Relatório de projeto

Mestrado em Engenharia Informática - Computação Móvel

Gerador automático de configurações de equipamentos de rede em ambientes virtualizados.

Rodrigo José Lopes Emiliano

Leiria, Setembro de 2015

Relatório de projeto

Mestrado em Engenharia Informática - Computação Móvel

Gerador automático de configurações de equipamentos de rede em ambientes virtualizados.

Rodrigo José Lopes Emiliano

Dissertação de Mestrado realizada sob a orientação do Doutor Mário João Gonçalves Antunes, Professor da Escola Superior de Tecnologia e Gestão do Instituto Politécnico de Leiria.

Leiria, Setembro de 2015

À família e amigos, pelo apoio no decorrer desta etapa.

Agradecimentos

Agradecimento à família e amigos, pela compreensão e apoio que demonstraram nesta etapa. Ao Professor Doutor Mário Antunes, pelo apoio, compreensão e orientação ao longo deste caminho.

Resumo

A Internet é composta por vários equipamentos interligados entre si, desde *routers*, *switches* e servidores até aos computadores e dispositivos utilizados para lhe aceder. Todos os ambientes onde é necessária troca de informação ou ligação com o meio exterior utilizam equipamentos de rede especializados para essas funções. Existem os *routers* e *switches* de alto desempenho que são utilizados em ambientes empresariais com centenas de utilizadores e um débito de informação e de dados elevado, mas também os *routers-modem*, que são utilizados nos ambientes domésticos onde só se necessita de uma ligação à Internet. Estes equipamentos têm funções específicas e necessitam de ser configurados de acordo com o seu papel na rede, como encaminhamento de pacotes, disponibilização de serviços, ligação com utilizadores, entre outras funções.

A existência de um grande número de equipamentos de rede em ambientes de média e grande dimensão traduz-se numa repetição de aspetos na configuração dos mesmos, quando estes necessitam ser configurados. A repetição constante na realização de configurações aumenta então a possibilidade de erro humano, devido a distrações causadas por esta repetição. Também existem *routers* e outros equipamentos de rede de diferentes modelos e marcas, o que implica um vasto conhecimento do funcionamento dos equipamentos por parte das pessoas que os configuram.

A repetição de comandos na implementação e configuração de redes acontece principalmente em ambientes educativos e empresariais, com a realização de cenários de teste, onde também se recorrem a redes simuladas ou virtualizadas por aplicações. É importante então automatizar o processo de escrita de configurações repetitivas em ambientes normais e heterogéneos, de modo a diminuir o erro humano na realização destas tarefas e a potenciar a produtividade do utilizador na realização de outras tarefas que requerem maior atenção.

Esta dissertação propõe uma solução na forma de uma aplicação protótipo que gera configurações para equipamentos de rede. A aplicação desenvolvida gera configurações para os *routers* que existam numa dada topologia desenhada com recurso ao GNS3, uma aplicação de desenho e virtualização de cenários de rede, de modo a colocar a topologia a comunicar.

Para a geração de configurações, a aplicação utiliza o ficheiro de especificação no formato JSON, que é criado pelo GNS3. Ao interpretar o ficheiro JSON, a aplicação obtém a informação dos *routers* e das suas ligações e gera as configurações de acordo, com recurso a *templates*. Os *templates* permitem a adição de diferentes configurações e o suporte a vàrios modelos e marcas de *routers*, havendo um *template* para cada configuração e marca de *router* diferente.

Após o desenvolvimento do protótipo da aplicação foram realizados testes funcionais, de desempenho e de carga. Estes testes revelaram que é possível desenvolver uma aplicação preparada para a heterogeneidade e que gera configurações para *routers* virtualizados e reais de forma simples e rápida, com recurso a ficheiros de projeto gerados no GNS3.

Abstract

The Internet is composed of various equipments connected with one another, from routers, switches and servers to the computers and devices used to access it. All environments where it is necessary to exchange information or to connect with the exterior use network equipments specialized to do those functions. There are high performance routers and switches that are used in corporate environments with hundreds of users and high information and data output, but also modem-routers, that are used in home environments where there is only the need of access to the Internet. These equipments have specific functions and need to be configured according to their role in the network, like packet forwarding, service provisioning, user connection, among other roles.

The existence of a large number of network equipments in medium and large scale environments involves a repetition of configurations, when those equipments need to be configured. The constant repetition of configurations increases the possibility of human errors, due to the distractions caused by this repetition. There are also routers and other network equipments from different manufacturers and models, which implies a great level of knowledge of the equipments by the people who configure them.

The repetition of commands in the implementation and configuration of networks happens mainly in corporate and learning environments, by carrying out test scenarios, which also use application simulated or virtualized networks. It's important to automate the process of writting repetitive configurations in normal and heterogeneous environments, in order to decrease human errors in these tasks and to increase user productivity in more complex tasks.

This dissertation proposes a solution in the form of a prototype application that generates configurations for network equipments. The application generates configurations for routers that exist in a given topology designed with GNS3, an application for the design and virtualization of network scenarios, in order to have the topology communicating.

For the generation of configurations, the application uses the specification file in JSON format, that is created by GNS3. By interpreting the JSON file, the application gathers information regarding routers and their connections and generates the configurations accordingly, using templates. The templates allow the adition of different configurations and the support of different manufacturers of routers, by having a template for each configuration and router manufacturer.

After developing the prototype of the application, functional, performance and load tests were carried out. These tests proved that it is possible to develop an application that is

prepared for heterogeneity and that generates configurations for real and virtualized routers, using project files generated by GNS3

Conteúdo

Lista de Tabelas

Lista de Figuras

Acrónimos

ADSL Asymmetric Digital Subscriber Line

ALU Alcatel-LUcent

BIOS Basic Input/Output System

CLI Command Line Interface

CPAN Comprehensive Perl Archive Network

CPU Central Processing Unit

DCE Data Communication Equipment

DTE Data Terminal Equipment

GNS Graphical Network Simulator

GPL GNU Public License

GUI Graphic User Interface

ICCSE International Conference of Computer Science and Education

IMDb Internet Movie Database

IOS Internetwork Operating System

IP Internet Protocol

JSON JavaScript Object Notation

NCG Netomata Config Generator

NVRAM Non-Volatile Random Access Memory

eNSP Network Simulation Platform

OSI Open Systems Interconnection

OSPF Open Shortest Path First

PC Personal Computer

POO Programação Orientada a Objetos

QEMU Quick EMUlator

RAM Random Access Memory

ROM Read Only Memory

RIP Routing Information Protocol

SO Sistema Operativo

TCP/IP Transmission Control Protocol/Internet Protocol

XML eXtensible Markup Language

Capítulo 1

Introdução

A Internet é uma rede à escala global, constituída por um vastíssimo conjunto de redes interligadas através de equipamentos específicos como *routers* e *switches*, com arquiteturas diferentes e de fabricantes distintos. Estes equipamentos são configurados geralmente de forma estática e de acordo com a topologia física e lógica da rede onde operam. A sua configuração tem igualmente em consideração os protocolos, aplicações e serviços TCP/IP (*Transmission Control Protocol/Internet Protocol*)onde os equipamentos irão operar e os serviços que irão suportar. A configuração tem de ser otimizada e adequada para que seja alcançado o nível de serviço esperado e previamente definido. As rotas de encaminhamento entre estes equipamentos podem ser automaticamente configuradas através de protocolos de encaminhamento dinâmico. No entanto, mesmo nessas circunstâncias, há lugar a configuração estática e manual efetuada pelo administrador de sistemas.

Uma alteração estática à topologia da rede implica necessariamente alterações à configuração dos equipamentos de rede envolvidos. Por exemplo, a inserção de uma nova placa de rede num router implica necessariamente que o administrador da rede proceda à configuração deste novo componente. Para o efeito, o administrador da rede deverá aceder à consola do equipamento e, através da linha de comandos disponibilizada para o efeito, executar os comandos necessários para que a nova configuração tome efeito. Por outro lado, algumas alterações à topologia da rede podem ser automaticamente descobertas pelos outros equipamentos existentes na rede, podendo nalguns casos traduzir-se numa atualização automática da configuração de cada um dos equipamentos. Um exemplo concreto é o uso de protocolos de encaminhamento dinâmico como o OSPF (*Open Shortest Path First* ou o RIP (*Routing Information Protocol*), que podem ser configurados para automaticamente descobrir novas rotas existentes para redes IP destino ou remover outras que tenham sido desativadas devido a uma alteração da topologia.

As empresas têm necessidade de efetuar vários testes de configurações, com vista a garantir que as mesmas terão os efeitos desejados num ambiente de produção. As alterações devem ser realizadas seguindo um conjunto de boas práticas, no sentido de garantir o sucesso da configuração e implementação em produção [1]. Uma dessas boas práticas na configuração de equipamentos consiste em testar em ambiente controlado as alterações que se pretendem implementar num ambiente de produção. Nesse sentido, podem realizar-se várias experiências e testes de modo a decidir sobre a eficácia das alterações que se pretendem implementar e, na medida do possível, ter uma noção aproximada do seu impacto num ambiente de produção. Estes testes podem ser realizados num laboratório com equipamentos reais, mas é cada vez mais frequente recorrer a ambientes virtualizados, com o uso de aplicações específicas para o efeito, como o GNS3 (`http://www.gns.com/`) (*Graphical Network Simulator*) e o Huawei eNSP simulator (`http://www.huawei.com/`) (*Network Simulation Platform*). Além de evitar a aquisição de equipamentos exclusivamente para testes, o recurso a aplicações de virtualização permite a integração de um vasto conjunto de equipamentos de redes distintas.

O teste de cenários de configuração de rede implica normalmente a realização de várias tarefas de configuração, num vasto conjunto de equipamentos, normalmente numa rede heterogénea onde os equipamentos não pertencem todos ao mesmo fabricante. A repetitividade destas tarefas, aliada à necessidade de dominar vários sistemas operativos, implica um aumento significativo do tempo gasto na configuração dos cenários de teste e por vezes a definição de configurações erradas ou pouco otimizadas, devido essencialmente a erros humanos. Tal deve-se à necessidade da escrita sistemática de comandos nos equipamentos através da linha de comando, ou CLI (*Command Line Interface*) dos equipamentos de rede. O uso do CLI para a configuração dos equipamentos implica normalmente morosidade na edição dos comandos, agravada pela necessidade de os repetir diversas vezes no mesmo equipamento (ou em equipamentos diferentes). Por exemplo, num router com 4 interfaces de rede, a configuração básica de IP terá de ser repetida em cada uma delas.

O nível de heterogeneidade da rede está diretamente relacionado com o grau de conhecimentos técnicos que os administradores da rede terão de possuir para conseguir lidar com a configurar de todos os equipamentos existentes. O aumento da heterogeneidade da rede implica necessariamente técnicos mais habilitados e com mais experiência em vários sistemas operativos.

Os desafios apresentados anteriormente, nomeadamente a necessidade de configurar cenários de teste que implicam lidar com a repetição de comandos de configuração em redes heterogéneas, realçam a necessidade de definir mecanismos eficientes que permitam a automatização total ou parcial dessas configurações, tentando assim minimizar erros humanos

e configurações erradas ao nível lógico da configuração da rede. Esta necessidade pode ser observada especialmente em empresas com redes de média/grande dimensão, com requisitos específicos de configuração.

Não é apenas nas empresas onde estes desafios são evidentes. O ensino das redes de computadores existe em várias instituições de ensino, desde as que ministram cursos técnicos e profissionais, até às de ensino superior que permitem a obtenção de um grau académico [2]. Nas aulas práticas de ensino das redes de computadores, existem várias situações onde é necessária a preparação de cenários de teste, com vista a proporcionar uma melhor experiência aos alunos e uma melhor compreensão dos conceitos que se pretendem transmitir numa determinada matéria.

À semelhança do que acontece nas empresas, os ambientes laboratoriais existentes nas instituições de ensino superior podem ser de três tipos principais: terem unicamente equipamentos físicos (*routers* e *switches*); usando exclusivamente aplicações de virtualização e/ou simulação, como o Packet Tracer ou o GNS3; terem ambientes virtuais interligados com equipamentos físicos. Os laboratórios com equipamentos físicos existem um pouco por todas as instituições de ensino, podendo integrar um número variável de equipamentos. Podemos constatar esse facto através da dispersão geográfica das academias Cisco e de outros fornecedores, onde as instituições de ensino adquirem configurações de equipamentos já definidas para o ensino dos conteúdos e a posterior certificação de conhecimentos [3].

As aplicações de virtualização são mais recentes e têm várias vantagens, comparativamente com os laboratórios de equipamentos físicos. Em primeiro lugar, permitem criar maior autonomia aos praticantes de redes de computadores na definição dos cenários de rede, não havendo necessidade de adquirir equipamentos físicos. De seguida, a escalabilidade dos cenários aumenta, permitindo definir redes de teste mais complexas e de maior dimensão. A limitação para a definição dos cenários de rede é essencialmente a configuração física, como o CPU (*Central Processing Unit*) e memória RAM (*Random Access Memory*), dos computadores onde as aplicações de virtualização estão instaladas. Por último e relacionado com as anteriores, o uso de ambientes virtuais para testes melhora o ritmo de aprendizagem dos protocolos e configurações de redes, facilita a reprodução dos testes realizados em cada cenário e permite facilmente testar configurações alternativas em cenários previamente configurados.

Estes cenários de teste são muito importantes devido ao acesso limitado a equipamentos físicos fora da empresa ou da escola [2]. Neste tipo de aulas práticas, do tipo "hands-on", é esperado que os alunos interajam com novas tecnologias e que aprendam a manuseá-las. Estas aulas requerem uma preparação da sala de aula, do cenário de teste e de todos

os equipamentos que irão ser usados. Especificamente nos conteúdos básicos de redes de computadores é necessária a retenção dos comandos fundamentais de configuração dos equipamentos, que são usados mais frequentemente. À medida que aumenta a experiência no uso destes conceitos básicos, são introduzidos novos comandos e métodos de configuração, mais complexos e específicos para alguns cenários. Por exemplo, para configurar uma funcionalidade avançada da rede, como a afinação dos parâmetros relativos à qualidade de serviço, é necessário proceder aos passos essenciais da configuração básica dos vários *routers* da rede, como a configuração básica de IP (*Internet Protocol*) em todas as interfaces e a configuração de protocolos dinâmicos de encaminhamento.

Assim, atendendo à dimensão das redes de teste e devido à frequência da realização desses mesmos testes em redes experimentais e da possível complexidade dos cenários de teste, muito trabalho é repetido na preparação e configuração dos equipamentos, havendo várias sequências de comandos que poderiam ser gerados automaticamente. Por exemplo, as interfaces são sempre configuradas no equipamento, tais como as rotas e protocolos de encaminhamento. Estes são alguns exemplos de configurações que poderiam ser automatizadas sempre que se pretende configurar um cenário de teste. É possível então perceber que existe um desafio a superar na configuração dos equipamentos, não só nos ambientes reais mas também nos virtualizados, que é a constante repetição de tarefas e comandos.

O repetitivo e moroso processo de configuração destes equipamentos está sempre sujeito a erros que poderiam ser evitados através de uma abordagem automática, que minimizasse a intervenção humana e o tempo dispendido na configuração dos equipamentos. O desafio da geração automática de configurações de equipamentos já tem sido estudado e, nesse sentido, existem várias soluções disponíveis. No entanto a maior parte das soluções estudadas apresentam ainda limites em termos de escalabilidade, adaptabilidade, custo e heterogeneidade nos equipamentos suportados. O trabalho realizado, conducente a esta dissertação, assenta no desenvolvimento de uma aplicação para gerar automaticamente ficheiros de configuração de equipamentos de routing de uma rede. Ou seja, foi apenas contemplada a configuração de *routers*, responsáveis por processar os pacotes ao nível da camada de rede (IP). Os cenários poderão incluir switches, mas a sua configuração não será abordada neste relatório.

1.1 Objectivos

O trabalho conducente à realização desta dissertação atinge os seguintes objetivos inicialmente propostos, designadamente:

- Instalar e testar um conjunto de soluções já existentes para a configuração automática de equipamentos de rede. Foram investigadas algumas aplicações existentes, de modo a perceber quais as funcionalidades deveriam ser consideradas e que novas funções poderiam ser incluídas. Para isso foram identificados os seus pontos fortes e fracos, determinando assim as características a manter e a melhorar na configuração dos *routers* numa rede.

- Avaliar as soluções, reconhecendo os seus pontos fortes e fracos. Esta comparação entre as várias soluções permitiu a criação de uma aplicação que inclui as principais vantagens observadas durante a investigação de aplicações semelhantes e que se adicionasse outras funcionalidades não existentes nas soluções observadas.

- Definir os requisitos principais da aplicação para a configuração automática de equipamentos de rede de vários vendedores. Estes requisitos resultam dos pontos fortes observados nas outras aplicações, complementando as funcionalidades em falta nas mesmas.

- Definir uma linguagem de especificação escalável e configurável para dar suporte à aplicação desenvolvida, tornando-a mais simples de utilizar e compreender. Esta linguagem servirá para a definição do *input* da aplicação, que servirá como base da geração de configuração.

- Desenvolver um protótipo da aplicação e acomodar a sua linguagem ao objetivo de heterogeneidade e escalabilidade, suportando *templates*. Estes *templates* permitem adicionar novas funcionalidades de diferentes construtores.

- Testar a aplicação através da implementação de vários cenários de configuração e analisar os resultados obtidos nos aspetos de integração de vários construtores de *routers*, complexidade da rede e complexidade da configuração. Estes testes servem para fortalecer a aplicação e observar o seu comportamento face a diferentes topologias e equipamentos.

- Apresentar um protótipo de um ambiente gráfico para configuração da rede, através da integração com o GNS3.

Foi desenvolvida uma aplicação protótipo como prova de conceito. O protótipo interage com a aplicação GNS3 e, ao interpretar o ficheiro na linguagem de especificação JSON (*JavaScript Object Notation*), gera automaticamente os ficheiros de configuração para os *routers* existentes na rede. A aplicação desenvolvida é *opensource*, de fácil utilização, e está disponível em `http://rcg-tool.sourceforge.net/`.A aplicação permite a configuração

automática de redes de grande dimensão, que pode ter equipamentos de rede de diferentes vendedores. O desenvolvimento realizado até ao momento contemplou os *routers* Cisco C2691 e os Alcatel-Lucent vSR-OS. Para facilitar o processo de desenho, configuração e teste da rede, a aplicação desenvolvida pode integrar-se com o GNS3, designadamente ao nível do desenho gráfico da rede e associação dos ficheiros de configuração aos equipamentos virtuais aí configurados. Pode igualmente ser executada autonomamente, permitindo o carregamento dos ficheiros de configuração nos equipamentos interligados no cenário.

A aplicação foi desenvolvida com a finalidade de integrar facilmente com o GNS3. A principal vantagem consiste na possibilidade de associar as facilidades gráficas e de organização dos equipamentos virtualizados do GNS3 à conversão automática das configurações do GNS3 para ficheiros de configuração dos respetivos equipamentos. Permite ainda tirar partido da escalabilidade que é possivel alcançar através deste ambiente virtual de configuração de rede, dando a possibilidade de definir cenários com redes de grande dimensão e com equipamentos de diferentes vendedores, bem como testar os ficheiros de configuração nesses mesmos cenários.

O processamento levado a cabo pela aplicação recebe dois tipos de entradas: configurações do GNS3, como por exemplo tipo de router e correspondente imagem do sistema operativo que será utilizada; configurações de execução, onde o utilizador pode definir vários parâmetros que condicionam a forma como a aplicação gerará os ficheiros de configuração. Por exemplo, o utilizador pode especificar o protocolo de encaminhamento que será utilizado para configurar a rede (nesta fase foram implementados os protocols OSPF e RIP) e as configurações específicas a implementar. A possibilidade de configurar redes heterogéneas é conseguida através do uso de *templates*, um para cada característica de configuração que se encontra implementada, por cada tipo de equipamento. Por exemplo, haverá um *template* para a configuração de protocolos de encaminhamento, um para cada fabricante suportado.

O uso de *templates* permite assim que não seja necessário alterar o código fonte da aplicação que está a ser usada para acomodar essas novas funcionalidades, bastando apenas adicionar um novo *template*. O uso de *templates* também vai ao encontro de outra motivação da tese, que consiste na possibilidade de configurar automaticamente redes heterogéneas. Como referido anteriormente, as redes não são constituídas por uma só marca de equipamento. Existem várias marcas e modelos para realizar a mesma tarefa, o que torna importante o suporte de vários construtores de equipamentos de rede por parte da solução proposta. Assim, os cenários reais existentes podem todos ser configurados automaticamente.

Após o desenvolvimento da aplicação foram realizados os seguintes testes: de **carga**, onde se testava a capacidade de geração de muitas configurações para uma rede de grande dimen-

são, testando a robustez da aplicação e do GNS3; de **tempo de execução**, onde se verificou a variação do tempo de execução da aplicação conforme a quantidade de *routers* na topologia; **funcionais**, que verificam o correcto output após a execução da aplicação. O resultado dos testes revelaram a eficiência da aplicação na geração automática de configurações base para *routers* em redes heterogéneas. É possível identificar que a configuração produzida assegura a comunicação de todos os equipamentos da rede. Usando a aplicação desenvolvida, é possível iniciar os equipamentos com uma configuração base, permitindo a execução de novos comandos para acomodar configurações mais complexas e específicas.

1.2 Contribuições

Do trabalho realizado resultaram as seguintes contribuições:

- Desenvolvimento de um protótipo de aplicação que gera automaticamente configurações para equipamentos de rede com base num ficheiro JSON desenhado no GNS3. As configurações podem ser carregadas em equipamentos reais ou virtualizados.
- Disponibilização da aplicação no serviço de partilha de aplicações e código *Sourceforge*. O *Sourceforge* é um serviço muito utilizado por equipas de desenvolvimento de software para partilha e divulgação de projetos e código *open-source*. A aplicação pode ser acedida no link `http://rcg-tool.sourceforge.net/`.
- Desenvolvimento de um tutorial para integração da aplicação desenvolvida no GNS3.
- Publicação de um artigo na conferência 10^{th} IEEE International Conference on Computer Science and Education (ICCSE) 2015, realizada em Cambridge, Inglaterra, premiado com uma menção de '*Best Student Paper Award*'.
  - Rodrigo Emiliano, Mário Antunes; *"Automatic network configuration in virtualized environment using GNS3"*; 10^{th} IEEE International Conference on Computer Science & Education (ICCSE) 2015; pp. 25-30; Editor: IEEE; Julho 2015; DOI: 10.1109/*ICCSE*.2015.7250212; Cambridge, UK

1.3 Estrutura da tese

O documento está estruturado da seguinte forma:

- No capítulo 2 apresentam–se algumas noções fundamentais que permitirão uma melhor compreensão do resto do documento. De seguida são apresentadas as aplicações de configuração automática existentes e realizada uma comparação entre elas, tirando algumas conclusões.
- O capítulo 3 é dedicado à descrição da solução implementada, designadamente a arquitetura geral, a descrição dos principais componentes e a sua interligação, bem como as decisões tomadas.
- No capítulo 4 descreve-se o desenvolvimento realizado, nomeando as principais escolhas tomadas e descrevendo os processos mais importantes da aplicação.
- No capítulo 5 são tratados os testes realizados à aplicação e as conclusões dos mesmos.
- No capítulo 6 é dada a conclusão sobre o trabalho realizado e delineado o trabalho futuro.
- Os anexos estão organizados da seguinte forma:
  - Anexo A.1: exemplo de ficheiro de configurações JSON gerado automaticamente pelo GNS3.
  - Anexo A.2: exemplo de ficheiro de configuração base dos routers Cisco.
  - Anexo A.3: código fonte da função `createConfiguration()`.

Capítulo 2

Estado da arte

Este capítulo inicia-se com a apresentação de alguns conceitos fundamentais relacionados com a temática das redes de computadores e com os métodos e aplicações existentes de configuração e teste dos equipamentos de rede, especificamente os *routers*. De seguida são abordadas algumas aplicações relacionadas com o desenho e configuração automática de redes, tanto em cenários reais como em ambientes simulados. As aplicações analisadas são: *Netomata Config Generator* [1], *Solarwinds Network Config Generator*[2], GEN-IT[3] e *AutoNetKit*[4]. Estas aplicações serviram de base para a definição da aplicação proposta e desenvolvida no âmbito da realização da dissertação [4]. No final deste capítulo é apresentada uma tabela comparativa das aplicações, tendo por base um conjunto de critérios definidos antecipadamente. Com base nessa tabela comparativa são apresentadas algumas conclusões ao funcionamento das aplicações, tendo em conta os seus pontos fortes e fracos.

2.1 Conceitos fundamentais

Existem vários conceitos fundamentais necessários à compreensão do tema da configuração automática de equipamentos de rede, que irá ser abordada nos próximos capítulos da dissertação. De modo a obter um melhor foco do objetivo da dissertação, foram identificados alguns pressupostos a ter em conta. A topologia de redes a ser configurada automaticamente é do tipo TCP/IP. Uma vez que o acesso a variados equipamentos de rede é por vezes limitado, é dado foco aos ambientes virtualizados, que são cada vez mais utilizados em cenários

[1] `http://www.netomata.com/tools/ncg/`

[2] `http://www.solarwinds.com/products/freetools/network-config-generator/`

[3] `http://gen-it.net/`

[4] `http://autonetkit.org/`

de teste e de aprendizagem, devido ao baixo custo da sua utilização e da sua disponibilidade a todos os utilizadores. A camada do modelo TCP/IP considerada na dissertação é a camada de rede, que foca maioritariamente o roteamento de pacotes pela rede usando *routers* como pontos principais na sua distribuição. Os *routers* ganham portanto o foco na configuração automática, ao contrário dos *switches*, que não são considerados neste trabalho.

Tendo em conta os objetivos delineados na resolução desta dissertação, é necessário perceber quais os principais intervenientes nas redes e a sua função, e o porquê da necessidade de estes serem configurados automaticamente, principalmente em cenários laboratoriais e de teste.

2.1.1 Routers

Os *routers* são equipamentos de interligação de redes IP e operam ao nível da camada de rede do modelo protocolar TCP/IP. É fundamental para um engenheiro de redes saber como configurar um router para obter um determinado comportamento na rede. É igualmente mandatório para todos os estudantes que frequentam cursos técnicos, profissionais e de grau académico o manuseamento destes equipamentos e a sua operação em ambiente laboratorial [5]. A principal função de um *router* é a ligação de diferentes redes IP entre si de modo a que estas comuniquem, uma vez que numa rede sem *router* só é possível obter ligação entre equipamentos que fazem parte dessa rede IP. Cada interface existente num *router* tipicamente está confinada numa rede IP, havendo uma rede diferente por cada interface. A necessidade de existência de um ou mais *routers* numa rede é definida pelo objetivo da rede em geral e com o tamanho da rede em si. Para existir a interligação entre os componentes das várias redes IP existentes numa rede, os *routers* necessitam ser configuráveis de modo a que possa ser realizada uma gestão mais eficaz de todos os elementos na rede [1].

A figura 2.1 ilustra uma simples rede doméstica que contém ou *router modem* utilizado pelos diversos aparelhos para comunicarem entre si e para fazer a ponte com a Internet.

Este tipo de *router* é utilizado para realizar a interacção entre os seus utilizadores e a Internet através de uma tecnologia que pode ser ADSL (*Asymmetric Digital Subscriber Line*) ou Fibra. Usualmente, estes *routers* permitem a ligação de vários aparelhos, seja por cabo *Ethernet* ou por ligação sem-fios. Este *router* foi concebido para realizar a tarefa de ligação entre redes domésticas e a Internet, dispondo de várias tecnologias específicas para este tipo de ligação. No entanto existem *routers* que apenas realizam o encaminhamento de pacotes na rede e não estão ligados directamente aos utilizadores. Estes *routers* têm como objetivo a recepção, processamento e envio rápido da informação, sendo utilizados em ambientes

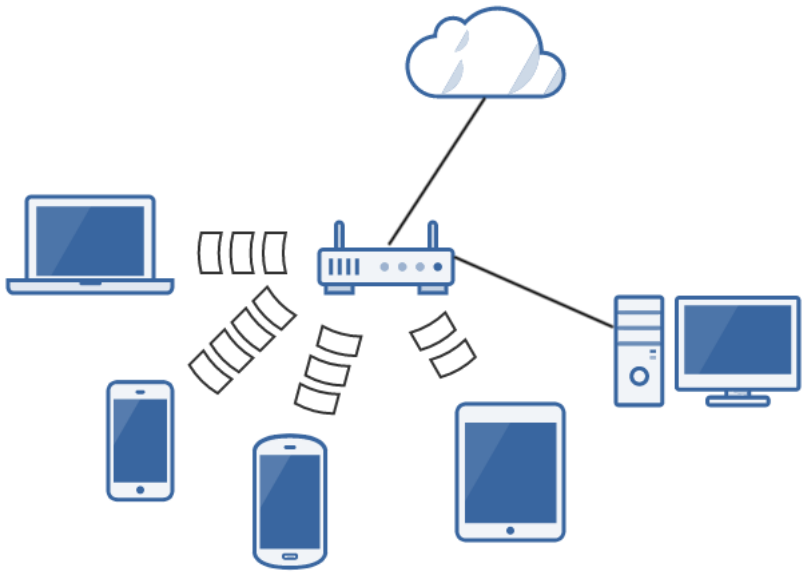

Figura 2.1: Exemplo de uma rede doméstica.

empresariais onde a rapidez na comunicação é expectável. A figura 2.2 mostra um exemplo de cenário que permite ter a ideia da forma como os *routers* são utilizados em ambientes empresariais.

É possível observar na figura 2.2 que os *routers* não comunicam diretamente com os PCs (*Personal Computer*) da rede, mas são utilizados na segregação das redes. Nas empresas, por motivos de organização e segurança, as redes são separadas de forma consciente, tipicamente para separar redes de acordo com a função dos seus utilizadores e retirar alguma visibilidade ao exterior. Na figura é possível observar essa mesma divisão, estando a rede geral subdividida em várias redes IP, onde quatro delas têm funções relevantes. Existe um *router* central que funciona como o centro da rede. Normalmente, esta zona na rede, que pode ser constituída por um ou mais *routers* é denominada por *backbone* ou *core* da rede. É nesta zona onde os vários *routers* intermédios se ligam e onde é realizada a separação da rede pela sua função. A rede à esquerda do *router* central tem como objetivo concentrar todos os servidores da empresa. Esta rede é separada uma vez que o acesso a este tipo de componentes em empresas é altamente condicionado e monitorizado. A rede abaixo do *router* central é considerada a rede *wireless* da empresa, onde os colaboradores e visitantes se podem ligar para navegar na Internet. Este tipo de redes normalmente é

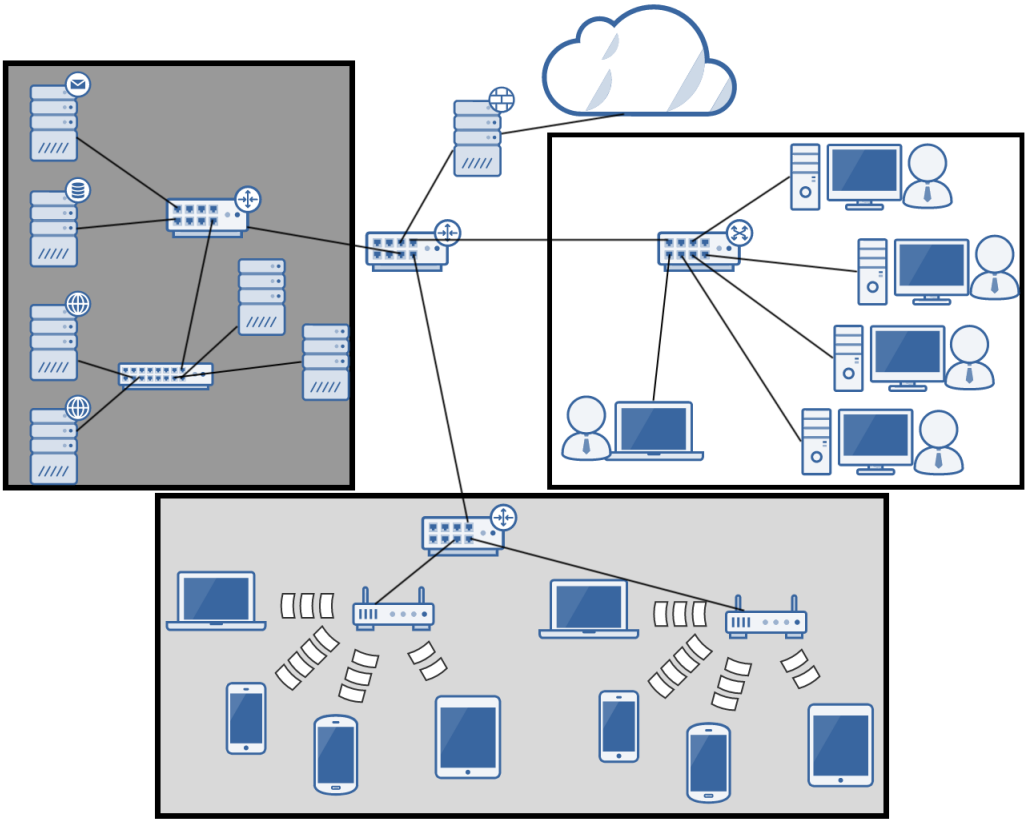

Figura 2.2: Exemplo de uma rede empresarial.

configurada de modo a que os seus utilizadores não tenham acesso a mais nenhuma rede interna que não a rede que faz a ligação à Internet. A rede à direita do *router* central é a rede interna da empresa, utilizada pelos colaboradores da empresa para realizarem as suas tarefas. Nesta rede geralmente os utilizadores têm acessos mais privilegiados a recursos da empresa, portanto é realizado algum tipo de controlo de acesso à mesma.

Dependendo das políticas de segurança e do tamanho da empresa, as redes internas podem estar também segregadas pelas tarefas que os utilizadores realizam dentro da empresa, havendo mais mecanismos de segurança em acção para restringir o acesso. Esta segregação é realizada com recurso aos *routers*, através da realização de diferentes configurações, e também a outro tipo de *hardware* e *software*, para melhor gestão de acessos. Os *routers* são então equipamentos complexos que permitem a realização das mais variadas tarefas, seja de separação de redes mas também no controlo de acesso às mesmas e aplicação de mecanismos de segurança.

Para melhor compreensão do funcionamento de um *router*, na Figura 2.3 é observada a sua arquitetura.

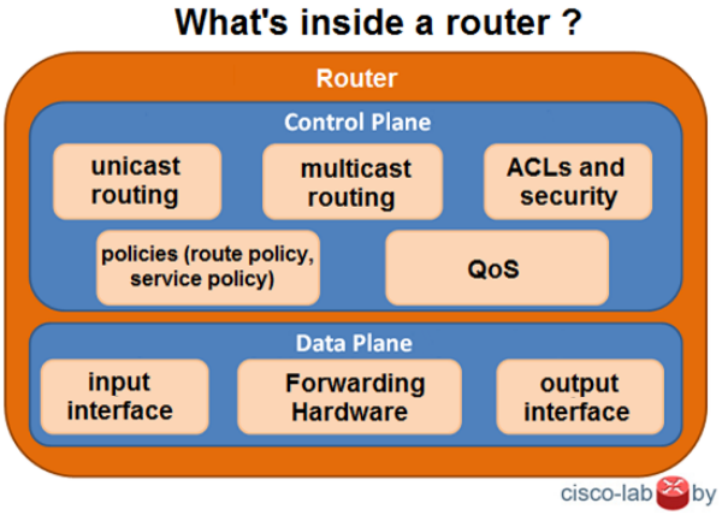

Figura 2.3: Arquitectura de um *router* (`http://cisco-lab.net/`)

Como é possível observar, um *router* tem dois componentes principais: *control plane* e *data plane*. O *control plane* funciona como cérebro do *router*. Este é responsável especialmente pela construção das tabelas de encaminhamento e políticas de segurança. Todos os mecanismos que afetem o comportamento da gestão de tráfego funcionam ao nível do *control plane*. O *data plane* é responsável pela receção e envio de tráfego, aplicando as regras definidas no *control plane* [6].

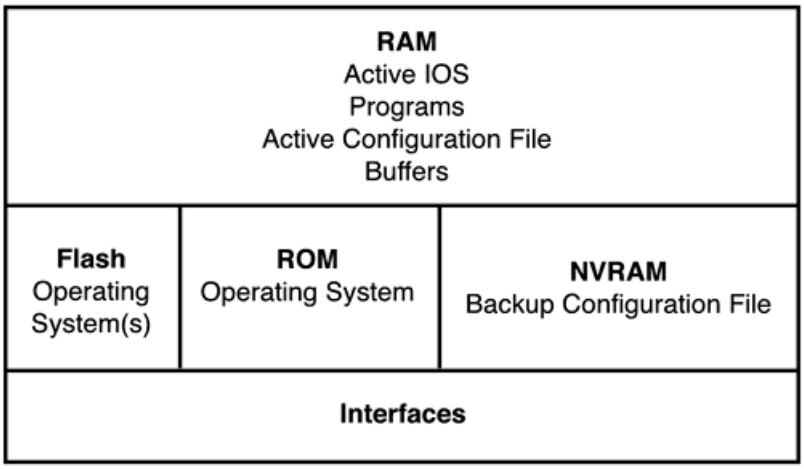

Figura 2.4: Componentes de um *router* Cisco

A figura 2.4 ilustra os componentes de memória de um *router* Cisco. Este componentes são usados para armazenar os dados a utilizar pelo *router* e cada um tem uma função específica. A RAM guarda as configurações em uso do *router* e dados como os pacotes a

serem processados e usados durante o funcionamento do *router*. A RAM também representa a memória não permanente e volátil do *router*. Ou seja, caso o *router* se desligue, todos os dados armazenados na RAM são perdidos.

A memória *flash* armazena o sistema operativo em uso pelo *router*. Trata-se de uma memória não volátil e que pode ser modificada, permitindo a atualização do sistema operativo a ser utilizado, à semelhança das atualizações de BIOS (*Basic Input/Output System*) realizadas normalmente nos computadores. A memória ROM (*Read Only Memory*) em *routers* Cisco mais antigos era utilizada para armazenar o sistema operativo em utilização, porém como este atualmente é armazenado na memória flash, a ROM é utilizada no arranque do *router*. Esta armazena então elementos que são utilizados durante o arranque do *router*, que auxiliam na verificação do correto funcionamento dos componentes presentes no *router*. A NVRAM (*Non-Volatile Random Access Memory*) tem como principal função armazenar a configuração de arranque do *router*. Sempre que o *router* arranca, todas as configurações presentes na NVRAM são carregadas para a RAM e entram em funcionamento. Esta memória não é volátil, o que significa que qualquer alteração realizada na configuração e consequentemente na RAM durante o funcionamento do *router* deverá ser transferida para a memória NVRAM de modo a que esta não seja perdida caso o *router* se desligue [7][8].

Todos os sistemas operativos de *routers* permitem portanto transferir as configurações realizadas na RAM para a NVRAM. A solução da criação automática de configurações para *routers* irá abordar as memórias RAM e NVRAM, uma vez que estas estão intrinsecamente relacionadas com a configuração do *router*.

2.1.2 Command Line Interface (CLI)

Os routers podem ser configurados através de duas formas principais: estatica ou dinâmica. A configuração estática está normalmente relacionada com a execução de comandos numa linha de comandos, que permite a interação com o sistema operativo. Este tipo de configuração pode também ser feita através de uma interface web disponibilizada para o efeito. Por exemplo, os routers domésticos instalados pelos operadores de telecomunicações nas casas dos clientes, têm normalmente apenas uma interface web para configurações essenciais do router [9].

A diferença do CLI face às aplicações gráficas (*Graphical User Interface*, ou GUI), é precisamente o nível de abstração que o utilizador tem ao configurar um equipamento. Através de uma GUI o utilizador interage com o router através da informação que é colocada numa inferface gráfica amigável. Pela manipulação dessa informação com o auxílio de

menus, *checkboxes*,*radio buttons*, entre outras formas, a configuração torna-se de mais alto nível, o que permite ao utilizador realizar mais alterações num tempo menor, eliminando a necessidade da escrita de vários comandos para realizar passos que podem ser feitos com um clique num ícone [9]. Estes pequenos passos são depois interpretados e alojados no ficheiro de configuração do router, para depois serem desencadeados para a concretização das alterações efetuadas. No entanto, apesar da amigabilidade da interface gráfica, o uso da GUI não permite automatizar tarefas de configuração. Ou seja, não permite replicar configurações semelhantes em vários equipamentos, quando por exemplo apenas são alterados poucos parâmetros [9].

Normalmente, um *router* ou *switch* é altamente configurável, existindo centenas de comandos para realizar as configurações. Numa rede de média ou grande dimensão, existem dezenas de aparelhos que fazem o encaminhamento de dados pela rede. Geralmente, todos estes equipamentos necessitam de ser configurados através da CLI. Numa configuração normal existem vários comandos fundamentais para configuração, desde o nome do *router* às várias interfaces e rotas de encaminhamento. Além destes comandos fundamentais, devido à complexidade da rede, existem contextos específicos de configuração e comandos específicos para a sua realização, como a configuração dos protocolos de encaminhamento e de ACLs (*ACcess Lists*). Algumas destas configurações são possíveis através de interfaces *Web*, embora só contenham alguns subconjuntos de comandos. A figura 2.5 mostra um exemplo de configuração simples de uma pequena rede funcional com *routers* Cisco, que usa o sistema operativo IOS e protocolo de encaminhamento OSPF.

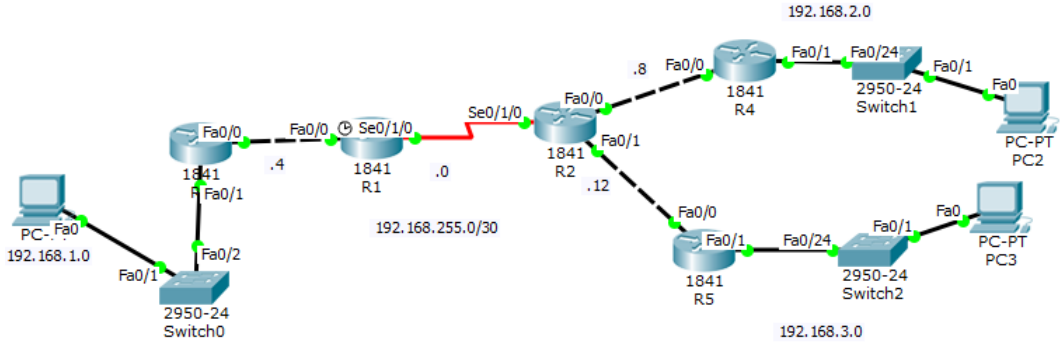

Figura 2.5: Exemplo de uma rede de pequena escala

Neste exemplo é necessário começar por configurar o endereçamento IP nas interfaces. De seguida deve definir-se uma estratégia para configuração do encaminhamento, que pode ser estático ou dinâmico. Se se optar por encaminhamento dinâmico é necessário iniciar a divulgação das redes IP nas quais cada *router* está configurado. Neste caso concreto, o encaminhamento é dinâmico e foi usado o protocolo OSPF (*Open Shortest Path First*). Este

protocolo permite definir áreas para distinção de zonas de divulgação de rotas, embora neste caso se tenha considerado que todas as redes IP anunciadas pertencem á mesma área do OSPF.

Em termos práticos, atendendo a que neste exemplo existem 5 *routers*, sendo que a configuração obrigatória inclui 10 comandos, significa que o número mínimo de comandos para configurar a rede será de pelo menos 50. Para a rede do exemplo, os comandos IOS obrigatórios utilizados para configuração do *router* R2 estão descritos nas figuras 2.6 e 2.7.

```
Router>enable
Router#configure terminal
Router(config)#interface fastEthernet 0/0
Router(config-if)#ip address 192.168.1.254 255.255.255.0
Router(config-if)#no shutdown
Router(config-if)#exit
```

Figura 2.6: Sequência de comandos IOS para configuração de interface.

```
Router>enable
Router#configure terminal
Router(config-router)#network 192.168.255.0 0.0.0.3 area 0
Router(config-router)#network 192.168.255.8 0.0.0.3 area 0
Router(config-router)#network 192.168.255.12 0.0.0.3 area 0
Router(config-router)#exit
```

Figura 2.7: Sequência de comandos IOS para configuração do protocolo OSPF.

Neste caso concreto apenas foram consideradas as tarefas de configuração das interfaces e divulgação das redes através de OSPF. Tal significa que existe uma variedade de comandos a serem repetidos para cada *router* e dentro de cada *router* existem comandos repetidos para cada interface. Dependendo da agilidade do utilizador com a linha de comando e também da sua experiência, poderão ser suprimidos alguns comandos intermédios de validação da configuração, como o comando '*do show running-config*', que permite visualizar a configuração em vigor sem sair do menu onde se está (usando o comando '*do*').

No entanto, a experiência não elimina a repetição existente na escrita de comandos. À medida que se vão adicionando *routers* e a rede vai aumentando de tamanho, o número de comandos a executar e as configurações necessárias vai igualmente aumentando. Os factores descritos anteriormente podem levar ao consumo de tempo na realização das tarefas de configuração e pode potenciar a existência de erros humanos.

Embora o exemplo ilustre uma rede constituída apenas por *routers* Cisco, com o seu sistema operativo IOS, na realidade os cenários reais são normalmente constituídos por *routers* de vários modelos e marcas, como a Alcatel-Lucent, Juniper e Huawei. Esta heterogeneidade nas redes existentes implica um domínio de vários sistemas operativos e um conhecimento dos diferentes comandos para cada marca por parte da pessoa que realiza a configuração. A necessidade do domínio dos CLIs de diferentes marcas aumenta a complexidade na configuração e a morosidade do mesmo processo.

Nos cenários de aprendizagem das redes de computadores são utilizadas várias aplicações que permitem aprender como configurar uma rede e perceber o seu funcionamento. O estudo através do exercício em redes fixas em laboratório é possível, no entanto existem vários obstáculos que impedem uma total experiência dos cenários reais, que passam pela aquisição de equipamentos para os laboratórios e a morosidade de implementação de uma rede física em tempos de aula relativamente pequenos. Uma alternativa ao uso de equipamentos físicos consiste na utilização de aplicações que simulam [10] ou virtualizam [11] o funcionamento dos routers e a sua interligação em cenários de redes, como por exemplo as aplicações *Cisco Packet Tracer* [12] e GNS3 [13] [14].

2.1.3 Aplicações de simulação de redes

No que toca a simulação de redes, existem várias aplicações que são utilizadas, como por exemplo o Cisco Packet Tracer or Huawei eNSP simulator [15] [16] [17]. Este tipo de aplicações são utilizadas nas fases de aprendizagem do funcionamento e configuração de redes de computadores e disponibilizam funcionalidades que permitem a criação de topologias de rede e configuração dos *routers* presentes na mesma. A vantagem do uso deste tipo de aplicações visuais é a visão que estas dão sobre o funcionamento da rede em si e a percepção de como os *routers* interagem entre si, dadas as configurações [18]. A simulação dos *routers* permite um controlo aprofundado do seu funcionamento por parte do software, sendo dada a possibilidade de abrandar o tempo para observar a comunicação entre os intervenientes na rede ao nível dos pacotes transmitidos. A figura 2.8 permite observar como a aplicação Cisco Packet Tracer faz a simulação das comunicações entre os routers e os pacotes enviados entre si.

Uma das vantagens da observação das transmissões de dados usando o modo de simulação do Cisco Packet Tracer é a possibilidade de realizar o *debugging* da rede ao observarmos onde os pacotes estão a ser retidos. Este modo ajuda então a perceber onde há erros de configurações e a reter melhor os conceitos aprendidos.

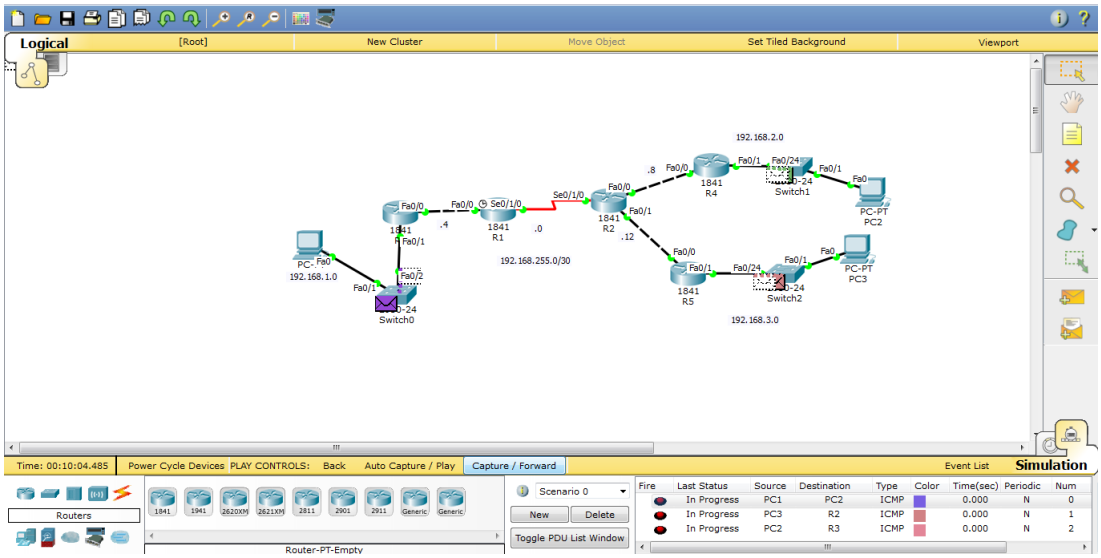

Figura 2.8: Aspecto geral do programa de simulação Cisco Packet Tracer.

2.1.4 Aplicações de virtualização de redes

Existem várias diferenças entre as aplicações que realizam a simulação de cenários de rede face às aplicações que virtualizam esses mesmos cenários. A diferença maior ocorre à medida que a complexidade dos cenários aumenta, quer em número de nós existentes ou nos protocolos em uso. Nos cenários mais complexos as funcionalidades das aplicações de simulação são bastante limitadas, não havendo ao dispor todas as funcionalidades presentes no *router* físico. Nas aplicações que virtualizam equipamentos de rede, há uma gestão de máquinas virtuais que correm o mesmo sistema operativo que um aparelho de rede, permitindo um acesso a todas as funções existentes num *router* real. Nos cenários de teste em empresas e escolas, a aplicação que mais se destaca é o GNS3.

2.1.5 GNS3

O GNS3 [19] [20] é uma aplicação muito utilizada atualmente para virtualizar cenários de interligação de redes. Embora tenha sido desenhada para virtualizar equipamentos Cisco, é também possível incorporar *routers* de outros construtores, como Alcatel-Lucent (ALU), e também interligar com uma rede real.

A Figura 2.9 ilustra os principais componentes do GNS3, que irão ser explicados de seguida.

O componente mais conhecido e utilizado no GNS3 é o `dynamips`, que é um emulador

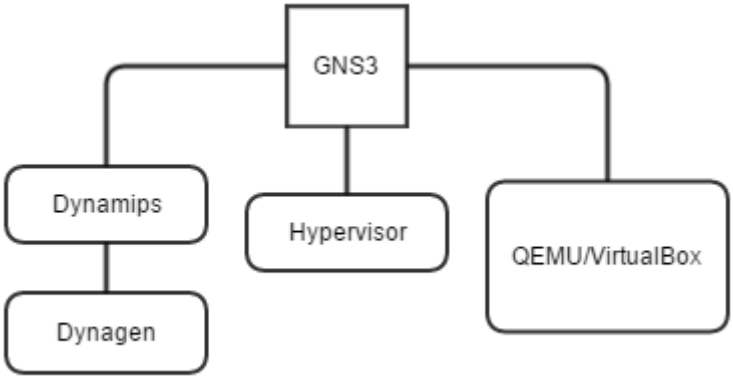

Figura 2.9: A arquitetura do GNS3.

desenvolvido para emular *routers* Cisco. Apesar de ter sido descontinuado, devido à sua utilização no GNS3 este foi mantido e desenvolvido pela equipa do GNS3 e por voluntários. O GNS3 é então a interface gráfica que utiliza o dynamips para emular os *routers* presentes na topologia desenhada, que é mantida e gerida pelo `dynagen`. O `dynagen` é um utilitário que interage com o `dynamips` e que realiza a gestão das redes virtuais criadas para os *routers* emulados. O *hypervisor* [21] usado pelo GNS3 faz a gestão da topologia de rede onde constam os aparelhos Cisco ligados entre si, criando processos dynamips, e permite o acesso às suas portas de consola virtual e, por sua vez, à suas linhas de comando. O GNS3 também permite a criação de máquinas virtuais com recurso a virtualizadores como o QEMU e VirtualBox, que permitem o arranque de sistemas operativos que vão interagir na rede.

Uma das vantagens da utilização de equipamentos virtualizados no GNS3 é a possibilidade destes poderem interagir com *routers* e equipamentos reais, através de pequenas configurações. Esta possibilidade permite uma integração de cenários virtualizados em cenários reais de forma transparente, melhorando casos de teste em cenários de laboratório e eliminando a necessidade de aquisição de equipamentos novos [20]. Através da virtualização com o QEMU ou VirtualBox é possível também a interacção com outros modelos de *routers*, como por exemplo *routers Alcatel-Lucent* [22], permitindo a criação de uma topologia heterogénea.

A Figura 2.10 ilustra a interface gráfica disponibilizada pelo GNS3.

Como é possível observar, existem várias opções disponíveis no GNS3, demarcadas na Figura 2.10 por números:

- 1 - Opção para criação de um *router* na topologia. Após a instalação de uma imagem

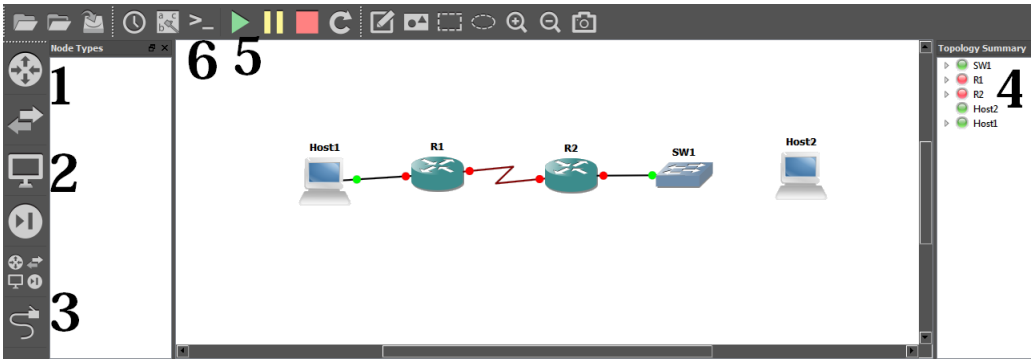

Figura 2.10: Aspecto geral da aplicação GNS3 (versão 1.3.4)

de um *router* no GNS3, estes encontram-se acessíveis aqui.

- 2 - Opção para criação de uma máquina virtual. As máquinas virtuais deverão ser previamente configuradas para constarem nesta opção.
- 3 - Opção para definir as ligações entre os aparelhos. Ao contrário do que permite a aplicação Packet Tracer, esta opção realiza as conexões automáticamente, dependendo apenas da interface a ser configurada.
- 4 - Lista de aparelhos na topologia.
- 5 - Opção para inicialização dos componentes a emular. Todos os *routers* e máquinas virtuais arrancam após a escolha desta opção.
- 6 - Opção para acesso aos componentes emulados em funcionamento. Esta opção abre uma janela de linha de comandos por cada componente emulado na topologia.

As definições de virtualização podem ser modificadas de acordo com o hardware da máquina anfitriã, como por exemplo a sua memória RAM e tipo de CPU. Dado que existe uma virtualização de uma máquina real, o número de nós em funcionamento no GNS3 implica um uso consideravel de memória e um decréscimo no desempenho do computador [23].

Todos os ficheiros de configuração dos *routers* emulados na rede são guardados em formato de texto na pasta do projeto criado no GNS3. Estes são guardados no mesmo formato que as configurações em *routers* reais, permitindo a sua implementação em redes físicas sem qualquer necessidade de alteração. O ficheiro relativo à topologia do projeto criado através do GNS3 é guardado em formato JSON, que permite uma fácil leitura e perceção da hierarquia de dados dentro do ficheiro. Um exemplo de um ficheiro JSON pode ser observado na secção A.1 presente nos anexos.

O GNS3 é uma aplicação muito versátil e que permite a realização de várias tarefas, através da integração com emulações de *routers* e máquinas virtualizadas. É dada uma aproximação à realidade da configuração de uma rede, sem a necessidade do uso de equipamentos reais. A interface gráfica do GNS3 serviu de suporte à aplicação desenvolvida no âmbito desta dissertação, uma vez que é bastante usada pela comunidade para suporte na configuração automática de equipamentos de rede. No entanto, mesmo tendo em conta essa decisão para o desenvolvimento de uma solução para a configuração automática de equipamentos de rede, foi realizado o levantamento das aplicações que já existem que o fazem.

2.2 Aplicações para configuração automática

Nesta secção são descritas várias aplicações que realizam a configuração aumtomática de equipamentos de rede. As aplicações de virtualização e simulação referidas anteriormente são muito usadas para aprendizagem das formas de configuração de uma rede. No entanto, apesar de possibilitarem a maior parte das opções existentes nos *routers*, não realizam a configuração automática. Tendo como objetivo a automatização, foram investigadas e abordadas algumas aplicações que também tentam auxiliar nesse aspeto.

2.2.1 Netomata Config Generator (NCG)

Figura 2.11: Logótipo Netomata

A aplicação Netomata Config Generator (NCG) foi desenvolvida pela empresa Netomata (`http://www.netomata.com`). "Netomata"resulta da junção das palavras *Network* e *Automata* e foca maioritariamente a automatização da configuração de redes. O NCG cria ficheiros de configuração completos e prontos a colocar em *routers*. Estes ficheiros são gerados automaticamente através de um modelo comum, conhecido por *template*, que determina a estrutura de comandos base para configuração de determinada funcionalidade.

Foram identificadas as seguintes características principais no NCG:

- Open-Source.

- Heterogeneidade em relação às marcas de routers.
- Funcionamento em sistema operativo Linux, com licensa GPL.
- Criação automática dos ficheiros de configuração dos *routers*.
- Especificação de características através da linguagem Ruby.

```
rodrigo@ubuntu:~/tese/ncg-0.10.3-r558/examples/point-to-point$ cat devices.neto_table
# List of devices in our network, along with their location,
# model number, and IP address to be used for managing them
#
# For each device, we simply create a node in "!devices" with the info for
# that device.
#
@ !devices!(+)!name = %{name}
@ !devices!(name=%{name})!location = %{location}
@ !devices!(name=%{name})!model = %{model}
@ !devices!(name=%{name})!management_ip = %{management_ip}
#
% name  location        model           management_ip
#
sfo     San Francisco   Cisco 3640      192.168.1.1
lax     Los Angeles     Cisco 3640      192.168.2.1
sjc     San Jose        Cisco 3620      192.168.3.1
oak     Oakland         Cisco 3620      192.168.4.1
san     San Diego       Cisco 3620      192.168.5.1
sba     Santa Barbara   Cisco 3620      192.168.6.1
rodrigo@ubuntu:~/tese/ncg-0.10.3-r558/examples/point-to-point$
```

Figura 2.12: Configuração de dispositivos no NCG.

A figura 2.12 ilustra o modo de criação de dispositivos. Esta configuração serve para adicionar os nomes, modelos e endereços IP dos dispositivos, de modo a que o NCG possa criar as configurações respetivas.

A figura 2.13 ilustra um exemplo da definição dos *links* entre os dispositivos de rede, evidenciando a forma como as interfaces estão ligadas entre si e a respetiva configuração de IP.

Esta aplicação, apesar de apresentar funcionalidades interessantes, tais como a definição da topologia através de ficheiros de texto simples e a geração de configuração automaticamente, tem as desvantagens de não ter documentação esclarecedora quanto à utilização e definição de topologias heterogéneas, bem como à criação de configurações. De revelar ainda que o NCG não sofre atualizações há mais de 2 anos (à data de 2015) e também não tem suporte ao utilizador.

```
########
# Data
########
% rtrA  ifcA          rtrZ    ifcZ          ip_net
# ----  ----          ----    ----          ------
sfo     Serial0/1     sjc     Serial0/1     172.31.1.0
sfo     Serial0/2     oak     Serial0/1     172.31.1.4
sfo     Serial0/3     san     Serial0/1     172.31.1.8
sfo     Serial0/4     sba     Serial0/1     172.31.1.12
sfo     Serial1/1     lax     Serial1/1     172.31.0.0
lax     Serial0/1     sjc     Serial0/2     172.31.2.0
lax     Serial0/2     oak     Serial0/2     172.31.2.4
lax     Serial0/3     san     Serial0/2     172.31.2.8
lax     Serial0/4     sba     Serial0/2     172.31.2.12
rodrigo@ubuntu:~/tese/ncg-0.10.3-r558/examples/point-to-point$
```

Figura 2.13: Configuração das ligações entre dispositivos.

2.2.2 Solarwinds Network Config Generator

Figura 2.14: Logótipo Solarwinds

A aplicação Network Config Generator da empresa Solarwinds[5], que desenvolve software de gestão de infraestruturas e equipamentos informáticos, foi também uma das aplicações a ser analisada. Esta aplicação torna possível ativar funcionalidades avançadas, como a configuração do *Netflow* [24] rapidamente nos equipamentos de rede Cisco. O Netflow é um protocolo desenvolvido pela Cisco que permite o armazenamento de tráfego IP gerado, para monitorização. Este necessita de ser configurado no *router* para que entre em funcionamento. Para isso, o utilizador é ligado remotamente através da aplicação ao aparelho que quer configurar. Após a ligação ao aparelho a ser configurado, são disponibilizadas configurações. A escolha de uma configuração e dos seus parâmetros através da interface é traduzida em comandos na CLI que são introduzidos automaticamente.

Entre as características da aplicação estão:

- Licença de utilização gratuita.
- Foco exclusivamente na configuração de equipamentos Cisco

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

- Interface gráfica para realização das configurações e abstração da escrita de comandos, através da mostragem e escolha de diferentes opções que existem.
- Ligação ao aparelho que se pretende configurar.
- Não permite configuração automática de equipamentos em conjunto, apenas um de cada vez.
- Uso de *templates* para configuração do equipamento.

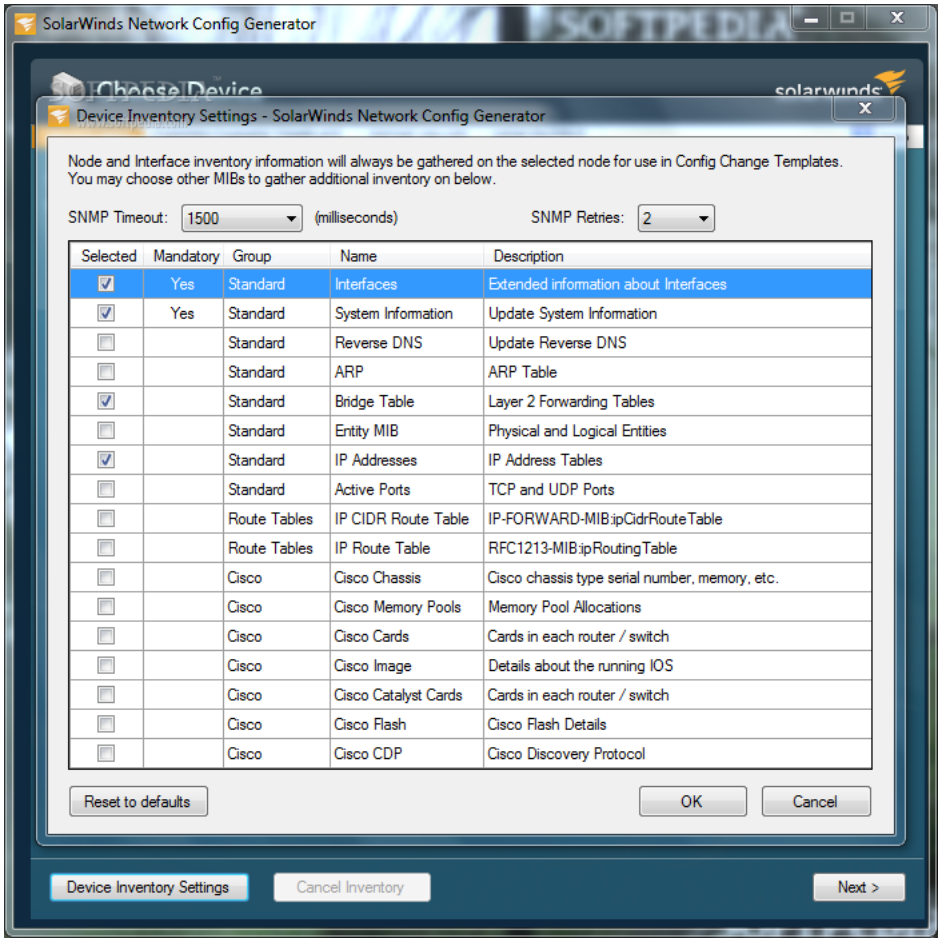

Figura 2.15: Exemplo de configuração de um equipamento

Na figura 2.15 estão ilustradas as várias opções disponíveis ao utilizador após conexão com o *router* Cisco. Apesar desta aplicação permitir ao utilizador explorar funcionalidades avançadas de um *router* e criar as configurações específicas automaticamente, o Network Config Generator possui alguns desafios ou desvantagens a superar. Entre eles está o facto de só se poderem configurar equipamentos Cisco, eliminando a capacidade de se configurar

uma rede heterogénea. Outra desvantagem é a limitação de apenas se poder configurar um equipamento de cada vez, tornando moroso o processo de configurar uma rede inteira, uma vez que não é possível a execução da configuração diretamente. Este desafio está associado também à limitação de só se poder configurar certas funcionalidades, através de *templates* específicos para um *router* já configurado com funcionalidades básicas, eliminando a possibilidade de configurar um equipamento de raiz e gerar uma configuração completa. A escassez de *templates* de configuração nativos obriga à utilização da comunidade *Thwack*[6] para a partilha e divulgação de novos *templates* para configuração.

Em suma, o Solarwinds Network Config Generator é uma boa aplicação para realização de tarefas complicadas através de poucos passos numa rede de pequenas dimensões, desde que haja o *template* associado. No entanto para configurações de raiz ou em redes de grandes dimensões, esta aplicação não é a mais adequada, uma vez que implica a ligação a vários equipamentos [25].

Para a realização de gestão de vários equipamentos com aplicações mais avançadas dentro da empresa Solarwinds, ter-se-ia de proceder à utilização da versão paga da aplicação Network Configuration Manager[7] da Solarwinds. Esta aplicação é então mais aconselhada para a realização de configurações específicas e complexas de um equipamento, ao contrário do NCG da Netomata, que realiza a configuração de vários equipamentos.

2.2.3 GEN-IT

Figura 2.16: Logótipo GEN-IT

Outra aplicação analisada foi o Gen.IT (`http://gen-it.net`). Uma breve análise a esta aplicação através do modo de *trial* permite perceber que utiliza ficheiros na forma de folha de cálculo *excel* como fonte de toda a informação a ser processada e que usa *templates* para configuração dos equipamentos.

Entre algumas das suas características estão:

- Aplicação proprietária paga (Standard - 150 €/ Enterprise - 1585 €).

[6] `https://thwack.solarwinds.com/`
[7] `http://www.solarwinds.com/network-configuration-manager.aspx`

- Heterogeneidade face à configuração aos equipamentos, usando *templates*.
- Interface gráfica.
- Capacidade de leitura de *templates* de configuração e variáveis associadas.
- Necessidade de criação dos respetivos *templates*.
- Não permite especificação da rede que se quer configurar.
- Uso das capacidades das folhas de cálculo excel para processamento de dados.

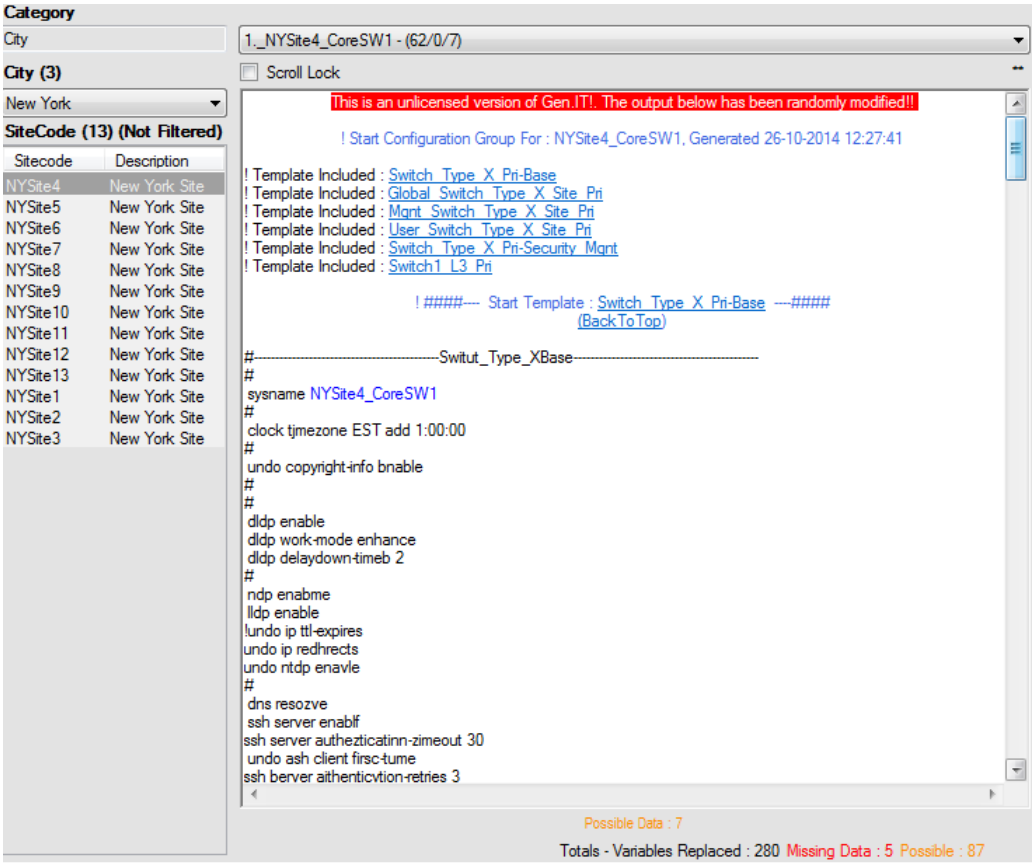

Figura 2.17: Exemplo de funcionamento da aplicação GEN-IT.

A figura 2.17 mostra a aplicação em funcionamento e o uso dos templates para geração de configurações.

A aplicação GEN-IT alega ser uma aplicação simples para configuração de redes de grandes dimensões. Apesar destas alegações, não existe mais informação disponível sobre a aplicação para além da sua própria página web. A heterogeneidade da aplicação deve-se

à necessidade da criação de *templates* específicos para esses equipamentos, por parte do utilizador. Uma das maiores desvantagens do uso do Gen-It passa por não estar disponível gratuitamente. Como vantagem, o Gen-It cria os ficheiros de configuração num todo, que podem ser carregados posteriormente nos equipamentos.

2.2.4 AutoNetKit

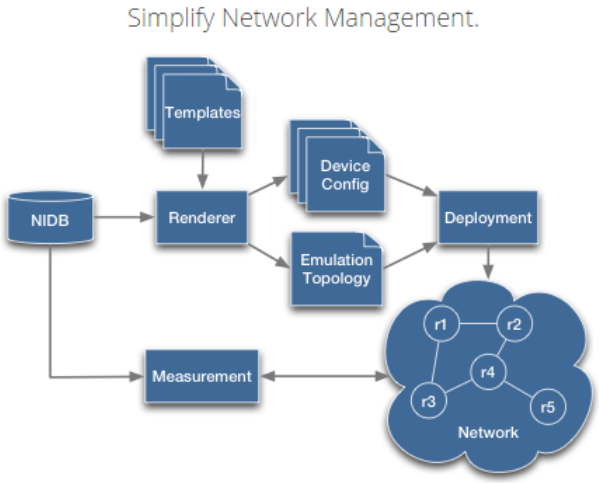

Figura 2.18: AutoNetKit (`http://www.autonetkit.org`)

O AutoNetKit (`http://autonetkit.org`) nasce da automação na geração e implementação de configurações em redes emuladas através da framework NetKit [26]. Assim, o AutoNetKit permite aos utilizadores do NetKit e não só criarem redes maiores e mais complexas, rápida e facilmente. Tem uma interface gráfica simples (usando o *yED*[8], onde se pode desenhar a topologia de rede e rapidamente configurar a rede criada e os seus serviços.

Entre as suas características estão:

- Aplicação *Open-Source*.
- Suporta redes heterogéneas.
- Interface gráfica através do uso da aplicação open-source *yED* para criação de topologia e interface Web para *feedback* visual.
- Uso de *templates*, baseadas em directórios, para criação das configurações.

[8] `http://www.yworks.com/en/products/yfiles/yed/`

- Desenvolvimento em *Python* e funcionamento em Linux.
- Transformação da topologia desenhada em routers e configurações automáticas respetivas.
- Emulação da topologia em NetKit, um ambiente para teste de redes de computadores.

Na Figura 2.19 é apresentado um exemplo de criação de uma topologia de rede utilizando o *yED*.

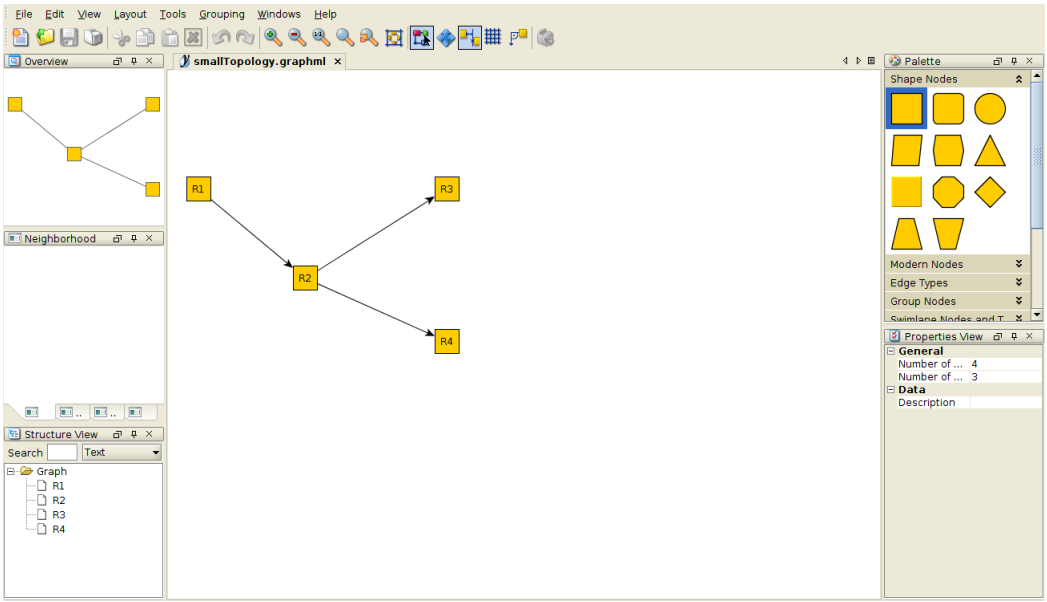

Figura 2.19: Desenho de uma topologia com auxílio da aplicação *yEd*.

O *yED* funciona como um editor gráfico comparável com o Visio (aplicação Microsoft Office) e o Gliffy [27]. Este editor guarda as relações entre os objetos desenhados, permitindo então a especificação da topologia inserida para interpretação e uso no AutoNetKit. Após o desenho da topologia e salvaguarda do ficheiro, a execução do AutoNetKit com o ficheiro criado no *yEd* como parâmetro irá criar automaticamente a hierarquia de pastas com as configurações. Esta sequência de eventos é mostrado na figura 2.20.

A hierarquia de pastas criadas pelo AutoNetKit é apresentada da seguinte forma, como ilustrado na figura 2.21.

É possível observar na figura 2.21 que cada *router* contém uma sequência de pastas e ficheiros. As várias configurações estão separadas por ficheiros (*bgpd.conf*,*ospfd.conf*), cada um com os comandos necessários para a configuração, não havendo uma configuração

```
rodrigo@ubuntu:~$ ls
Desktop  Documents  Downloads  escola  Music  Pictures  psmware  Public  scriptingStuff  smallTopology.graphml
rodrigo@ubuntu:~$ autonetkit -f smallTopology.graphml
INFO AutoNetkit 0.8.38
INFO Automatically assigning input interfaces
INFO IPv4 allocations: Infrastructure: 10.0.0.0/8, Loopback: 192.168.0.0/22
INFO Allocating v4 Infrastructure IPs
INFO Allocating v4 Primary Host loopback IPs
INFO Skipping iBGP for iBGP disabled nodes: []
INFO Unable to connect to visualisation server http://127.0.0.1:8000/publish
INFO All validation tests passed.
INFO Compiling configurations for netkit on localhost
INFO Compiling Netkit for localhost
INFO Unable to connect to visualisation server http://127.0.0.1:8000/publish
INFO Rendering Configuration Files
INFO Finished
rodrigo@ubuntu:~$
```

Figura 2.20: Execução do AutoNetKit, usando como parâmetro o ficheiro da topologia criada com o *yEd*

```
rodrigo@ubuntu:~$ tree rendered/
rendered/
└── localhost
    └── netkit
        ├── lab.conf
        ├── R1
        │   ├── etc
        │   │   ├── hostname
        │   │   ├── shadow
        │   │   ├── ssh
        │   │   │   └── sshd_config
        │   │   └── zebra
        │   │       ├── bgpd.conf
        │   │       ├── daemons
        │   │       ├── isisd.conf
        │   │       ├── motd.txt
        │   │       ├── ospfd.conf
        │   │       └── zebra.conf
        │   └── root
        ├── R1.startup
        ├── R2
        │   ├── etc
        │   │   ├── hostname
        │   │   ├── shadow
        │   │   ├── ssh
        │   │   │   └── sshd_config
        │   │   └── zebra
        │   │       ├── bgpd.conf
        │   │       ├── daemons
        │   │       ├── isisd.conf
        │   │       ├── motd.txt
        │   │       ├── ospfd.conf
        │   │       └── zebra.conf
        │   └── root
        ├── R2.startup
        ├── R3
```

Figura 2.21: Hierarquia de pastas e ficheiros, dada a topologia definida

geral onde estes estejam todos agregados. Como dito anteriormente, o AutoNetKit também disponibiliza uma página web onde se podem realizar mais visualizações da topologia, desde as interfaces em utilização ao fluxo da informação. O ficheiro da topologia criado no *yEd* é introduzido como parâmetro ao arrancar a página. A figura 2.22 ilustra essa topologia apresentada na página Web.

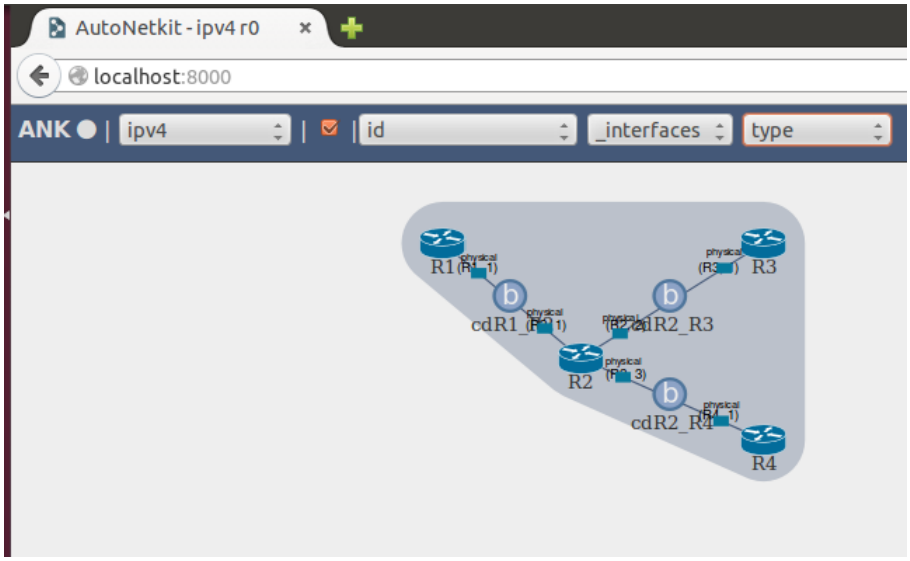

Figura 2.22: Visualização da topologia criada através da interface Web.

O AutoNetKit é uma aplicação com diversos pontos fortes, como a utilização de uma interface gráfica e demonstração da comunicação entre os componentes da topologia. No entanto dá preferência a equipamentos Cisco (embora seja alegado e se preveja a implementação de outros modelos), e não permite a utilização de outras redes que não as pré-definidas. A utilização de uma interface gráfica para auxílio na definição da topologia de rede funciona como uma linguagem de especificação transparente ao utilizador, que é interpretada pelo AutoNetKit. Como maior desvantagem, esta aplicação não gera configurações completas num só ficheiro para aplicação em cenários reais.

Após a análise de algumas aplicações que alegam realizar a configuração automática de equipamentos, foi feita uma análise e comparação dos seus pontos fortes e fracos, dados alguns critérios e realizando no fim uma pequena conclusão..

2.2.5 Comparação das aplicações

A tabela 2.1 apresenta a comparação das aplicações testadas, segundo os seguintes critérios:

- **Licenciamento:** Necessidade de pagamento de licença de uso.
- **Heterogeneidade:** Capacidade de uso de várias marcas de *routers* para configuração.
- **Suporte:** Apoio ao utilizador por parte da comunidade e da entidade que desenvolve a aplicação.
- **Usabilidade e eficiência:** Curva de aprendizagem na utilização da aplicação, facilidade do uso da mesma e eficiência na geração de configurações.
- **GUI:** Uso de interface gráfica que permita uma utilização mais intuitiva

Os critérios mais relevantes a considerar são: a usabilidade e eficiência, uma vez que o objetivo é a geração de configurações automaticamente e, caso tal não aconteça, a aplicação perde o sentido; o uso de interface gráfica, uma vez que o objetivo é a rapidez na configuração automática de topologias, uma GUI para definir as mesmas irá acelerar o processo; heterogeneidade, visto que as redes não são só constituídas por um modelo ou fabricante de *routers*. O licenciamento e suporte não são pontos de grande relevância porque não influenciam diretamente o objetivo de configuração automática de redes.

Tabela 2.1: Comparação das aplicações de configuração automática de *routers*

	Netomata Config Gen.	**Solarwinds**	**GEN-IT**	**AutoNetKit**
Licenciamento	GPL	GPL	Windows	GPL
Heterogeneidade	Sim	Não	Sim	Sim
Suporte	Não	Sim	Não	Sim
Usabilidade e eficiência	Sim	Sim	Não	Sim
GUI	Não	Sim	Sim	Sim

As aplicações estudadas diferem entre si em vários aspetos, além dos descritos acima, como por exemplo no modo de abordagem da configuração dos equipamentos de rede. Esta comparação baseia-se nos testes *blackbox* realizados com as aplicações. Existe uma necessidade de melhoria na abordagem à configuração automática de equipamentos e, através dos testes efetuados com as aplicações, as características mais importantes a melhorar são:

- **Especificação** - As aplicações ou requeriam vários ficheiros para especificação da rede (NCG) ou não permitiam uma especificação de rede(GEN-IT e Solarwinds Network Config Generator). Especificações complicadas e pouco apelativas.

- **Geração de configuração** - Algumas aplicações criavam configurações específicas a uma tarefa e não uma configuração base completa.
- **Heterogeneidade** Apesar de algumas aplicações alegarem suporte a variadas marcas de *routers*, há a necessidade de utilização de *templates* específicos disponibilizados por outros utilizadores. Outras aplicações não apresentam suporte de como fazer a configuração de rede, tendo em conta uma rede heterogénea.
- **Suporte gráfico** A existência de um suporte gráfico simples e robusto para o desenho da topologia é uma mais valia que deve ser explorada. O uso de uma interface familiar ajuda no processo de desenho da topologia a configurar e permitirá uma melhor associação das configurações criadas com o cenário desenhado.

O AutoNetKit é a aplicação que cumpre mais os requisitos avaliados nas várias aplicações e que mais se assemelha ao objetivo definido na dissertação. No entanto, como dito anteriormente, não cria as configurações completas para os equipamentos e não se observa a heterogeneidade alegada. O uso de uma aplicação que gera automaticamente as configurações completas e utilize uma interface mais reconhecível e amigável seria então a solução ideal para a configuração automática de redes. Com o intuito de combater as dificuldades observadas na criação de uma configuração automática para a rede, foi decidido criar um protótipo de uma aplicação, com vista a melhorar as características apresentadas anteriormente e, acima de tudo, que permita a configuração automática de redes heterogéneas, tanto em ambientes reais como em ambientes virtualizados.

Este capítulo descreveu as noções teóricas a reter e as aplicações investigadas para configuração automática de uma rede de comunicação. Foi concluído que, apesar de as aplicações realizarem determinados objetivos, haviam pontos a melhorar, tais como o modo de especificação e a heterogeneidade. No capítulo seguinte irá ser apresentada a arquitectura proposta para a aplicação de configuração de rede a desenvolver.

Capítulo 3

Arquitetura proposta

Este capítulo descreve a arquitetura considerada para aplicação desenvolvida e as principais ações tomadas durante o desenho da arquitetura. De seguida são apresentados os seus componentes e o seu funcionamento. Esta fase antecede o desenvolvimento da aplicação, delineando alguns pressupostos e objetivos para definir de uma melhor forma os pontos a ter em consideração durante o desenvolvimento.

3.1 Enquadramento

A aplicação a desenvolver deverá assentar nos seguintes princípios gerais:

- Usabilidade simples, permitindo a definição fácil das características de configuração da rede;
- Bom desempenho na geração automática dos ficheiros de configuração dos equipamentos de rede;
- Permitir a configuração de uma rede heterogénea, considerando a existência de equipamentos de rede de construtores diferentes;
- Utilização de uma interface gráfica para permitir uma visão global do desenho da rede e auxiliar a sua configuração inicial.

Com base nestes princípios gerais e no teste de várias aplicações existentes, a primeira decisão tomada consistiu no desenvolvimento de raiz de uma nova aplicação de geração

automática de configurações de rede, que colmatasse alguns pontos fracos das aplicações estudadas no capítulo anterior.

3.2 Pressupostos

Para além dos pressupostos apresentados no capítulo 2 para delineação dos objetivos da tese, foram definidos ainda os pressupostos mais orientados para o desenvolvimento do protótipo da aplicação:

- Dos equipamentos que são normalmente utilizados numa rede de computadores, apenas foram considerados os *routers* para geração dos ficheiros de configurações.
- Foram consideradas nesta fase as operações básicas de configuração dos equipamentos de rede, designadamente a configuração IP das interfaces, os protocolos de encaminhamento dinâmico e outras características gerais como o nome do *router* ou um *banner* de boas vindas na consola. Ou seja, nesta fase de desenvolvimento do protótipo, a preocupação é gerar uma rede funcional, que assegure a conectividade entre todos os equipamentos da rede.
- A máscara de rede associada à configuração de endereçamento IP das interfaces é sempre 255.255.255.0/24. Este pressuposto é justificado pela necessidade de agilizar a atribuição de endereços IP às várias redes na fase inicial de desenvolvimento, delegando para segundo plano as estratégias de poupança na atribuição de endereços.
- Nas interfaces *Serial* do tipo ponto-a-ponto um dos *routers* assume a função de DCE (*Data Communication Equipment*), responsável por sinalizar a ligação (*clock rate*) com o *router* remoto, que será o DTE (Data Terminal Equipment). Para efeitos de simplificação na descoberta dos *routers* que desempenham as funções de DTE e DCE e devido à sua inocuidade na definição, a atribuição do comando *Clock Rate* é realizada em ambos os pontos da ligação, ou seja em ambos os *routers*.

Na primeira fase de delineação da aplicação, foi equacionado o desenvolvimento de uma aplicação na linguagem Perl que recebia um ficheiro de configuração em XML com as características de configuração que se pretendiam implementar, conforme se ilustra na Figura 3.1.

O XML foi inicialmente escolhido como linguagem de especificação para registar as características da rede e configurações que se pretendiam incluir. A facilidade do seu uso e

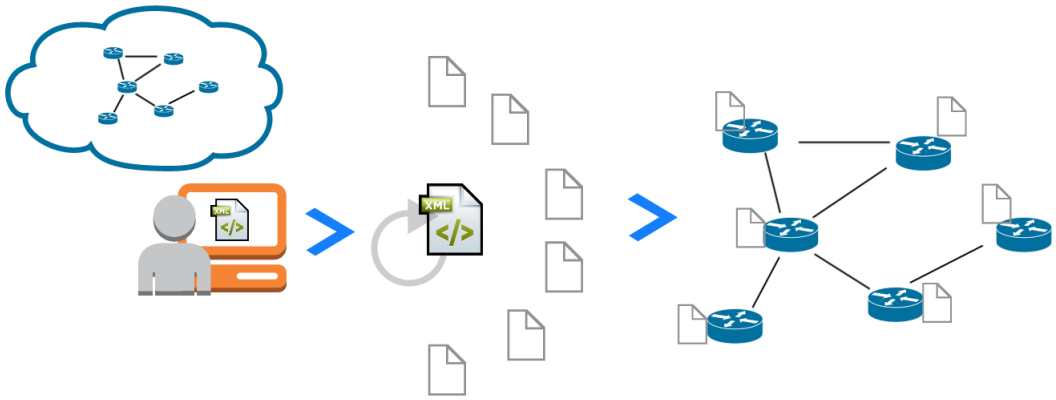

Figura 3.1: Utilização de XML

compreensão, aliada á sua escalabilidade permitem uma especificação simples da configuração da rede e também da sua conversão para os ficheiros de configuração dos equipamentos de rede. O XML respondia assim a alguns dos requisitos iniciais desta aplicação que era a simplicidade de leitura e processamento, de modo a obter o melhor desempenho possível na geração da configuração em redes de grande dimensão.

A abordagem e desenvolvimento inicial da aplicação trouxe vários desafios após a realização de alguns testes. O XML funcionaria como linguagem universal, adaptando o modelo do equipamento ao *template* de configuração armazenado. Porém, a necessidade de guardar no XML as características de cada tipo de configuração que se quer fazer torna o processo de especificação moroso mesmo eliminando a redundância na especificação de algumas dessas características, como a definição da máscara de rede IP para os vários segmentos de rede existentes, a ativação automática das interfaces de rede, a definição do(s) protocolo(s) de encaminhamento em uso, entre outras.

A morosidade na especificação de cada nó e a definição da sua ligação ao ficheiro de especificação iam claramente contra o objetivo de automatização e poupança de tempo na configuração de equipamentos. A criação de uma interface gráfica minimizaria este problema mas implicaria um esforço razoável de desenvolvimento. Os problemas descritos anteriormente motivaram a utilização de uma aplicação gráfica de desenho de rede, cujo ficheiro de especificação servisse de base à configuração dos equipamentos. A solução proposta encontra-se descrita de seguida e assenta nas funcionalidades de desenho do GNS3 e da informação guardada no ficheiro de especificação em JSON.

3.3 Solução proposta

Tendo em conta os objetivos inicialmente traçados para a aplicação e o facto de existirem aplicações com interfaces gráficas que contemplam o desenho da rede e a gestão dos ficheiros de configuração, optou-se por desenvolver uma aplicação que tirasse partido dessas funcionalidades gráficas, automatizasse e otimizasse a geração automática da configuração dos equipamentos de rede.

Existem várias aplicações com interfaces que permitem o desenho e teste de cenários práticos de redes de computadores. Estas aplicações são essencialmente simuladores, como o Cisco Packet Tracer ou assentes em tecnologias de virtualização, como o GNS3, descritas no capítulo 2. Permitem o desenho de topologias com inúmeros nós e conexões entre eles e guardam essas mesmas características em ficheiro de especificação. Esses ficheiros contém a especificação dos nós criados na topologia e a relação entre eles, bem como outras características gerais do cenário.

O GNS3 é uma aplicação que faz a virtualização de cenários desenhados através de uma interface gráfica, usando os vários componentes descritos no capítulo 2. Uma vez que o ficheiro onde as topologias são guardadas está no formato JSON (com extensão `.gns3`), que pode ser facilmente lido e interpretado, é possível à nossa aplicação criar as configurações com base nessa estrutura de dados, que já se encontra organizada automaticamente. O GNS3 é uma das aplicações mais utilizadas nos ambientes de ensino e profissionais, que a torna assim bastante popular na área das redes de computadores e entre os utilizadores. Essa popularidade traduz-se na existência de uma comunidade e um suporte ativo à aplicação.

Como dito anteriormente, o GNS3 promove a heterogeneidade permitindo o uso de vários modelos de *routers* através de máquinas virtuais, o que também vai ao encontro do objetivo de desenvolvimento de uma aplicação heterogénea. Devido à capacidade de virtualização da rede criada, o GNS3 pode ser usado para validar as configurações existentes nos *router*, sem a necessidade de implementação das mesmas noutras aplicações ou em *routers* reais, embora seja possível. Por estas razões, o GNS3 foi a aplicação escolhida para a integração com a aplicação de geração automática de configurações de *routers*.

A arquitectura do funcionamento da aplicação usando o GNS3 está organizada como é descrito na figura 3.2.

Após o desenho da topologia através do GNS3, o ficheiro que define o projeto desenhado é criado, tendo o formato JSON, e contém todas as especificações da rede ao nível de nós e da relação entre si. A secção A.1 presente nos anexos exemplifica um ficheiro JSON com a especificação de um cenário. Este ficheiro JSON é então introduzido como um dos parâ-

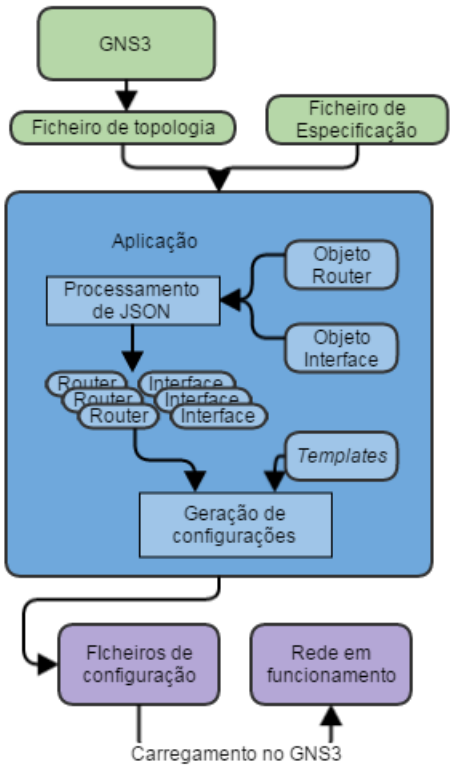

Figura 3.2: Arquitectura da aplicação de geração da configuração de redes e integração no GNS3.

metros necessários para a aplicação através de um ficheiro de especificação. Este ficheiro de especificação é utilizado como parâmetro de entrada ao executar a aplicação e contém toda a informação necessária à aplicação, como por exemplo o ficheiro JSON a processar, as configurações a efectuar, entre outras, que serão descritas no capítulo de desenvolvimento. A aplicação, após receber os parâmetros definidos pelo ficheiro de especificação, lê o ficheiro JSON e retira a informação relativa à topologia desenhada. A informação retirada do ficheiro da JSON corresponde aos *routers* presentes na topologia desenhada e às ligações que estes têm com outros elementos da rede.

A metodologia de Programação Orientada a Objetos (POO) é utilizada para o armazenamento da informação, ao serem utilizadas classes de objetos para definir os dados a serem armazenados. As classes em uso são a classe *Router* e a classe *Interface* (podem ser observadas no capítulo 4 na figura 4.2), que guardam a informação relativa a cada *router* e interface presente no ficheiro JSON lido. Após realizado o armazenamento de toda a informação necessária, são geradas dinamicamente pela aplicação todas as informações relativas ao endereçamento de rede, tendo em conta os pressupostos descritos anteriormente. No final do

processamento e armazenamento da informação, cada objeto *router* irá gerar o seu ficheiro de configuração, que poderá ser utilizado numa topologia real ou simulada para validação do seu funcionamento. A geração dos comandos para o ficheiro de configuração é realizada com o recurso a *templates* que guardam todos os comandos necessários para a configuração. No final da geração dos ficheiros de configuração, estes poderão ser carregados novamente no GNS3 nos *routers* correspondentes e a rede estará configurada de modo a que todos os elementos comuniquem entre si.

A figura 3.3 mostra a organização por módulos da aplicação, estando representados os módulos *Router.pm* e *Interface.pm*, o *script* principal e os *templates* utilizados para a geração das configurações.

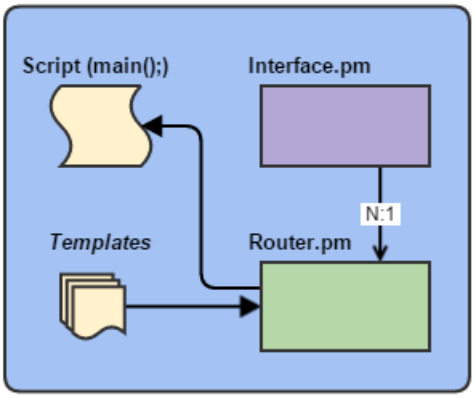

Figura 3.3: Organização da aplicação

Para o seu funcionamento, a aplicação utiliza o *script* principal, os módulos *Router.pm* e *Interface.pm* e também *templates* para auxílio na geração dos comandos para o ficheiro de configuração. O *script* é o interveniente principal, que é responsável pela recepção, processamento e armazenamento dos dados presentes no ficheiro JSON criado pelo GNS3. Para armazenar a informação processada, o *script* utiliza os módulos *Router.pm* e *Interface.pm*. Estes módulos são responsáveis pelo armazenamento de toda a informação obtida e gerada pelo *script*.

O módulo *Router.pm* contém a classe *Router* que é utilizada para armazenar toda a informação relativa aos *routers* presentes na rede, desde o seu *hostname*, as configurações a realizar e as interfaces pertencentes a si. A classe *Router* também é responsável pela geração dos ficheiros de configuração de cada *router* presente no ficheiro JSON. O módulo *Interface.pm* representa a classe *Interface*, que armazena a informação relativa às interfaces de cada *router* na topologia. A informação guardada por esta classe é: O *router* ao qual a interface pertence, o seu tipo de interface, o IP e máscara definidos dinâmicamente pela aplicação e por fim o

nó destino ao qual está conectada. Cada objeto *Router* contém um ou mais objetos *Interface*. Finalmente, os *templates* são utilizados na geração das configurações.

3.3.1 Noção de *template*

Na fase de geração de ficheiros de configuração, os dados armazenados são então combinados com os *templates* que definem os comandos a inserir no ficheiro de configuração. Os *templates* são pequenos ficheiros de texto segregados por função na configuração a gerar para um *router*. Por exemplo, para configurar o endereçamento IP numa interface, em *routers* Cisco, é necessário inserir o comando `ip address a.b.c.d x.y.z.w`, sendo "`a.b.c.d`" e "`x.y.z.w`" o endereço IP e máscara de rede/sub-rede, respectivamente. Este comando é executado no contexto de configuração da interface em causa. Existe um template para cada função específica. Por exemplo, existe um template para cada configuração de protocolo de encaminhamento, outro para a configuração de interface, e assim sucessivamente. Cada comando num template pode ter uma ou mais máscaras para serem substituídas pelos valores gerados e armazenados pelo script ou pelo utilizador. A figura 3.4 mostra o template usado para configurar uma interface.

```
interface __INTERFACE__
ip address __ADDRESS__ __MASK__
description __DESCRIPTION__
no shutdown
```

Figura 3.4: Exemplo de um *template*.

As máscaras `__INTERFACE__`, `__ADDRESS__`, `__MASK__`, e `__DESCRIPTION__` são substituídas pelos valores correspondentes que foram armazenados pelo objecto Interface definido no módulo *Interfaces.pm*, nomeadamente o endereço IP, máscara e descrição da ligação. Tomemos como exemplo os seguintes dados armazenados por uma interface, ilustrados pela figura 3.5.

```
Interface = "Serial 0/0/0"
IP = "192.168.1.2"
Máscara = "255.255.255.0"
Descrição = "Ligação ao Router R2"
```

Figura 3.5: Exemplo de dados armazenados para uma interface.

Dados estes valores, o resultado dos comandos através da interface pode ser observado na figura 3.6.

```
interface Serial 0/0/0
ip address 192.168.1.2 255.255.255.0
description Ligação ao Router R2
no shutdown
```

Figura 3.6: Exemplo de resultado final.

Após a criação dos ficheiros de configuração para os nós desenhados na topologia, estes poderão substituir os ficheiros de configuração dos routers no GNS3, (que não contêm nenhuma configuração), carregados nas máquinas virtuais dos routers e ser usados no GNS3 para teste das configurações criadas para cada um dos nós existentes.

3.3.2 Heterogeneidade

Um dos objetivos traçados para esta aplicação consiste na possibilidade de definir redes heterogéneas, em que os equipamentos de rede são de vendedores diferentes, com sistemas operativos distintos. Esta funcionalidade é igualmente alcançada com recurso aos *templates*, uma vez que podem ser orientados não só a funções diferentes para o mesmo tipo de *router*, mas também para *routers* de marcas diferentes. Como a sintaxe do CLI e dos comandos constituem uma mudança significativa entre marcas, onde os parâmetros principais a configurar são os mesmos, os *templates* permitem uma configuração mais abrangente. A figura 3.7 mostra então um *template* que poderá ser utilizado para configurar *routers* Alcatel-Lucent, em contraste à figura 3.4, usado para a mesma função mas para routers Cisco.

```
interface "__DESCRIPTION__"
address __ADDRESS____MASK__
port __PORT__
exit
```

Figura 3.7: Exemplo de configuração em *routers* ALU vSR-OS

Assim, para adicionar um *router* diferente à aplicação, apenas são necessários *templates* específicos para esse mesmo *router*, não implicando a alteração de código. A aplicação só terá de associar as máscaras aos dados que têm de ser substituídos. Uma vez alcançado esse objetivo, a implementação e configuração de topologias heterogéneas será assim um processo simples e intuitivo.

Com base no desenho criado no GNS3 e do correspondente ficheiro de texto contendo as especificações do cenário, como equipamentos e suas interligações, a aplicação desenvolvida gera automaticamente uma configuração de rede funcional, cujos ficheiros podem ser carregados para os equipamentos virtualizados no GNS3. É assim possível testar vários cenários de forma rápida, com a automatização das tarefas de configuração, não sendo necessário conhecimentos profundos de redes. Também para a configuração de protocolos mais avançados, onde é necessária a configuração de base dos equipamentos de rede, esta aplicação poderá minimizar o tempo dispendido.

3.3.3 Integração com cenários reais

O GNS3 é uma aplicação muito poderosa que dispõe de um leque variado de funcionalidades, entre elas a possibilidade de integração do cenário virtual criado com uma rede real física. A vantagem desta capacidade é a possibilidade de complementação de uma rede física com uma virtual e a realização de testes de uma forma transparente e em simultâneo. A figura 3.8 ilustra o conceito de ligação de um cenário virtualizado no GNS3 a uma rede física.

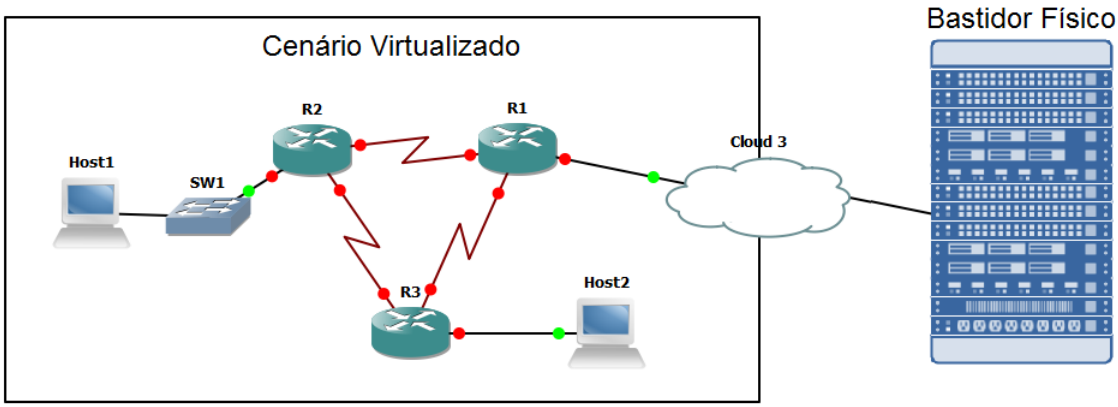

Figura 3.8: Ligação de um cenário no GNS3 a uma rede física

A realização de testes com recurso a esta funcionalidade permite que não seja implementada uma rede física para teste e permite também o teste de funcionamento de redes experimentais com a rede real em utilização antes da sua implementação, não prejudicando a rede real. Para realização desta integração com cenários reais podem ser seguidos vários tutoriais [28] disponíveis na Internet. A solução passa pela criação de um adaptador virtual *loopBack* na máquina onde está instalado o GNS3, que servirá de ponto de ligação ao GNS3 através do

nó virtual representado pela *cloud*. No nó *cloud* está configurada a ligação da rede virtual do GNS3 ao adaptador *loopBack*, e consequentemente à rede física.

Este capítulo descreveu a arquitetura delineada para a aplicação de geração automática de configurações. Com a solução proposta delineada e os objetivos e processos definidos, procedeu-se ao desenvolvimento da aplicação, que é descrito no capítulo seguinte.

Capítulo 4

Desenvolvimento

Este capítulo descreve a fase de desenvolvimento da aplicação, tendo em conta os vários pressupostos definidos no capítulo anterior. O capítulo descreve a linguagem de programação utilizada para desenvolvimento, a linguagem de especificação utilizada para descrever a topologia de rede, a forma como a aplicação foi desenvolvida para armazenar a informação recolhida do ficheiro que define a topologia de rede, e as funções principais da aplicação e o seu funcionamento. No final do capítulo é descrita a forma como é possível integrar a aplicação como extensão do GNS3.

A aplicação foi desenvolvida na linguagem Perl (`https://www.perl.org/`). A linguagem Perl é utilizada em vário domínios, desde *websites* como o IMDb (`http://www.imdb.com/`), que foi inicialmente construído através de *scripts* escritos na linguagem Perl a scripts de processamento de dados geralmente utilizados por administradores de sistemas. Tal como as *libraries* para o Java, existem módulos para Perl que permitem realizar diferentes tarefas, como processamento de determinados tipos de ficheiro, debugging, entre outras. Estes módulos, disponíveis no CPAN (`http://www.cpan.org/`) são pedaços de código escritos em Perl ou C, que são desenvolvidos pela comunidade e são disponibilizados para utilização por outros utilizadores. Esta capacidade de processamento de dados e o seu alto desempenho na realização de tarefas foram algumas das razões pelas quais a linguagem Perl foi escolhida para ser utilizada no desenvolvimento. Embora seja maioritariamente utilizado em sistemas UNIX a linguagem Perl pode ser utilizada em várias plataformas, o que permite que o código seja executado em várias plataformas sem a necessidade de realização de grandes alterações no código fonte.

Para desenvolvimento e execução da aplicação foi utilizado o *Strawberry Perl* 5.22.0.1 (Windows 64 bit `http://strawberryperl.com/`).

4.1 Linguagem de Especificação

Na criação de uma aplicação que necessita de configurações de entrada é importante definir uma linguagem de especificação. Esta linguagem é o ponto de entrada para o processamento, com vista à criação do resultado final. Ao contrário das linguagens de programação, que são de mais baixo nível e são executadas, as linguagens de especificação não definem o *como* mas sim o *quê*. No exemplo de especificação de uma topologia de rede, quando não existe uma GUI, esta linguagem serve então para definir os nós, o seu tipo, a ligação entre estes, a gama de redes a usar, as interfaces em ligação, entre outras características. Tendo em conta a sua relevância no resultado final, é importante que esta linguagem seja do mais alto nível possível, para que se possam definir o número maior de características no menor tempo possível e não limitando a escalabilidade. Esta linguagem deve ser simples e fácil de entender, maximizando a produtividade do seu uso [29].

Antes da decisão em utilizar a interpretação do formato JSON, foi considerada a linguagem XML (`http://www.xml.com/`) para definição da topologia de rede a configurar. O XML é uma linguagem muito usada como sintaxe para vários formatos de documentos [30] e como linguagem base para protocolos de comunicação. Apesar de ser uma linguagem mais orientada a aplicações Web, o XML preenche os requisitos desejados para implementação da linguagem para definição da topologia e características da rede.

O XML usa "unidades de armazenamento"designadas entidades, que contêm dados. Estas entidades são constituidas por "marcações"que designam os dados e a hierarquia dos mesmos.

Apesar da sua fácil compreensão e organização de dados, o XML tem como desvantagem a morosidade na especificação de topologias de grande tamanho. A necessidade de escrita de todos os nós sempre que é necessário desenhar uma topologia nova, faz com que esta fase de desenho da topologia seja muito morosa e repetitiva, o que vai contra os objetivos do trabalho incluído nesta dissertação. Para eliminar este problema, seria necessário o desenho de uma interface que realizasse a geração do ficheiro de topologia em XML ou adaptar uma que já existisse. A integração do GNS3 na solução veio então então resolver este problema.

4.2 JSON

Uma vez que a utilização de uma interface gráfica eliminava a necessidade de rescrita do ficheiro de especificação sempre que se necessitava realizar uma alteração, foram pesquisa-

das várias alternativas ao desenvolvimento de uma interface gráfica de raiz. O GNS3 foi a alternativa escolhida, pelas várias razões descritas no capítulo 2, entre as quais o armazenamento de configurações dos *routers*, a sua virtualização e o armazenamento da topologia em JSON (`http://json.org/`), que pode ser interpretado. O JSON permite o armazenamento de dados em estruturas que podem ser facilmente interpretadas por aplicações e utilizadores, sendo uma alternativa viável ao XML. Uma vez que o ficheiro JSON define uma topologia gerada automaticamente pelo GNS3, a aplicação poderá interpretar o ficheiro gerado pelo GNS3 e consequentemente criar as configurações dos equipamentos de acordo.

O GNS3 armazena os dados da topologia (ou projeto) da seguinte forma:

- Dados do projeto: Inclui o nome do projeto e a sua identificação.
- Dados da topologia: Dentro dos dados da topologia estão presentes duas estruturas de dados, a estrutura referente às ligações entre nós da topologia e a estrutura que guarda a informação dos nós presentes na topologia.

O Anexo A.1 presente no capítulo de anexos mostra um exemplo de um ficheiro JSON completo gerado pelo GNS3.

Uma vez que o ficheiro de especificação é agora um ficheiro JSON gerado por outra aplicação, ainda é necessário especificar outros detalhes como a configuração a realizar, o local de criação das configurações, entre outros parâmetros. Para isso foi decidido utilizar um ficheiro de especificação auxiliar que define estes parâmetros adicionais, e que vai servir de parâmetro de entrada para a execução da aplicação. Assim, este ficheiro define o directório de saída do resultado das configurações, as configurações a efetuar, o ficheiro JSON a processar e o local onde estão definidas as configurações base utilizadas para adicionar outras configurações genéricas, dependendo do modelo de *router*. O ficheiro de especificação pode ser observado na figura 4.1 e um exemplo de ficheiro base de configuração está disponível no anexo A.2 na secção de anexos.

```
ProjectDir = C:\Users\Script\scenario.gns3
BaseConfigPath = C:\Users\Basefiles\
OutputDir = C:\Users\
Configurations = interface,ospf
```

Figura 4.1: Exemplo de ficheiro de especificação usado na execução da aplicação.

4.3 Programação Orientada a Objetos

Houve a decisão de adaptar o modelo de Programação Orientada a Objetos [31] ao desenvolvimento da aplicação de forma a simplificar a estrutura de armazenamento dos dados retirados do ficheiro JSON, simplificar o uso dos dados recolhidos, e proporcionar uma melhor organização do código. A POO usa o conceito de "Objeto"para designar uma estrutura de dados, que permite armazenar dados referentes ao objeto. Essa estrutura de dados, para além de armazenar informação relevante para a instância do Objeto, também pode realizar operações (métodos) sobre esses mesmos dados.

Por exemplo, assumindo que é pretendido desenvolver uma aplicação que entre as suas funções armazene informação relativa a um conjunto de pessoas, como o nome, sexo, contacto telefónico, morada, entre outras. Usando uma abordagem de Programação Orientada a Objetos basta ser criada uma classe de objeto genérica denominada *Pessoa* que armazene essa informação e, para cada pessoa, ser instanciada essa classe para recolher os dados. Assim cada pessoa tem uma instância do objeto classe *Pessoa* que contém toda a informação relativa a si. A forma mais utilizada para aceder aos dados de cada instância é através do uso de métodos que permitem obter e definir os dados que cada instância guarda.

No caso concreto da aplicação desenvolvida, os dados mais relevantes a armazenar são os dados referentes aos *routers* presentes na topologia e a sua interligação através das suas interfaces, portanto foram criadas as classes *Router* e *Interface* representadas pelos módulos Perl *Router.pm* e *Interface.pm*, respectivamente, como é possível observar na figura 4.2.

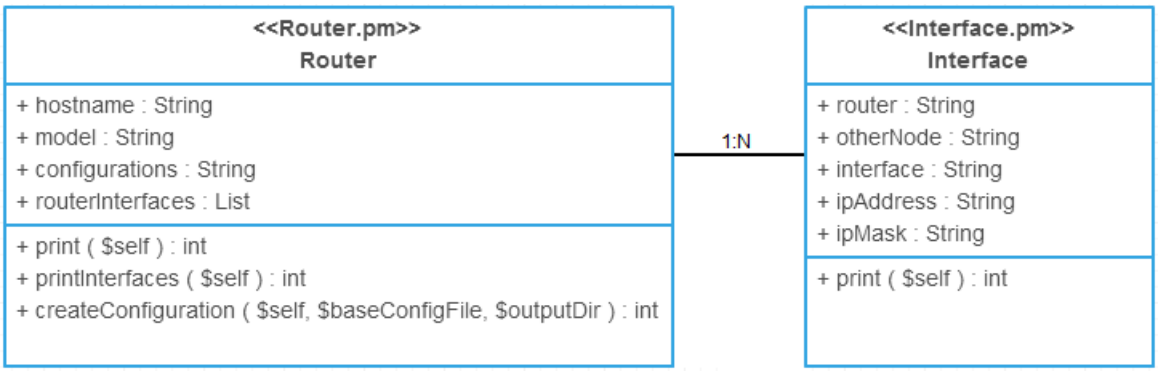

Figura 4.2: Classes definidas nos módulos Router.pm e Interface.pm

O módulo *Router.pm* guarda toda a informação presente num dado *router* no ficheiro de topologia que a aplicação está a processar. Por cada *router* existente na topologia, existe uma instância da classe *Router* para o representar. A classe armazena os dados referentes

ao *hostname* do *router*, que será igual ao nome dado na topologia do GNS3 para o mesmo; a variável *model* guarda o modelo do router, permitindo a distinção de vários tipos e marcas de routers presentes na topologia, indo de encontro ao objectivo da heterogeneidade; a variável *configurations* guarda as configurações a implementar no *router*, desde o protocolo de encaminhamento às interfaces; a variável *routerInterfaces* armazena todas as instâncias das interfaces existentes na topologia, de modo a que se consiga gerar a configuração das interfaces pertencentes à instância do router.

As funções existentes nesta classe são o *print()*, que mostra os valores das variáveis referentes à instancia em questão; o *printInterfaces()* mostra a informação relativa às instâncias de interface referentes ao *router* em questão; finalmente, a função *createConfiguration* vai criar o ficheiro de configuração pertencente à instância do *router* usando toda a informação relativa ao mesmo e às suas interfaces. O modo como o ficheiro de configuração é gerado será detalhado mais à frente.

O módulo *Interface.pm* é utilizado para definir a classe *Interface*, que irá armazenar os dados relativos às instâncias de interfaces que estão em uso na topologia. Os dados armazenados são: a variável *router*, que guarda o nome do *router* ao qual a instância da interface pertence; a variável *otherNode* guarda o destino ao qual a interface está ligada; a *interface* armazena os dados referentes ao tipo de interface que a instância representa, como por exemplo *Serial 0/0/0*; o *ipAddress* e *ipMask* armazenam o IP gerado automaticamente para esta instância de interface e a sua máscara, respectivamente.

A única função existente na classe *Interface* é a função *print()*, que mostra os dados relativos às variáveis faladas anteriormente, que pertencem à instância em questão. As funções principais na aplicação, onde estes módulos participam, são descrevidas de seguida.

4.4 Funções principais

A aplicação realiza vários passos durante o seu funcionamento, a interpretação do ficheiro de especificação, interpretação do ficheiro JSON criado pelo GNS3, armazenamento de dados, entre outros. Cada um destes passos está dividido por funções específicas. Para a extração de dados do ficheiro JSON é utilizado a função `extractData()`, que usa os módulos descritos anteriormente para armazenamento dos dados extraídos e geração de dados como endereços IP. A função `createConfiguration()` é utilizada para a geração dos ficheiros de configuração. Existem também mais funções que realizam outras tarefas.

O algoritmo 1 mostra a forma como a aplicação realiza as várias tarefas, de um modo geral.

Algorithm 1: `script`

Data: Ficheiro especificação

Result: Ficheiros configuração Routers

```
1  while Processar ficheiro Especificação do
2      Obter nome ficheiro JSON;
3      Obter localização ficheiro base;
4      Obter configurações;
5      Obter localização Output;
6  end
7  while Processar ficheiro JSON do
8      Criar instâncias objetos 'Interface';
9      Gerar endereços IP para interfaces;
10     Criar instâncias objetos 'Router';
11 end
12 Gerar pasta output for Cada instância router existente do
13     Gerar configuração com as configurações;
14     Colocar ficheiro na localização output;
15 end
```

Descreve-se de seguida, de forma resumida, o funcionamento geral do algoritmo. Primeiro, a aplicação interpreta o ficheiro de especificação, onde obtém os dados referentes ao ficheiro JSON a processar, ficheiros base a utilizar, configurações a efectuar e a localização onde o utilizador deseja que a aplicação coloque as configurações finais. De seguida, através do ficheiro JSON recebido para processar, a aplicação interpreta os dados existentes e cria as várias instâncias dos objetos '*Router*' e '*Interface*', gerando os endereços IP para as várias ligações existentes. Finalmente, após geração e armazenamento de dados consoante o ficheiro JSON, a aplicação ordena a cada instância de um *router* para gerar a sua configuração. Os dois procedimentos mais relevantes, que se distinguem devido à sua importância, são o tratamento do ficheiro JSON e a realização da configuração automática. Devido ao seu impacto na aplicação, cada um destes passos são descritos sucintamente adiante.

4.4.1 Processamento de ficheiros JSON

Uma das tarefas iniciais realizadas na execução da aplicação é o processamento do ficheiro JSON que é fornecido pelo ficheiro de especificação na chamada da aplicação. Este proces-

samento permite a extração dos dados referentes à topologia e o seu armazenamento. Os dados extraídos do ficheiro correspondem aos *routers* existentes e à ligação existente entre *routers* e os restantes nós.

A função *extractData()* é responsável pelo tratamento do ficheiro JSON. Nesta função foram utilizados os módulos para Perl designados *JSON*[1] e *Data::Dumper*[2], para além do ficheiro da topologia. O módulo *Data::Dumper* foi utilizado inicialmente para interpretar o ficheiro JSON e mostrar como os dados referentes à topologia se encontravam organizados hierarquicamente, ajudando à sua interpretação e à navegação entre a informação armazenada. O módulo *JSON* permite interpretar dados organizados no formato JSON e é utilizado para descodificar o ficheiro da topologia, dando uma referência para variável que é utilizada para extração dos dados.

A figura 4.3 ilustra o modo como o módulo *Data::Dumper* mostra como os dados de um ficheiro de topologia JSON estão organizados hierarquicamente.

Sabendo a forma como os dados em JSON estão organizados hierarquicamente, é possível perceber a melhor forma de aceder aos dados. Tomemos como exemplo o excerto JSON para a especificação dos portos, representado pela figura 4.4.

A variável '*ports*' representa os portos presentes num dado nó, que neste exemplo pertence a um '*host*' no GNS3. Esta variável pertence a uma *hash*. As *hashes* são estruturas de dados onde existem chaves e valores ligados entre si, ou seja, cada valor é acedido através da sua chave. A variável '*ports*' é uma chave que tem como valor um *array* ou vetor de dados, que na figura 4.4 é representada pelo símbolo '=>'. Este array aponta para os dados que a chave contém, e o símbolo '[]' que designa que os dados estão organizados na forma de um vetor. Cada índice existente dentro desse vetor é separado por uma vírgula e pode ser acedido através da sua posição, representada por um número que começa no '0'. Ou seja, o primeiro índice é acedido na posição '0', o seguinte na '1' e por aí adiante. É possível observar que a variável '*ports*' contém 4 posições preenchidas com dados referentes a *hashes*, representados entre duas chavetas ''.

Tendo por base o exemplo, existe um *host* na topologia representada em JSON criada pelo GNS3 que contém 4 portos. Cada um desses portos tem informação relativa a si, representada na forma de *hash*, com chaves e valores. Cada uma das chaves representa uma característica do porto e o valor representa os dados dessa característica. Programaticamente, caso se necessitasse saber o valor do nome do segundo porto desse '*host*', esse valor seria acedido da seguinte forma `Host->ports->[1]->name`. Sabendo a forma

[1] `http://search.cpan.org/~makamaka/JSON-2.90/lib/JSON.pm`
[2] `http://perldoc.perl.org/Data/Dumper.html`

```
            'name' => "nio_gen_eth:Liga\x{e7}\x{e3}o de \x{c1}rea Local"
          }
        ]
      },
      {
        'id' => 5,
        'description' => 'Host',
        'type' => 'Host',
        'properties' => {
            'name' => 'Host2',
            'nios' => [
              "nio_gen_eth:Liga\x{e7}\x{e3}o de rede sem fios",
              'nio_gen_eth:VMware Network Adapter VMnet1',
              'nio_gen_eth:VMware Network Adapter VMnet8',
              "nio_gen_eth:Liga\x{e7}\x{e3}o de \x{c1}rea Local"
            ]
        },
        'x' => '284.5',
        'ports' => [
            {
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'},
              'id' => 23,
              'name' => "nio_gen_eth:Liga\x{e7}\x{e3}o de rede sem fios"
            },
            {
              'name' => 'nio_gen_eth:VMware Network Adapter VMnet1',
              'id' => 24,
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'}
            },
            {
              'name' => 'nio_gen_eth:VMware Network Adapter VMnet8',
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'},
              'id' => 25
            },
            {
              'nio' => 'NIO_Generic_Ethernet',
              'link_id' => 4,
              'id' => 26,
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'},
              'description' => 'connected to SW1 on port 2',
              'name' => "nio_gen_eth:Liga\x{e7}\x{e3}o de \x{c1}rea Local"
            }
        ],
        'label' => {
          'font' => 'TypeWriter,10,-1,5,75,0,0,0,0,0',
          'color' => '#000000',
          'y' => '-25',
          'text' => 'Host2',
          'x' => '9.5'
        },
        'y' => '-49.5',
        'server_id' => 1
      }
    ]
  },
  'name' => 'cenario2-2routers'
};
```

Figura 4.3: Exemplo de *output* do *Data::Dumper* para um ficheiro JSON

```
'ports' => [
            {
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'},
              'id' => 23,
              'name' => "nio_gen_eth:Liga\x{e7}\x{e3}o de rede sem fios"
            },
            {
              'name' => 'nio_gen_eth:VMware Network Adapter VMnet1',
              'id' => 24,
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'}
            },
            {
              'name' => 'nio_gen_eth:VMware Network Adapter VMnet8',
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'},
              'id' => 25
            },
            {
              'nio' => 'NIO_Generic_Ethernet',
              'link_id' => 4,
              'id' => 26,
              'stub' => $VAR1->{'topology'}{'servers'}[0]{'local'},
              'description' => 'connected to SW1 on port 2',
              'name' => "nio_gen_eth:Liga\x{e7}\x{e3}o de \x{c1}rea Local"
            }
          ],
```

Figura 4.4: Dados para os portos mostrados pelo *Data::Dumper*.

de como aceder a todos os valores guardados no ficheiro JSON, foi possível armazenar todos os dados relevantes à aplicação. A função `extractData()` mostra a forma como esse armazenamento foi realizado.

4.4.1.1 Função `extractData()` - interface

Após a perceção de como a informação é guardada no formato JSON, é realizada a sua leitura e interpretação. Os primeiros dados a serem interpretados e armazenados são os dados relativos às conexões existentes na topologia. Por cada conexão existente na topologia é atribuída uma rede. As redes atribuídas nesta fase são redes 192.168.a.b/24, onde 'a' é incrementado consoante as conexões existentes na topologia e 'b' é atribuído com valores diferentes (tipicamente '1' e '2' ou '254' e '253') às interfaces que integram essas conexões. É nesta fase que são criadas as instâncias das interfaces presentes na topologia. Isto acontece uma vez que o ficheiro JSON da topologia contém uma estrutura designada *links* onde estão retratadas todas as interfaces que desempenham um papel nas conexões rede, e a sua informação. Por cada interface integrante numa conexão é criada uma instância da classe *Interface*, atribuído um IP e máscara, e guardados os dados referentes ao *router* a que pertence, ao destino a que está ligada e porque tipo de interface. Para isso é necessário determinar se as interfaces integrantes na conexão pertencem a *routers*, através da função *checkIsRouter()*. As instâncias são todas armazenadas numa lista.

Algorithm 2: `extractData() (Interface)`

Data: Ficheiro JSON
Result: Lista de objetos interface

```
1  while Existem conexões em 'links' do
2      if Interface origem pertence a Router then
3          Associa rede e IP a essa interface;
4          Cria uma instância da interface e adiciona-a a uma lista de interfaces;
5      end
6      if Interface destino pertence a Router then
7          Associa a mesma rede mas IP diferente a essa interface;
8          Cria uma instância da interface e adiciona-a a uma lista de interfaces;
9      end
10 end
```

O algoritmo 2 representa a função `extractData()` na geração dos dados para as interfaces.

4.4.1.2 Função `checkIsRouter()`

Como explicado no capítulo anterior, foram retirados alguns pressupostos de modo a simplificar o desenvolvimento da aplicação. Um deles foi somente a consideração de *routers* para a realização da configuração. Portanto os únicos dados a serem armazenados relativamente às conexões e nós existentes na topologia são os dados referentes a interfaces de *routers* que integram essas conexões e a nós que sejam *routers*.

A função `checkIsRouter()` utiliza os dados referentes ao nó e determina se o nó em análise é um *router* ou não. A forma como a função o determina é através da interpretação dos dados dos nós em análise, no ficheiro JSON. Todos os nós no ficheiro de topologia JSON têm uma propriedade chamada *text* que determina o seu tipo. No caso dos *routers* essa propriedade começa com a designação "*Router XXXXX*", nos *switches* "*SW*"e nos *hosts* "*Host*". É através desta propriedade que função consegue distinguir os vários nós presentes na topologia e então devolver uma resposta afirmativa ou negativa quer o nó seja ou não um *router*.

No que toca a *routers* ALU, uma vez que estes são representados por máquinas virtuais, esta propriedade será "*QEMU VM*". Esta propriedade poderá causar ambiguidades na deteção de *routers* ALU, uma vez que podem existir outras máquinas virtuais QEMU que representem outros sistemas operativos que não de *routers* ALU.

4.4.1.3 Função `extractData()` - routers

Após o armazenamento da informação relativa às interfaces de *routers* integrantes nas conexões, a função *extractData()* passa a armazenar todos os nós que são *routers*. O ficheiro JSON contém uma propriedade denominada *nodes* onde estão definidos todos os nós existentes na topologia.

Por cada nó existente na topologia, a função verifica que esse nó pertence a um router, através da função `checkIsRouter()` e, caso este pertença, é criada uma instância da classe *Router* que armazena o seu tipo, o seu hostname e as interfaçes existentes na topologia.

Assim termina a fase de extração de dados do ficheiro JSON. Após esta fase são criadas as configurações para cada instância de *Router* na topologia.

O algoritmo 3 representa a função `extractData()` na geração dos dados para os *Routers*.

Algorithm 3: `extractData() (Router)`

Data: Ficheiro JSON

Result: Lista de objetos *Router*

```
1 while Existem 'nodes' do
2     if checkIsRouter() devolver Router then
3         Cria uma instância Router com dados JSON;
4         Adiciona a instância a uma lista de Routers;
5     end
6 end
```

4.4.2 Configuração automática

Após o processamento do ficheiro JSON, a criação das interfaces com IP atribuído e a criação dos *routers*, a aplicação já tem armazenada toda a informação necessária para proceder à criação das configurações para cada um dos *routers* na topologia. Esta configuração é realizada através da função `createConfiguration()` que é definida para a classe *Router*, ou seja, cada instância da classe *Router* consegue chamar a função e gerar a sua configuração automaticamente.

4.4.2.1 `createConfiguration()`

É nesta função que é criada a configuração para cada router. No final da execução da aplicação, todos os *routers* encontrados vão chamar a função `createConfiguration()` e gerar a configuração para a sua instância.

O algoritmo 4 ilustra os passos efetuados durante a função, que irão ser descritos posteriormente.

O primeiro passo na função é a criação do ficheiro de configuração, que irá ter o nome do *router* seguido da extensão '*.cfg*'. O ficheiro é criado no directório especificado no ficheiro de especificação auxiliar, caso este exista (figura 4.1). Dentro do directório, todas as configurações são colocadas numa pasta com o nome do projeto definido no ficheiro JSON, seguido da data de execução da aplicação (exemplo: '`testTopology_2015-06-27__16-13-32`'). O nome da pasta onde as configurações são criadas é definido no *script* principal.

Após a criação do ficheiro de configuração na pasta definida pelo ficheiro de especificação,

todos os comandos definidos no ficheiro base são copiados para o ficheiro de configuração. A designação do *router* é o primeiro comando a ser definido. Este comando é executado para ser possível fazer a associação do *router* criado na topologia com o *router* que irá receber a configuração, quer no ambiente virtualizado ou real. Finalmente, os novos comandos de configuração são agora adicionados.

De seguida são efectuadas as configurações definidas no ficheiro de especificação auxiliar. Estas configurações no ficheiro estão separadas por vírgulas, pelo que a função assume que cada configuração a efetuar está definida entre vírgulas. As configurações estão intrísecamente ligadas com os *templates*. É nesta fase que os *templates* entram em acção.

Os *templates* são definidos com o modelo do *router* como nome de ficheiro, seguido pela extensão que designa a configuração que o *template* realiza. Por exemplo, o *template* que realiza a configuração de uma interface para o modelo de router '`Cisco123X`' irá ter como nome '`Cisco123X.interface`'. Esta definição permite que sejam adicionados novos *templates* e modelos sem que se tenha de alterar o código para os suportar. Como cada configuração tem um *template* associado, a aplicação usa um *template* específico para a configuração e modelo a configurar.

Como foi referido anteriormente na secção 3.2, os *templates* usam máscaras para definir os locais no comando a implementar que devem ser substituídos por configurações guardadas ou geradas pela aplicação. Por exemplo, a máscara `__INTERFACE__` irá ser substituída pela interface pertencente ao *router* a configurar, como por exemplo *Serial 0/0/0*. Esta substituição é realizada com recurso a expressões regulares (exemplo: `$row = s/__REVERSEMASK__/$reverseMask/;`).

O uso de *templates* possibilita uma redução da complexidade e tamanho do código de criação das configurações, mantendo também a heterogeneidade da aplicação. Devido ao pressuposto descrito no capítulo 3, sempre que se configuram os dados de uma interface serial, é inserido o comando '*clock rate*'.

Devido à necessidade desta heterogeneidade, é utilizado apenas um ciclo para a implementação das substituições das máscaras por cada *template* e configuração a efectuar. Este ciclo vai correr por cada configuração definida no ficheiro de especificação auxiliar. De seguida, para cada configuração definida são iteradas todas as interfaces pertencentes ao *router* e aplicadas as configurações definidas para cada interface usando o template definido para essa configuração (seja da definição da interface ou do protocolo de encaminhamento) e para o modelo de *router* específico. Tal acontece porque através das configurações de IP e máscara geradas para cada interface é possível configurar ambas as interfaces e o protocolo

de encaminhamento para a rede à qual a interface está ligada.

Algorithm 4: Função `createConfiguration()`

Data: Lista Interfaces, Objeto *Router*, Directorio ficheiro base, Directorio destino
Result: Ficheiro de configuração para o *router*

```
1  nomeFicheiroConfiguração = hostnameRouter + ".cfg";
2  ficheiroBase = Directorio ficheiro base + "\"+ modeloRouter + ".baseConfig";
3  if Directorio destino definido no ficheiro de especificação then
4      nomeFicheiroConfiguração = Directorio destino + "\"+ nomeFicheiroConfiguração;
5      Abre o ficheiro para escrita;
6  else
7      Abre o ficheiro para escrita;
8  end
9  abrir ficheiro de configuração base;
10 while Ler linha a linha do ficheiro de configuração base do
11     if Linha para hostname then
12         escreve o hostname no ficheiro de configuração do router;
13     else
14         escreve linha no ficheiro de configuração do router;
15     end
16 end
17 listaConfiguraçõesAEfectuar = configurações fornecidas pelo ficheiro base;
18 for cada configuração da lista do
19     templateConfiguração = "templates\"+ hosnameRouter + "."+ configuração;
20     for cada interface pertencente ao Router do
21         Calcula rede através do IP;
22         Calcula Máscara através da rede;
23         Calcula a WildCard da máscara através da máscara;
24         for Cada linha do templateConfiguração do
25             substituir o valor na máscara do templateConfiguração;
26             escrever linha no ficheiro de configuração;
27         end
28         if template para configurar interface e interface é 'serial' then
29             escrever o valor de Clock Rate;
30         end
31     end
32     fechar ficheiro de configuração;
33 end
```

Todas as configurações de interface e protocolo de encaminhamento para cada interface existente para o *router* são então adicionadas ao ficheiro de configuração gerado, e a função *createConfiguration()* termina para a instância que o invocou.

A função *createConfiguration()* desenvolvida em Perl pode ser vista na secção de anexos. Esta função é a última fase da execução da aplicação. Após o seu término para cada instância de *router* existente na topologia definida no ficheiro JSON, a pasta criada pela aplicação irá conter todos os ficheiros de configuração para os *routers*. A sua implementação nos *routers* virtualizados ou reais irá fazer com que a rede funcione e comunique entre si.

4.5 Integração com o GNS3

Uma vez que a aplicação desenvolvida está intimamente ligada com o GNS3 e, sendo esta uma aplicação *opensource*, foi planeada a integração como extensão da aplicação com o GNS3. Após contacto com a equipa de desenvolvimento do GNS3 e da percepção da forma como o GNS3 funciona em termos de código, os passos a realizar para integração estão descritos de seguida.

O GNS3 está desenvolvido na linguagem Python (`http://www.python.org/`). Como o GNS3 contém uma interface gráfica desenvolvida em Python, ao acrescentar alguns parâmetros ao código fonte do GNS3, será possível utilizar a aplicação como uma opção no menu '*Tools*' do GNS3. Em sistemas operativos Windows, ao realizar o *download* da aplicação, obtemos um ficheiro '*.exe*' que não pode ser alterado. Portanto, para conseguir realizar alterações no código fonte do GNS3 é necessário obter a versão para sistemas Linux, que vem em formato '*zip*' e necessita de ser descomprimido e instalado manualmente. Existe um tutorial de como realizar a instalação manual em sistemas Linux, onde são descritos os passos de como instalar manualmente todos os ficheiros e componentes do GNS3 [32]. Na versão para Linux temos acesso a todos os ficheiros e código que compõe o GNS3, desde o seu funcionamento interno à sua interface gráfica. O ficheiro de código que comanda os aspetos a aparecer na janela principal é o '*main_window.py*'. É neste ficheiro de código onde estão definidos os menus a aparecer na interface gráfica do GNS3. Para adicionar mais um menu, é necessário criar uma '*Action*' para a aplicação e ligá-la a um '*slot*', que é chamado quando é feito um clique na opção. A figura 4.5 ilustra como são definidas no código fonte do GNS3 as opções de menu, as diferentes '*Actions*' para as opções e os '*slots* associados.

Os '*slots*' são invocados quando é feito clique na opção escolhida. A figura 4.6 mostra

```
# control menu connections
self.uiStartAllAction.triggered.connect(self._startAllActionSlot)
self.uiSuspendAllAction.triggered.connect(self._suspendAllActionSlot)
self.uiStopAllAction.triggered.connect(self._stopAllActionSlot)
self.uiReloadAllAction.triggered.connect(self._reloadAllActionSlot)
self.uiAuxConsoleAllAction.triggered.connect(self._auxConsoleAllActionSlot)
self.uiConsoleAllAction.triggered.connect(self._consoleAllActionSlot)
# device menu is contextual and is build on-the-fly
self.uiDeviceMenu.aboutToShow.connect(self._deviceMenuActionSlot)
# tools menu connections
self.uiVPCSAction.triggered.connect(self._vpcsActionSlot)
```

Figura 4.5: Definição das opções de menu no GNS3

alguns exemplos de código nos '*slots*' existentes do GNS3.

```
def _suspendAllActionSlot(self):
"""
Slot called when suspending all the nodes.
"""

for item in self.uiGraphicsView.scene().items():
if isinstance(item, NodeItem) and hasattr(item.node(), "suspend")
and item.node().initialized():
item.node().suspend()

def _stopAllActionSlot(self):
"""
Slot called when stopping all the nodes.
"""

for item in self.uiGraphicsView.scene().items():
if isinstance(item, NodeItem) and hasattr(item.node(), "stop")
and item.node().initialized():
item.node().stop()
```

Figura 4.6: Definição dos *slots* no GNS3

A definição de uma '*Action*' e um '*Slot*' para a aplicação desenvolvida no âmbito desta disseração irá permitir que a mesma fique disponível para execução no GNS3. No entanto

existem parâmetros para definir sempre que a aplicação é executada, como por exemplo o diretório destino, as configurações a executar e a localização dos ficheiros base. O ficheiro JSON do GNS3 pode ser acedido através do código da aplicação e enviado como parâmetro. A solução passa então por criar uma sub-aplicação que cria o ficheiro de especificação e posteriormente executa a aplicação com esse ficheiro.

A função, no formato de um *script* denominado '*genSpecFile.pl*' foi criada para este efeito. Recebe como parâmetro o ficheiro JSON da topologia em questão e, através da especificação dos parâmetros desejados, cria o ficheiro de especificação para a topologia e executa-o automaticamente.

Para melhor automatização do processo de integração também é possível substituir os ficheiros que o GNS3 usa para cada *router* na topologia criada pelos ficheiros gerados pela aplicação. O GNS3 guarda os ficheiros de configuração de todos os nós em formato de texto, cuja localização pode ser obtida através do ficheiro JSON. Assim, a aplicação poderá substituir automaticamente os ficheiros dos *routers* configurados, ficando já carregados no GNS3 e colocando a rede virtualizada em funcionamento. Isto poderá ser possível através do parâmetro '*AutoSubstituteGNS3Configs*' no ficheiro de especificação. Deverá ser necessário no entanto que o ficheiro JSON esteja no local pré-definido pelo GNS3, de modo a que as configurações possam ser substituídas. Numa execução realizada pelo GNS3 irá sempre acontecer.

A implementação da integração no GNS3 não foi realizada uma vez que o ambiente de desenvolvimento e de teste estava assentado no sistema operativo Windows, o que não permitia o acesso ao código fonte do GNS3 pelas razões descritas anteriormente.

Este capítulo descreveu a fase de desenvolvimento da aplicação,tratando os aspetos principais tidos em conta durante o desenvolvimento. Foram descritas algumas decisões tomadas durante a fase e as funções principais presentes na aplicação. Concluída a fase de desenvolvimento, o próximo capítulo irá descrever os testes realizados à aplicação para apurar a sua eficácia e rapidez na geração das configurações.

Capítulo 5

Testes

Este capítulo trata a fase de testes efectuados à aplicação desenvolvida ao longo da dissertação. Serão abordados os testes de aceitação, tempo de execução, carga na máquina, entre outros considerados importantes. No final de cada iteração de testes, é tecida uma conclusão sobre os resultados obtidos. Esta fase é de grande importância pois permite auferir o cumprimento dos objetivos definidos ao longo do desenvolvimento da aplicação e a sua mais valia como alternativa à configuração manual de equipamentos de rede.

5.1 Ambiente de teste

Para realizar o teste da aplicação face a vários cenários, foi definido o ambiente a ser utilizado para a realização dos testes. Os testes foram conduzidos numa só máquina, cujas especificações podem ser verificadas na tabela 5.1.

Tabela 5.1: Especificações do computador de teste

Componentes	Definição
CPU	Intel Core i5-3337U 1.8GHz (Ivy Bridge)
Motherboard	ASUSTeK X550CL
RAM	2x2048MB DDR3 1600MHz
Disco Rígido	Travelstar Z5K500 500Gb 5400RPM
SO	Windows 7 Ultimate 64bits
Distribuição Perl	*Strawberry Perl* 5.22.0.1 (64 bit)
Versão GNS3	GNS3 1.3.4

O ambiente de teste foi mantido inalterado durante a fase de testes de modo a que todos os dados fossem retirados de uma só fonte e fossem assim coerentes entre si, não existindo discrepâncias no software e hardware utilizado durante os testes.

5.2 Testes de aceitação

Os primeiros testes a serem realizados durante o desenvolvimento tinham o objetivo de verificar a correta criação de configurações automaticamente, que pudessem ser posteriormente carregadas nos *routers* com o bojetivo de ter a rede a funcionar corretamente. Este tipo de testes são designados testes de aceitação ou *blackbox*, onde o conhecimento do funcionamento interno não é considerado e onde só é dada importância ao *output* da aplicação. Para a realização destes destes, foram desenhados diferentes cenários de teste onde a aplicação foi executada, para ser verificada a criação das configurações individuais de cada *router* presente. Posteriormente foram realizados testes de comunicação e conectividade entre os elementos da rede de modo a verificar a correta configuração da rede. A figura 5.1 ilustra um cenário de teste criado no GNS3 com 4 *routers Cisco c2691*, onde foi executada a aplicação e realizado o teste de aceitação.

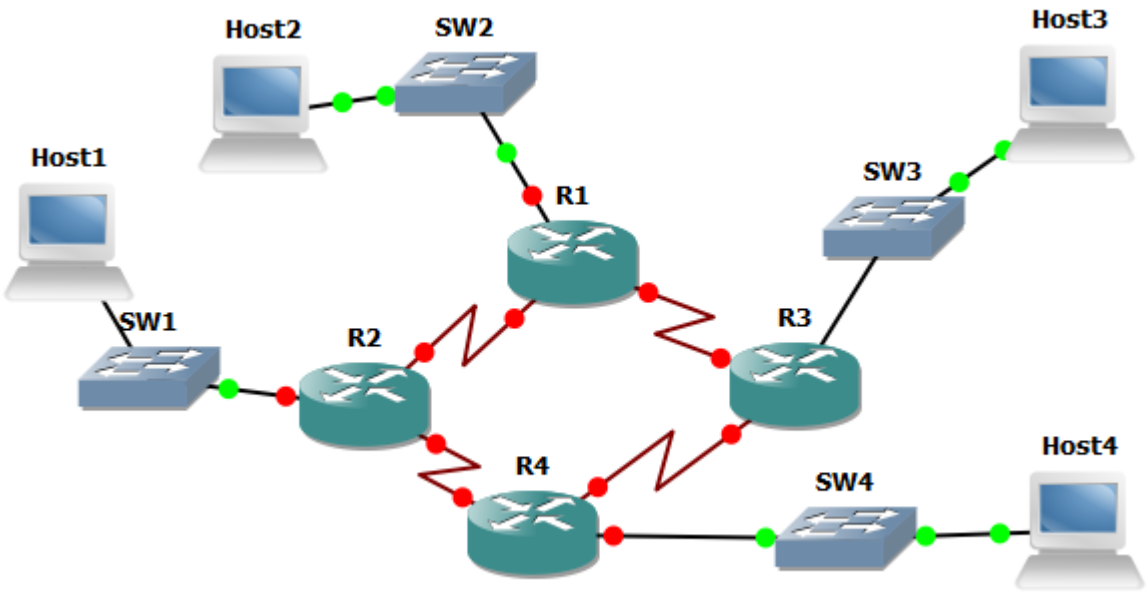

Figura 5.1: Cenário para teste de aceitação

Ao ser executada a aplicação com base neste cenário, o *output* esperado pela aplicação é

uma pasta criada num diretório definido pelo ficheiro de *.cfg* de especificação contendo 4 ficheiros de configuração com nomes relativos aos *routers* criados no GNS3. Para isso a aplicação interpretou corretamente o ficheiro de especificação que é usado como parâmetro de entrada. Na figura 5.2 está definido o ficheiro de especificação utilizado para o cenário de teste especificado na figura 5.1.

```
ProjectDir = C:\Users\Script\scenario.gns3
BaseConfigPath = C:\Users\Basefiles\
OutputDir = C:\Users\
Configurations = interface,ospf
```

Figura 5.2: Ficheiro de especificação para o cenário de teste

Ao interpretar o ficheiro de entrada ilustrado na figura 5.2, podemos esperar que o *output* da aplicação deverá ser criado no directório *C:\Users*, que foi especificado pelo parâmetro *OutputDir*, na forma de uma pasta com o nome do projeto criado no GNS3 seguido da data de execução da aplicação. Também é esperado que o conteúdo dos ficheiros base específicos para os *routers* utilizados na topologia, presente no diretório *C:\Users\Basefiles*, seja copiado para as configurações finais. Ou seja, que o ficheiro *Router c2691.baseConfig*, caso esteja presente no diretório seja utilizado para a configuração base dos *routers Cisco c2691* utilizados. Finalmente também são esperados que os comandos para as configurações de interface e *OSPF* estejam presentes na configuração final dos *routers*.

Após a execução da aplicação, foi verificada a criação da pasta no directório indicado pelo ficheiro de especificação, contendo as 4 configurações a serem instaladas nos *routers*. As figuras 5.3,5.4,5.5 e 5.6 mostram as configurações geradas para os *routers* R1, R2, R3 e R4, respectivamente. Nestas configurações não são contempladas as configurações copiadas do ficheiro de configuração base, embora tenham sido copiadas com êxito.

Analisando a figura 5.3 é possível observar que ao *router* R1 foram atribuídas as redes 192.168.5.0/24, 192.168.6.0/24 e 192.168.12.0/24, onde foi atribuído o seu endereço IP específico na ligação. Nas ligações *serial* está descrito o *router* destino ao qual o *router* R1 está ligado, que neste caso é o *router* R2 e R3 pelas interfaces *Serial0/1* e *Serial0/0*, respectivamente. Finalmente também está configurada a ligação ao *switch* SW2 e o protocolo definido, sendo divulgadas as redes atribuídas por *OSPF*.

```
!
hostname R1
!
interface Serial0/1
ip address 192.168.5.2 255.255.255.0
description Connection to R2
no shutdown
clock rate 4000000
!
interface Serial0/0
ip address 192.168.6.1 255.255.255.0
description Connection to R3
no shutdown
clock rate 4000000
!
interface FastEthernet0/0
ip address 192.168.12.2 255.255.255.0
description Connection to SW2
no shutdown
!
router ospf 1
network 192.168.5.0 0.0.0.255 area 0
network 192.168.6.0 0.0.0.255 area 0
network 192.168.12.0 0.0.0.255 area 0
!
end
```

Figura 5.3: Ficheiro de configuração para o *router* R1

Na figura 5.4 pode ser observado que o *router* R2 ficou ligado às redes 192.168.4.0/24, 192.168.5.0/24 e 192.168.10.0/24, onde foi atribuído o seu endereço IP específico na ligação. Como o *router* R2 está ligado ao R1 na rede 192.168.5.0/24, pode ser constatado que foram atribuídos endereços IP diferentes aos dois *routers*, 192.168.5.1 no *router* R2 e 192.168.5.2 no *router* R1, não havendo colisão na atribuição dos endereços.

Este *router* também está ligado por ligação *serial0/0* ao *router* R4 na rede 192.168.4.0/24 pelo IP 192.168.4.1/24 e por *FastEthernet0/0* ao *switch* SW1 pelo IP 192.168.10.2 e rede 192.168.10.0/24. O protocolo de encaminhamento *OSPF* também se encontra corretamente configurado, havendo a divulgação das redes referidas.

```
!
hostname R2
!
interface Serial0/0
ip address 192.168.4.1 255.255.255.0
description Connection to R4
no shutdown
clock rate 4000000
!
interface Serial0/1
ip address 192.168.5.1 255.255.255.0
description Connection to R1
no shutdown
clock rate 4000000
!
interface FastEthernet0/0
ip address 192.168.10.2 255.255.255.0
description Connection to SW1
no shutdown
!
router ospf 1
network 192.168.4.0 0.0.0.255 area 0
network 192.168.5.0 0.0.0.255 area 0
network 192.168.10.0 0.0.0.255 area 0
!
end
```

Figura 5.4: Ficheiro de configuração para o *router* R2

A figura 5.5 ilustra a configuração gerada para o *router* R3. Estão contempladas as interfaces com ligações a nós presentes na rede, tendo sido utilizadas para este *router* as redes IP 192.168.6.0/24, 192.168.7.0/24 e 192.168.8.0/24 para ligação aos nós *router* R1, R4 e *switch* SW3. Mais uma vez é possível observar a não colisão de endereços de IP na ligação ao *router* R1 e a correta configuração do protocolo *OSPF*, onde as redes utilizadas são divulgadas.

```
!
hostname R3
!
interface Serial0/0
ip address 192.168.6.2 255.255.255.0
description Connection to R1
no shutdown
clock rate 4000000
!
interface Serial0/1
ip address 192.168.7.1 255.255.255.0
description Connection to R4
no shutdown
clock rate 4000000
!
interface FastEthernet0/0
ip address 192.168.8.2 255.255.255.0
description Connection to SW3
no shutdown
!
router ospf 1
network 192.168.6.0 0.0.0.255 area 0
network 192.168.7.0 0.0.0.255 area 0
network 192.168.8.0 0.0.0.255 area 0
!
end
```

Figura 5.5: Ficheiro de configuração para o *router* R3

Finalmente na figura 5.6 conseguem-se observar as configurações geradas pela aplicação para o *router* R4. As redes 192.168.4.0/24, 192.168.7.0/24 e 192.168.9.0/24 foram utilizadas para configuração deste *router*. Observa-se que os endereços gerados para o *router* R4 não colidem os dos outros *routers* ao qual este está ligado, mantendo-se a coerência dos endereços na rede. O endereço 192.168.4.2 é utilizado face ao endereço 192.168.4.1 existente no *router* R2, tal como o endereço 192.168.7.2 conjuga com o endereço 192.168.7.1 existente no *router* R3. Também a interface e protocolo de encaminhamento *OSPF* se encontram bem configurados e coerentes.

```
!
hostname R4
!
interface Serial0/0
ip address 192.168.4.2 255.255.255.0
description Connection to R2
no shutdown
clock rate 4000000
!
interface Serial0/1
ip address 192.168.7.2 255.255.255.0
description Connection to R3
no shutdown
clock rate 4000000
!
interface FastEthernet0/0
ip address 192.168.9.2 255.255.255.0
description Connection to SW4
no shutdown
!
router ospf 1
network 192.168.4.0 0.0.0.255 area 0
network 192.168.7.0 0.0.0.255 area 0
network 192.168.9.0 0.0.0.255 area 0
!
end
```

Figura 5.6: Ficheiro de configuração para o *router* R4

A figura 5.7 permite uma visão de como os endereços IP são utilizados no cenário e como a topologia fica configurada ao serem aplicadas as configurações nos *routers*.

Todas as configurações de endereços de rede IP foram geradas automaticamente pela aplicação. Após o carregamento das configurações nos *routers*, todos os intervenientes na rede comunicaram entre si, havendo conectividade entre todos os nós do cenário. Assim, foi verificado o funcionamento correto da aplicação, cumprindo todos os parâmetros a nível funcional definidos para o projeto.

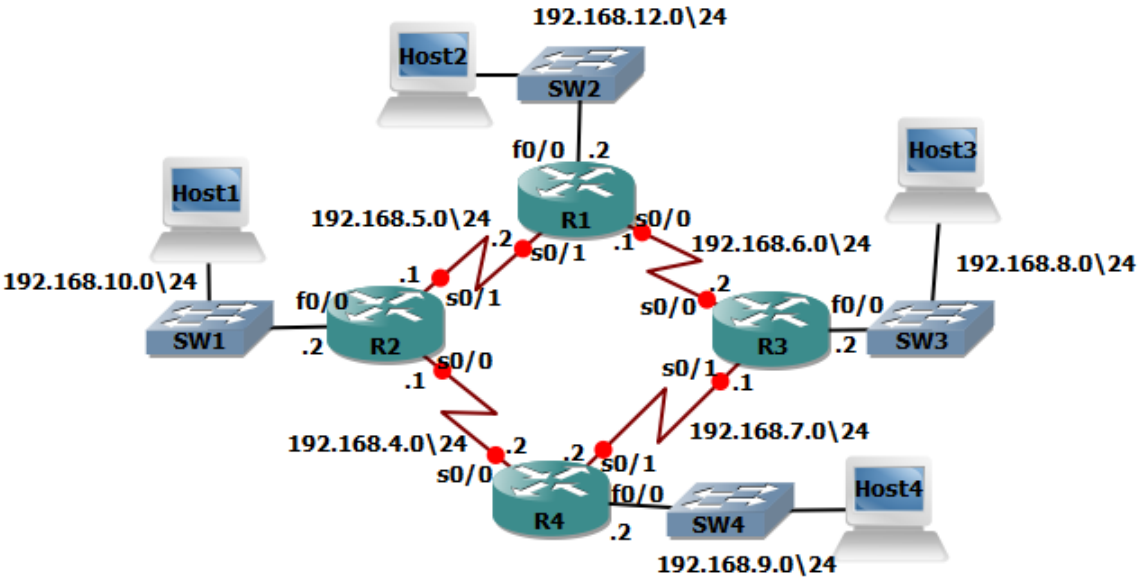

Figura 5.7: Cenário geral para testes de aceitação

5.3 Testes de desempenho.

Um dos desafios mais importantes a ser cumprido com a aplicação é a redução da morosidade do processo de configuração dos equipamentos. Cada *router* necessita ser configurado individualmente com os comandos específicos para o seu papel na rede. Este processo é bastante moroso quando realizado manualmente e está sempre sujeito a erros humanos. A automação e aceleração deste processo é então um grande fator a considerar na fase de testes.

A tabela 5.2 mostra o tempo que a aplicação demorou a executar as tarefas de leitura e criação das configurações face a cenários com diferentes quantidades de *routers*. Cada cenário foi testado um mínimo de 10 vezes, onde foi apontado o tempo de cada execução em milissegundos e posteriormente realizada a média do tempo de execução. Para cronometragem do tempo de execução foi utilizado o módulo *Time::HiRes*, que fez leitura do tempo de execução da aplicação desde que é executada até terminar.

É possível observar que o tempo de execução aumenta à medida que são inseridos mais elementos na rede, designadamente *routers*. A discrepância entre os tempos de cada execução não é muito significativa, havendo uma uniformidade entre os tempos registados dentro de cada cenário testado.

O gráfico 5.8 ilustra a relação entre a quantidade de *routers* presentes na topologia e o tempo

Tabela 5.2: Tempos de execução na geração de configurações (milissegundos)

Routers	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Run #1	0.017	0.028	0.061	0.166	0.488	1.696	5.735
Run #2	0.017	0.027	0.105	0.134	0.471	1.656	6.122
Run #3	0.016	0.026	0.065	0.160	0.450	1.461	6.006
Run #4	0.012	0.023	0.074	0.164	0.652	1.640	6.204
Run #5	0.015	0.025	0.065	0.163	0.542	1.447	5.699
Run #6	0.014	0.031	0.041	0.164	0.585	1.700	5.822
Run #7	0.015	0.022	0.077	0.169	0.487	1.577	6.086
Run #8	0.014	0.024	0.097	0.129	0.520	1.641	6.050
Run #9	0.014	0.027	0.086	0.180	0.535	1.647	5.784
Run #10	0.013	0.023	0.112	0.149	0.648	1.678	6.069
Média	**0.015**	**0.026**	**0.078**	**0.158**	**0.538**	**1.614**	**5.958**

médio de execução da aplicação para a mesma, considerando as 10 experiências realizadas para cada cenário.

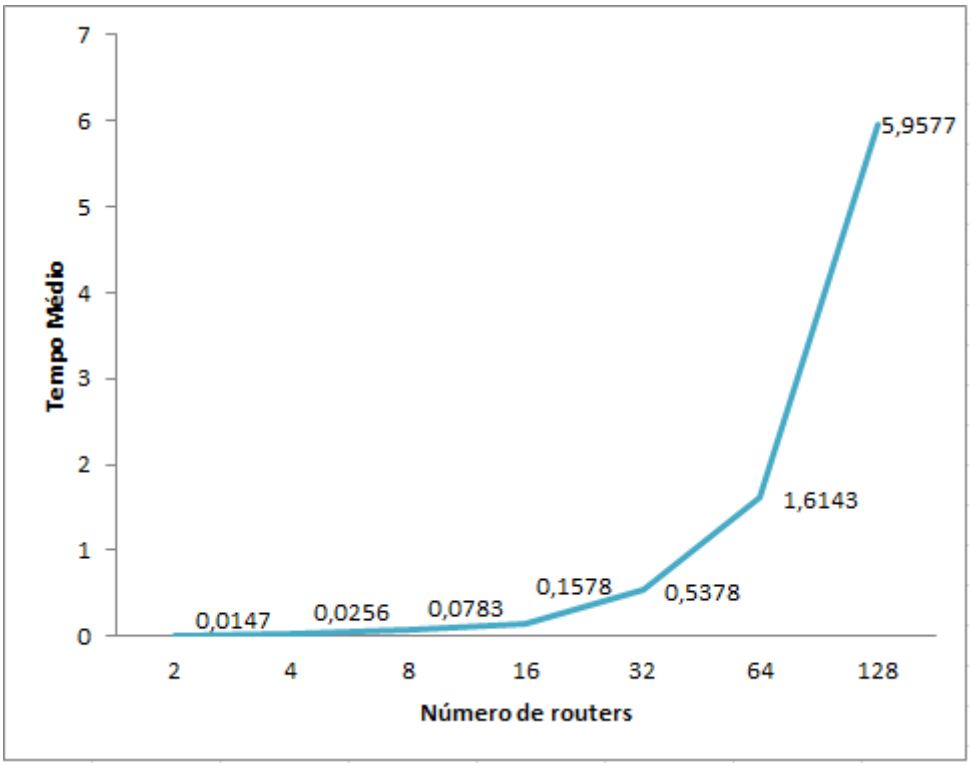

Figura 5.8: Média de tempos de execução

Foi escolhida a duplicação da quantidade de *routers* em cada cenário para melhor realizar

uma comparação entre o tempo de execução e a quantidade de *routers* existentes. É possível observar que o tempo de execução aumenta com a quantidade de *routers*. No entanto não há uma relação de crescimento direta entre o tempo de execução e a quantidade de *routers*. À medida que se difere entre cada cenário, a quantidade de *routers* duplica. Em contraste, o tempo de execução não duplica em conformidade com esse aumento, tem um crescimento mais acentuado, demorando até alguns segundos na execução em cenários com mais *routers*.

Este fato deve-se a problemas de otimização de código da aplicação. Para verificar onde a aplicação demora mais tempo a realizar tarefas foi utilizado o módulo *Devel::NYTProf*, que faz a auditoria do tempo de execução das várias rotinas e funções da aplicação, mostrando depois onde a aplicação consome mais tempo e recursos durante a sua execução. O uso deste módulo em conjunto com a aplicação faz com que esta demore um pouco mais de tempo na execução, devido ao registo dos vários tempos que cada função e rotina toma. A figura 5.9 mostra onde a aplicação consome mais tempo na execução, usando o cenário com 128 *routers* como parâmetro. A execução demorou 8.700 segundos usando o módulo.

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
84224	3	1	6.06s	6.06s	Router::CORE:open (opcode)
128	1	1	1.13s	8.00s	Router::createConfiguration
83968	1	1	339ms	339ms	Interface::router
256	2	1	256ms	256ms	Router::CORE:close (opcode)
84224	3	1	164ms	164ms	Router::hostname
713	3	1	153ms	153ms	main::checkIsRouter
1	1	1	84.4ms	211ms	main::BEGIN@19
16010	13	1	65.2ms	65.2ms	diagnostics::CORE:subst (opcode)
2074	2	1	35.5ms	35.5ms	diagnostics::CORE:readline (opcode)
1	1	1	29.2ms	30.3ms	DateTime::Locale::BEGIN@11
1	1	1	28.9ms	219ms	main::BEGIN@24
5752	2	1	19.3ms	19.3ms	Router::CORE:readline (opcode)
466	1	1	15.4ms	25.8ms	DateTime::Locale::_register
1	1	1	13.7ms	13.9ms	DateTime::TimeZone::BEGIN@8
12136	8	1	11.9ms	11.9ms	Router::CORE:subst (opcode)

Figura 5.9: Consumo de tempo em execução não otimizada (128 *routers*).

Como se pode observar, a rotina *Router::Core:open* ocupou cerca de 6 segundos na execução. Esta rotina tem como função tratar da abertura de ficheiros para leitura ou escrita. A demora nesta execução acontece devido a esta função consultar o disco rígido da máquina para leitura e escrita de ficheiros, que é um dos componentes mais lentos da máquina. Esta

função é chamada 84224 vezes, como pode ser observado na figura 5.9. Este número de chamadas à função é demasiado elevado, uma vez que só se abrem de 3 a 4 ficheiros por *router*. Este número mostra que existe alguma otimização a realizar na consulta ao disco.

Após a análise mais aprofundada ao código-fonte da aplicação, foi encontrada uma chamada à função de abertura de um ficheiro dentro de um ciclo de execução, como pode ser observado na figura 5.10.

```
for ($x=0;$x<scalar @routerInterfaces;$x++){
    open($tempFile, '<', $templateName);
```

Figura 5.10: Abertura de ficheiros dentro de ciclo.

O que esta função realizava era a abertura de um *template* sempre que era iterada uma interface da topologia. Após a otimização do código, retirando a chamada à função do ciclo, os tempos de execução e o número de chamadas reduziram consideravelmente, como pode ser observado na figura 5.11. A execução demorou apenas 1.88 segundos para o mesmo cenário de 128 *routers*.

Top 15 Subroutines

Calls	P	F	Exclusive Time	Inclusive Time	Subroutine
128	1	1	569ms	1.21s	Router::createConfiguration
512	3	1	240ms	240ms	Router::CORE:open (opcode)
713	3	1	152ms	152ms	main::checkIsRouter
83968	1	1	152ms	152ms	Interface::router
84224	3	1	134ms	134ms	Router::hostname
256	2	1	79.4ms	79.4ms	Router::CORE:close (opcode)
1	1	1	78.1ms	197ms	main::BEGIN@19
16010	13	1	59.6ms	59.6ms	diagnostics::CORE:subst (opcode)
2074	2	1	33.5ms	33.5ms	diagnostics::CORE:readline (opcode)
1	1	1	27.6ms	207ms	main::BEGIN@24
1	1	1	26.6ms	27.5ms	DateTime::Locale::BEGIN@11
466	1	1	14.5ms	24.2ms	DateTime::Locale::_register
1	1	1	13.4ms	13.6ms	DateTime::TimeZone::BEGIN@8
4752	2	1	13.4ms	13.4ms	Router::CORE:readline (opcode)
1	1	1	10.1ms	10.5ms	diagnostics::BEGIN@186

Figura 5.11: Consumo de tempo em execução otimizada (128 *routers*).

Pode ser observada a diminuição considerável no tempo e número de chamadas à função de abertura de ficheiros. Tendo alguns aspetos do código fonte otimizados, foi feita uma nova

cronometragem dos tempos de execução para os vários cenários de teste. Os novos tempos estão ilustrados na tabela 5.3

Tabela 5.3: Tempos de execução em código-fonte otimizado

Routers	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Run #1	0,016	0,025	0,024	0,039	0,063	0,188	0,507
Run #2	0,013	0,020	0,029	0,046	0,071	0,186	0,472
Run #3	0,012	0,022	0,027	0,042	0,073	0,179	0,585
Run #4	0,019	0,024	0,029	0,039	0,079	0,193	0,478
Run #5	0,012	0,023	0,027	0,039	0,074	0,234	0,495
Run #6	0,013	0,022	0,024	0,037	0,080	0,179	0,552
Run #7	0,013	0,023	0,029	0,035	0,084	0,195	0,492
Run #8	0,017	0,022	0,027	0,036	0,085	0,201	0,520
Run #9	0,016	0,021	0,027	0,037	0,063	0,211	0,525
Run #10	0,016	0,017	0,028	0,040	0,076	0,218	0,497
Média	**0,015**	**0,022**	**0,027**	**0,039**	**0,075**	**0,198**	**0,512**

Pode-se observar uma diminuição considerável no tempo de execução da aplicação face à tabela 5.2. No entanto, face à redução do tempo de execução, como é possível observar na figura 5.12, a barra de crescimento apresenta a mesma tendência no cenário não otimizado, embora não tão acentuada . Esta tendência de crescimento não é causa para preocupação, devido ao tempo de execução ser mais reduzido, o crescimento não ser tão acentuado e uma vez que o teste de cenários com mais que várias dezenas de *routers* não são comuns em casos de teste.

A comparação da média de tempo na solução não otimizada com a solução otimizada pode ser observada na figura 5.13.

Devido ao tempo de execução da aplicação face às diferentes topologias ser bastante reduzido, demorando menos de um segundo mesmo em cenários excessivamente grandes e complexos, não é realizada nenhuma carga excessiva em processamento no ambiente de teste. Os ambientes de teste também incluíam outros nós que não *routers*, como *switches* e *hosts*. Os nós foram adicionados de forma uniforme em todos os cenários, relativamente ao número de *routers*. Os dados destes nós não eram relevantes à aplicação, uma vez que não são utilizados para geração de configuração. No entanto, a existência de dados não relevantes no ficheiro JSON não constituiu um atraso no seu processamento, por parte da aplicação. A escolha do Perl, linguagem bastante conhecida pela sua rapidez no processamento de dados, foi então acertada para reduzir os tempos e otimizar o processamento de todos os dados utilizados na aplicação.

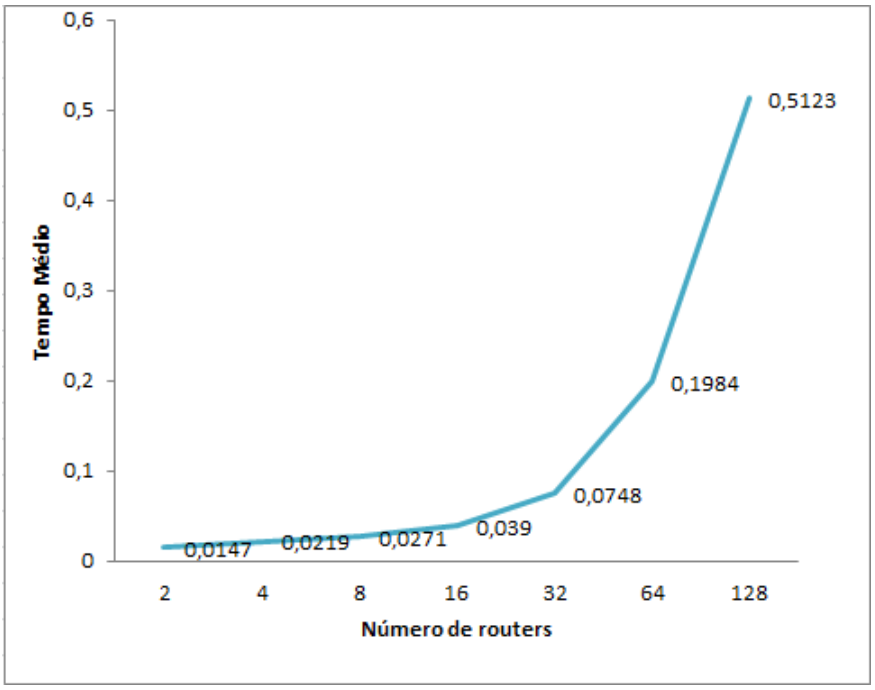

Figura 5.12: Média de tempos na solução otimizada.

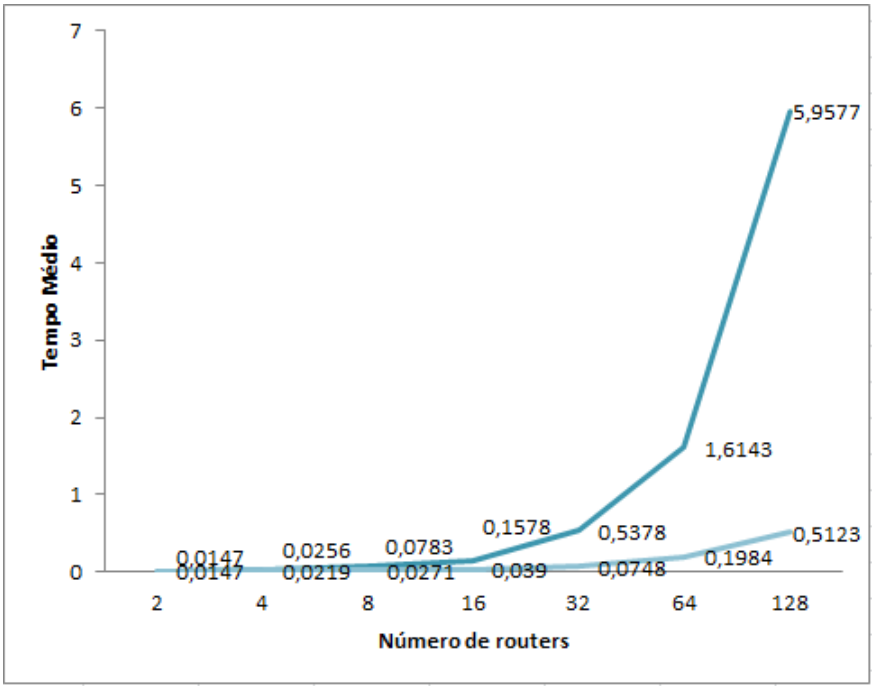

Figura 5.13: Comparaçao de médias.

5.4 Análise de resultados

Os testes realizados permitem concluir que a aplicação cumpre os objetivos de interpretação de uma topologia armazenada no ficheiro criado pelo GNS3, processa e gera os dados tendo em conta os *routers* e outros intervenientes na topologia e cria os ficheiros de configuração para cada *router* presente no cenário.

Esses ficheiros de configuração são válidos para utilizar em *routers*, quer sejam virtualizados, simulados ou reais, e permitem configurar automaticamente a rede desenhada. Todo este processo ocorre num curto espaço de tempo e com um desempenho aceitável. Os testes ao protótipo desenvolvido permitem concluir que é possível desenvolver um gerador automático simples que gera configurações num curto espaço de tempo e de forma e transparente para o utilizador, independentemente do número de elementos na rede. O desafio de uso de uma interface gráfica simples também foi ultrapassado, através do uso do GNS3, uma aplicação bastante completa, não só na componente gráfica mas também ao nível do leque de funções disponíveis para o utilizador.

Através do uso de *templates* e da capacidade da aplicação conseguir distinguir os diferentes nós, é possível concluir que o desafio da configuração de ambientes heterogéneos é possível ultrapassar, aproximando os cenários de teste aos ambientes heterogéneos reais.

Este capítulo tratou os testes realizados à aplicação desenvolvida e as conclusões retiradas dos mesmos. O capítulo seguinte apresenta as conclusões do trabalho realizado no âmbito da dissertação, e apresenta algumas indicações para trabalho futuro.

Capítulo 6

Conclusão

A Internet é um conjunto de redes ligadas entre si, através de equipamentos especializados como *routers* e *switches*. Todos estes equipamentos necessitam de ser configurados de forma específica, tendo em conta o seu objetivo na rede, por utilizadores especializados na área das redes de computadores. Devido à sua importância cada vez mais significativa nas tecnologias utilizadas hoje em dia, o tópico de redes de computadores é cada vez mais abordado nas áreas de ensino, onde os alunos têm acesso a vários equipamentos e aplicações para auxiliar na sua aprendizagem. São nos ambientes de ensino e também em ambientes empresariais onde vários cenários de teste são utilizados para aprendizagem e teste de novas configurações nos equipamentos de rede.

A necessidade da utilização e teste de vários cenários implica uma constante configuração dos equipamentos presentes nos mesmos, seja para realização de configurações mais básicas ou para implementação de cenários mais complexos para testes e aprendizagem. Esta configuração constante e repetitiva está sujeita à ocorrência de erros humanos, que poderiam ser evitados com a utilização de uma abordagem automática. Foi então proposta uma abordagem que consiste na geração de configurações automáticas para os equipamentos de uma dada topologia, evitando assim a repetição de equipamentos e a necessidade de conhecimento do funcionamento de vários tipos e modelos de equipamentos.

Foram investigadas várias aplicações de configuração automática de equipamentos, de modo a auferir a necessidade de desenvolvimento e melhoria de aspetos em falta nas aplicações de configuração automática dos equipamentos. Foram encontrados vários pontos a melhorar, nomeadamente a não existência de uma interface gráfica onde fosse possível o desenho da topologia a testar, a não geração de ficheiros que pudessem ser carregados diretamente nos equipamentos de modo a colocá-los em funcionamento e também a falta de heterogeneidade

no suporte de configuração de equipamentos de diferentes vendedores, que normalmente existem nas redes. Estes vários aspetos encontrados serviram de motivação ao desenho e desenvolvimento de uma aplicação que conseguisse colmatar estes desafios.

O desenvolvimento efetuado realiza a geração de configurações automáticas para equipamentos de rede, através de uma aplicação *opensource*, desenvolvida na linguagem Perl. A aplicação realiza a configuração de topologias de rede ao gerar as configurações para os *routers* presentes, independentemente do tamanho da rede. Como suporte gráfico foi utilizado o GNS3, que é uma aplicação de desenho e virtualização de redes. A aplicação usa assim o ficheiro de especificação no formato JSON criado pelo GNS3 como base para a geração das configurações para a rede desenhada. A aplicação também deteta a existência de equipamentos de diferentes vendedores, como *routers* ALU. A aplicação está abrangida pela licensa GPL e está disponível no serviço *sourceforge* através do link (`http://rcg-tool.sourceforge.net/`).

Os testes realizados com a aplicação provam que a mesma cumpre os requisitos de geração de configurações para os *routers* de uma dada rede definida, que podem ser aplicadas em equipamentos de rede, colocando a topologia em funcionamento e a comunicar. Foram realizados testes funcionais para testar a correta interpretação dos parâmetros dados à aplicação e a geração de configurações válidas para a topologia de rede desejada. Nestes testes, a aplicação gerava automaticamente as configurações para os *routers* e, ao colocar as configurações de volta ao GNS3, todos os elementos na topologia fornecida comunicavam entre si. Foram realizados também testes de desempenho e carga ao testar a aplicação em cenários com várias quantidades de *routers*, onde se media o tempo de execução e a correta interpretação dos dados e geração de configurações.

A aplicação atingiu os objetivos definidos nesta dissertação, ao gerar automaticamente e de forma rápida configurações completas para *routers*, sejam virtualizados ou reais, interpretando corretamente uma topologia fornecida, através. A aplicação utiliza corretamente os *templates*, que podem ser modificados ou acrescentados para adicionar novos comandos e funcionalidades aos *routers*.

Para além do trabalho realizado, ainda estão definidos alguns passos a realizar no futuro para melhorar a aplicação e o seu potencial. Os passos são descritos de seguida:

- Gerar uma configuração base otimizada, tendo em conta o número de nós e as ligações entre eles. Por exemplo, usar uma rede /30 para otimizar a atribuição de endereços IP em ligações ponto a ponto.
- Adicionar mais templates com a implementação de diferentes configurações que po-

dem ser aplicadas em *routers*.

- Otimizar a configuração de redes heterogéneas. A aplicação consegue detetar os diferentes tipos de *routers*, embora o uso de *routers* ALU no GNS3 não esteja totalmente otimizado.

Apêndice A

Anexos

A.1 Exemplo de ficheiro JSON criado pelo GNS3


```
{
    "auto_start": false,
    "name": "cenario1-3routers",
    "resources_type": "local",
    "topology": {
        "links": [
            {
                "description": "Link from Host1 port
nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local to SW1 port 1",
                "destination_node_id": 4,
                "destination_port_id": 16,
                "id": 1,
                "source_node_id": 5,
                "source_port_id": 27
            },
            {
                "description": "Link from SW1 port 2 to R2 port FastEthernet0/0",
                "destination_node_id": 2,
                "destination_port_id": 6,
                "id": 2,
                "source_node_id": 4,
                "source_port_id": 17
            },
```

```
            {
                "description": "Link from R2 port Serial0/0 to R1 port Serial0/0",
                "destination_node_id": 1,
                "destination_port_id": 3,
                "id": 3,
                "source_node_id": 2,
                "source_port_id": 8
            },
            {
                "description": "Link from R1 port Serial0/1 to R3 port Serial0/1",
                "destination_node_id": 3,
                "destination_port_id": 15,
                "id": 4,
                "source_node_id": 1,
                "source_port_id": 5
            },
            {
                "description": "Link from R3 port Serial0/0 to R2 port Serial0/1",
                "destination_node_id": 2,
                "destination_port_id": 10,
                "id": 5,
                "source_node_id": 3,
                "source_port_id": 13
            },
            {
                "description": "Link from R3 port FastEthernet0/0 to Host2 port
nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local",
                "destination_node_id": 6,
                "destination_port_id": 31,
                "id": 6,
                "source_node_id": 3,
                "source_port_id": 11
            }
        ],
        "nodes": [
            {
                "description": "Router c2691",
                "id": 1,
                "label": {
                    "color": "#000000",
```

```
            "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
            "text": "R1",
            "x": 20.5,
            "y": -25.0
        },
        "ports": [
            {
                "id": 1,
                "name": "FastEthernet0/0",
                "port_number": 0,
                "slot_number": 0
            },
            {
                "id": 2,
                "name": "FastEthernet0/1",
                "port_number": 1,
                "slot_number": 0
            },
            {
                "description": "connected to R2 on port Serial0/0",
                "id": 3,
                "link_id": 3,
                "name": "Serial0/0",
                "nio": "NIO_UDP",
                "port_number": 16,
                "slot_number": 0
            },
            {
                "id": 4,
                "name": "FastEthernet1/0",
                "port_number": 0,
                "slot_number": 1
            },
            {
                "description": "connected to R3 on port Serial0/1",
                "id": 5,
                "link_id": 4,
                "name": "Serial0/1",
                "nio": "NIO_UDP",
                "port_number": 32,
```

```
            "slot_number": 0
        }
    ],
    "properties": {
        "aux": 2501,
        "console": 2001,
        "disk0": 1,
        "exec_area": 16,
        "image": "c2691-adventerprisek9-mz.124-25d.image",
        "mac_addr": "c001.13c4.0000",
        "name": "R1",
        "nvram": 256,
        "private_config": "configs\\i1_private-config.cfg",
        "ram": 128,
        "slot1": "NM-1FE-TX",
        "startup_config": "configs\\i1_startup-config.cfg",
        "wic0": "WIC-1T",
        "wic1": "WIC-1T"
    },
    "router_id": 1,
    "server_id": 1,
    "type": "C2691",
    "x": -106.0,
    "y": -129.0
},
{
    "description": "Router c2691",
    "id": 2,
    "label": {
        "color": "#000000",
        "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
        "text": "R2",
        "x": 20.5,
        "y": -25.0
    },
    "ports": [
        {
            "description": "connected to SW1 on port 2",
            "id": 6,
            "link_id": 2,
```

```
            "name": "FastEthernet0/0",
            "nio": "NIO_UDP",
            "port_number": 0,
            "slot_number": 0
        },
        {
            "id": 7,
            "name": "FastEthernet0/1",
            "port_number": 1,
            "slot_number": 0
        },
        {
            "description": "connected to R1 on port Serial0/0",
            "id": 8,
            "link_id": 3,
            "name": "Serial0/0",
            "nio": "NIO_UDP",
            "port_number": 16,
            "slot_number": 0
        },
        {
            "id": 9,
            "name": "FastEthernet1/0",
            "port_number": 0,
            "slot_number": 1
        },
        {
            "description": "connected to R3 on port Serial0/0",
            "id": 10,
            "link_id": 5,
            "name": "Serial0/1",
            "nio": "NIO_UDP",
            "port_number": 32,
            "slot_number": 0
        }
    ],
    "properties": {
        "aux": 2502,
        "console": 2002,
        "disk0": 1,
```

```
                    "exec_area": 16,
                    "image": "c2691-adventerprisek9-mz.124-25d.image",
                    "mac_addr": "c002.12e8.0000",
                    "name": "R2",
                    "nvram": 256,
                    "private_config": "configs\\i2_private-config.cfg",
                    "ram": 128,
                    "slot1": "NM-1FE-TX",
                    "startup_config": "configs\\i2_startup-config.cfg",
                    "wic0": "WIC-1T",
                    "wic1": "WIC-1T"
                },
                "router_id": 2,
                "server_id": 1,
                "type": "C2691",
                "x": -179.0,
                "y": -39.0
            },
            {
                "description": "Router c2691",
                "id": 3,
                "label": {
                    "color": "#000000",
                    "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
                    "text": "R3",
                    "x": 20.5,
                    "y": -25.0
                },
                "ports": [
                    {
                        "description": "connected to Host2 on port
nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local",
                        "id": 11,
                        "link_id": 6,
                        "name": "FastEthernet0/0",
                        "nio": "NIO_Generic_Ethernet",
                        "port_number": 0,
                        "slot_number": 0
                    },
                    {
```

```
            "id": 12,
            "name": "FastEthernet0/1",
            "port_number": 1,
            "slot_number": 0
        },
        {
            "description": "connected to R2 on port Serial0/1",
            "id": 13,
            "link_id": 5,
            "name": "Serial0/0",
            "nio": "NIO_UDP",
            "port_number": 16,
            "slot_number": 0
        },
        {
            "id": 14,
            "name": "FastEthernet1/0",
            "port_number": 0,
            "slot_number": 1
        },
        {
            "description": "connected to R1 on port Serial0/1",
            "id": 15,
            "link_id": 4,
            "name": "Serial0/1",
            "nio": "NIO_UDP",
            "port_number": 32,
            "slot_number": 0
        }
    ],
    "properties": {
        "aux": 2503,
        "console": 2003,
        "disk0": 1,
        "exec_area": 16,
        "image": "c2691-adventerprisek9-mz.124-25d.image",
        "mac_addr": "c003.13bc.0000",
        "name": "R3",
        "nvram": 256,
        "private_config": "configs\\i3_private-config.cfg",
```

```
                    "ram": 128,
                    "slot1": "NM-1FE-TX",
                    "startup_config": "configs\\i3_startup-config.cfg",
                    "wic0": "WIC-1T",
                    "wic1": "WIC-1T"
                },
                "router_id": 3,
                "server_id": 1,
                "type": "C2691",
                "x": -29.0,
                "y": -39.0
            },
            {
                "description": "Ethernet switch",
                "id": 4,
                "label": {
                    "color": "#000000",
                    "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
                    "text": "SW1",
                    "x": 17.0,
                    "y": -25.0
                },
                "ports": [
                    {
                        "description": "connected to Host1 on port
nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local",
                        "id": 16,
                        "link_id": 1,
                        "name": "1",
                        "nio": "NIO_Generic_Ethernet",
                        "port_number": 1,
                        "type": "access",
                        "vlan": 1
                    },
                    {
                        "description": "connected to R2 on port FastEthernet0/0",
                        "id": 17,
                        "link_id": 2,
                        "name": "2",
                        "nio": "NIO_UDP",
```

```
            "port_number": 2,
            "type": "access",
            "vlan": 1
        },
        {
            "id": 18,
            "name": "3",
            "port_number": 3,
            "type": "access",
            "vlan": 1
        },
        {
            "id": 19,
            "name": "4",
            "port_number": 4,
            "type": "access",
            "vlan": 1
        },
        {
            "id": 20,
            "name": "5",
            "port_number": 5,
            "type": "access",
            "vlan": 1
        },
        {
            "id": 21,
            "name": "6",
            "port_number": 6,
            "type": "access",
            "vlan": 1
        },
        {
            "id": 22,
            "name": "7",
            "port_number": 7,
            "type": "access",
            "vlan": 1
        },
        {
```

```
                "id": 23,
                "name": "8",
                "port_number": 8,
                "type": "access",
                "vlan": 1
            }
        ],
        "properties": {
            "name": "SW1"
        },
        "server_id": 1,
        "type": "EthernetSwitch",
        "x": -312.5,
        "y": -34.0
    },
    {
        "description": "Host",
        "id": 5,
        "label": {
            "color": "#000000",
            "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
            "text": "Host1",
            "x": 9.5,
            "y": -25.0
        },
        "ports": [
            {
                "id": 24,
                "name": "nio_gen_eth:Liga\u00e7\u00e3o de rede sem fios",
                "stub": true
            },
            {
                "id": 25,
                "name": "nio_gen_eth:VMware Network Adapter VMnet1",
                "stub": true
            },
            {
                "id": 26,
                "name": "nio_gen_eth:VMware Network Adapter VMnet8",
                "stub": true
```

```
            },
            {
                "description": "connected to SW1 on port 1",
                "id": 27,
                "link_id": 1,
                "name": "nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local",
                "nio": "NIO_Generic_Ethernet",
                "stub": true
            }
        ],
        "properties": {
            "name": "Host1",
            "nios": [
                "nio_gen_eth:Liga\u00e7\u00e3o de rede sem fios",
                "nio_gen_eth:VMware Network Adapter VMnet1",
                "nio_gen_eth:VMware Network Adapter VMnet8",
                "nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local"
            ]
        },
        "server_id": 1,
        "type": "Host",
        "x": -294.5,
        "y": -146.5
    },
    {
        "description": "Host",
        "id": 6,
        "label": {
            "color": "#000000",
            "font": "TypeWriter,10,-1,5,75,0,0,0,0,0",
            "text": "Host2",
            "x": 9.5,
            "y": -25.0
        },
        "ports": [
            {
                "id": 28,
                "name": "nio_gen_eth:Liga\u00e7\u00e3o de rede sem fios",
                "stub": true
            },
```

```
                {
                    "id": 29,
                    "name": "nio_gen_eth:VMware Network Adapter VMnet1",
                    "stub": true
                },
                {
                    "id": 30,
                    "name": "nio_gen_eth:VMware Network Adapter VMnet8",
                    "stub": true
                },
                {
                    "description": "connected to R3 on port FastEthernet0/0",
                    "id": 31,
                    "link_id": 6,
                    "name": "nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local",
                    "nio": "NIO_Generic_Ethernet",
                    "stub": true
                }
            ],
            "properties": {
                "name": "Host2",
                "nios": [
                    "nio_gen_eth:Liga\u00e7\u00e3o de rede sem fios",
                    "nio_gen_eth:VMware Network Adapter VMnet1",
                    "nio_gen_eth:VMware Network Adapter VMnet8",
                    "nio_gen_eth:Liga\u00e7\u00e3o de \u00c1rea Local"
                ]
            },
            "server_id": 1,
            "type": "Host",
            "x": 135.5,
            "y": -43.5
        }
    ],
    "servers": [
        {
            "cloud": false,
            "host": "127.0.0.1",
            "id": 1,
            "local": true,
```

```
                "port": 8000
            }
        ]
    },
    "type": "topology",
    "version": "1.2.3"
}
```

A.2 Exemplo de ficheiro de configuração base (Cisco).

```
!
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname %h
!
ip cef
no ip domain lookup
no ip icmp rate-limit unreachable
ip tcp synwait 5
no cdp log mismatch duplex
!
line con 0
 exec-timeout 0 0
 logging synchronous
 privilege level 15
 no login
line aux 0
 exec-timeout 0 0
 logging synchronous
 privilege level 15
 no login
!
!
end
```

A.3 Função `createConfiguration()`

```
sub createConfiguration{
my ($self, $baseConfigPath, $outputDir) = @_;
my $filename = $self->hostname.".cfg";
my $confFile;
my $tempFile;
my $templateName;
my $baseConfigFile = $baseConfigPath.$self->model.".baseConfig";
my $row = "";
my $x = 0;
my $exclamation = 0;
my ($interface,$address,$mask, $auxAddress ,$otherNode);
my $reverseMask;
my ($a,$b,$c,$d);

##Open configuration file##
if($outputDir ne ""){
$filename = $outputDir."\\".$filename;
#print $filename;
chomp $filename;
open ($confFile,'>',$filename) or die "Could not open file '$filename' $!";
}else{
open ($confFile,'>',$filename) or die "Could not open file '$filename' $!";
}
##Copy Base Configuration File into configuration file%%

open (my $bcf ,'<',$baseConfigFile) or die
"Could not open base config file '$baseConfigFile' $!";

while (my $bcfRow = <$bcf>){
chomp $bcfRow;
$_ = $bcfRow;
if ($bcfRow ne "end"){
if ($bcfRow =~ m/^hostname.+/){
print $confFile "hostname ".$self->hostname."\n!\n";
}else{
print $confFile "$bcfRow\n";
}
```

```
}
}

##Write configurations##
my @configurationArray = split(',',$self->configurations);
for(my $count = 0; $count < scalar @configurationArray; $count++){
$exclamation = 1;
$templateName = "templates/".$self->model.".".$configurationArray[$count];
open($tempFile, '<', $templateName);
if(lc $configurationArray[$count] eq "ospf"){
print $confFile "!\nrouter ospf 1\n";
$exclamation = 0;
}
if(lc $configurationArray[$count] eq "ripv2"){
print $confFile "!\nrouter rip\nversion 2\n";
$exclamation = 0;
}

for ($x=0;$x<scalar @routerInterfaces;$x++){
if($routerInterfaces[$x]->router eq $self->hostname){
($interface,$address,$mask,$otherNode) = (
$routerInterfaces[$x]->interface,$routerInterfaces[$x]->ipAdress,
$routerInterfaces[$x]->ipMask,$routerInterfaces[$x]->otherNode);
$auxAddress = $address;
$auxAddress =~ s/\d+$/0/;
$mask =~ /(\d+)\.(\d+)\.(\d+)\.(\d+)/i;
$a = 255 - $1;
$b = 255 - $2;
$c = 255 - $3;
$d = 255 - $4;
$reverseMask = "$a.$b.$c.$d";
while (my $row = <$tempFile>) {
chomp $row;
$row =~ s/__NETWORK__/$auxAddress/;
$row =~ s/__REVERSEMASK__/$reverseMask/;
$row =~ s/__DESTINATION__/$otherNode/;
$row =~ s/__INTERFACE__/$interface/;
$row =~ s/__ADDRESS__/$address/;
$row =~ s/__DESCRIPTION__/Connection to $otherNode/;
$row =~ s/__MASK__/$mask/;
```

```
print $confFile "$row\n";
}
if ((index(lc($interface), 'serial') != -1) &&
(index(lc($templateName), 'interface') != -1) ) {
print $confFile "clock rate 4000000\n";
}
if($exclamation==1){
print $confFile "!\n";
}
}
}
print $confFile "!\n";
}

close($tempFile);

##Close configuration file##
print $confFile "!\nend";
close $confFile;
}
```

Bibliografia

[1] J. Neumann, *Cisco Routers for the Small Business: A Practical Guide for IT Professionals*. Apress, 2008. 26, 34

[2] A. Nogueira and P. Salvador, "Teaching networking: A hands-on approach that relies on emulation-based projects," in *INFOCOMP 2014, The Fourth International Conference on Advanced Communications and Computation*, pp. 149–155, 2014. 27

[3] ESTG, "Academia Cisco no Instituto Politécnico de Leiria." `http://www.dei.estg.ipleiria.pt/academias/cisco/`. Accessed: 2015-1. 27

[4] R. Emiliano and M. Antunes, "Automatic network configuration in virtualized environment using gns3," in *Computer Science & Education (ICCSE), 2015 10th International Conference on*, pp. 25–30, IEEE, 2015. 33

[5] ACM, IEEE, "Computer science curricula 2013 - curriculum guidelines for undergraduate degree programs in computer science," 2013. 34

[6] Cisco, "Cisco lab virtual networks." `http://cisco-lab.net/about-virtual-networks-2/`. Accessed: 2015-01. 37

[7] M. Ford, T. Stevenson, H. K. Lew, and S. Spanier, *Internetworking technologies handbook*. Macmillan Publishing Co., Inc., 1997. 38

[8] D. DiNicolo, "Memories of a Cisco router." `http://archive.networknewz.com/networknewz-10-20050125MemoriesofaCiscoRouter.html`. Accessed: 2014-11. 38

[9] CLI, "Software user interface design," 1997. 38, 39

[10] S. Siraj, A. Gupta, and R. Badgujar, "Network simulation tools survey," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 1, no. 4, pp. 199–206, 2012. 41

[11] A. Wang, M. Iyer, R. Dutta, G. N. Rouskas, and I. Baldine, "Network virtualization: Technologies, perspectives, and frontiers," *Lightwave Technology, Journal of*, vol. 31, no. 4, pp. 523–537, 2013. 41

[12] Cisco, "Cisco Packet Tracer." https://www.netacad.com/web/about-us/cisco-packet-tracer. Accessed: 2014-10. 41

[13] GNS3, "GNS3." http://www.gns3.com/. Accessed: 2014-10. 41

[14] L. Sun, Y. Zhang, H. Yin, *et al.*, "Comparison between physical devices and simulator software for cisco network technology teaching," in *Computer Science & Education (ICCSE), 2013 8th International Conference on*, pp. 1357–1360, IEEE, 2013. 41

[15] A. Jesin, *Packet Tracer Network Simulator*. Packt Publishing Ltd, 2014. 41

[16] W.-J. Hsin, "Learning computer networking through illustration," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pp. 515–515, ACM, 2015. 41

[17] C. S. Tan, *Network simulator test engine for Huawei eNSP and Cisco GNS3*. PhD thesis, UTAR, 2014. 41

[18] J. Janitor, F. Jakab, and K. Kniewald, "Visual learning tools for teaching/learning computer networks: Cisco networking academy and packet tracer," in *Networking and Services (ICNS), 2010 Sixth International Conference on*, pp. 351–355, IEEE, 2010. 41

[19] Y. Liu, "The application of gns3 in network equipment of the internet course teaching," *Computer Knowledge and Technology*, vol. 8, p. 057, 2012. 42

[20] C. Welsh, *GNS3 network simulation guide*. Packt Publ., 2013. 42, 43

[21] RedNectar, "GNS3 Hypervisor." http://rednectar.net/2013/07/06/how-the-gns3-hypervisor-manager-works/. Accessed: 2014-10. 43

[22] R. Brezula, "Alcatel-Lucent router in GNS3." https://community.gns3.com/community/connect/community-blog/blog/2014/12/18/alcatel-lucent-virtualized-simulator-on-gns3. Accessed: 2015-03. 43

[23] "GNS Performance." https://community.gns3.com/docs/DOC-1708. Accessed: 2014-10. 44

[24] B. Claise, "Cisco systems netflow services export version 9," 2004. 47

[25] A. Leskiw, "Solarwinds network config generator review." http://www.networkmanagementsoftware.com/solarwinds-network-config-generator-review. Accessed: 2014-10. 49

[26] M. Pizzonia and M. Rimondini, "Netkit: easy emulation of complex networks on inexpensive hardware," in *Proceedings of the 4th International Conference on Testbeds and research infrastructures for the development of networks & communities*, p. 7, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. 51

[27] "Gliffy." https://www.gliffy.com/. Accessed: 2014-10. 52

[28] Smart PC Tricks, "Tutorial de integração do GNS3 numa rede física." http://www.smartpctricks.com/2014/06/connecting-gns3-real-networks.html. Accessed: 2014-12. 65

[29] D. Bjørner and M. C. Henson, *Logics of specification languages*. Springer Science & Business Media, 2007. 68

[30] "XML Applications." http://xml.coverpages.org/xmlApplications.html. Accessed: 2014-12. 68

[31] I. Bond and Q. Room, "Object oriented programming," 1993. 70

[32] M. Raio, "Quick start guide for Linux users." https://community.gns3.com/community/connect/community-blog/blog/2014/10/21/installation-guide-for-linux-loversi-mean-users. Accessed: 2015-05. 80