanto, comparativamente com outras bibliotecas abordadas, carecia de funcionalidades que podem ser úteis e vantajosas para o módulo de gestão gráfico.

2.6.0.6 JSPlumb

JSPlumb é uma biblioteca *JavaScript* que permite a visualização gráfica de elementos numa página web, utilizando SVG (*Scalabale Vector Graphics*), linguagem XML W3C standard que permite descrever diferentes tipos de gráficos de forma vetorial. Esta biblioteca está disponível em duas versões, *toolkit* e *community*, sendo que a versão que não necessita de uma licença paga é a versão *community* presente no GitHub. Apresenta uma documentação detalhada e a facilidade de uso, uma vez que não requer a instalação de *plugins* adicionais ou outras dependências. Esta biblioteca, com todas as suas potencialidades, será uma solução adequada, tendo em conta as necessidades da interface gráfica em questão.

2.6.0.7 MxGraph

MxGraph é uma biblioteca *open-source* sendo a base de implementação do *Draw.io*, ferramenta da Google. É usada para criação de vários tipos de grafos com o uso de diferentes blocos conectáveis e editáveis. Segundo [46], é um projeto que suporta vários tipos de *browsers*, não usa *third-party software*, não necessita de *plugins* adicionais e pode ser virtualmente integrada com qualquer *framework*. Para além de estar em constante atualização, com possíveis correções de bugs existentes. Apresenta uma documentação detalhada e vários exemplos de aplicação, pelo que, considerando também a vasta lista de potencialidades da mesma, é uma solução viável para implementação da interface gráfica de gestão.

Comparação das bibliotecas A tabela 2.1 apresenta um resumo da avaliação das bibliotecas mencionadas, realçando os critérios mais relevantes que as permitem qualificar. A curva de aprendizagem relaciona-se com a complexidade associada à biblioteca e às suas tecnologias inerentes. A arquitetura, relaciona-se com as potencialidades técnicas que permitam desenvolver interfaces mais completas. A compatibilidade relaciona-se com o facto de a biblioteca poder ser, ou não,

integrada no sistema da IPBrick. No caso da documentação, relaciona-se com o que é disponibilizado ao utilizador para que seja reduzida a curva de aprendizagem. As funcionalidades referem-se ao que é possível de implementar com a biblioteca em questão. A licença é um fator fundamental, uma vez que, é necessário que seja uma biblioteca *open-source* e que possa ser incluída em produtos comerciais. Por fim, a fiabilidade relaciona-se com a credibilidade da biblioteca e com a sua manutenção. A legenda correspondente é:

- ++ : Muito boa opção
- + : Boa opção
- +- : Opção Suficiente
- - : Má opção
- –: Muito má opção

Tabela 2.1: Tabela comparativa das bibliotecas *JavaScript*

	jQuery UI	Flow Chart jQuery	GoJS	D3 node editor	JointJS	JSPlumb	MxGraph
Curva de Aprendizagem	++	++	+-	++	+-	+-	-
Arquitetura	–	–	++	+-	+-	++	++
Compatibilidade	++	++	++	+-	–	++	++
Documentação	++	++	++	+-	–	++	++
Licença	++	++	–	++	+-	+-	++
Funcionalidades	–	–	++	+-	+-	+-	++
Fiabilidade	++	++	++	-	++	+-	++

Capítulo 3

Caracterização do Problema

Neste capítulo é apresentada a caracterização e definição do problema, assim como a solução proposta e quais as tecnologias envolvidas.

3.1 Definição do problema

A gestão do dialplan pode ser tornar-se numa tarefa hercúlea para um administrador do sistema, uma vez que, há várias dependências associadas e inúmeras configurações de rotas possíveis. O IPBrick OS, a partir da sua interface de administração, disponibiliza ao administrador a capacidade de configurar os vários serviços de rede do sistema. Os serviços poderiam ser configurados a partir de uma linha de comandos mas a interface de gestão administrativa do IPBrick OS, tem como objetivo principal, simplificar os processos de configuração. O dialplan é uma das várias páginas associadas à interface, ilustrada na figura 3.1. As rotas configuradas (rotas de saída, entrada e internas) são mostradas por intermédio de tabelas, no entanto, o dialplan interno, que engloba as regras de entrada e internas a um certo domínio, será o foco do módulo gráfico.

Figura 3.1: Página atual dialplan

As rotas de entrada podem assumir vários caminhos, como referido no capítulo anterior. Como tal, ao serem criadas funções, endereços para utilizadores ou telefones através das páginas correspondentes, essas configurações serão depois visíveis na página do dialplan. As diferentes regras e recursos disponibilizados pelo Asterisk, no que diz respeito a rotas de entrada, são acedidas como ilustra a figura 3.2.

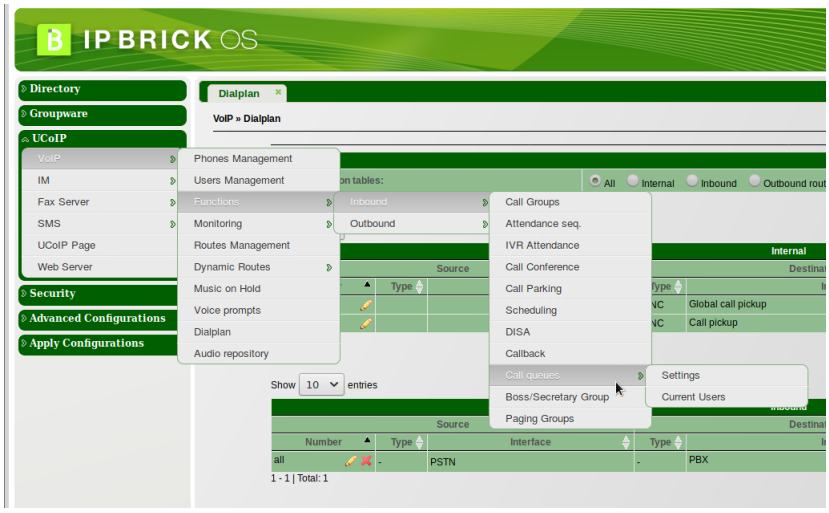


Figura 3.2: Configurações de rotas de entrada

As chamadas podem conter fluxos originados pelas rotas criadas e que são formadas a partir de funções VoIP disponibilizadas para configuração através da interface de administração. Das funções disponibilizadas, nem todas serão incluídas na gestão gráfica do dialplan, uma vez que, nem todas são igualmente comuns numa configuração num cenário real. Deste modo, as que se destacam pela sua usabilidade e relevância são: sequências de atendimento, grupos de chamada, conferências, filas de espera, IVRs e escalonadores. Estas funções podem ter associações com telefones e utilizadores registados, sendo a sua gestão feita a partir das páginas *Phones Management* (Figura 3.3) e *Users Management* (Figura 3.4), os quais, no caso de terem endereços, são vistos como regras internas e serão mapeados na tabela que corresponde às *internal routes*.

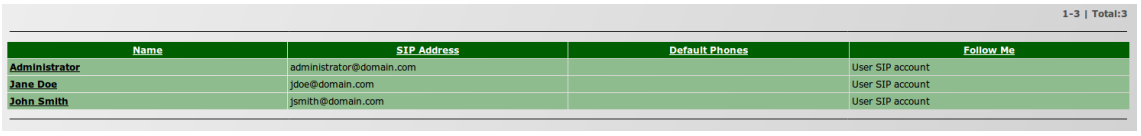


Figura 3.3: Página *Users Management*

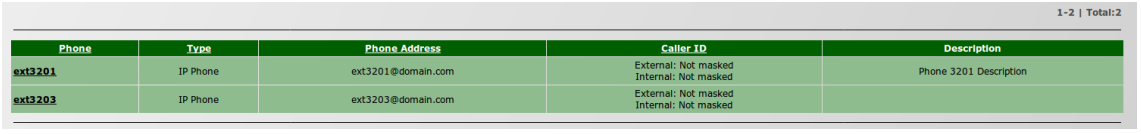


Figura 3.4: Página *Phones Management*

Tendo em conta a página atual do dialplan, é sensato afirmar que esta não permite uma visão clara das associações que existem, cuja a combinação resulta em rotas que determinam o caminho que uma determinada chamada pode ter. O módulo de gestão gráfico irá colmatar essa falha, proporcionando uma visão clara do fluxo de chamadas configurado em sistema.

O esquema da figura 3.5 serve como visão geral das dependências e das ligações que estão adjacentes à gestão do dialplan. A aplicação *web* caracteriza-se como sendo o serviço central de toda esta arquitetura, onde se registam telefones, utilizadores e onde se faz a configuração das funções VoIP. Os *exports* feitos para o sistema, permitem que todas essas alterações feitas a partir da interface, as quais se refletem na página do dialplan, sejam traduzidas para os ficheiros de configuração do Kamailio e Asterisk, em particular, para o ficheiro *extension.conf* e *sip.conf*. A relação com a base de dados tem também um forte impacto, dado que, todas as propriedades associadas às funções, telefones e utilizadores são guardadas nas respetivas tabelas. Qualquer alteração ou criação feita na interface, após submissão, reflete-se de imediato nas tabelas da base de dados, enquanto que, para se refletir em sistema é necessário aplicar configurações a partir da interface de administração. Só depois de serem aplicadas configurações é que os *exports* para o sistema são efetuados.

Deste modo, interface web é um elemento chave para proporcionar integridade dos dados e das configurações. As tecnologias inerentes à interface podem ser usadas como uma forma de comprometer o sistema, através de injeção de código malicioso. A ocorrência de vulnerabilidades que comprometam o sistema, podem levar a que, por exemplo, neste caso particular, rotas indesejadas sejam criadas ou que utilizadores sejam adicionados de forma a beneficiar do serviço VoIP configurado num sistema IPBrick.

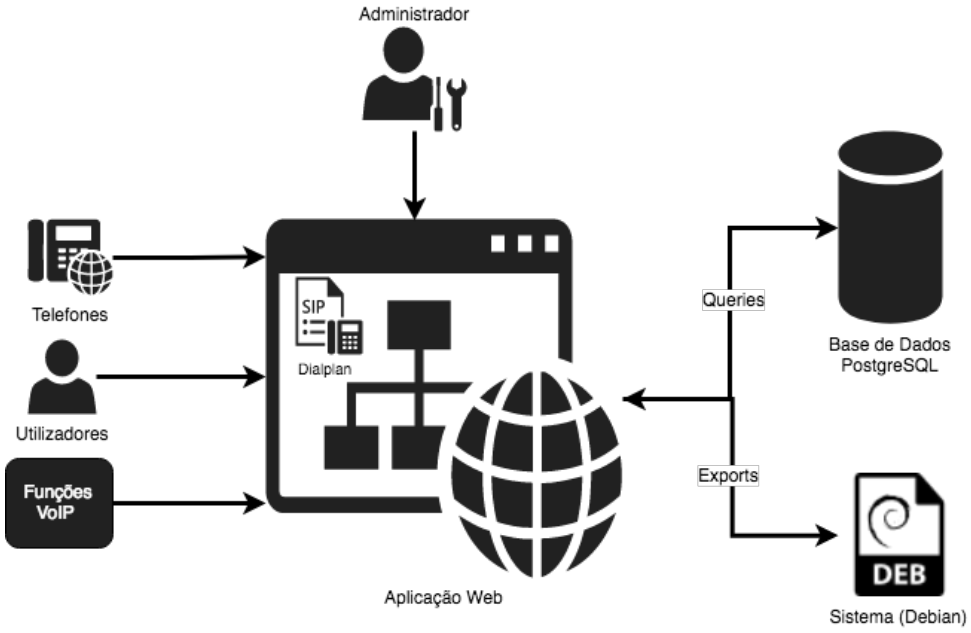


Figura 3.5: Arquitetura do sistema

Por outro lado, há problemas que podem advir de uma má gestão do dialplan, sendo os mais

problemáticos, a existência de colisões nas rotas ou a existência de *loops* causados por uma configuração descuidada. Como consequência, podem existir sessões que são interrompidas ou uso excessivo de recursos do sistema se os *loops* não forem detetados. Parte integrante deste projeto será procurar manter a integridade dos ficheiros de configuração, validar as configurações efetuadas e garantir que robustez de código suficiente na construção do módulo gráfico de forma a tornar o sistema o menos vulnerável possível. Para além de que, este projeto deve permitir a análise de todos os fluxos de chamadas e a sua facilidade de interpretação para que a administração do sistema se torne mais eficiente, uma vez que, permite detetar mais facilmente comportamentos indesejados (*loops* ou encaminhamentos para destinos que já não fazem sentido).

3.2 Dialplan IPBrick

A criação e configuração das diferentes funções VoIP é feita através da páginas correspondentes a cada uma, sendo a página web do dialplan uma visualização, em forma de tabela, das funções criadas. As funções criadas na IPBrick têm a possibilidade de lhes serem atribuídas acessos diretos e restrições de *caller* ID.

3.2.0.1 Caller ID

Caller ID é o um recurso que serve para identificar um chamador, ou seja, é o número de telefone que será exibido no telefone do destinatário quando a chamada tem uma determinada origem. Desta forma, é garantida a integridade entre as ligações, apresentando uma informação que seja útil ao destinatário - pode ser o número original da chamada, ou até uma *string* composta pelo nome de uma função e do chamador (e.g. FilaSuporte-91XXXXXXX").

Caller ID Restrictions Um *caller* ID é um identificador de um determinado elemento no sistema VoIP, podendo ser atribuído a endereços que fazem parte do domínio. Deste modo, *caller ID restrictions* é definida como uma *whitelist* dos números que podem ligar para os acessos definidos.

3.2.0.2 Acessos Diretos

Os acessos diretos permitem que determinadas funções sejam acedidas através de um endereço específico atribuído, ou seja, dentro de um determinado domínio irão ter um endereço atribuído, o qual poderá estar ou não limitado a certos números. Os acessos diretos podem ser de dois tipos: SIP e DDI.

Acesso Direto SIP Um endereço SIP é similar a um endereço *email*, serve como um identificador e é composto por um nome e um domínio: *example@domain.com*.

Acesso Direto DDI DDI significa *Direct Inward Dialing*, é um recurso dado pelas operadoras para uso num sistema PBX, que permite que uma empresa atribua um número a uma entidade

sem necessidade de uma linha física separada para cada um. Num sistema de comunicações VoIP, permite que utilizadores da rede telefónica pública comuniquem com utilizadores VoIP, sendo o DDI usado como *gateway*.

3.2.1 Gestão de Telefones

O menu lateral correspondente à gestão de telefones (*Phones Management*) permite obter uma lista de todas as contas VoIP registados numa IPBrick. Os dados atribuídos a cada telefone são o nome, *password*, um endereço numérico, *Caller* ID externo e interno, opções de *voicemail* e de auto-aprovisionamento, assim como uma descrição do mesmo (Figura 3.6).

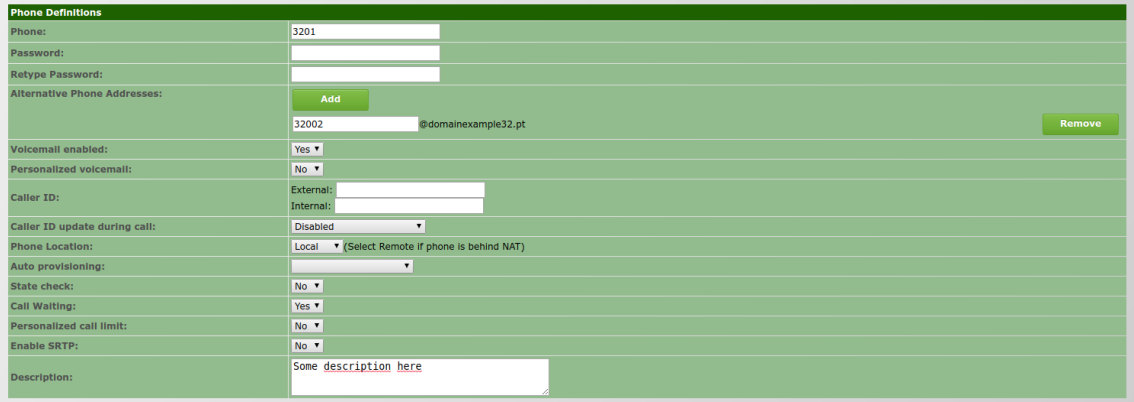


Figura 3.6: VoIP - Gestão de Telefones

3.2.2 Gestão de Utilizadores

Na página (*Users Management*) é apresentada uma lista dos utilizadores, os quais têm associado um endereço do tipo SIP. Aos utilizadores, podem ser associados endereços alternativos e telefones. Pode ser configurado opcionalmente um PIN numérico que serve para aceder ao telefone, às classes de acesso, ao *voicemail* e a filas de espera caso o utilizador esteja associado a alguma dessas funções. Tal como no caso dos telefones, é possível definir um *caller* ID interno e externo (Figura 3.7).

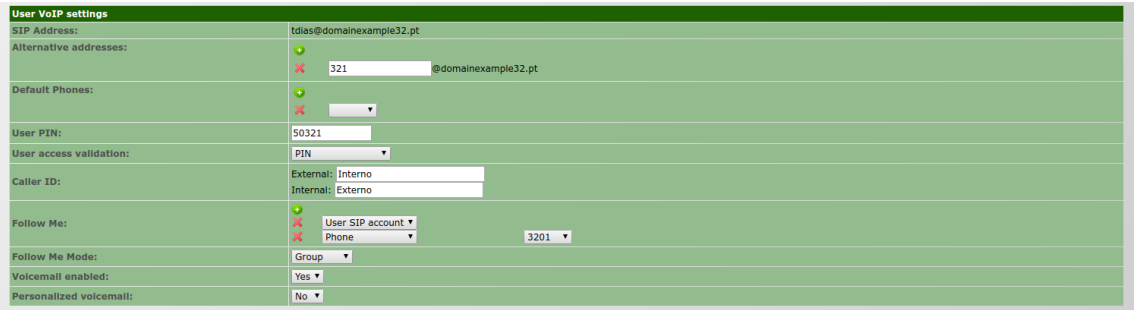


Figura 3.7: VoIP - Gestão de Utilizadores

Follow Me Esta função permite que a um determinado utilizador seja associado um encaminhamento da chamada, podendo ser configurado o encaminhamento para um telefone, outro utilizador ou um endereço externo, como por exemplo um número móvel. O modo do encaminhamento pode ser feito em grupo ou em sequência, em grupo será feito o encaminhamento simultâneo para o destino configurado, em sequência será feito por ordem de inserção.

3.2.3 Funções

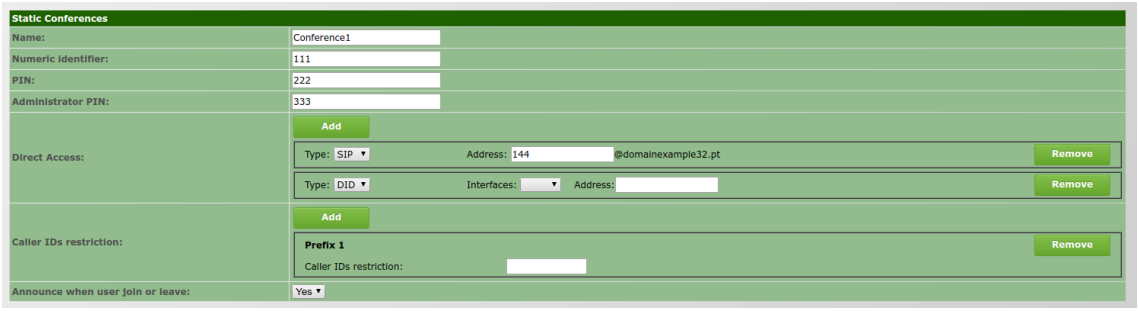


Figura 3.8: VoIP - Conferências

Conferências Uma conferência tem como atributos nome, identificador numérico e PIN de acesso para os utilizadores se conectarem, assim como um PIN para o administrador da conferência. É possível definir também acessos diretos e restrições de *caller* ID. (Figura 3.8).

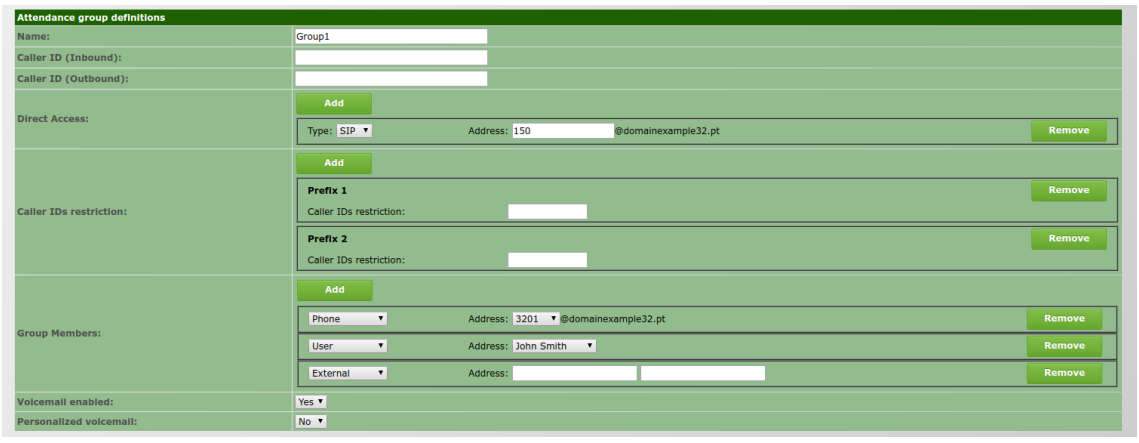


Figura 3.9: VoIP - Grupos de Chamada

Grupos de chamada Nos grupos de chamada são inseridos membros, os quais são distinguidos pelo tipo, podendo ser utilizadores, telefones ou endereços externos. Quando o acesso ao grupo de chamada é feito, os endereços associados irão tocar ao mesmo tempo. É possível configurar um *caller* ID interno ou externo, acessos diretos e restrições de *caller* ID (Figura 3.9).

Sequências de atendimento Sequências de atendimento são similares aos grupos de chamada, apenas diferindo no facto de que os membros têm uma ordem associada quando é feito o acesso

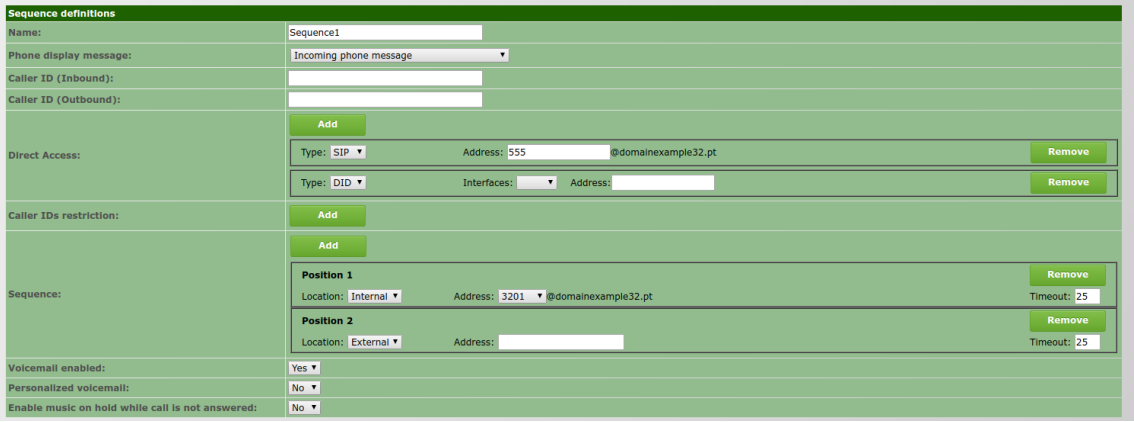


Figura 3.10: VoIP -Sequências de atendimento

à sequência (Figura 3.12). Para além da ordem que irão tocar, também têm um *timeout* definido caso não haja resposta. Os membros internos são telefones registados no servidor local, enquanto que os membros externo podem ser um endereço externo ou um endereço local.

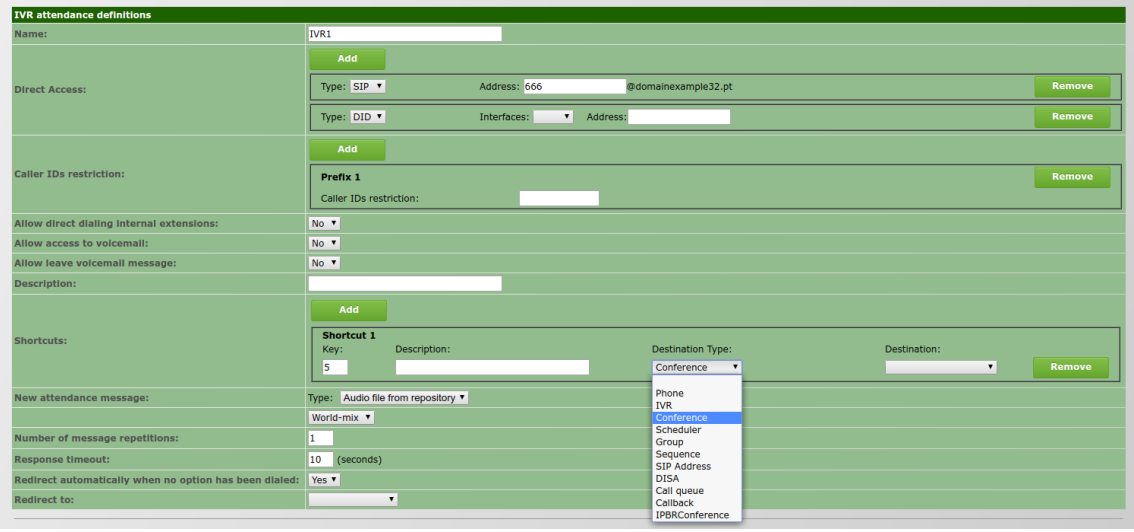


Figura 3.11: VoIP -Interactive Voice Response

IVR IVR (*Interactive Voice Response*) é uma função que serve como elemento de resposta automática, o qual tem associado um áudio que pode servir de orientação ou de informação. No caso de ser um processo de orientação para um determinado utilizador que aceda ao IVR, este pode ter atalhos que servem como caminhos para o utilizador, cujas opções encaminham para outras funções. Qualquer função ou telefone presente no sistema pode ser associada a este elemento (Figura 3.11), assim como há possibilidade de redirecionar a chamada, também para qualquer função ou elemento, após um certo *timeout*.

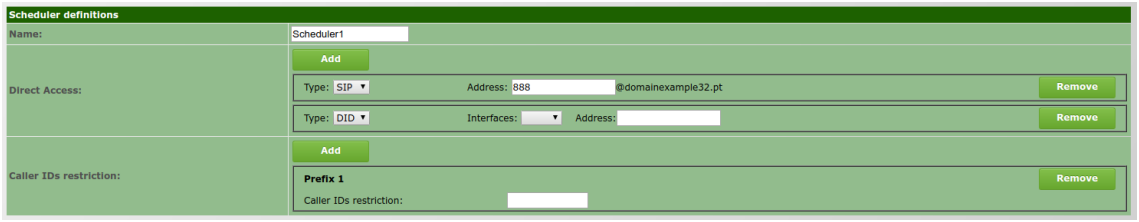


Figura 3.12: VoIP - Propriedades Escalonador

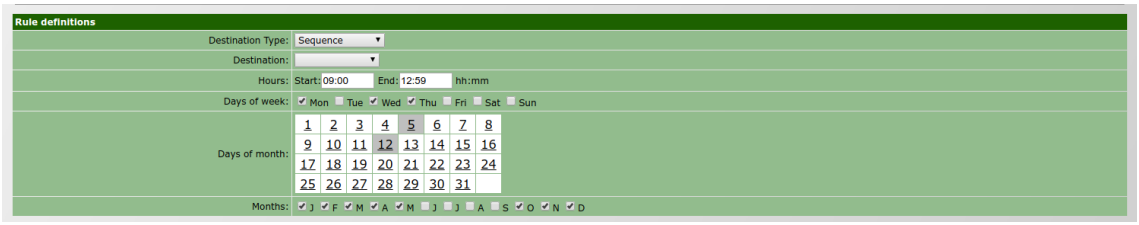


Figura 3.13: VoIP - Regras Escalonador

Escalonadores Os escalonadores servem para definir o encaminhamento das chamadas consoante horários definidos, podendo ser definidas horas, dias e meses. Estas regras de encaminhamento devem ter associado um destino, que pode ser qualquer uma das funções VoIP configuradas no sistema, ou um telefone ou um endereço SIP específico (Figura 3.13).

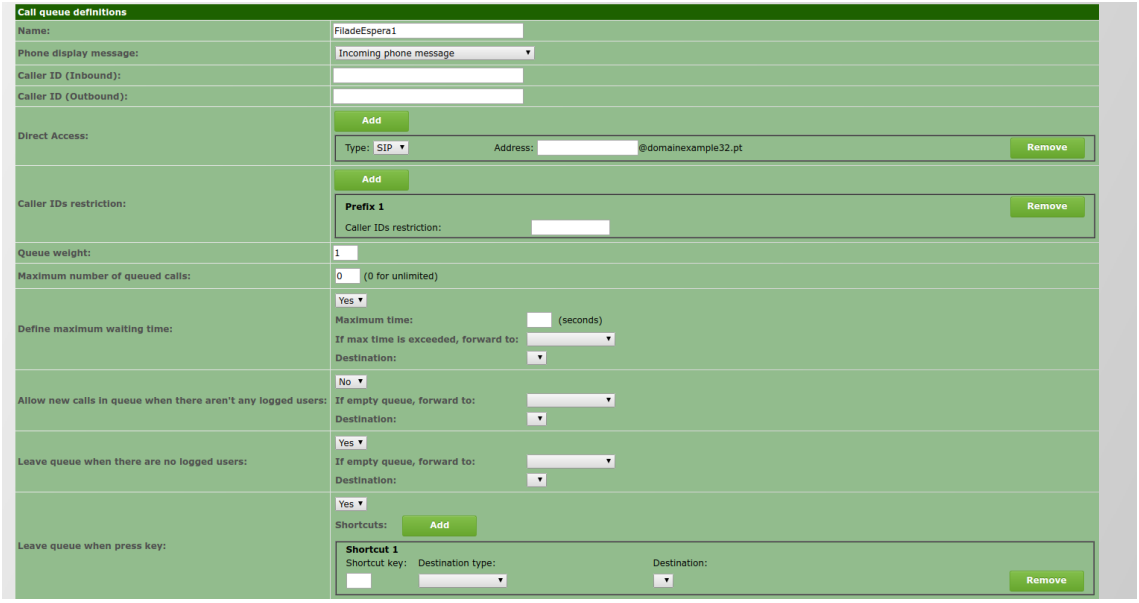


Figura 3.14: VoIP - Filas de Espera

Filas de Espera As filas de espera servem para colocar chamada em espera para ser respondida, podendo ser definida uma mensagem de resposta enquanto a chamada estiver em espera. Esta função é a mais complexa, uma vez que dispõe de várias propriedades que podem ou não ser configuradas. É possível definir o número máximo de chamadas em espera, o tempo máximo

de espera, a possibilidade de permitir ou não novas chamadas no caso da fila se encontrar vazia e a possibilidade da chamada deixar a fila caso não haja utilizadores disponíveis para atender ou deixar a fila premindo uma tecla. Em todos estes casos, no caso de serem configurados, é necessário encaminhar a chamada para outro destino, sendo que, o destino pode ser qualquer uma das outras funções definidas, telefones, ou mesmo um endereço externo (Figura 3.14).

3.3 Solução proposta

Atualmente, a visualização do dialplan da IPBrick carece de clarividência, visto que, não é possível ter uma perceção clara das configurações feitas. Por conseguinte, a solução proposta passa por, em termos gerais, usar tecnologias *web* atuais e plenamente compatíveis com o sistema atual da IPBrick, de forma criar um módulo gráfico com um design *user-friendly* que permita, primeiramente, transportar a informação que se encontra tabelada na página no dialplan para uma representação gráfica, na qual se deve visualizar de forma clara e percetível todas configurações e associações que já existam. Em segundo, deve permitir que sejam realizadas configurações de rotas, através de um mecanismo *drag and drop*, utilizando blocos arrastáveis que se possam interligar entre si e que tenham atributos associados. Dado que, para cada tipo de função, existe uma página que permite a listagem da mesma, assim como uma página de inserção, de modificação e de eliminação, sendo que, os telefones e utilizadores que seguem o mesmo modelo, o módulo deverá também deve ser capaz de abstrair essas ações numa só, ou seja, cada função pode ser criada, eliminada ou alterada a partir da interface e, logicamente, ao serem exportados os elementos, a página de listagem também fará parte dessa abstração.

Portanto, tendo em conta esta abstração necessária e considerando os elementos descritos em 3.2, é notória a complexidade envolvida, devido à quantidade de informação relacionada com cada bloco e com as possíveis interações que pode ter. Existem elementos mais e menos complexos, sendo que, esta complexidade traduz-se nas possíveis configurações que podem ser feitas a parte do mesmo e as informações que lhes estão associadas. Desta forma, é necessário ponderar quais das informações são relevantes serem visíveis na interface. Dentro das que são necessárias ser visíveis, é necessário distinguir o que serão atributos do elemento ou que podem ser representados através de uma ligação para o mesmo.

Como tal, para que esta solução seja conseguida, é necessário modularizar os elementos ao máximo, de modo a criar um cenário gráfico, em que cada elemento do dialpan irá ser visto como um bloco arrastável, que poderá ser conectado a outros. Para que esta modularização seja o mais congruente possível é necessário avaliar e distinguir, de entre os elementos, o que será visto como um atributo de um bloco e o que deve ser estruturado como um bloco.

Uma vez que, cada função, pode ter na sua configuração vários tipos de acesso direto e restrições de *caller* ID, estes elementos não serão vistos como um atributo da função mas sim como um bloco de entrada para a mesma, definido como regras de entrada (*Inbound Rules*). Para promover a unanimidade desta abordagem, os telefones e utilizadores também podem ter endereços alternativos, estes endereços serão vistos como um bloco de entrada, uma vez que, são igualmente vistos

como acessos ao elemento. Em termos gráficos, não existe mais nenhum atributo que seja lógico modularizar, portanto serão apenas associados aos elementos.

Transportar a gestão do dialplan para um ambiente visual requer que cada elemento seja pensado e modulado de forma diferente, dado que, por exemplo, um IVR não tem as mesmas características que uma conferência, e o mesmo aplica-se aos outros elementos. Assim sendo, cada bloco deverá ter uma estrutura definida face às suas propriedades.

Para além da estrutura, a sua aparência também deve ser considerada, dado que, este módulo apesar de ser orientado à gestão, deverá possuir uma interface que conquiste pela sua usabilidade. *UX - User Experience* é a disciplina de estudo para que, numa interface, sejam alcançada a utilidade, a facilidade e satisfação de uso. Uma metodologias usadas envolve o *benchmarking* de serviços similares e, como evidenciado em 2.5, as implementações já existentes de uma interface gráfica para gestão do dialplan, apresentam cada elemento da interface com um *design* adequado às suas propriedades e de com algum tipo de identificador do tipo de elemento em questão.

Desta forma, cada elemento que irá compor este módulo gráfico deverá ter associado algum tipo de identificador, para que a representação gráfica do dialplan seja o mais clara e evidente possível. Para além da estrutura definida em conformidade com os elementos, será associada a cada elemento uma cor identificativa. Estas cores foram pensadas de modo a não pôr em causa a experiência do utilizador, ou seja, não foram escolhidas cores com tons fortes nem com demasiado impacto visual para tornar a leitura do dialplan o mais legível possível e o design da interface o mais limpo possível.

Para além deste aspeto, foi necessário considerar que, em certos casos, adotar o bloco no seu todo como sendo o *input* e *output* não iria fazer sentido. O caso que permite evidenciar este fator, é o caso da sequência de atendimento, em que, caso os membros da sequência se ligassem todos ao bloco da sequência, não iria ser possível ter a perceção da ordem que lhes está associada. Ou seja, apesar de termos uma representação gráfica do objeto, não iríamos conseguir ter uma visão clara das configurações que lhes estavam associadas. Deste modo, faz sentido que os blocos que têm vários destinos possíveis, em que, cada um deles deve ser diferenciado, contendo um *output* diferente para cada um deles. Nos casos em que a diferenciação não é significativa que seja destacada, o próprio bloco é definido como *input* e *output* nas conexões que serão efetuadas.

Interface gráfica O *mockup* apresentado na figura 3.15 ilustra, uma proposta de estrutura para interface de gestão gráfica. Será composta por menu lateral do lado esquerdo, onde estarão os blocos, uma divisão central onde será feita a representação gráfica e onde será possível realizar as operações de configuração e um menu lateral do lado direito que será visível quando um determinado bloco for selecionado, contendo os atributos do mesmo. As alterações feitas às atributos dos blocos serão guardadas *on demand* sendo o *export* para sistema feito quando se aplicar as configurações. Dado que, em cenários reais, um dialplan pode ser composto por vários elementos que dão origem a diferentes rotas, existem funcionalidades que são uma mais valia para melhor a usabilidade da interface, destacam-se o facto ser possível ajudar o zoom, agrupar, copiar e remover blocos e, não menos importante, a funcionalidade de poder reverter ações.

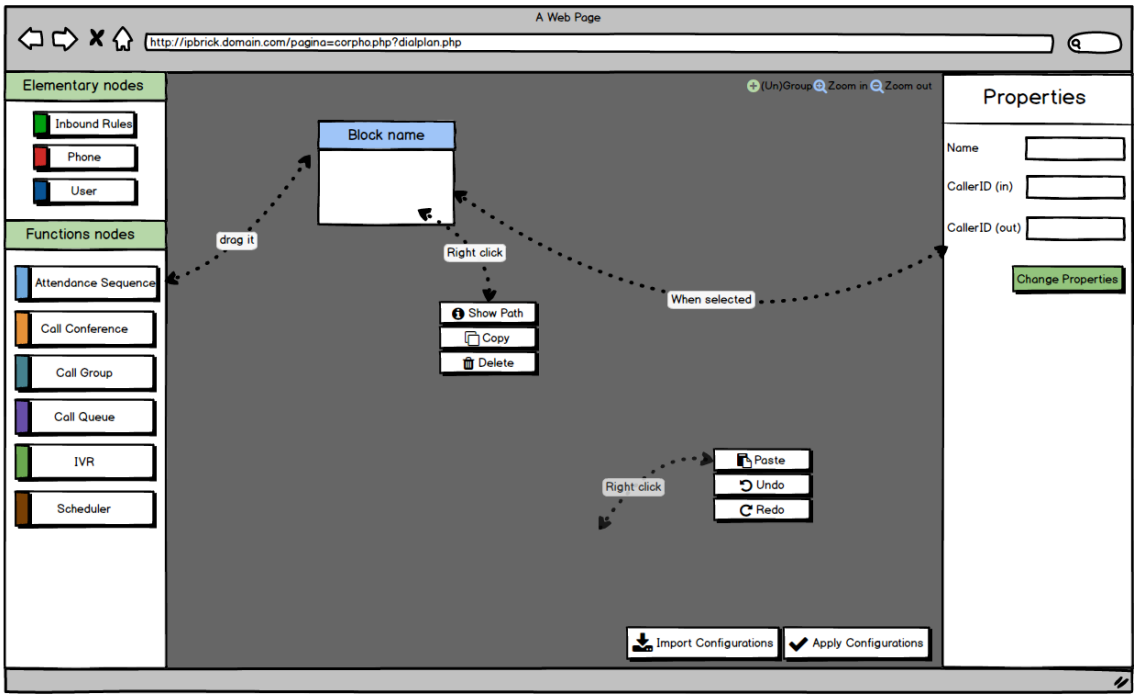


Figura 3.15: Proposta de *layout* para módulo gráfico

Acessos Diretos e Restrições de *Caller* ID Os acessos diretos e as restrições de *caller* ID foram pensados de forma a constituírem o bloco de regras de entrada para uma determinada função, figura 3.16 . Esta abordagem deve-se ao facto de ser intuitivo para o administrador olha para este par em conjunto, dado que, os acessos diretos mostram quais os acessos que uma determinado elemento pode ter e, as restrições de *caller* ID, constituem o conjunto de números que podem aceder aos acessos diretos desse mesmo elemento. No caso de não haver restrições, qualquer número poderá aceder aos acessos do elemento. Juntamente a este aspeto, ao aglomerar estas duas propriedades, reduz-se o ruído introduzido no caso de serem elaborados dois blocos em separado. No entanto, existem casos em que faz sentido haver acessos diretos e não existir restrições de *caller* ID, como é o caso dos endereços alternativos associados aos telefones e utilizadores, em que apenas se atribui acessos diretos à função. Posto isto, a solução passa por manter a mesma estrutura do bloco ilustrado na figura 3.16 mas apenas com a hipótese de adicionar acessos diretos.

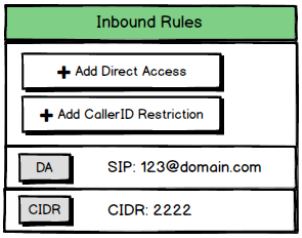


Figura 3.16: *Mockup do bloco Inbound Rules*

Telefones e Utilizadores Estes dois elementos serão representados por blocos simples 3.17, sem nenhuma complexidade associada uma vez que em termos visuais basta ter a perceção do nome e do tipo de elemento. Serão visíveis, no meu lateral do lado esquerdo, os atributos mais fundamentais de cada um.

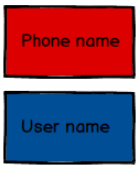


Figura 3.17: *Mockup* do blocos *Phone* e *User*

Conferências As conferência apresentam-se como o elemento, no grupo das funções VoIP, como o de menor complexidade. Todos os seus atributos serão visíveis no menu lateral. No que toca à sua estrutura, figura 3.18, apenas é necessário a perceção de que se trata de uma conferência daí a ausência de complexidade na sua estrutura.

Figura 3.18: *Mockup* do bloco *Call Conference*

Grupos de chamada Os grupos de chamada 3.19 foram pensados de modo a que, consoante cada ligação feita a este bloco, o elemento ao qual se liga será inserido neste mesmo bloco, figura. No entanto, quando for exportado, apresentará desde logo os elementos na sua estrutura e as ligações correspondentes.

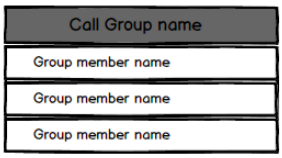


Figura 3.19: *Mockup* do bloco *Call Group*

Sequências de atendimento A estrutura das sequências de atendimento, figura 3.20 é definida de forma a que a ordem dos elementos seja visível e mantida consoante as alterações, como indica a figura. Ou seja, cada membro adicionado terá a sua ordem definida tendo em consideração a posição no bloco e, terá associado a si um *output* específico. Com isto, o bloco no seu todo é visto como um *input* e pode ter vários *outputs*.

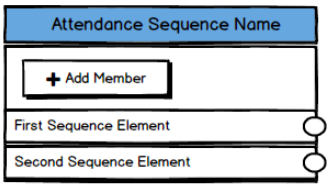


Figura 3.20: *Mockup* do bloco *Attendance Sequence*

IVRs Os IVR's seguem a mesma estrutura que as sequências, no entanto, neste caso a ligação de um IVR ao próximo nó depende dos atalhos adicionados ao mesmo. Desta forma, como ilustra a figura 3.21, será adicionado uma nova linha com o valor da tecla e output para o próximo nó. A ordem dos atalhos não é relevante, apenas é necessário atribuir-lhe um valor numérico. A opção de redirecionar para outro destino após um certo tempo é evidenciada pela opção *redirect* apresentada no bloco.

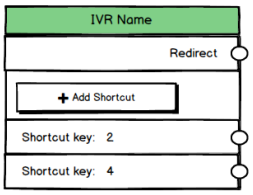


Figura 3.21: *Mockup* do bloco IVR

Figura 3.22: *Mockup* do bloco *Scheduler*

Escalonadores Os escalonadores têm associados regras de escalonamento às quais se define um destino, assim sendo, como mostra na figura 3.22, este bloco terá a possibilidade de adicionar regras à lista. As informações de cada regra serão visíveis no menu lateral direito.

Filas de Espera As filas de espera mostram-se como o bloco de maior complexidade, de modo que terá várias opções de ligação consoante o tipo de rota que irá ser criada. As opções de configuração mostradas na figura 3.14 serão transportas para o bloco, como indica a figura 3.23. Assim, este bloco pode ter associado a si qualquer um dos outros elementos que fazem parte do módulo gráfico.

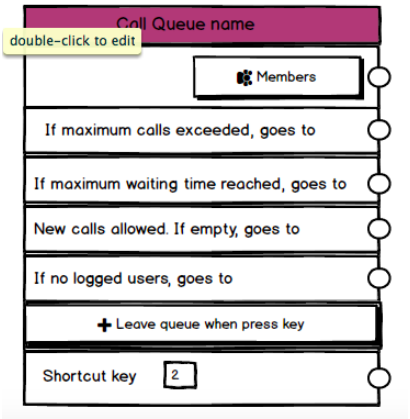


Figura 3.23: Mockup do bloco Call Queue

3.4 Tecnologia associada

Para realizar a implementação da solução proposta, a escolha da biblioteca *JavaScript* assenta em dois princípios essenciais para o sucesso da implementação. Primeiramente deverá ser possível, através das potencialidades dadas pela mesma, criar representações gráficas de objetos através de uma estrutura de código pré-definida, sendo que, esses objetos devem ter propriedades que lhes possam ser atribuídas. Desta forma, é possível criar uma visualização gráfica das configurações de rotas já implementadas numa IPBrick, com todas as rotas de entrada e internas ao sistema importadas sob forma de fluxo, tendo os números de entrada explícitos, os pontos de saída de cada uma das rotas e ainda a possibilidade de aceder às propriedades de cada um dos blocos, fazendo alterações que sejam necessárias.

Em segundo, este mesmo módulo deverá ter a capacidade de servir como ferramenta de configuração de rotas. Isto é, deverá ser uma interface que permita ao administrador de um sistema IPBrick programar visualmente o fluxo de chamadas, manipulando cada elemento graficamente e fazer as conexões necessárias para criar a configuração desejada. No entanto, será mantida a possibilidade de fazer configurações utilizando as atuais interfaces de configuração, através das páginas correspondentes a cada uma das funções VoIP disponíveis ou poderá utilizar o módulo de gestão gráfico para proceder às configurações. De uma forma, ou de outra, todas as configurações deverão ser importadas numa estrutura pré-definida para que a sua visualização seja possível na interface de gestão gráfica.

Contudo, seja qual for a abordagem e a ferramenta escolhida para o desenvolvimento, é necessário ter em consideração toda a arquitetura do sistema envolvente para que seja o mais adaptável possível, sem que haja a necessidade de alteração de todas as funções de *backend* desenvolvidas, nem que seja necessário a instalação de nenhum tipo de *plugin* adicional de forma a minimizar problemas de compatibilidade que possam advir após integração com o sistema operativo IPBrick.

A arquitetura do sistema operativo IPBrick utiliza um *backend* assente em PHP, desde o acesso à base de dados, até aos *exports* para alterar os ficheiros de configuração presentes no sistema. Portanto, algumas das ferramentas atuais que permitem criar modelos gráficos através de mecanismos

de *drag and drop* exigem dependências incompatíveis, isto é, atualmente determinadas bibliotecas *JavaScript* têm a necessidade de um interpretador de código *JavaScript* no servidor.

3.5 MxGraph

MxGraph foi a biblioteca utilizada para implementação do módulo, apesar de ser uma biblioteca com uma elevada curva de aprendizagem devido às suas potencialidades inerentes, apresenta-se como uma biblioteca de arquitetura ágil, sendo apenas necessário incluir um *script* que contém toda as funções necessárias da biblioteca, que serão carregadas numa página Web, com o intuito de utilizar e modificar a arquitetura da HTML definida da página. Para criação dos diagramas, usa SVG (*Scalable Vector Graphics*), tal como a biblioteca JSPlumb. Ainda que esta tecnologia esteja dependente do suporte dos *browsers*, atualmente os *browsers* mais recentes já a suportam e para além disso, esta biblioteca tem a capacidade de abstrair essas dependências, colocando os recursos necessários numa classe comum.

É, a par com JSPlumb, uma das hipóteses mais legítimas, consistentes e fiáveis para implementação da interface gráfica de gestão segundo os seus requerimentos. Apesar do JSPlumb não apresentar problemas a nível de licenciamento, dado que também dispõe de uma versão *open-source*, para além de se apresentar como um projeto com uma documentação extensa, com exemplos de aplicação, perfeitamente compatível com o sistema considerado para este projeto, MxGraph supera em termos de funcionalidades e exemplos concretos e aplicabilidade o que permitiu desde logo, antecipar o que era possível explorar com esta biblioteca. Os vários pontos abordados que permitiram avaliar e escolher esta biblioteca foram:

- Compatibilidade: MxGraph oferece uma API consistente e completa que permite criar uma aplicação em qualquer ambiente sem quaisquer dependências com sistema ou versões de *browsers*.

- Documentação: Esta biblioteca apresenta uma documentação detalhada de cada classe disponível, assim como vários exemplos de aplicação que, apesar de serem simples, permitem ter uma ideia básica de como se devem aplicar certas funcionalidades. A documentação detalhada e a lista de exemplos permite à priori descobrir quais as potencialidades da biblioteca e a sua aplicação prática.

- Licença: MxGraph está sob licença da *Apache Software Foundation* mas tem, de forma explícita indicações de permissão para uso comercial, modificação, distribuição, uso de patente e uso privado.

- Funcionalidades: MxGraph apresenta mais funcionalidades, uma vez que, é o projeto base que deu origem ao *draw.io*, uma ferramenta com inúmeras funcionalidades e altamente utilizada, para além de ter implementações em mais do que uma linguagem, que pode ser uma mais valia ou não, dependendo dos obstáculos que irão aparecer ao longo do desenvolvimento. Entre os vários recursos disponibilizados, destacam-se a possibilidade de arrastar,

clonar, redimensionar, remodelar, conectar e desconectar e editar os diferentes componentes que constituem os grafo. De salientar o facto de, ser possível a aplicação de *layouts* automáticos à representação gráfica criada.

- Fiabilidade: MxGraph apresenta um código consistente com atualizações constantes de *bugs* ou ajustes de funcionalidades. É um projeto sólido e consistente, que já foi uma solução paga e como tal, mostra-se como uma solução fiável, pela sua já enorme contribuição e garantias dadas no que diz respeito à criação de representações gráficas.

3.5.1 Arquitetura

A base para entender a arquitetura da biblioteca MxGraph é perceber que o foco desta API é a representação visual de grafos, assim sendo, dispõe de vários recursos que permitem a visualização gráfica, podendo os vértices ser definidos com diferentes *designs* [47]. A esta visualização pode ser aplicada funções que permitem a criação *layouts* automáticos, onde é aplicado um algoritmo específico às *cells* de modo a que a disposição das mesmas vá de encontro ao tipo de *layout*. Estes algoritmos evitam o *overlap* de *cells*, permitem que o utilizador faça alterações sobre o *layout* e que estas sejam aplicadas.

Para criação de uma página web que faça uso das funções desta biblioteca será necessário incluir a pasta onde se encontram os recursos de necessários, como imagens e o ficheiro que contém todo o código fonte - *mxClient.js*. De salientar que o facto de ser apenas um ficheiro reduz o *overhead* criado aquando dos *requests* feitos para gerar a página [47]. mxGraph irá manusear os componentes HTML criados na página, isto é, os eventos e alterações nesses componentes serão gerados pelas funções inerentes à biblioteca.

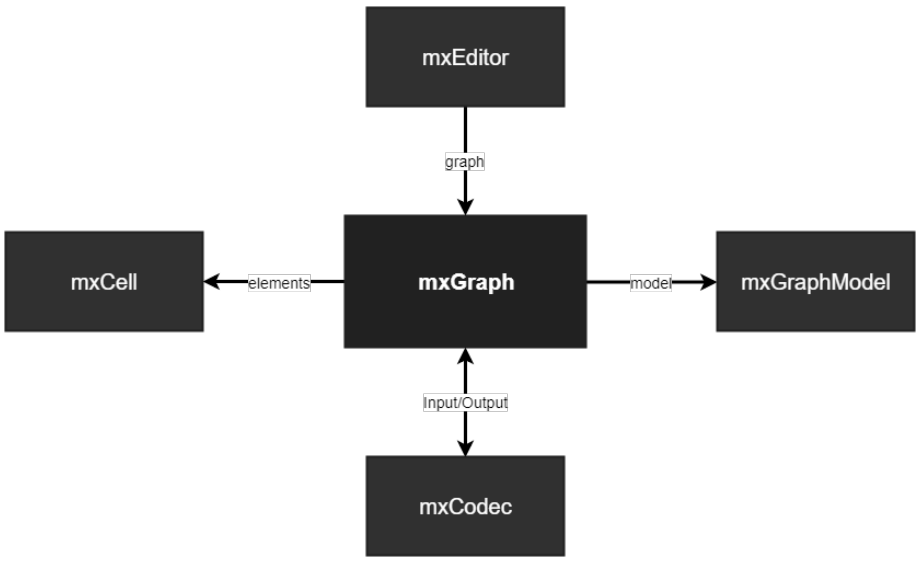


Figura 3.24: Visão geral da arquitetura da biblioteca mxGraph

A API *JavaScript* está dividida em várias classes. A classe de nível superior é a classe *mxClient* que inclui tudo o resto. A classe *mxEditor* é a classe principal para implementação de um grafo,

concebendo-lhe as capacidades necessárias para edição e tratamento de eventos. A visualização e modelo do grafo, dependem de uma classe primária *mxGraph*, a qual tem interdependências com a classe *mxGraphModel*, responsável pelo estado e eventos que ocorrem nos elementos do grafo. A classe mxCell define todos os elementos que fazem parte de um grafo, sejam ligações ou nós. A renderização dos elementos é feita pela classe *mxCellRenderer* e a sua aparência definida pela classe *mxStylesheet*. A classe diretamente ligada à importação ou exportação do grafo é a classe *mxCodec*.

Teoria dos Grafos Um grafo é definido como *G = (V, E)* e consiste num conjunto de vértices (nós) e arestas (ligações entre nós). Um caminho num grafo é a sequência de vértices entre um nó de inicio e um nó de fim. Os grafos são utilizados para definir trajetos, mapear caminhos ou representação de redes. Existem vários algoritmos associados, que permitem alcançar caminhos mais curtos ou mais longos entre dois nós, podendo as ligações ter algum custo ou não. Podem ser representados segundo uma matriz de adjacências, em que é apresentado, caso esteja definido o peso de cada ligação.

Analogamente, nesta biblioteca, o termo *cell* é usado para definir os elementos do grafo, sejam eles arestas, definidos pelo conceito de *edge* ou vértices, definidos pela conceito de *vertex*.

3.5.1.1 mxGraph Model

A classe *mxGraph model* é a classe principal que descreve a estrutura do grafo. As alterações feitas à estrutura, como mudança de visibilidade ou de estilo, ocorrem por intermédio desta classe [47].

No entanto, apesar desta classe ser responsável pelas transações e mudanças que ocorrem no modelo do grafo, a classe *mxGraph* é usada como primária (figura 3.24), ou seja, é através desta que as alterações são efetuadas. Isto porque, o conceito de adicionar um nó ao grafo é mais natural e intuitivo do que o conceito de adicionar um nó ao modelo do grafo [47]. *mxGraph* usa um sistema transacional para fazer alterações no modelo, sendo os métodos principais:

- *mxGraphModel.beginUpdate()* - dá inicio a uma nova transação.
- *mxGraphModel.endUpdate()* - dá por terminada uma transação.

Transaction Model Os métodos *mxGraphModel.beginUpdate()* e *mxGraphModel.endUpdate()* são usados em conjunto de forma a que as alterações feitas no modelo sejam encapsuladas entre estes dois métodos. Deste modo, cada secção de código que tenha como objetivo fazer algum tipo de alteração ao modelo deve estar cercada pela combinação de *beginUpdate()/endUpdate()* [47].

As funções que permitem executar um *layout* automático são um exemplo da aplicação, uma vez que o algoritmo de posicionamento automático será executado entre as chamadas de atualização de inicio e fim.

É possível adicionar vértices e ligações com os método *add()* da classe *mxGraphModel*, no entanto, para uso geral, os métodos primários são *mxGraph.insertVertex()* e *mxGraph.insertEdge()*,

da classe *mxGraph*. A grande diferença reside no facto de que, o nó tem de já estar criado quando se usa os métodos *add()*, enquanto que, os métodos de inserir criam a *cell* e inserem-na no grafo.

3.5.1.2 *mxCell*

mxCell é a classe que define os componentes que fazem parte do grafo, ou seja, ligações e nós. Os componentes são inseridos no grafo através dos métodos *mxGraph.insertVertex* e *mxGraph.insertEdge*. Para inserir um vértice os parâmetros são o modelo, o valor, a posição relativa em coordenadas x e y e, opcionalmente, o seu *design*. Para inserir uma ligação é necessário atribuir a origem e o terminal da ligação. Os parâmetros necessários para inserir um vértice ao grafo são:

- *parent*: elemento pai onde o vértice será inserido. No caso de ser inserido no grafo, é usado o método *graph.getDefaultParent()*, no entanto, se o elemento fizer parte de um vértice já criado, o elemento pai será esse mesmo vértice.
- ID: Identificador do vértice. Cada elemento definido no grafo tem um ID, que pode ser definido de forma automática ou definido pelo utilizar, deverá, no entanto, ser único e numérico.
- *value*: Valor associado ao elemento criado. Cada *cell* de um grafo tem um valor associado a si, no caso de ser criado pelo utilizador um objeto específico. No entanto, esse objeto pode ter várias propriedades associadas que podem definidas como valores do elemento.
- x, y: Valores da localização no grafo face ao canto superior esquerdo do divisão HTML da página onde será inserido o grafo.
- *width, height*: Valores que definem as dimensões do vértice.
- *style*: Variável que define o *design*, em específico, que o vértice poderá ter. Porém, caso não seja passada nenhuma variável como parâmetro, o vértice irá assumir o design *default*.

Para criar uma ligação, é necessário, de forma similar, o parâmetro *parent*, o ID da ligação, e os IDs dos vértices de origem e destino da ligação.

3.5.1.3 mxGraph

Labels Os elementos do grafo (*cells*), podem ter rótulos (*labels*) associados, que são definidos pelas funções *getLabel*, a qual usa *convertValueToString* se a variável booleana *islabelsVisible* estiver definida a *true*. Desta forma, a *label* é renderizada para HTML e o método *isHtmlLabel* retorna *true*. As *labels* são fixas por definição. no entanto, podem ser movidas a partir dos métodos *edgeLabelsMovable* e *vertexLabelsMovable* [48].

Edição Caso os elementos do grafo sejam editáveis, isto é, *isCellEditable* definida a *true*, estes podem ser editados através de um clique-duplo ou premindo F2. O métodos chamado é *startEditingAtCell*, que invoca *mxCellEditor.startEditing*.

Após o valor ser mudado, o método *labelchanged* é chamado, o qual invoca *mxGraphModel.setValue*, o que por sua vez chama *mxGraphModel.valueForCellChanged* [48].

Tooltips *Tooltips* são implementadas pelo método *getTooltip*, o qual chama *getTooltipForCell* se o elemento do grafo estiver sobre o ponteiro do rato. A implementação *default* verifica que a *cell* tem uma função *getTooltip* e chama-a no caso de existir [48].

Multiplicidades e Validações Para controlar as conexões feitas no grafo, o método *getEdgeValidationError* é usado. A implementação *default* da função usa multiplicidades, *mxMultiplicity*, para assim estabelecer multiplicidades entre nós [48].

3.5.1.4 mxEditor

Esta classe é usada para implementar um *wrapper* para o grafo, definido pela classe *mxGraph*, de forma a adicionar ações, *input* e *output*, *auto-layout*, histórico de comandos e implementação de menus *pop-up* [49].

Ações O método *addAction()* providenciado pela classe *mxEditor*, recebe-a como seu primeiro parâmetro e, como segundo argumento o elemento do tipo *mxCell* onde serão executadas as ações.

Propriedades Esta classe permite que sejam associadas propriedades a cada elemento do grafo e que serão visíveis no XML definido para cada objeto presente no grafo, como é definido no seguinte exemplo:

```
1  <Task label="Task" description="">
2    <mxCell vertex="true">
3      <mxGeometry as="geometry" width="72" height="32"/>
4    </mxCell>
5  </Task>
```

3.5.1.5 Input/Output

A classe *mxGraph* apresenta funções capazes de codificar e descodificar o modelo definido por um grafo. Desta forma, é possível criar uma estrutura XML capaz de ser descodificada e representada graficamente de acordo com o código definido nesse mesmo grafo, ou seja, ao criar um grafo em que os seus componentes obedecem a uma estrutura pré-definida, o XML gerado pelo grafo poderá também ser lido e o resultado visual será o mesmo.

Desta forma, as classes necessárias são *mxUtils*, que tem métodos capazes de fazer o *parse* do código XML e *mxCodec*, que usa *codecs* registados na classe *mxCodecRegistry* para fazer o

encode e *decode* de cada objeto *JavaScript*. A representação XML do diagrama, é feita usando o seguinte código:

```
1    var enc = new mxCodec(mxUtils.createXmlDocument());
2    var node = enc.encode(graph.getModel());
3    var xml = mxUtils.getPrettyXml(node);
```

3.5.1.6 Estrutura XML

O XML definido por esta biblioteca é uma estrutura organizada, com os elementos que compõe o grafo, ou seja, objetos do tipo mxCell. No caso de serem definidos objetos diferentes, com determinados atributos, vão ser reproduzidos no XML como indicado no exemplo 3.5.1.6.

```
1  <mxGraphModel>
2    <root>
3      <mxCell id="0"/>
4      <mxCell id="1" parent="0"/>
5      <Person Name="John" age="18" id="2">
6        <mxCell vertex="1" parent="1">
7          <mxGeometry x="40" y="40" width="80" height="30" as="geometry"/>
8        </mxCell>
9      </Person>
10     <mxCell id="3" value="" vertex="1" parent="1">
11       <mxGeometry x="620" y="160" width="120" height="120" as="geometry">
12     </mxCell>
13   </root>
14 </mxGraphModel>
```

Capítulo 4

Implementação

Neste capítulo, é descrito todo o processo envolvido na implementação do projeto, estando dividido em duas partes, importação e exportação de configurações.

4.1 Importação das configurações

A importação das configurações é o requisito mais importante da implementação deste módulo, para que, quando integrado numa IPBrick, seja feita a representação gráfica das configurações já existentes em sistema.

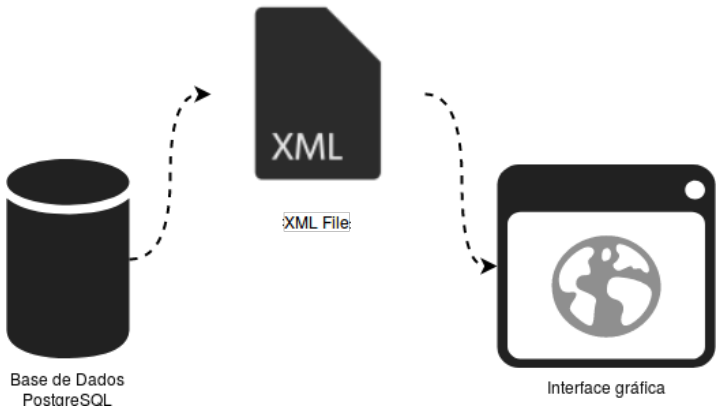


Figura 4.1: Importação - Fluxo de dados

Uma vez que a biblioteca MxGraph define o modelo do grafo sobre forma da linguagem XML, permitindo que seja feita uma exportação do mesmo, assim como uma leitura e representação gráfica a partir da estrutura descrita no XML, esta linguagem foi definida como a camada de abstração entre a interface e a base de dados. Deste modo, a estrutura a seguir, apresentada na figura 4.1, passou por obter as informações individuais de cada função, telefone e utilizador VoIP, para, posteriormente, ser feita a criação de um ficheiro XML que seja possível ser lido e interpretado pela interface, por forma a gerar a representação gráfica de acordo com a sua estrutura.

4.1.1 Interface Gráfica

Na implementação da interface gráfica foram usadas algumas classes da API, procurando usar as necessárias para satisfazer os requisitos iniciais definidos para a mesma. O diagrama de classes apresentado na figura 4.3, representa as classes que foram usadas e as associações resultantes. Relativamente às classes representadas, não estão referidos todos os atributos e métodos correspondentes a cada uma, mas sim os que foram instanciados na implementação.

Primeiramente, foi definido um grafo através da classe *mxEditor* uma vez que este grafo terá de lidar com ações de *input/output*, *auto-layout*, reverter ações e criação de um *pop-up menu*. O grafo foi criado dentro de um *container*, que é definido por uma divisão da página criada em HTML, com a *tag* <*div*>, como é ilustrado no presente excerto de código.

```
1    var editor = new mxEditor();
2    var graph = editor.graph;
3    var model = graph.model;
4    editor.setGraphContainer(container);
```

Após definidas estas variáveis, foi possível começar as implementações adicionais necessárias, uma vez que, qualquer alteração feita no grafo ou na sua estrutura poderia ser feita acedendo a classe primária, *graph*, ou através do seu modelo, *graph.getDefaultModel*. Esta classe, *graph*, herda todos atributos e métodos da classe *mxGraph*, no entanto, foram criados métodos adicionais para criação dos ícones arrastáveis do menu lateral esquerdo, como estruturado em 3.15, para criação de uma *toolbar* com ações necessárias, criação de um *pop-menu*, configuração da aparência das ligações e de cada vértice em específico e ainda criação de uma função que gera de forma dinâmica o menu das propriedades de cada elemento quando o mesmo é selecionado. De destacar que, de entre os recursos disponibilizados pela classe *mxGraph*, foram usados alguns relevantes para a interface gráfica, entre os quais, a capacidade de introduzir rótulos aos elementos através de HTML, a capacidade os blocos serem colapsados e ainda, o facto não permitir que existam ligações penduradas no grafo.

O uso da classe *mxEditor*, também é necessária para lidar com o facto dos blocos com maior complexidade assumirem um *design* diferente, que pode ser definido com um método desta classe. A aparência definida foi pensada em forma de tabela de forma a ir de encontra com os *mockups* definidos. Para tal, foi necessário criar um *layout* automático para esses blocos, juntando dois conceitos de *layout* abordados por esta biblioteca, *SwimlaneLayout* e *StackLayout*. O *layout* em *swimlane*, permite agrupar e organizar elementos que fazem parte do mesmo grupo. Ao organizar estes elementos através do *StackLayout*, esta organização é feita em pilha (*stack*), ou seja, os elementos ficam alinhados e com as dimensões definidas pela *swimlane*. Esta abordagem é ilustrada no presente excerto de código.

```
1    // Configures the automatic layout for the table columns
2    editor.layoutSwimlanes = true;
3    editor.createSwimlaneLayout = function ()
4    {
5            var layout = new mxStackLayout(this.graph, false);
6            layout.fill = true;
```

```
7             return layout;
8     };
```

De forma a manter a coerência do código, foi definido o objeto *Table*, com os atributos necessários a cada um associados ao seu *value*. Esta abordagem é importante para depois ser possível alterar os seus atributos. Assim, as tabelas apresentam um *SwimlaneLayout* onde são definidos os vértices pai e os vértices que são incluídos em *StackLayout*, definidos como vértices filho. Nos casos dos elementos de menor complexidade, como é caso dos telefones, utilizadores e conferências, foram vistos como objetos do tipo *Block*, os quais herdam os atributos e métodos da classe *mxCell*. A classe *mxCodec* é parte fundamental na estrutura da interface, uma vez que, permite codificar e descodificar o XML correspondente ao grafo gerado. Para o fazer, necessita de invocar a classe *mxUtils*, que irá fazer um *parse* do XML, antes de ser feito o seu *decode*.

A classe *mxMultiplicity* foi instanciada para descrever as conexões permitidas entre os blocos do grafo, para que as associações permitidas atualmente na interface de administração gráfico, se mantenham no módulo gráfico. Para que seja possível gerar eventos no momento em que certas conexões são realizadas, a classe *mxConnectionHandler* permite controlar esses eventos. A classe *mxGraphHandler* permitiu que fossem ativadas linhas que ajudam a orientar a colocação dos blocos no *container* do grafo.

De forma a reproduzir na totalidade os *mockups* definidos em 3.3, foi usado o modelo transacional definido pela biblioteca, ou seja, sempre que havia alterações no modelo, estas eram encapsuladas pelos métodos *mxGraphModel.beginUpdate()* e *mxGraphModel.endUpdate()*. Estas alterações surgiam pela dinâmica introduzida em cada bloco de maior complexidade, nos quais era necessário adicionar um vértice filho por intermédio de um determinado *trigger*, sendo este um evento de clicar no botão ou a existência de uma nova ligação a esse mesmo bloco.

4.1.2 Fluxo de dados

A recolha de dados de cada função é feita a partir da base de dados, uma vez que, as configurações feitas ou modificadas na aplicação *web* são logo inseridas e atualizadas, respetivamente, nas tabelas correspondentes.

A figura 4.3, mostra os ficheiros envolvidos na comunicação entre a aplicação *web* e a base de dados. A estrutura pré-definida passa por usar funções descritas nas bibliotecas, que por sua vez, usam as *queries* definidas em sistema. Cada elemento integrante da estrutura VoIP, seja do tipo função, telefone ou utilizador, tem associado a si um ficheiro com as várias *queries* que lhe são associadas, ou seja, permitem obter os mesmos dados com informações diferentes acerca do elemento, ou seja, com diferentes parâmetros de entrada na *query*. No caso das funções VoIP, existe uma biblioteca capaz de recolher os dados de cada uma, invocando para tal as *queries* definidas nesses ficheiros correspondentes. Este fluxo de dados permite que seja criada uma camada de abstração entre a aplicação *web* e a base de dados, para além de simplificar o processo de obter informação acerca de algum elemento.

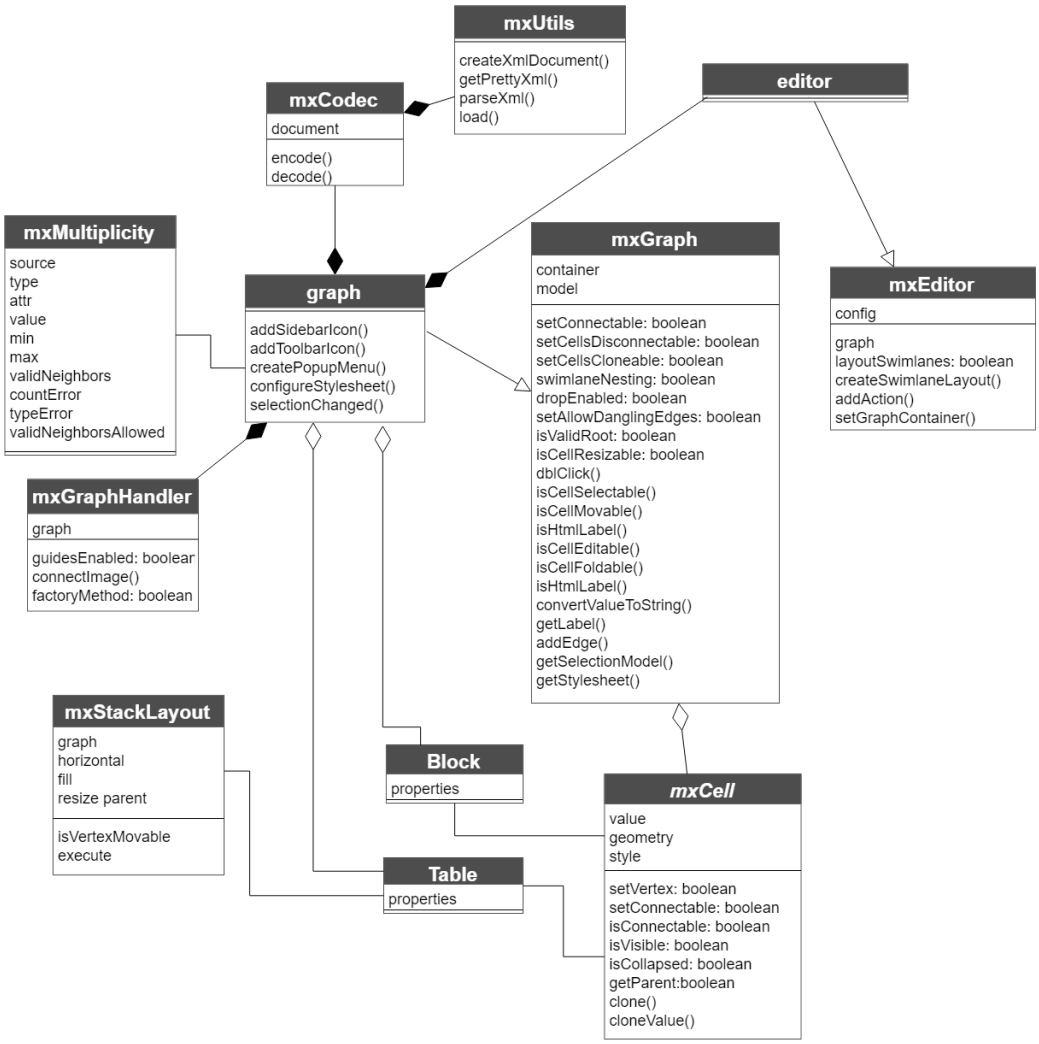


Figura 4.2: Interface - Diagrama de classes

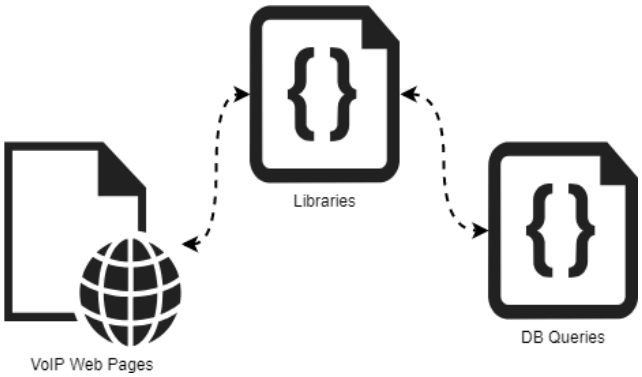


Figura 4.3: Fluxo de dados - Importação das configurações

Atualmente, a página da interface *web* do dialplan da IPBrick apenas mostra as funções com algum tipo de acesso direto, no entanto, no módulo gráfico o *export* não irá seguir o mesmo princípio, pois todas as funções definidas em sistema vão ser exportadas e representadas graficamente.

Portanto, o processo não passou por aproveitar a estrutura de código já existente na página do dialplan da interface de administração mas sim por usar as funções usadas em cada página de visualização de cada função. Como tal, o fluxo de dados referido na figura 4.3 não foi possível de reproduzir em pleno, uma vez que nas bibliotecas definidas não existiam funções que permitissem fazer a recolha dos dados na totalidade, isto é, com todas as informações relevantes para uma construção gráfica do dialplan. Deste modo, dado que não fazia parte do âmbito alterar ou incluir funções no *backend* do sistema mas sim, tentar compatibilizar ao máximo o módulo gráfico, sem que fosse necessário incluir funções ou alterar as existentes, foram usadas diretamente as *queries* definidas nos ficheiros, para os casos em que havia essa inexistência de funções nas bibliotecas.

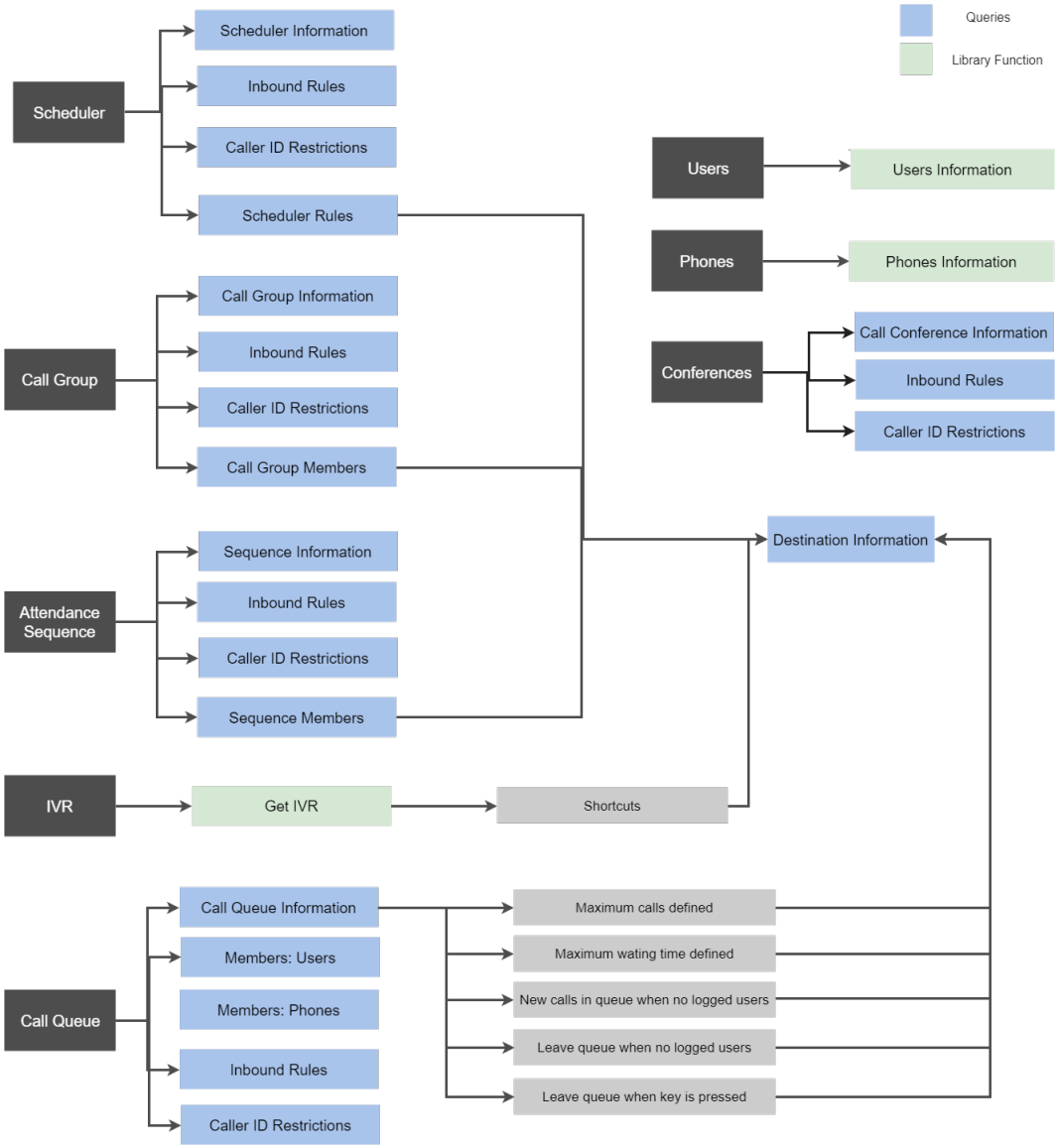


Figura 4.4: Informações obtidas para cada elemento do dialplan

A figura 4.4, mostra o dados que foram necessários obter para cada elemento, assim como a distinção de onde foram obtidas considerando o fluxo de dados ilustrado na figura 4.3. As infor-

mações relativas aos telefones e utilizadores são facilmente obtidas a partir das funções definidas nas bibliotecas correspondentes. No entanto, para todas as funções VoIP excetuando o caso do IVR, existiu sempre a necessidade de realizar outras *queries* que permitissem obter as suas regras de entrada (*inbound rules*) e as suas respetivas restrições de *caller* ID, para além, da informação referente à função. A figura 4.5, ilustra esta dependência entre a função, ou seja, é através do ID da função que é possível obter as outras informações acerca da mesma.

No caso das sequências e grupos de chamada, foi necessário obter através de uma *query* específica, os seus membros, que em termos gráficos são declarados e representados como o seu destino. Esta informação acerca dos membros está dividida segundo o tipo de membro, portanto, foi necessário consoante o tipo procurar as respetivas informações do destino. No caso dos escalonadores, o princípio foi o mesmo, após obter as regras associadas, também foi necessário, consoante o tipo, obter as informações do destino de cada regra. Nas filas de espera, foram determinados os seus membros (telefones e utilizadores), assim como as respetivas configurações efetuadas, ou seja, para cada uma, era necessário obter o destino submetido. No caso de IVR, foi usada uma função da biblioteca, que permitiu desde logo obter as propriedades todas do elemento, apenas foi necessário obter as informações de cada destino de cada *shortcut* criado.

Os destinos não se caracterizavam por ser os mesmos para cada função, ou seja, as sequências e grupos de chamada não podiam ter como seus membros qualquer outro tipo de elemento, no entanto, as outras funções permitiam ter associadas a si vários tipos de destino.

A um grupo de chamada é possível associar, como tipo de destino, utilizadores, telefones e endereços SIP. Nas sequências, é possível atribuir como seus membros, elementos do tipo telefone ou um endereço SIP. Já os IVRs permitem configurar atalhos que podem encaminhar para qualquer tipo de função, telefone ou endereço SIP. Às regras definidas no escalonador, podem ser-lhes atribuídas um destino de qualquer tipo. As filas de espera são um elemento com maior complexidade, dado que, podem ter associações com qualquer tipo de função, com utilizadores e telefones, dependendo do tipo de configuração da mesma. Os telefones, para serem associados têm de estar registados no servidor local e que os endereços SIP podem ser externos ou mesmo endereços internos que apontem para alguma função, utilizador ou telefone.

Os endereços SIP, foram substituídos, em termos de interface gráfica, pelo conceito geral de acesso direto, ou seja, quando um destino de uma das funções é para um determinado endereço SIP, este é visto como um bloco do tipo Direct Access. Na exportação, quando um endereço SIP corresponde a algum endereço SIP interno, a ligação é feita para o acesso direto do elemento em questão.

A tabela 4.2, mostra as possíveis ligações entre elementos que poderão existir e que serão visíveis na interface. Estas possíveis ligações são referenciadas ao tipo de função no seu todo, logo, se assumirmos o número total de possíveis destinos como N e o número de possíveis elementos, atalhos, regras ou configurações de cada função, iremos ter um resultado de possibilidades evidenciado na tabela 4.1:

Dada a variedade de associações possíveis entre elementos do dialplan, os quais apresentam uma heterogeneidade face às singularidades que apresentam na sua estrutura, devido às configura-

Tabela 4.1: Possibilidades de ligação

Outputs from each function	Nº of destinations	Destinations Possibilides
Call Group members	N	3
Sequence members	N	2
IVR shortcuts	N	9
Scheduler rules	N	9
CQ - Maximum calls defined	1	9
CQ - Maximum waiting time defined	1	9
CQ - New calls when no logged users	1	9
CQ - Leave when no logged users	1	9
CQ - Leave when key is pressed	N	9

ções que os caracterizam, o dialplan assume uma complexidade indubitável. Tendo em consideração este facto, foi necessário desenvolver uma metodologia que permitisse descomplicar todo este cenário característico do dialplan.

4.1.3 Mapeamento dos dados

O grande obstáculo da exportação era encontrar um método sistemático, que permitisse recolher a informação acerca de cada bloco, criar a sua estrutura XML e saber as ligações associadas a cada um. O processo de configuração dos diferentes fluxos, devido às funções que as podem compor e as propriedades das mesmas, assume elevada complexidade, uma vez que, cada função pode fazer parte de vários fluxos de chamada e pode ter associada a si vários elementos. Tendo em conta cada elemento e as suas respetivas propriedades, que irão ser incluídos no módulos de gestão gráfico do dialplan da IPBrick, é inegável que a criação de rotas pode traduzir-se num conjunto de inúmeras associações entre várias funções VoIP.

Ao recolher as informações de cada bloco, apenas era possível saber desde logo os *outputs* do bloco, isto é, os elementos aos quais o bloco se vai conectar e, caso a função tivesse algum tipo de acesso direto, seria o único input do bloco que era possível de determinar à priori. Ou seja, era

Tabela 4.2: Interações entre elementos dialplan IPBrick

	Atten. Sequence	Call Conf.	Call Group	Call Queue	IVR	Scheduler	Phone	User	SIP Address
Atten. Sequence							x		x
Call Conf.									
Call Group							x	x	x
Call Queue	x	x	x	x	x	x	x	x	x
IVR	x	x	x	x	x	x	x	x	x
Scheduler	x	x	x	x	x	x	x	x	x

impossível saber logo quais os elementos que vão ligar a um determinado bloco, isto é, os seus *inputs*.

Dada esta premissa, ao criar um bloco em XML que tivesse uma ligação para outro bloco, essa ligação só podia ser criada quando os dois blocos estivessem gerados, senão era inconcebível definir a origem e destino da ligação como referido em 3.5.1. Portanto, primeiramente teria de ser criada a estrutura XML de todos os blocos e só depois é que se poderia gerar as conexões. Deste modo, foi criado um *array* onde as conexões existentes eram guardadas, sendo a chave do *array* o ID da origem da ligação e o valor da chave, o ID de destino da ligação. Mas apesar de se ter esta informação agrupada, faltava definir quais os valores que iriam compor este *array*.

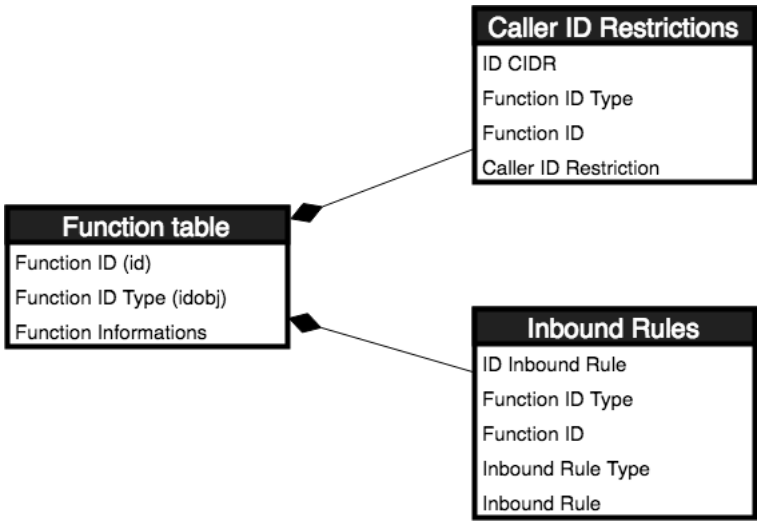


Figura 4.5: Informação relativa a cada função do dialplan

Os valores teriam de ser os identificadores da função, não os definidos na base de dados mas os identificadores que lhe iam ser atribuídos no XML, no entanto, uma função que tivesse uma ligação para outra que ainda não tinha sido criada, teria logo atribuído um valor que depois era introduzido na estrutura da função de destino. Esse valor não poderia ser de natureza aleatória senão seria inconcebível este método, portanto, teria de lhe ser atribuída alguma lógica.

Deste modo, a estrutura de dados definida no sistema IPBrick, foi conveniente para este processo. Como ilustra a figura 4.5, a tabela de cada função na base de dados tem associada a si um identificador do tipo e um identificador próprio. Como referido em 3.5.1, cada elemento do grafo tem de ter um identificador único associado (ID) que é usado para definir as ligações entre nós. Os identificadores teriam de ter alguma coerência e ao mesmo tempo, teriam de ser numéricos e, obrigatoriamente, únicos para evitar colisões ou maus resultados na exportação. Deste modo, foi elaborado um mapeamento entre os IDs presentes na base de dados, e os IDs no XML. Este mapeamento foi a chave para todo o processo de exportação das configurações para a interface gráfica.

Na base de dados, cada função tem associada a si um ID do tipo de função e um ID da função, assim sendo, o ID em XML de cada função será uma concatenação entre esses IDs como mostra a tabela 4.3. Esta concatenação, permitia que o ID XML fosse sempre definido com os dois

Tabela 4.3: Mapeamento IDs Funções

Function	ID Type of Function	ID XML
IVR	2	02FunctionID
Conferences	3	03FunctionID
Scheduler	4	04FunctionID
Call Groups	5	05FunctionID
Attendance Sequence	6	06FunctionID
Call Queues	11	11FunctionID

primeiros dígitos referentes ao tipo da função e os seguintes como o ID da função, dado que este último pode assumir valores superior a dois dígitos. No entanto, os blocos correspondentes às *inbound rules*, telefones e utilizadores, não têm definido um ID em sistema que identificasse o seu tipo, como tal, foi definido um ID para manter a lógica definida, ilustrado na tabela 4.4.

O preenchimento do *array* com as conexões foi composto com o resultado da concatenação, desta forma, após percorrer todos os elementos e gerar o XML para cada um, era percorrido este *array* e as conexões eram geradas. Assim, foi conseguida uma estrutura intuitiva e fácil de trabalhar, assim como, um XML onde era percetível identificar o tipo de função em questão.

No entanto, esta abordagem teria de ser ajustada e replicada para todos os elementos que compõe o XML gerado, dado que, para a estrutura dos elementos definidos em 3.3 ser concebida, é certo que irá haver elementos do tipo *mxCell* que serão elementos filho de outros elementos *mxCell*. Como tal, para que haja uma perceção em termos de XML desta dependência e de forma a manter a coerência do mapeamento, a tabela 4.5 ilustra o resultado. Qualquer vértice da tabela tem como seu ID, a concatenação entre o ID tipo que o caracteriza e no caso de assumir uma complexidade acrescida, terá os vértices filhos definidos pela concatenação do seu próprio ID com a combinação *0+ChildNumber*. O *childNumber* é definido em função da posição que ocupa na estrutura do *layout* que irá ser definido, e o 0 serve como delimitador de distinção entre ID do vértice pai e ID vértice filho. Os *outputs*, devem assumir também um valor diferente porque é o seu ID que irá ser usado para definir as ligações cuja a origem é o bloco onde estão inseridos, dessa forma, o seu ID será acrescentar 25 ao ID definido para o vértice filho onde estão inseridos. Não obstante, todas ligações respeitam a mesma abordagem, sendo identificadas pelo número 23 seguidas do respetivo ID.

Tabela 4.4: Mapeamento IDs

Bloco	ID Type	ID XML
Telefones	20	20ID
Utilizadores	21	21ID
Inbound Rules	7	07ID

Tabela 4.5: Mapeamento elementos XML

Graph Element Type	ID	Childs ID	Ouputs ID
Vertex	TypeID + ID	(TypeID + ID) + 0 + ChildNumber	25 + Childs ID
Edge	23 + ID		

4.1.4 Criação do XML

Após definida a interface e mapeamento dos dados, foi necessário perceber qual o algoritmo a desenvolver de forma a construir a estrutura XML de uma forma sistemática, e que em termos de *performance* não colocasse em causa o sistema. A abordagem passou por iterar caso a caso, dependendo do tipo de elemento, ou seja, são obtidos os elementos de um certo tipo, depois, para cada um, são guardadas e estruturadas as informações relativas ao mesmo.

Para estruturar essas informações e de forma a que o código esteja preparado para inclusões futuras, foram definidas funções para criação de cada tipo de objeto, as quais recebem as informações obtidas de cada um para introduzir no código XML. Se a função tiver acessos diretos ou restrições de *caller* ID é desde logo criado também o bloco respetivo. O retorno das funções será sempre o ID que essa função representa, de forma a que seja adicionado no *array* das conexões, a conexão entre o bloco do tipo *Direct Access* ou *Inbound Rules* caso exista, e as conexões para outros blocos, de acordo com o mapeamento feito, caso também existam. Nos casos em que é necessário verificar os destinos, é usado *switch statement*, de forma a encontrar o ID de cada um dos elementos e adicionar um vértice necessário à estrutura do elemento de origem. A figura 4.6 apresenta uma visão geral da forma como é feita a elaboração do ficheiro XML.

Apesar de ser bastante informação para ser obtida, considerando cada elemento e as associações respetivas, o comportamento do código em termos de performance não colocou em causa o sistema nem houve perturbações significativas em termos de atrasos.

4.1.5 Mecanismo

Após o mapeamento dos elementos, foi necessário importar a estrutura XML para a interface e, para tal, a abordagem foi utilizar as classes que a biblioteca MxGraph disponibiliza para criação de *layouts* automáticos, uma vez que, foi uma das causas que pesaram na escolha desta mesma biblioteca.

Esta capacidade da biblioteca foi crucial para que o *export* fosse conseguido, uma vez que, caso não fosse possível a aplicação do *layout* num cenário de elevada complexidade, a abordagem seria pensar na estrutura XML resultante da exportação das rotas como um aglomerado de vários grafos, de entre os quais haveria um caminho mais longo e esse era o ponto inicial, pois seriam os elementos que a constituam que iriam ser primeiramente criados. Ou seja, seria uma abordagem bastante hermética devido à variedade de combinações com as quais o algoritmo se iria debater.

De entre as classes disponibilizadas pela API, existiam *layouts* circulares, *tree-based layouts*, hierárquicos e orgânicos. Para que fosse possível perceber qual é que seria a escolha para aplicar à estrutura XML, foram elaborados testes, inicialmente com grafos de estrutura simples, depois

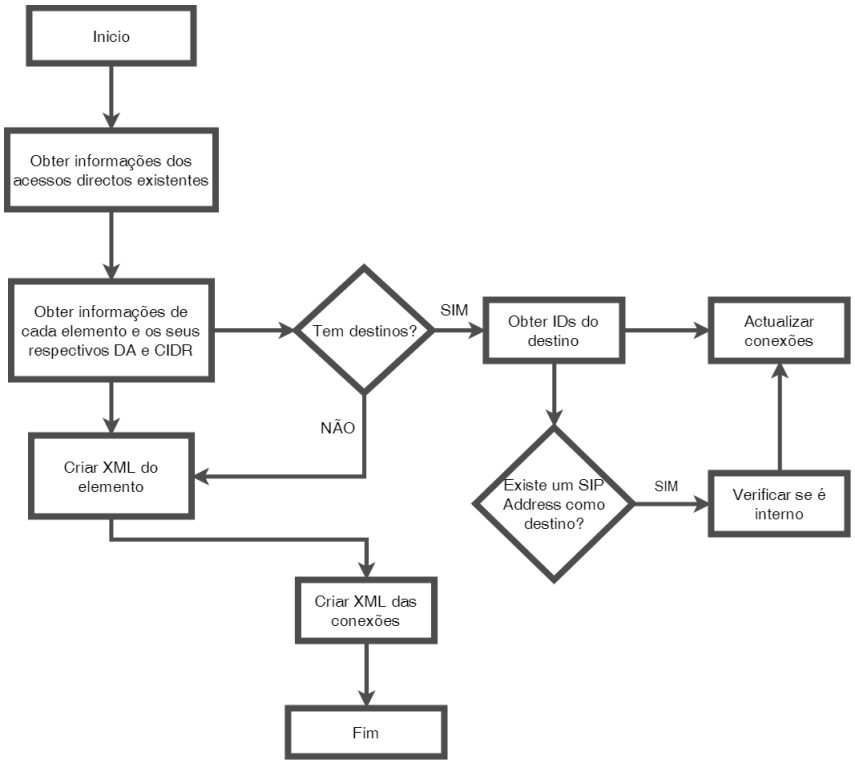


Figura 4.6: Algoritmo exportação

com a estrutura já adotada pelo XML que iria ser exportado e depois com essa mesma estrutura mas com um cenário mais complexo. As figuras 4.7 e 4.8 mostram, respetivamente, o resultado da aplicação de um *layout* automático do tipo circular e orgânico. Foi possível concluir que não são adequados, uma vez que não iriam promover a interpretação do dialplan, considerando o tipo de representação que é obtida. Para além de que, no caso de existirem nós sem qualquer tipo de ligações, são excluídos da representação gráfica quando é aplicado o *layout* orgânico.

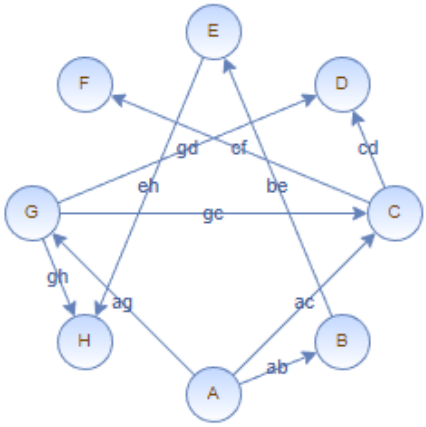


Figura 4.7: *Circle Layout*

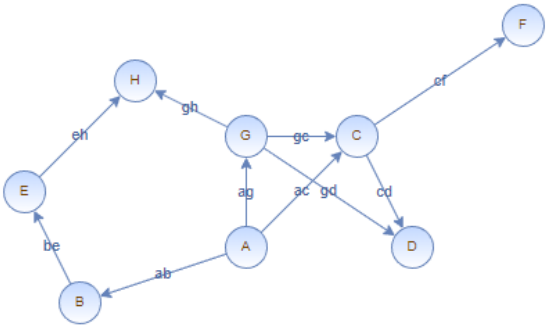


Figura 4.8: *Organic Layout*

A figura 4.9, apresenta o resultado da execução do *layout* automático do tipo hierárquico.

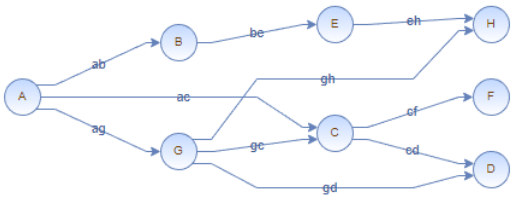


Figura 4.9: *Hierarchical Layout*

Apesar de ser aplicado a uma estrutura simples, sem nenhuma complexidade acrescida, nem em termos de número de nós nem na estrutura aplicada a cada um, este *layout*, não tem limitações associadas ao número de nós nem ao respetivo tipo. Deste modo, foram realizados testes aplicando este *layout* à estrutura XML gerada a partir das configurações armazenadas na base de dados.

4.1.6 Testes e Resultados

Os testes realizados pela exportação de um dialplan interno foram realizados com base em configurações feitas num cenário real. Estas configurações foram aplicadas numa máquina de testes criada num cenário virtual, que foi acedida através de outra máquina IPBrick, com duas interfaces de rede, que servia como *gateway*. Para importar configurações numa IPBrick é necessário que não haja problemas de incompatibilidade, ou seja, o tipo de máquina onde as configurações foram efetuadas deve ser igual à máquina onde serão aplicadas.

Desta forma, foi criado um cenário de teste, indicado na figura 4.10. Na máquina de teste, a introdução do módulo não foi feita através da substituição da página do dialplan mas sim, através da criação de um *virtual host* no Apache no mesmo domínio da máquina. Os resultados das exportações, com a aplicação do *layout* automático do tipo hierárquico, estão evidenciados nas figuras B.8, B.9 e B.10. Estas figuras ilustram a representação gráfica e as informações acerca de cada elemento disponíveis no menu lateral, sendo a primeira, uma visão geral e as seguintes imagens mais focadas que mostram com mais nitidez o aspeto dos blocos e das ligações.

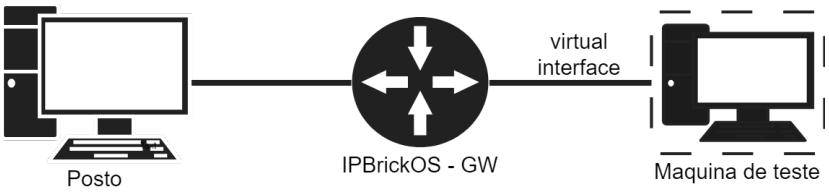


Figura 4.10: Exportação - Cenário de teste

Comparativamente à página atual do dialplan da IPBrick, a representação gráfica promove um entendimento mais claro das interações entre os elementos que o compõe. Para além de serem percetíveis os acessos definidos a cada elemento e os números que os poderão aceder. A figura B.1, mostra as mesmas configurações representadas graficamente nas tabelas que se encontram na página atual do dialplan da interface de administração do IPBrickOS. É seguro afirmar que a interface gráfica permite uma visão mais limpa e interativa das mesmas rotas. A navegação pela

interface é feita através do controlo do *zoom* no scroll do rato e premindo o botão direito do rato é possível arrastar a imagem, para que seja possível visualizar todas os caminhos definidos pela representação gráfica. As figuras B.2,B.3,B.4,B.5,B.6 e B.7, representam as listagens das funções que fazem parte deste dialplan. Nem todas aparecem atualmente na interface do dialplan, como referido anteriormente, uma vez que, só são tabeladas as que possuem algum tipo de acesso direto. No entanto, a representação gráfica apresenta-as todas, de forma completa.

4.2 Exportação das configurações

Do lado oposto, na exportação, as configurações que são feitas na interface são passadas para XML. Com o uso de um pedido AJAX, o XML é passado para ficheiro PHP - *XML handler*, que vai percorrer a estrutura, retirando os dados referentes a cada elemento. Neste caso, há dois cenários a considerar:

1. Configuração isolada: Numa configuração isolada, o tratamento de informação será diferente uma vez que não haverá o cuidado de percorrer o XML de forma a encontrar *flags* de alteração de certos blocos, isto porque a configuração que será feita é nova e será apenas validada e exportada para a base de dados.

2. Alteração de um dialplan importado: No caso de uma importação de um dialplan já configurado de um determinado sistema, é necessário um processo mais complexo. É necessário percorrer o XML, procurando por *flags* que definam novos elementos criados e/ou elementos alterados.

4.2.1 Fluxo de dados

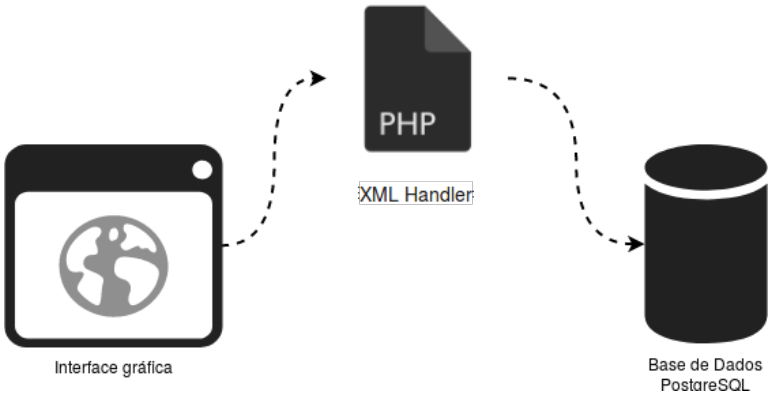


Figura 4.11: Exportação - Fluxo de dados

Como indicado na figura 4.1, o fluxo de dados é análogo ao demonstrado na figura 4.1, ou seja, existe na mesma a camada de abstração providenciada pelo XML. No entanto, enquanto que no caso da importação de dados, a leitura é feita a partir de um ficheiro XML, neste caso não é o ficheiro que é enviado mas sim, um pedido AJAX que vai enviar o XML para um ficheiro PHP. O

ficheiro PHP é responsável por retirar e organizar a informação para que esta seja introduzida na base de dados.

4.2.2 Tratamento XML

Neste caso, o fluxo de dados, apesar de manter o XML como camada intermédia, não será gerado um ficheiro XML a partir da interface, dado que, iria ser criado um *overhead* ao fluxo desnecessário. Para além de que, o mecanismo da interface possa fazer modificações no dialplan *on demand*, ou seja, o administrador poderá criar elementos, modificar e depois de gravar as alterações feitas, poderá continuar a alterar o que ainda achar necessário. Para que fosse possível criar este dinamismo, foi necessário usar a tecnologia AJAX, de modo a que seja enviado o XML para um ficheiro PHP capaz de o tratar. Na interface gráfica, o XML foi gerado através do seguinte excerto de código:

```
1 var enc = new mxCodec(mxUtils.createXmlDocument());
2 var node = enc.encode(editor.graph.getModel());
3 var text = mxUtils.getPrettyXml(node);
```

Depois, foi feito um pedido AJAX para um ficheiro PHP, *xml_handler.php*, que usa uma extensão dada por esta linguagem, o *SimpleXML Parser*, a partir do qual foi possível manipular e receber os dados no formato XML. A manipulação é feita tendo como base o conceito de *tree-based*, ou seja, existe uma hierarquia nos elementos e a cada um estão associados os seus atributos.

A figura 4.12, mostra a relação entre a estrutura XML descrita e como deve ser entendida a hierarquia para assim ser possível aceder a cada elemento da estrutura. Ou seja, a *layer* na qual era necessário trabalhar, era a correspondente a todos os objetos do tipo mxCell.

Para o processo de exportação seja coerente, é necessário que haja uma metodologia associada, para garantir a coerência dos dados que são inseridos na base de dados. Na interface atual da IPBrick só é possível apontar um destino, seja em que função for, caso já se encontre criada. No módulo gráfico, no caso de serem feitas configurações, os elementos são criados na interface gráfica, procedendo-se as ligações necessárias de seguida. No entanto, depois de aplicar as configurações, a metodologia passa por primeiro verificar os elementos criados e guardar todos os dados associados e, após este processo, o procedimento passa por procurar as ligações nas quais estes elementos se inserem, ou seja, as ligações em que os elementos em causa são a origem para assim, serem inseridos na base de dados os dados relativos ao elemento e os destinos do mesmo, caso os tenha.

O algoritmo representado na 4.13, é uma visão geral da lógica a ser seguida. Apesar da sua estrutura simples, é um algoritmo que se irá deparar com elementos de elevada complexidade, com várias propriedades que lhes estão associadas. Ou seja, em termos de arquitetura de código, será necessário criar estruturas que permitam, ao percorrer o XML, guardar todas as informações e associá-las ao bloco correspondente. Para além disso, quando o elemento fizer parte de várias ligações, essas estruturas terão de ser dinamicamente atualizadas de forma a que seja possível guardar as informações relativas na base de dados.

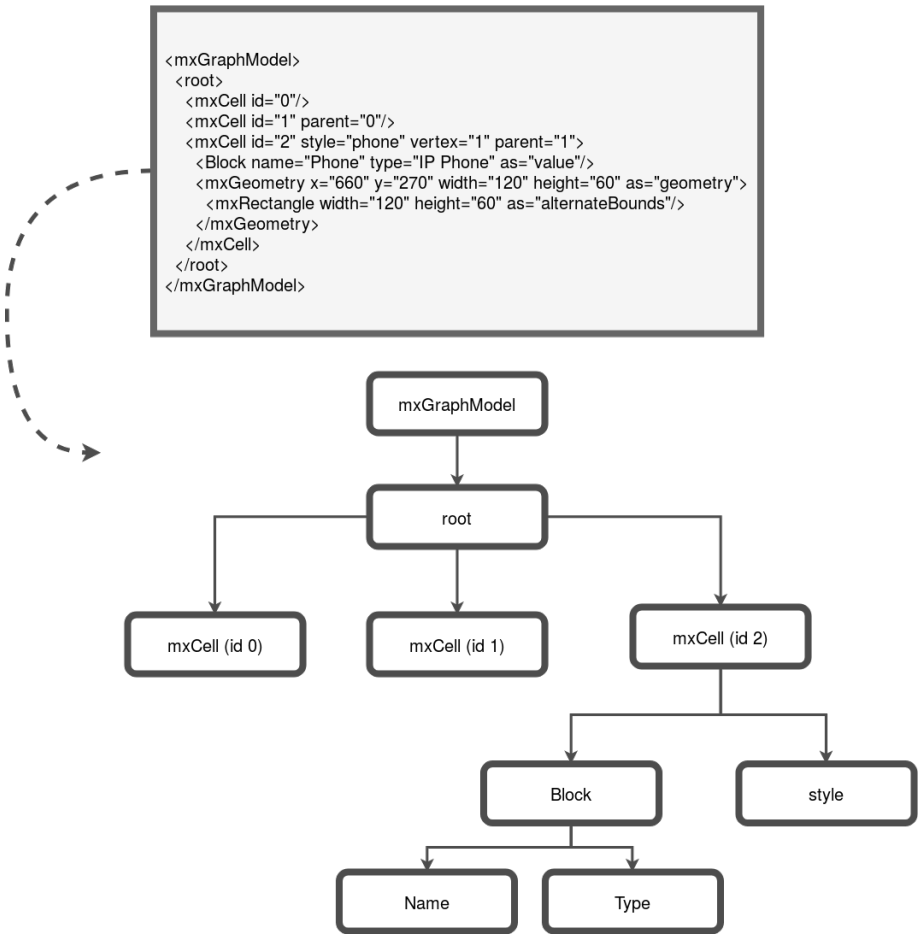


Figura 4.12: *Tree-based XML*

4.2.3 Validações

Existem validações a ser feitas que podem estar relacionadas com a configuração das rotas ou com os dados que podem ser introduzidos. Estes dados são introduzidos pelo administrador, no entanto, não deixa de ser um utilizador e, no que diz respeito a implementações de interfaces ou aplicações *web*, deve-se ter em conta a premissa de que estes dados podem não ser confiáveis.

Conexões entre elementos De forma a que o módulo de gestão gráfico mantenha a coerência no que toca a possíveis configurações que possam ser feitas, terá que ter em conta regras de associação entre os diferentes blocos, mantendo a lógica definida atualmente na IPBrick. Esta coerência, baseia-se nas interações possíveis entre os elementos que compõe o dialplan, 4.2 e na estrutura do bloco. Ou seja, os blocos de maior complexidade, para além da sua estrutura característica, para se ligarem a outro elemento têm *outputs* definidos e esses *outputs* devem ser respeitados. Posto isto, como regra geral, os *outputs* de cada bloco não podem ser o destino de uma ligação, apenas a origem, e só é permitida uma e só uma ligação por *output*. No entanto, para cada bloco foram feitas validações de acordo com a lógica dos mesmos, sendo estas as regras definidas:

- *Inbound Rules*: Este bloco poderá ligar-se a qualquer bloco que seja uma função.

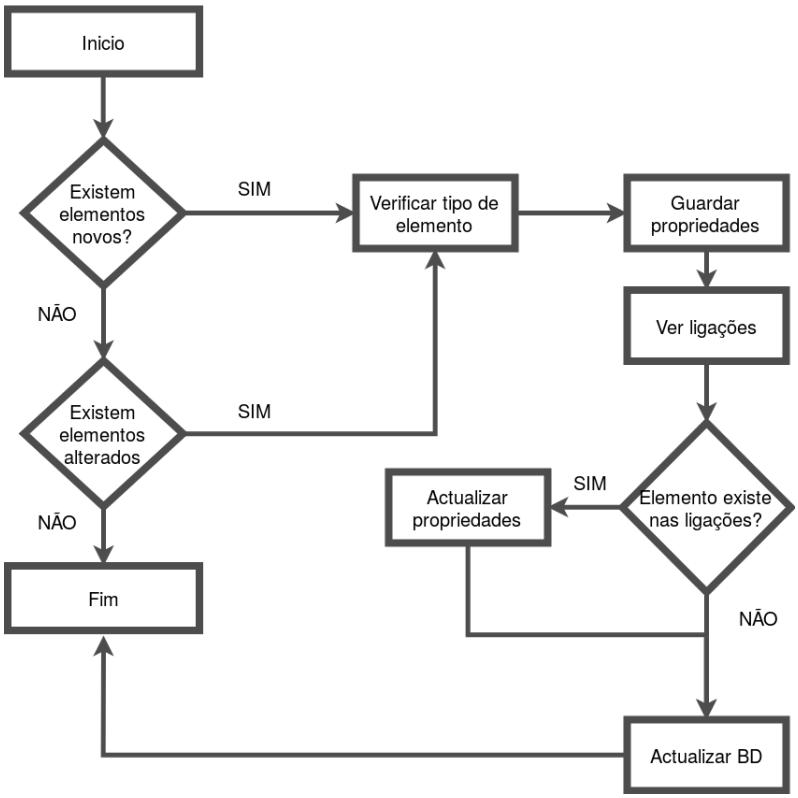


Figura 4.13: Algoritmo exportação

- *Direct Access*: Este elemento será o bloco que irá conter os acessos definidos para um utilizador ou telefone. No entanto, qualquer função que se ligue a um endereço SIP, irá ser representado por este bloco. Caso este endereço SIP corresponda a uma função interna do dialplan, a ligação irá ser feita para a função.

- Sequências de atendimento: Os *outputs* da sequência de atendimento só podem ligar aos elementos do tipo telefone ou do tipo *Direct Access* e, essas ligações são feitas a partir do output de cada membro constituinte.

- Conferência: As conferências podem ter associadas a si várias funções, no entanto, não se podem ligar a nenhum elemento.

- Grupos de chamada: O bloco *Call Group* não tem na sua estrutura *outputs* e, conforme definido em sistema, o bloco em si só se pode conectar com elementos do tipo utilizador, telefone ou *Direct Access*.

- IVR: Tal como as Sequências de Atendimento, este bloco pode ser conectado a outros a partir do output definido para cada atalho, sendo que o destino possível poderá ser de qualquer tipo, à exceção do bloco utilizador.

- Escalonador: O escalonador segue a mesma lógica que o IVR e que a Sequência de Atendimento, cada regra tem um output definido que se poderá ligar a qualquer tipo de elemento, à exceção do bloco utilizador.
- Fila de Espera: Este bloco, apresenta um output de acordo com possíveis configurações que possam ser feitas, como definir o número máximo de utilizadores ou o tempo máximo de espera, deste modo, cada output é associada a estas propriedades. As validações a considerar neste caso são nos membros da fila de espera, que só podem ser utilizadores ou telefones.

A figura 4.14, mostra uma possível má ligação a mero exemplo, sendo este tipo de ligações automaticamente detetadas, alertando o administrador do sucedido.

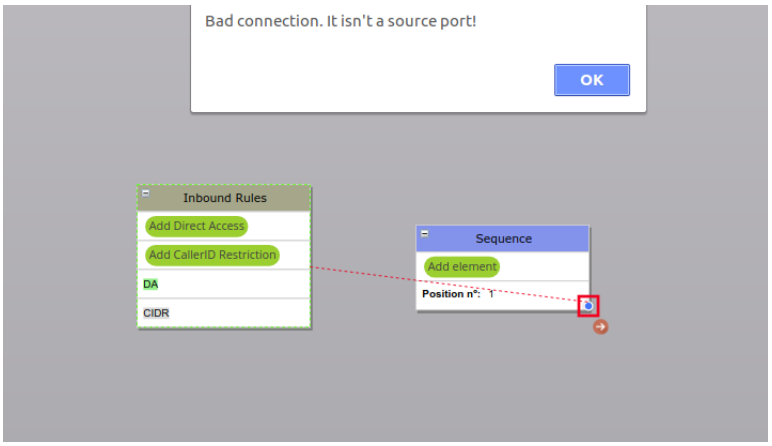


Figura 4.14: Exemplo de má conexão entre elementos

User input A partir do menu lateral direito, o administrador tem possibilidade de alterar os dados referentes a elementos que já existam ou então configurar elementos novos. Para que essa configuração seja válida, existem campos a serem preenchidos e, de seguida, validados. Esta validação não se remete apenas ao facto de verificar se os campos foram preenchidos, mas sim se foram preenchidos de acordo com o formato e se não apresentam caracteres inválidos que podem ser associados a possíveis injeções de código malicioso. Desde modo, a primeira etapa após a ação de aplicar os dados introduzidos passa por validar todos esses *user inputs* de forma a fazer escape de todos os caracteres inválidos, que corresponde à etapa número um, ilustrada na imagem 4.15. Usar os dados introduzidos pelo utilizador sem assegurar a integridade dos mesmos fragiliza a aplicação. O princípio base para que os dados estejam sempre o mais íntegros possíveis, é validar o input e fazer escape do output. Para validar, existem duas abordagens, validar se o input tem caracteres não permitidos (*blacklisting*) ou se o input tem caracteres aceitáveis (*whitelisting*). A abordagem mais segura, é a *whitelisting*, desta forma, sabemos que os dados validados são os que estamos a espera de obter. *Blacklisting*, assenta na premissa do programador antecipar os dados inválidos que possam aparecer, o que pode levar a erros. A validação dos dados, em termos gerais, passa por, verificar se os campos estão preenchidos, o tipo de informação que contém e o formato da mesma.

Figura 4.15: Validações

Portanto, a validação feita antes de os atributos serem introduzidos no XML é importante, mas não deve ser única. Estes atributos irão ser introduzidos na base de dados, nas tabelas correspondentes, ou seja, a segunda fase (figura 4.15) de validação é feita antes da inserção dos dados na base de dados. Desta forma, garantimos uma validação feita no lado do cliente e ainda a validação do lado do servidor.

No que diz respeito às validações de recursos, isto é, se os elementos criados entram em conflito com outros, por exemplo, se é definido para IVRs diferentes o mesmo acesso direto, são feitas pelas funções já definidas pela IPBrick para adicionar ou modificar dados na base de dados. Validações de preenchimento de dados, ou do seu formato estão, em grande parte, asseguradas pela interface gráfica.

4.2.4 Testes e Resultados

Neste caso, apesar de toda a metodologia ter sido preparada para que os testes se tornassem o mais simples possível, só foi conseguido testar a exportação das configurações num cenário de configuração isolada. Está configuração caracterizou-se por ser simples, no entanto, serve como uma prova de conceito da metodologia envolvida.

O cenário envolveu a criação de um bloco do tipo IVR, com um acesso direto do tipo SIP, uma restrição de *caller* ID e um *shortcut* configurado para um destino do tipo *Call Group*, a figura 4.16 mostra o cenário desta simples configuração. Em termos de configuração gráfica, é o equivalente a termos um bloco do tipo *Inbound Rules*, conectado a um bloco do tipo IVR, o qual tem um *shortcut* configurado para um bloco do tipo *Call Group*, como é ilustrado na figura 4.16.

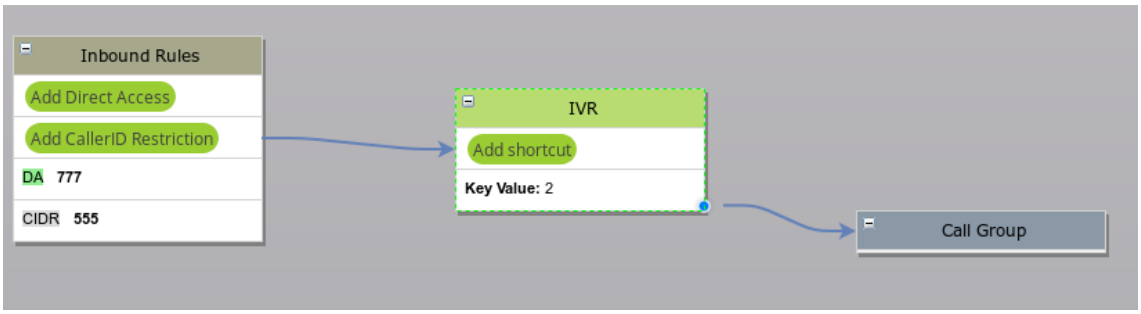
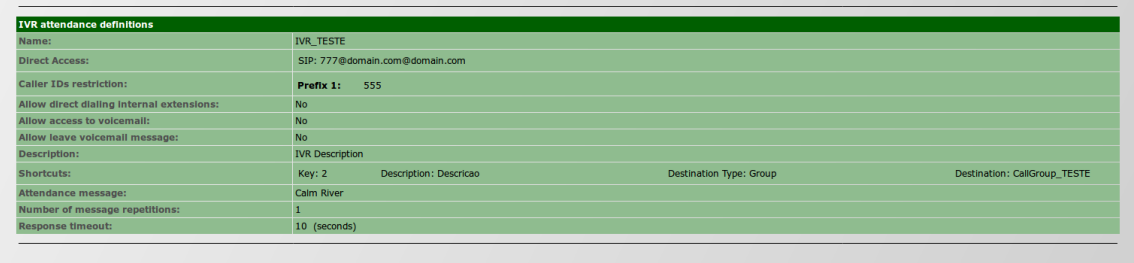


Figura 4.16: Cenário prova de conceito - *Export* das configurações

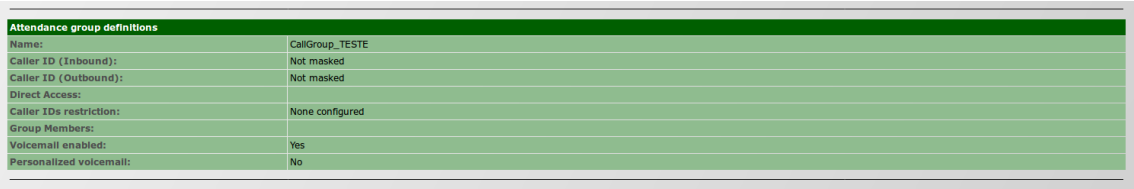
Após serem aplicadas configurações, o XML é então enviado através de um pedido AJAX, depois as informações acerca de cada bloco foram guardadas para que no fim fossem chamadas as funções já definidas pelo sistema que permitiram inserir os dados na base de dados. Apesar de ter sido feita uma configuração isolada, na próxima exportação realizada, este elementos já

constam na representação gráfica do dialplan. A figura 4.17 e 4.18, mostra os elementos criados na interface de administração da IPBrick.



IVR attendance definitions				
Name:	IVR_TESTE			
Direct Access:	SIP: 777@domain.com@domain.com			
Caller IDs restriction:	**Prefix 1:** 555			
Allow direct dialing internal extensions:	No			
Allow access to voicemail:	No			
Allow leave voicemail message:	No			
Description:	IVR Description			
Shortcuts:	Key: 2	Description: Descricao	Destination Type: Group	Destination: CallGroup_TESTE
Attendance message:	Calm River			
Number of message repetitions:	1			
Response timeout:	10 (seconds)			

Figura 4.17: IVR inserido - *Export* das configurações



Attendance group definitions	
Name:	CallGroup_TESTE
Caller ID (Inbound):	Not masked
Caller ID (Outbound):	Not masked
Direct Access:	
Caller IDs restriction:	None configured
Group Members:	
Voicemail enabled:	Yes
Personalized voicemail:	No

Figura 4.18: Grupo de chamada inserido - *Export* das configurações

Extensions.conf As configurações efetuadas são reproduzidas no ficheiro de configuração principal da estrutura do dialplan, o extensions.conf. Estas configurações são elaboradas num contexto específico para o elemento, com base em combinação das aplicações disponibilizadas pela infraestrutura que engloba o dialplan, o Asterisk.

Capítulo 5

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas conclusões acerca deste trabalho de dissertação e o trabalho futuro a desenvolver.

5.1 Conclusões

A implementação do módulo gráfico debateu-se com vários obstáculos, que se relacionam diretamente com a complexidade que assume a gestão do dialplan. Esta complexidade traduz-se nas potencialidades que o definem e apesar de toda a sua gestão se remeter a um ficheiro de configuração, existem dependências e configurações possíveis de ser efetuadas que se ampliam em toda a sua elaboração. Deste modo, o facto de ser possível visualizar através de uma representação gráfica, permite que o administrador consiga ter uma noção mais clara e percetível dos fluxos de chamada relativos ao sistema. Esta visão, por consequência, permite um *debug* mais prático e intuitivo, ou seja, permite ao administrador identificar algum problema de configuração caso exista.

O módulo está preparado para que, qualquer elemento que seja introduzido, seja o mais linear possível, ou seja, simplesmente é necessário usar as funções de *backend* correspondentes, criar um identificador para o mesmo e depois é replicar o que foi elaborado para os elementos que compõe atualmente o módulo gráfico.

Fundamentalmente, a modularização de cada elemento permitiu que toda a experiência de configuração e visualização se tornasse numa associação lógica entre elementos que possuem atributos e que, as associações com outros elementos se traduzem em conexões entre os mesmos.

Os objetivos definidos inicialmente, referidos no capítulo 1, focavam-se essencialmente em quatro pontos fundamentais apresentados na tabela 5.1. Constata-se que a implementação do módulo não colocou em causa a compatibilidade com o sistema, ou seja, foi pensado de modo a que a tecnologia utilizada não entrasse em conflito com dependências no sistema e a camada de abstração criada a partir da estruturação dos elementos no XML, permite que qualquer que seja a versão da IPBrick, se for mantida a comunicação com a base de dados, está assegurado o funcionamento em pleno do módulo gráfico.

Tabela 5.1: Objectivos definidos

Objectivo	Resultado
Representação gráfica do dialplan	Concluído
Compatibilidade do módulo com o sistema	Concluído
Alterações on demand através do mecanismo drag&drop	Concluído parcialmente
Integridade dos dados e configurações	Concluído parcialmente

Para além desse objetivo, a representação gráfica foi conseguida com sucesso, o módulo é capaz de importar os elementos presentes no dialplan interno, sendo que, tem toda uma lógica documentada a qual permite que caso seja adicionado um elemento novo.

As alterações estão preparadas para ser feitas assim que o administrador desejar, no entanto, o *export* para o sistema foi conseguido apenas para um cenário muito simples. Contudo, está definido todo um planeamento para ser aproveitado futuramente.

A integridade das associações foi assegurada, no entanto, pode ser ainda mais evasiva no que toca à deteção de possíveis más configurações, como a possibilidade de detetar automaticamente *loops* causados por configurações descuidadas. No que diz respeito à integridade dos dados e validações de *user input*, foram introduzidos mecanismos de verificação dos dados inseridos, contudo, em termos de *backend* ainda podem existir mais mecanismos incorporados, assim como a realização de testes de intrusão que podem detetar possíveis falhas.

5.2 Trabalho Futuro

Os propósitos do trabalho futuro, relacionam-se, primeiramente com a conclusão da exportação das configurações tendo em conta a modularização já desenvolvida, depois, com funcionalidades que podem ser implementadas na interface gráfica, com as validações que podem ser adicionadas na parte de *backend* do sistema e com possíveis elementos que podem ser adicionados ao módulo gráfico de gestão do dialplan de forma a aproveitar as potencialidades dadas pelo mesmo.

5.2.1 Exportação das Configurações

Apesar de toda a metodologia e algoritmia relacionada com a exportação da configurações, os testes realizados foram feitos com um cenário muito simples.

O trabalho futuro neste âmbito, passar por replicar o processo envolvente para este cenário e reproduzir com configurações mais complexas.

5.2.2 Interface gráfica

A interface gráfica já dispõe de várias funcionalidades valiosas para o administrador, no que diz respeito ao processo de visualização e configuração. No entanto, apesar de ser possível filtrar certos elementos do dialplan consoante o seu tipo, poderia ser implementada a capacidade de

destacar um certo caminho através da opção *Show Path* já presente no *pop-up menu*. Esta implementação faz mais sentido ser implementada num bloco do tipo *Inbound Rule*, desta forma, o administrador poderia através de uma regra de entrada de um certo bloco, ter uma perceção mais objetiva de quais os caminhos que uma chamada pode ter, ao entrar em algum dos acessos diretos presentes nesse mesmo bloco. Esta funcionalidade seria vantajosa não só em termos visuais, porque a ideia seria abstrair os outros caminhos, mas também em termos de *debug* e controlo de erros.

Neste momento, as propriedades disponíveis para serem configuradas a partir da interface gráfica do bloco, concentram-se nas que são fundamentais. Um dos focos do trabalho futuro poderá transportar toda a informação que é possível de alterar nas páginas de cada elemento para a interface gráfica.

5.2.3 Testes de usabilidade

Os testes de usabilidade são uma mais valia para qualquer aplicação, uma vez que permitem obter *feedback* por parte de quem a utiliza, de forma a avaliar a flexibilidade e funcionalidade da interface.

O módulo implementado, permite que certas ações como copiar ou eliminar elementos e reverter ações sejam feitas a partir de um menu que é acessível a partir do botão direito do rato mas a biblioteca permite facilmente adicionar *listeners* para que esses atalhos sejam feitos a partir do teclado caso, após testes de usabilidade concluam que essa abordagem seja mais benéfica para utilizador. Para além de funcionalidades que podem ser acrescentadas ou alteradas, existe também a possibilidade de alterar facilmente o design e as cores subjacentes ao mesmo, caso os testes de usabilidade falhem nesse aspeto.

5.2.4 Validações

Apesar dos cuidados associados ao módulo gráfico, de forma a que não haja comprometimento da aplicação a partir dos dados introduzidos pelo utilizador, é importante que esta metodologia seja replicada para as outras páginas que fazem ou possam fazer parte da aplicação. Existem também cuidados a ter do lado do servidor que permitem tornar a aplicação mais robusta.

Deste modo, seria importante procurar colmatar algumas falhas existentes atualmente na aplicação que podem, ou não, serem exploradas de forma a causar estragos. Após uns testes realizados na aplicação, precisamente nas páginas relacionadas com a infraestrutura VoIP, que estão diretamente relacionadas com o módulo gráfico, através de ferramentas capazes de detetar possíveis falhas, foi evidenciado que havia campos na aplicação que permitam a injeção de código *JavaScript* (XSS), que havia a falta de um *token* de sessão (CSRF) e que, em certas páginas, também existia a possibilidade de injeção de código SQL (SQLi). Os testes foram feitos usando ferramentas disponíveis, como *Burp Suite*, *Arachni* e *Sqlmap*.

Não foi prioridade a correção destas vulnerabilidades, no entanto, existem procedimentos que podem, futuramente, ser implementados por forma a que não existam falhas deste tipo que possam

comprometer a aplicação. Os procedimentos passam, em primeiro lugar, por procurar validar qualquer input realizado a partir da aplicação web, porque pode bastar um para comprometer a aplicação. Em segundo lugar, como mitigação para prevenir ataques do tipo CSRF, será usar uma biblioteca disponibilizada pela *OWASP*, a *CSRF Protector Project*, que assegura a proteção contra ataques deste tipo. A sua utilização apesar de ser simples, é necessário que todas as páginas associadas à aplicação a incluam. Em terceiro lugar, por forma a mitigar *SQLi*, é importante que as *queries* sejam parametrizadas, com o uso de *PHP Data Objects*, assim permite à base de dados fazer a distinção entre código malicioso ou código SQL.

5.2.5 Rotas externas

O foco deste módulo gráfico foi o dialplan interno implementado uma IPBrick, no entanto, existem ainda regras externas que permitem ao dialplan interno comunicar com outros servidores. A comunicação com servidores externos ou com operadoras é feita através de um *SIP trunk*, que elimina a necessidade de usar uma ligação física entre os terminais. Assim sendo, uma adição a este módulo será a inclusão de um bloco *SIP trunk*, para que seja possível, quando a saída de uma função for um endereço SIP externo, que esse bloco esteja ligado ao *trunk* correspondente.

No entanto, esta implementação apresenta um grau de dificuldade elevado, uma vez que as configurações inerentes ao *SIP trunk* são várias, como é ilustrado na figura 5.1. Contudo, a existência deste bloco permitiria perceber quais os números internos que estariam autorizados a comunicar com o exterior, através das regras de *pattern* definidas. Isto é, determinados números, caso obedecessem aos prefixos configurados no *SIP trunk*, como ilustrado na figura 5.2, poderiam ter ligações com o servidor externo. O ponto fulcral seria mesmo este, ou seja, quando um número interno fizesse *match* com o *pattern* de um respetivo *trunk*, teríamos na interface gráfica uma ligação para o bloco *SIP trunk* correspondente.

Route definitions	
Type:	SIP
Name:	SIPTRUNKEXAMPLE
SIP server address:	192.168.19.66
SIP server port:	5060
Authentication:	Fixed IP
Video support:	No
Caller ID:	Use external callerid
Registration realm:	192.168.19.251
Outbound proxy:	192.168.19.251
Available to internet:	No
Simetric signaling:	No
Search:	Disabled
DTMF type:	rfc2833
Call limit:	Unlimited
State check:	No
No far-end NAT detection by provider:	No
Mandatory Route:	No
P-Asserted-Identity:	Yes
P-Asserted-Identity Format:	Caller ID
Caller ID on blind tranfers:	Use the Caller ID of the phone that transfers
Caller ID update during call:	Disabled
Enable SRTP:	No

Figura 5.1: Exemplo SIP Trunk - Interface IPBrick OS

Prefix | Number Pattern	Include prefix in address	Postrouting prefix	Caller IDs restriction	Fallback available	Generate local ringing tone	Priority
0 | XXXXXXXXX	No	-	No	No	No	Disabled
2 | -	Yes	-	No	No	No	Disabled
3 | -	Yes	-	No	No	No	Disabled
7 | -	Yes	-	No	No	No	Disabled
8 | -	Yes	-	No	No	No	Disabled
9 | -	Yes	-	No	No	No	Disabled
00 | -	Yes	-	No	No	No	Disabled
09 | XXXXXXXX	No	9	No	No	No	Disabled
8555 | -	No	-	No	No	No	Disabled

Figura 5.2: Exemplo Lista Prefixos- Interface IPBrick OS

Anexo A

Interface Gráfica

Figura A.1: Visão geral da interface gráfica

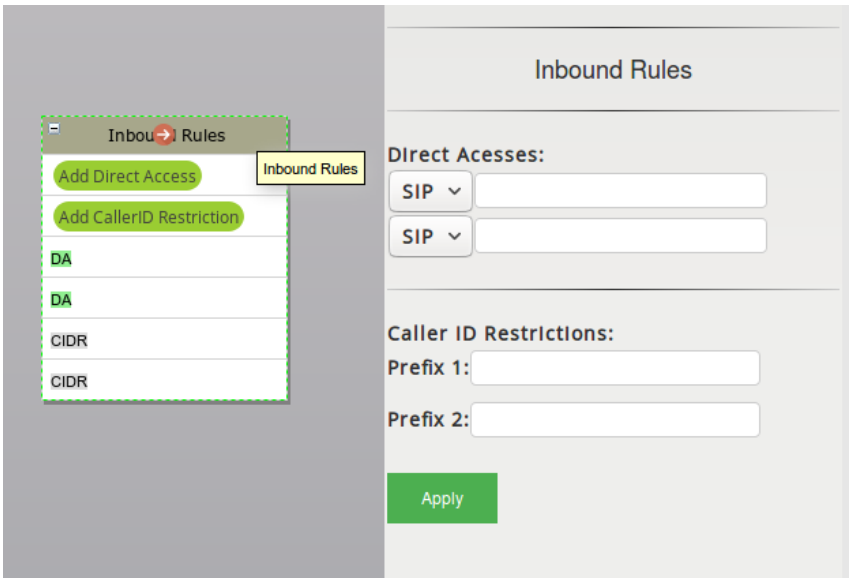


Figura A.2: Bloco *Inbound Rules*

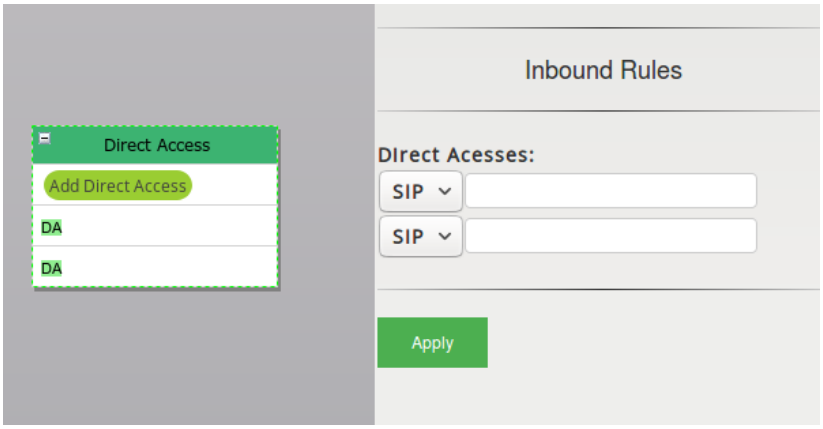


Figura A.3: Bloco *Direct Access*

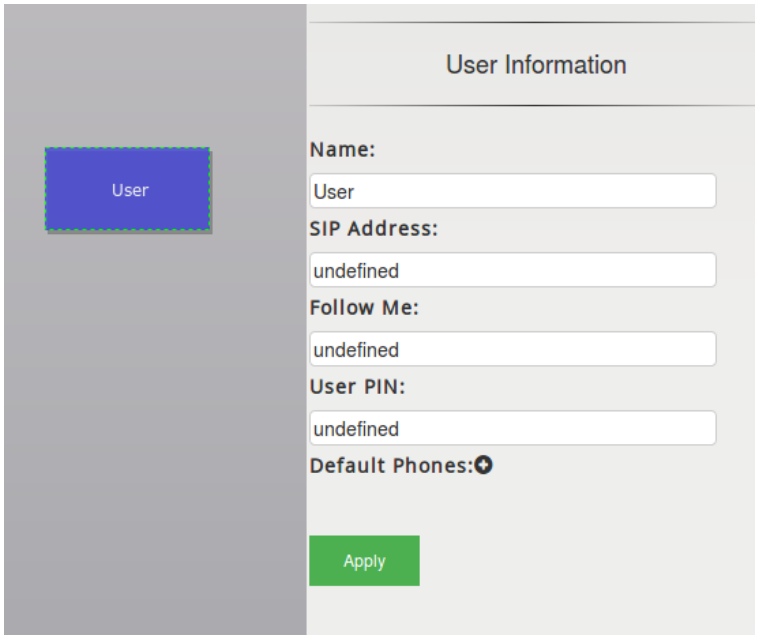


Figura A.4: Bloco *User*

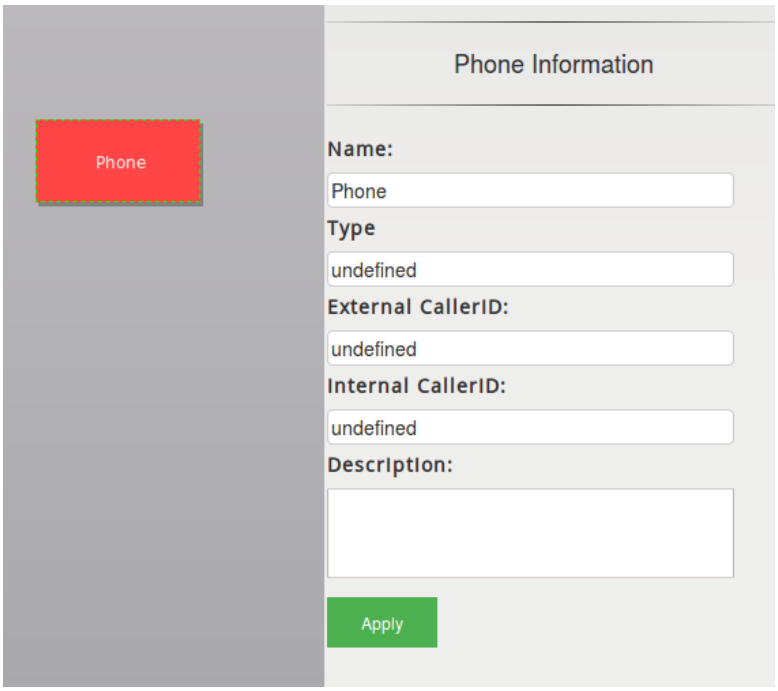


Figura A.5: Bloco *Phone*

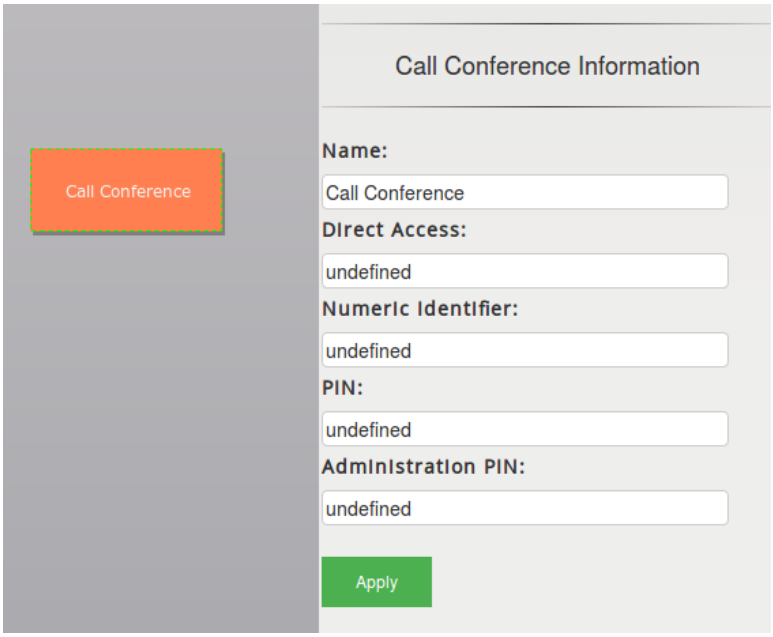


Figura A.6: Bloco *Call Conference*

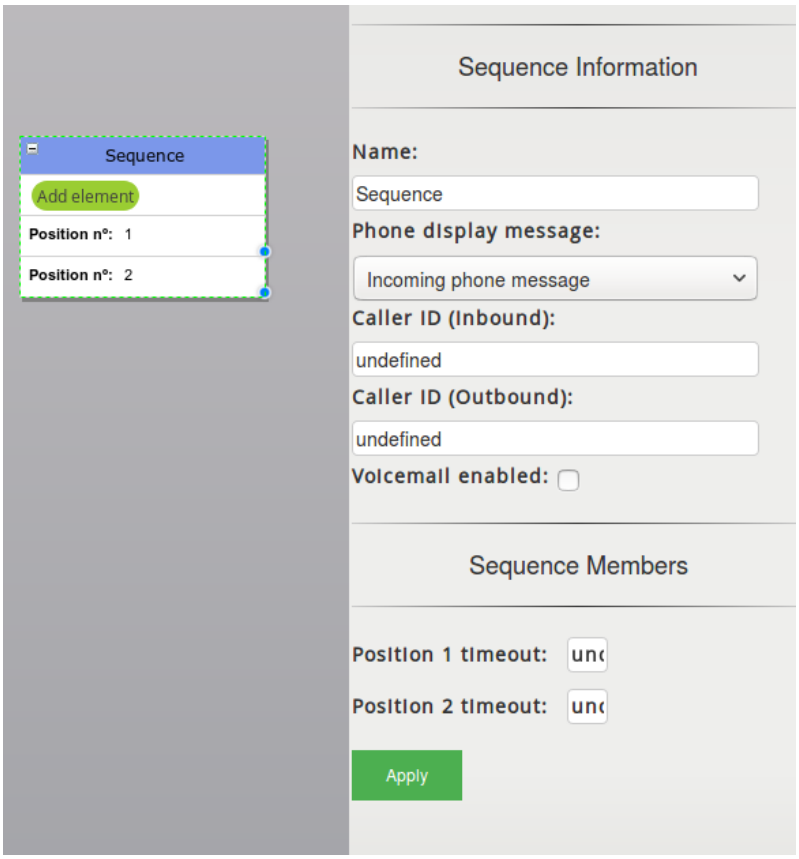


Figura A.7: Bloco *Attendance Sequence*

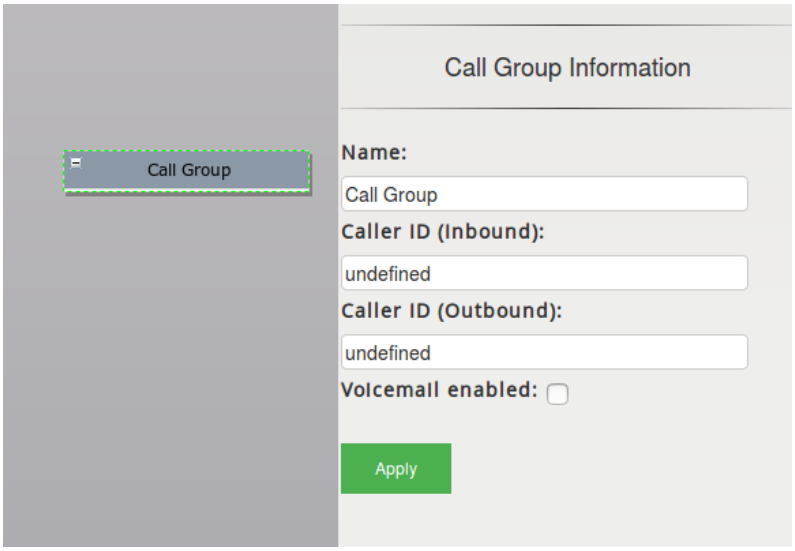


Figura A.8: Bloco *Call Group*

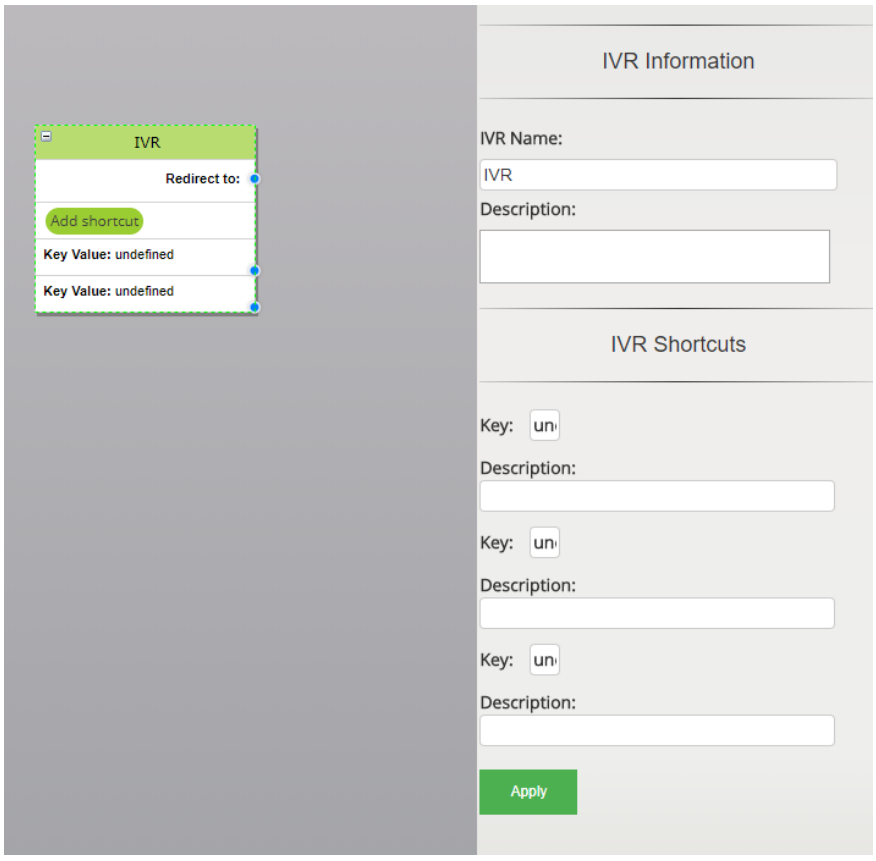


Figura A.9: Bloco *IVR*

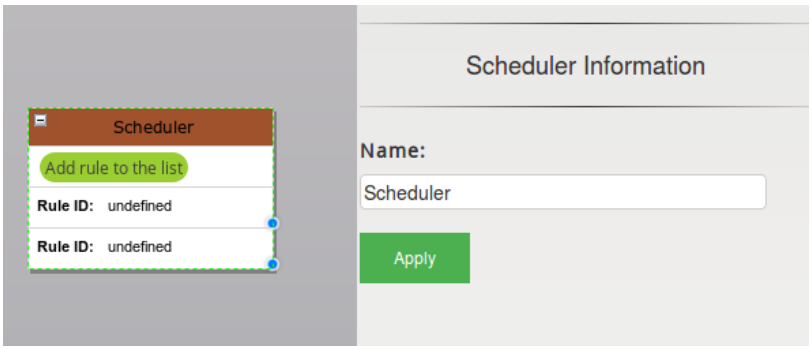


Figura A.10: Bloco *Scheduler*

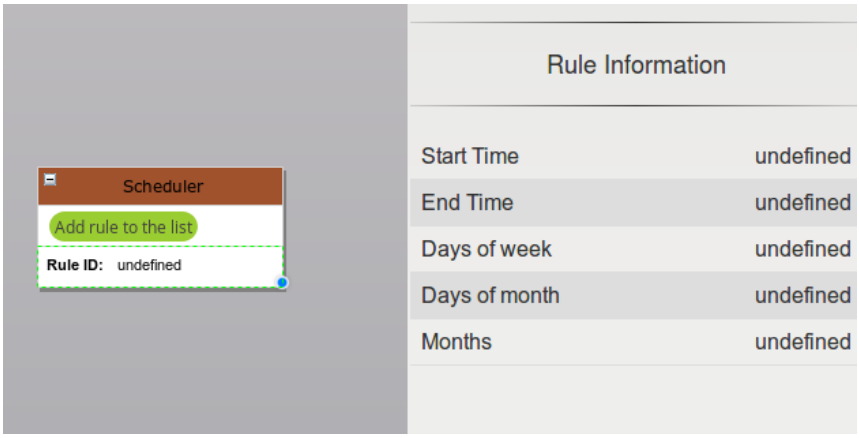


Figura A.11: Bloco *Scheduler* - Regras

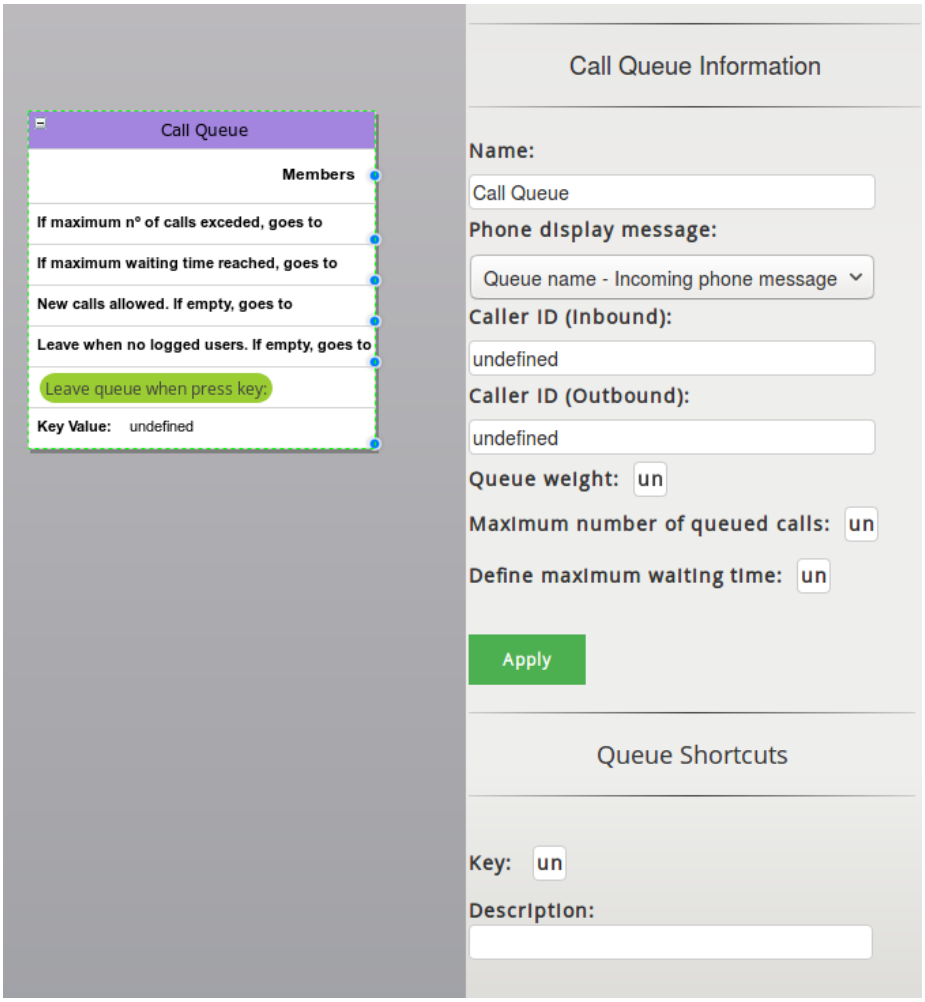


Figura A.12: Bloco *Call Queue*

Anexo B

Dialplan Importado

Internal

Source			Destination		Description
Number	Type	Interface	Type	Interface	
309			FUNC	ext309	Phone VoIP alternate address of phone
308			FUNC	ext308	Phone VoIP alternate address of phone
307			FUNC	ext307	Phone VoIP alternate address of phone
306			FUNC	ext306	Phone VoIP alternate address of phone
305			FUNC	ext305	Phone VoIP alternate address of phone
304			FUNC	ext304	Phone VoIP alternate address of phone
303			FUNC	ext303	Phone VoIP alternate address of phone
302			FUNC	ext302	Phone VoIP alternate address of phone
301			FUNC	ext301	Phone VoIP alternate address of phone
300			FUNC	ext300	Phone VoIP alternate address of phone

41 - 50 | Total: 126

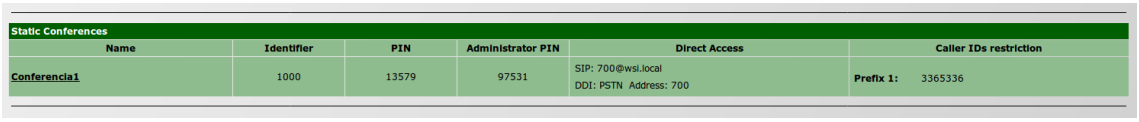
Show 10 entries Search:

Inbound

Source			Destination		Description
Number	Type	Interface	Type	Interface	
8585	SIP		FUNC	Sequence2	Attendance Sequences
+351222111333	SIP		FUNC	Sequence1	Attendance Sequences
+351222111333	SIP		FUNC	Sequence1	Attendance Sequences
999	SIP		FUNC	Sequence1	Attendance Sequences
900	SIP		FUNC	SchedulerGeral	Scheduling
+351858585	SIP		FUNC	Scheduler99	Scheduling
+351888777	SIP		FUNC	Scheduler1	Scheduling
+351969633	SIP		FUNC	Scheduler1	Scheduling
35188888	SIP		FUNC	Scheduler1	Scheduling
800	SIP		FUNC	Scheduler1	Scheduling

1 - 10 | Total: 47

Figura B.1: Página dialplan - *Internal and Inbound Routes*



Static Conferences

Name	Identifier	PIN	Administrator PIN	Direct Access	Caller IDs restriction
Conferencia1	1000	13579	97531	SIP: 700@wsi.local DDI: PSTN Address: 700	Prefix 1: 3365336

Figura B.2: Conferências no dialplan

List of configured IVR Attenders

Name	Type	Direct Access	Caller IDs restriction
IVR1	Normal		None configured
IVR_123	Normal		None configured
IVR2	Normal	SIP: 666@wsi.local	None configured
IVR333	Normal		None configured
IVR4Voicemail	Normal		None configured
IVR5	Normal		None configured
IVR55	Normal		None configured
IVR60	Normal	SIP: 6517@wsi.local	None configured
IVR77	Normal		None configured
IVR88	Normal		None configured
IVRInicial	Normal	SIP: 123456789@wsi.local	None configured
IVR_Outros	Normal		None configured

Figura B.3: IVRs no dialplan

List of configured Call Groups

Name	Caller ID	Voicemail enabled	Direct Access	Caller IDs restriction
Call Group1	Not masked	No		None configured
Call Group2	Not masked	No		None configured
Call Group3	Not masked	No		None configured
Call Group4	Not masked	No		None configured
Call Group5	Not masked	No		None configured

Figura B.4: Grupos de Chamada no dialplan

Attendance Sequences List

Name	Caller ID	Voicemail enabled	Direct Access	Caller IDs restriction
Sequence1	Not masked	No	SIP: +351222111333@wsi.local SIP: 999@wsi.local SIP: +351222111333@wsi.local DDI: PSTN Address: 800	**Prefix 1:** 99999999
Sequence2	Not masked	No	SIP: 8585@wsi.local	**Prefix 1:** 6666
SequenceAsVoicemail	Not masked	Yes		None configured

Figura B.5: Sequências de Atendimento no dialplan

Schedulers

Name	Direct Access	Caller IDs restriction
Scheduler1	SIP: 35188888@wsi.local SIP: 999@wsi.local SIP: +351888777@wsi.local SIP: +351969633@wsi.local DDI: PSTN Address: 800 SIP: 800@wsi.local	None configured
SchedulerGeral	DDI: PSTN Address: 900 SIP: 900@wsi.local	None configured
Scheduler99	SIP: +351858585@wsi.local	None configured
NoScheduler		None configured
SchedulerV		None configured

Figura B.6: Escalonadores no dialplan

List of call queues

Name	Caller ID	Direct Access	Caller IDs restriction
CallQueueGeral	Not masked		None configured
CallQueueGeral1	Not masked		None configured
CallQueueGeral88	Not masked		None configured
CallQueueGeralInfo	Not masked		None configured
CQOutbound	Not masked		None configured
CQOutrasInfos	Not masked		None configured
CQ_Suges	Not masked		None configured

Figura B.7: Filas de Espera no dialplan

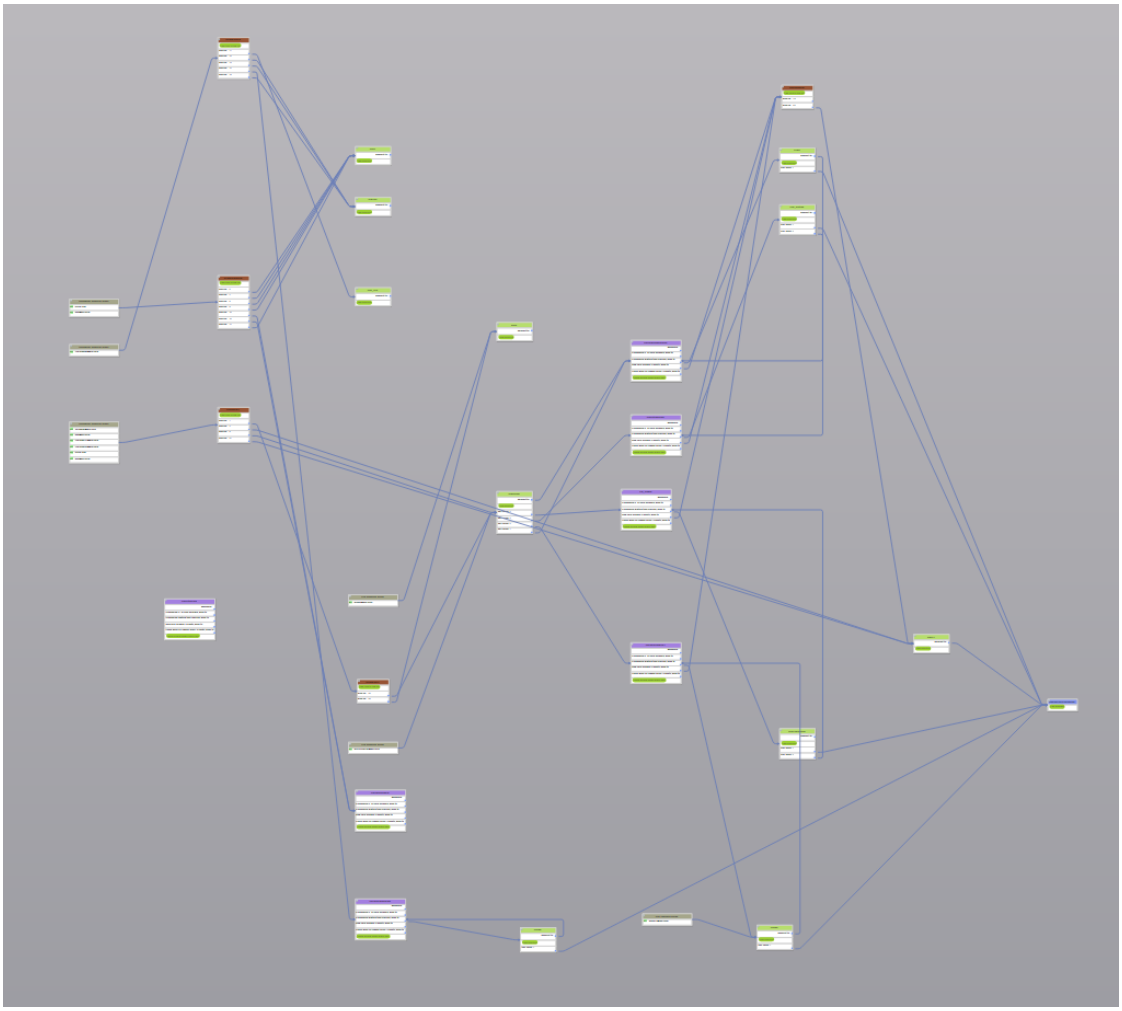


Figura B.8: Visão geral - Exportação

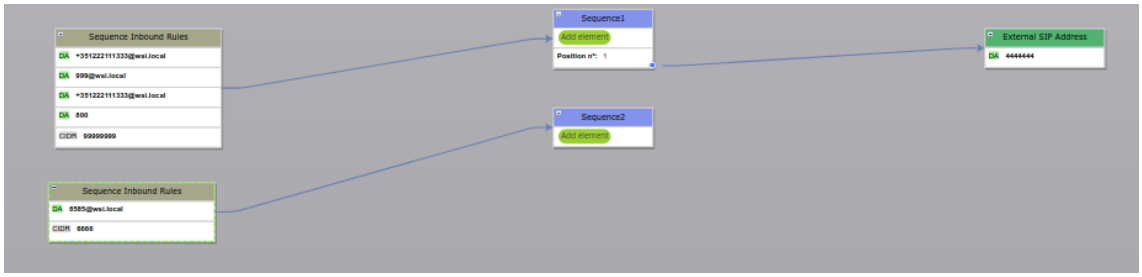


Figura B.9: Exemplo 1

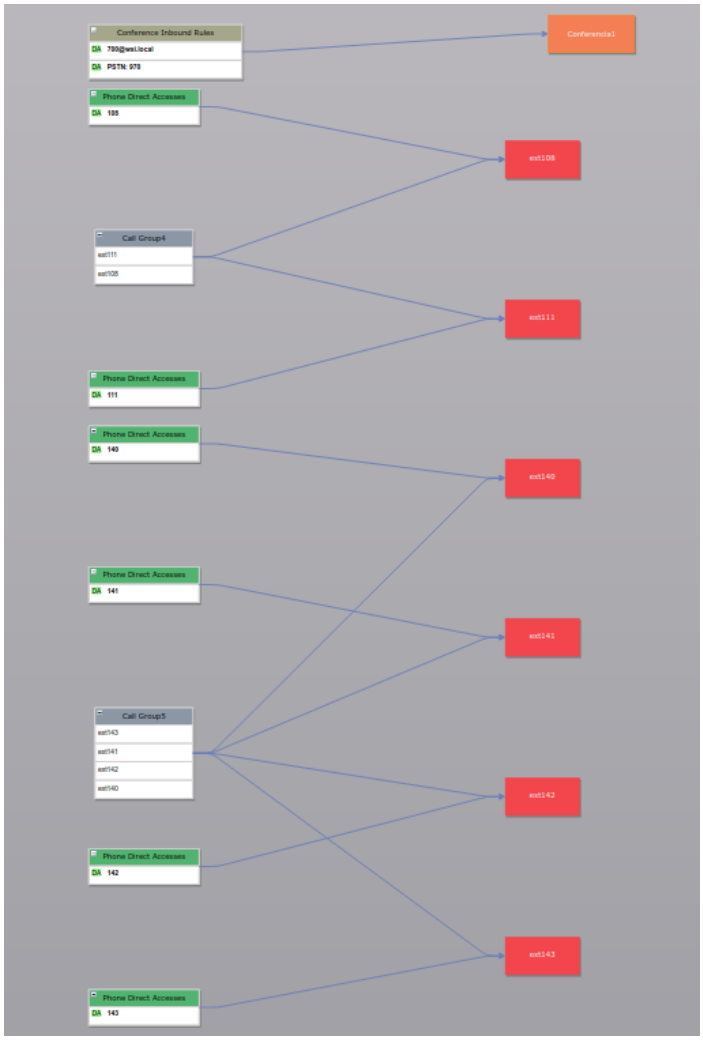


Figura B.10: Exemplo 2

Anexo C

XML - Exportação

```
1
2  <mxGraphModel>
3    <root>
4      <mxCell id="0"/>
5      <mxCell id="1" parent="0"/>
6      <mxCell id="2" style="da" vertex="1" parent="1">
7        <Table name="Inbound Rules" as="value"/>
8        <mxGeometry x="70" y="198" width="160" height="208" as="geometry">
9          <mxRectangle width="160" height="30" as="alternateBounds"/>
10       </mxGeometry>
11     </mxCell>
12     <mxCell id="3" value="addDA" vertex="1" connectable="0" parent="2">
13       <mxGeometry y="28" width="160" height="30" as="geometry"/>
14     </mxCell>
15     <mxCell id="4" value="addCIDR" vertex="1" connectable="0" parent="2">
16       <mxGeometry y="58" width="160" height="30" as="geometry"/>
17     </mxCell>
18     <mxCell id="9" value="888" vertex="1" connectable="0" DA="1" parent="2">
19       <mxGeometry y="88" width="160" height="30" as="geometry"/>
20     </mxCell>
21     <mxCell id="10" value="777" vertex="1" connectable="0" DA="1" parent="2">
22       <mxGeometry y="118" width="160" height="30" as="geometry"/>
23     </mxCell>
24     <mxCell id="11" value="222" vertex="1" connectable="0" CIDR="1" parent="2">
25       <mxGeometry y="148" width="160" height="30" as="geometry"/>
26     </mxCell>
27     <mxCell id="12" value="555" vertex="1" connectable="0" CIDR="1" parent="2">
28       <mxGeometry y="178" width="160" height="30" as="geometry"/>
29     </mxCell>
30     <mxCell id="5" style="table_ivr" vertex="1" parent="1">
31       <Table name="IVR_TESTE" as="value">
32       <mxGeometry x="300" y="255" width="180" height="88" as="geometry">
33         <mxRectangle width="180" height="30" as="alternateBounds"/>
34       </mxGeometry>
```

```
35      </mxCell>
36      <mxCell id="6" value="addIVRShortcut" vertex="1" connectable="0" parent="5">
37        <mxGeometry y="28" width="180" height="30" as="geometry"/>
38      </mxCell>
39      <mxCell id="14" value="keyIVR" vertex="1" connectable="0" parent="5" keyValue="
            2">
40        <mxGeometry y="58" width="180" height="30" as="geometry"/>
41      </mxCell>
42      <mxCell id="13" value="OutputIVR" style="port" vertex="1" parent="14">
43        <mxGeometry x="1" y="0.75" width="16" height="16" relative="1" as="geometry">
44          <mxPoint x="-8" y="-4" as="offset"/>
45        </mxGeometry>
46      </mxCell>
47      <mxCell id="7" style="table_cgroup" vertex="1" parent="1">
48        <Table name="CallGroup_TESTE" as="value"/>
49        <mxGeometry x="580" y="225" width="160" height="88" as="geometry">
50          <mxRectangle width="160" height="30" as="alternateBounds"/>
51        </mxGeometry>
52      </mxCell>
53      <mxCell id="16" style="sourcePort=13;" edge="1" parent="1" source="14" target="
            7">
54        <mxGeometry relative="1" as="geometry"/>
55      </mxCell>
56    </root>
57  </mxGraphModel>
```

Referências

[1] IPBRICK SA. Whitepaper-UCoIP. disponível em https://www.ipbrick.com/wp-content/uploads/2015/12/WhitePaper-UCOIP.PT_1.pdf, acedido a última vez em 28 de Novembro de 2017.

[2] IPBRICK SA. IPBRICK.GT Guide. disponível em http://downloads.ipbrick.com/IPBrick/documentation/EN/ipbgt_en.pdf, acedido a última vez em 28 de Novembro de 2017.

[3] Russell Bryant. *Asterisk: The Definitive Guide*. O'Reilly, Beijing : Sebastopol, 4a edição, 2013.

[4] Debian Stretch. disponível em https://wiki.debian.org/DebianStretch, acedido a última vez em 5 de Fevereiro de 2018.

[5] 3CX. VoIP definition. disponível em https://www.3cx.com/pbx/voip-definition/, acedido a última vez em 1 de Fevereiro de 2018.

[6] Ubaid Ur Rehman e Abdul Ghafoor Abbasi. Security analysis of VoIP architecture for identifying SIP vulnerabilities. Em *Emerging Technologies (ICET), 2014 International Conference on*. IEEE, 2014.

[7] Internet Engineering Task Force IETF. SIP: Session Initiation Protocol. disponível em https://www.ietf.org/rfc/rfc3261.txt/, acedido a última vez em 2 de Fevereiro de 2018.

[8] 3CX PBX. SIP: Protocol. disponível em http://www.en.voipforo.com/SIP/SIP_components.php, acedido a última vez em 2 de Fevereiro de 2018.

[9] David Gomillion e Barrie Dempster. *Building Telephony Systems with Asterisk*. PACKT - Publishing, February 2006.

[10] Internet Engineering Task Force IETF. IAX: Inter-Asterisk eXchange Version 2. disponível em https://tools.ietf.org/html/rfc5456, acedido a última vez em 2 de Fevereiro de 2018.

[11] Internet Engineering Task Force IETF. RTP: A Transport Protocol for Real-Time Applications. disponível em https://tools.ietf.org/html/rfc3550, acedido a última vez em 1 de Fevereiro de 2018.

[12] Internet Engineering Task Force IETF. Hypertext Transfer Protocol – HTTP/1.1. disponível em https://tools.ietf.org/html/rfc2616, acedido a última vez em 2 de Fevereiro de 2018.

[13] Dafydd Stuttard e Marcus Pinto. *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. John Wiley, 2a edição, 2011.

[14] Mozilla Web Documentation. HTTP cookies. disponível em https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies, acedido a última vez em 2 de Fevereiro de 2018.

[15] Pavan Podila. HTTP: The Protocol Every Web Developer Must Know - Part 1. disponível em https://code.tutsplus.com/tutorials/http-the-protocol-every-web-developer-must-know-part-1--net-31177, acedido a última vez em 30 de Janeiro de 2018.

[16] Inc Digium. What Is A IP PBX? disponível em https://www.asterisk.org/get-started/applications/what-is-an-ip-pbx, acedido a última vez em 2 de Fevereiro de 2018.

[17] GetVoip Rober Pepper. What is a SIP Proxy Server? disponível em https://getvoip.com/library/what-is-a-sip-proxy-server/, acedido a última vez em 30 de Janeiro de 2018.

[18] The Apache Software Foundations. The number one HTTP server on the Internet. disponível em https://httpd.apache.org/, acedido a última vez em 30 de Janeiro de 2018.

[19] Acunetix. Defence in Depth – Part 4 – Validate everything, Parameterize SQL queries. disponível em https://www.acunetix.com/blog/articles/defence-in-depth-part-4-validate-everything-parameterize-sql-queries/, acedido a última vez em 02 de Junho de 2018.

[20] Acunetix. Prevent SQL injection vulnerabilities in PHP applications and fix them. disponível em https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/, acedido a última vez em 02 de Junho de 2018.

[21] OWASP. Cross-site Request Forgery (CSRF). disponível em https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF), acedido a última vez em 02 de Junho de 2018.

[22] OWASP. Cross-site Request Forgery (csrf) Prevention Cheat Sheet. disponível em https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet, acedido a última vez em 02 de Junho de 2018.

[23] Origin http header. disponível em https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Origin, acedido a última vez em 02 de Junho de 2018.

[24] OWASP. Owasp - Cross-site Scripting (XSS). disponível em https://www.owasp.org/index.php/Cross-site_Scripting_(XSS), acedido a última vez em 02 de Junho de 2018.

[25] OWASP. XSS (Cross Site Scripting) Prevention Cheat Sheet. disponível em https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet, acedido a última vez em 02 de Junho de 2018.

[26] Acunetix. Preventing XSS Attacks. disponível em https://www.acunetix.com/blog/articles/preventing-xss-attacks/, acedido a última vez em 02 de Junho de 2018.

[27] OWASP. SQL Injection. disponível em https://www.acunetix.com/blog/articles/preventing-xss-attacks/, acedido a última vez em 02 de Junho de 2018.

[28] OWASP. SQLi Prevention Cheat Sheet. disponível em https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet, acedido a última vez em 02 de Junho de 2018.

[29] Ale Imran, Qadeer Mohammed A., e M. J. R. Khan. Asterisk VoIP Private Branch Exchange. Em *Multimedia, Signal Processing and Communication Technologies, 2009. IMPACT'09. International*. IEEE, 2009.

[30] Asterisk Project. Types of Asterisk Modules. disponível em https://wiki.asterisk.org/wiki/display/AST/Types+of+Asterisk+Modules, acedido a última vez em 28 de Novembro de 2017.

[31] Benjamin Jackson e Champ Clark. *Asterisk Hacking*. Syngress, Burlington, MA, 2007.

[32] Nir Simionovich. *Asterisk Gateway Interface 1.4 And 1.6 Programming*. Packt Publishing, Burlington, MA, 2009.

[33] Matt Jordan. Asterisk REST Interface. disponível em https://wiki.asterisk.org/wiki/pages/viewpage.action?pageId=29395573, acedido a última vez em 2 de Fevereiro de 2018.

[34] The Kamailio SIP Server Project. Welcome To Kamailio – The Open Source SIP Server. disponível em https://www.kamailio.org/w/, acedido a última vez em 30 Janeiro de 2018.

[35] W3Schools. JavaScript tutorial. disponível em https://www.w3schools.com/js/L, acedido a última vez em 1 de Fevereiro de 2018.

[36] XMLFiles.com. Introduction to XML. disponível em https://www.xmlfiles.com/xml/xml-intro/, acedido a última vez em 2 de Fevereiro de 2018.

[37] The PHP Group. Manual do PHP. disponível em https://secure.php.net/manual/pt_BR/preface.php, acedido a última vez em 2 de Fevereiro de 2018.

[38] Flávio Eduardo de Andrade Gonçalves. *Como construir e configurar um PBX com Software Livre*. BISAC, Julho 2006.

[39] Asterisk Project. Dialplan - Variables. disponível em https://www.wiki.asterisk.org/wiki/display/AST/Variables, acedido a última vez em 2 de Fevereiro de 2018.

[40] VOIP-Info.org. Asterisk variables. disponível em https://www.voip-info.org/wiki/view/Asterisk+variables, acedido a última vez em 2 de Fevereiro de 2018.

[41] Asterisk Project. The Verbose and NoOp Applications. disponível em https://wiki.asterisk.org/wiki/display/AST/The+Verbose+and+NoOp+Applications, acedido a última vez em 28 de Novembro de 2017.

[42] VisualPBX. disponível em http://www.apstel.com/site/wp-content/uploads/2015/05/VisualPBX.png, acedido a última vez em 30 de Janeiro de 2018.

[43] VisualDialplan. disponível em http://saghul.net/blog/wp-content/uploads/2007/07/vdp_validation.png, acedido a última vez em 30 de Janeiro de 2018.

[44] Jive Hosted VoIP Dial-plan Editor. disponível em https://assets.pcmag.com/media/images/531632-jive-hosted-voip-dial-plan-editor.png, acedido a última vez em 30 de Janeiro de 2018.

[45] Flowchart jQuery. disponível em http://sebastien.drouyer.com/jquery.flowchart-demo/, acedido a última vez em 31 de Maio de 2018.

[46] MxGraph. disponível em https://github.com/jgraph/mxgraph, acedido a última vez em 31 de Maio de 2018.

[47] mxGraph User Manual – JavaScript Client. disponível em https://jgraph.github.io/mxgraph/docs/manual.html, acedido a última vez em 02 de Junho de 2018.

[48] mxGraph classs - JavaScrip API Doc. disponível em https://jgraph.github.io/mxgraph/docs/js-api/files/view/mxGraph-js.html#mxGraph, acedido a última vez em 10 de Junho de 2018.

[49] mxEditor JavaScrip API Doc. disponível em https://jgraph.github.io/mxgraph/docs/js-api/files/editor/mxEditor-js.html, acedido a última vez em 10 de Junho de 2018.