Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática

Curso de Graduação em Engenharia Elétrica

ANDRÉ ROLIM ALMEIDA GUIMARÃES

ESTÁGIO NA GERÊNCIA DE SOFTWARE DE CONTROLE DO CPQD

Campina Grande, Paraíba
Dezembro de 2013

ANDRÉ ROLIM ALMEIDA GUIMARÃES

ESTÁGIO NA GERÊNCIA DE SOFTWARE DE CONTROLE DO CPQD

Relatório de Estágio Integrado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Sistemas de Automação

Orientador:
Professor Dr. Angelo Perkusich

Campina Grande, Paraíba
Dezembro de 2013

ANDRÉ ROLIM ALMEIDA GUIMARÃES

ESTÁGIO NA GERÊNCIA DE SOFTWARE DE CONTROLE DO CPQD

Relatório de Estágio Integrado submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Sistemas de Automação

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Dr. Angelo Perkusich
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho a minha família, que sempre me apoiou nas decisões mais importantes de minha vida.

Agradecimentos

Agradeço primeiramente a minha família pelo apoio durante todo o período de estágio no CPqD como também durante todo o período de graduação.

Agradeço aos colegas de trabalho por terem me dado a oportunidade de estagiar no CPqD e por todo o conhecimento passado durante o período do estágio.

Agradeço aos professores e funcionários da Universidade Federal de Campina Grande por terem contribuído para a minha formação acadêmica e pessoal.

Por fim agradeço aos meus amigos e colegas de universidade sem os quais nada seria possível.

"Eu posso não ter ido aonde
eu pretendia ir, mas eu acho que
acabei terminando onde
eu pretendia estar."

Douglas N. Adams

Resumo

Este trabalho consiste num relato das atividades desenvolvidas pelo aluno André Rolim Almeida Guimarães durante o estágio realizado na fundação Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPqD) realizado no primeiro semestre do ano de 2013. As atividades realizadas durante o período de estágio estiveram no âmbito do projeto ROADM-NG. O foco principal do trabalho desenvolvido foi o desenvolvimento e a realização de testes em software de controle de dispositivos embarcados que atuam diretamente em nós de redes baseados em ROADM (*Reconfigurable Optical Add/Drop Multiplexers)*. O estágio foi realizado na Gerência de Software de Controle (GSC) alocado na Diretoria de Redes Convergentes (DRC). Neste documento são descritas as atividades desenvolvidas durante o período de estágio no CPqD, como também são explanados alguns aspectos teóricos que foram fundamentais para a realização das atividades.

Palavras-chave: CPqD, sistemas embarcados, Linux, *Reconfigurable Optical Add-Drop Multiplexer*, testes de unidade.

Abstract

This document is a report of the activities developed by the student André Rolim Almeida Guimarães during the internship at the Fundação Centro de Pesquisa e Desenvolvimento em Telecomunicações (CPqD) held in the first semester of 2013. The activities performed during the internship period were executed within the ROADM-NG project. The main focus of the work was in the development and testing of control software deployed in embedded devices which act directly on ROADM (*Reconfigurable Optical Add/Drop Multiplexers*) based networks. The internship was performed in the Gerência de Software de Controle (GSC) allocated on the Diretoria de Redes Convergentes (DRC). The text describes the activities undertaken during the internship period within CPqD, and also describes some theoretical aspects that were fundamental to the work carried out.

Keywords: CPqD, embedded systems, Linux, *Reconfigurable Optical Add-Drop Multiplexer*, unit tests.

Lista de Figuras

SUMÁRIO

1 Introdução

1.1 INTRODUÇÃO GERAL

Com o constante desenvolvimento de tecnologias e serviços voltados para a internet, juntamente com o grande aumento na inclusão digital em todas as camadas da população, um dos grandes desafios para o setor de telecomunicações brasileiro é o aumento da capacidade das redes de comunicações para suprir a banda demandada. Uma estrutura de telecomunicações moderna e capaz de acompanhar a diversificação do mercado de serviços da internet é um pré-requisito necessário para o crescimento econômico e a competitividade de um país (Wright, Da Silva, & Spers, 2010).

Sistemas de transmissão ópticos foram desenvolvidos para prover uma solução para o grande aumento na demanda por maiores taxas de transmissão, devido a grande largura de banda da fibra óptica.

Com o desenvolvimento de novas tecnologias na área de transmissão óptica, possibilitando cada vez mais uma maior capacidade de transmissão, se faz necessário também o desenvolvimento de sistemas computacionais capazes de gerenciar os equipamentos e o tráfego de informação na rede. Sistemas embarcados por sua vez são amplamente utilizados nestes equipamentos, que são projetados e configurados para prover performances elevadas e alta disponibilidade.

Para o desenvolvimento destas tecnologias na área de transmissão óptica, é necessário um grande investimento devido a necessidade de qualificação de pessoal e compra de equipamentos muitas vezes com um preço elevado, e é de extrema importância para um país em desenvolvimento como o Brasil possuir centros de pesquisas focados no avanço deste segmento. No Brasil, o CPqD é um dos principais centros de pesquisa responsável pela inovação tecnológica neste setor.

1.2 FUNDAÇÃO CPQD

O CPqD foi criado em 1976 como Centro de Pesquisa e Desenvolvimento da Telebrás, empresa estatal que detinha o monopólio dos serviços públicos de telecomunicações no Brasil. Desde então, ocupa posto de vanguarda tecnológica, sintonizado com o futuro e antecipando-se às necessidades de uma sociedade que se modifica e evolui em alta velocidade. Em 1998, com a privatização do sistema Telebrás, o CPqD tornou-se uma fundação de direito privado, ampliando a sua atuação, tanto no escopo como na abrangência do mercado.

Centrais digitais, antenas, sistemas de transmissão digital, equipamentos de transmissão óptica, fibra óptica, laser semicondutor, centrais de comutação por pacote, telefone público a cartão indutivo, centrais de telex, complexos sistemas de suporte a operações e negócios e tantas outras tecnologias desenvolvidas no CPqD, que somadas ao conhecimento sobre o mercado e a sociedade brasileira, em seus aspectos culturais, socioeconômicos e históricos envolvidos com TICs, consolidaram no CPqD potencial inovador único na área.

O CPqD é uma instituição focada na inovação com base nas TICs (Tecnologia da Informação e Comunicações) tendo como objetivo contribuir para a competitividade do país e para a inclusão digital da sociedade. Desenvolve amplo programa de pesquisa e desenvolvimento, o maior da América Latina em sua área de atuação, gerando soluções em TICs que são utilizadas em diversos setores: telecomunicações, financeiro, energia elétrica, industrial, corporativo e administração pública.

1.3 OBJETIVOS DO ESTÁGIO

Desenvolver atividades voltadas para a gerência e configuração de equipamentos inseridos em redes ópticas, dando ênfase aos que utilizam sistemas embarcados com o sistema operacional Linux. Auxiliar no desenvolvimento e no teste de software que utiliza o protocolo Netconf e a linguagem YANG utilizado na modelagem de dispositivos de rede, descritos na seção 2.3.

2 Fundamentação teórica

2.1 *RECONFIGURABLE OPTICAL ADD/DROP MULTIPLEXERS*

Hoje em dia amplificadores EDFA (*Erbium-Doped Fiber Amplifier*) são visto como uma tecnologia madura, enquanto novas aplicações de rede requerem avanços mais profundos em amplificadores de fibra óptica. Uma aplicação em ascensão são redes ópticas dinâmicas que usam multiplexadores ópticos add/drop reconfiguráveis (ROADM, *Reconfigurable Optical Add/Drop Multiplexers*) de forma remota (COLLINGS, B. C.; ROORDA, P., 2011).

Antes da introdução das redes reconfiguráveis, os sistemas WDM (*Wavelength-Division Multiplexing*) eram normalmente constituídos por transmissões ponto-a-ponto interconectando regeneradores ou repetidores com filtros ópticos fixos e transições O-E-O (*Optical-Electrical-Optical*) para todos os canais do sistema. Com o aumento do número de canais e alcance dos sistemas WDM, se tornou possível e desejável o tráfego transparente nas redes ópticas, simplificando os nós e eliminando o uso de transições O-E-O indesejadas. Para permitir essas operações, multiplexadores insere/deriva ópticos (*Optical Add/Drop Multiplexers* – OADM) fixos, foram introduzidos nas redes. Esses dispositivos possuem filtros ópticos fixos que determinam quais canais são removidos e inseridos em cada nó de uma rede.

Redes baseadas em ROADMs disponibilizam um nível mais alto de monitoração, gerenciamento de potência e roteamento de comprimentos de onda na camada de transporte de uma rede óptica. Essas funcionalidades resultam em reduções de custos operacionais e, principalmente, intervalos para ativações de novos serviços na rede.

2.1.1 CHAVES WSS

A tecnologia da terceira geração dos ROADMs e também a dominante nos sistemas atuais é a chave seletora de comprimento de onda (*Wavelength Selective Switch* – WSS). Na Figura 1 apresenta-se uma ilustração do funcionamento básico de um WSS 1x4, em que um sinal óptico com vários canais chega à chave WSS e em seguida podem ser separados na saída de

acordo com a configuração aplicada na chave. WSS são dispositivos capazes de rotear qualquer canal presente na porta de entrada para qualquer porta de saída. Operações de equalização e monitoração de canais são realizadas nessas chaves, que normalmente são dispositivos bidirecionais.

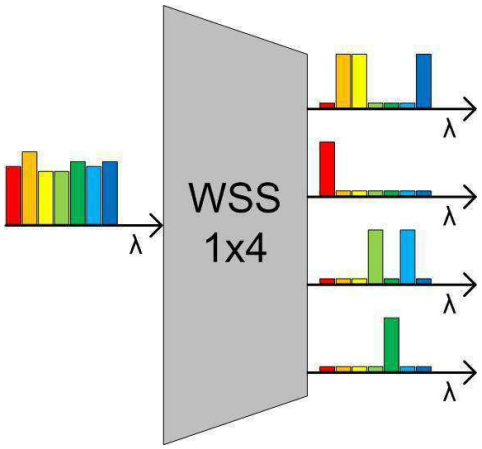

Figura 1 - Diagrama de funcionamento de uma chave WSS 1x4

O projeto de WSS pode ser realizado com o uso de diferentes tecnologias, tais como sistemas micro-eletro-mecânicos (*Micro-Electro-Mechanical Systems* – MEMS) e cristal líquido em silício (*Liquid Crystal on Silicon* – LCoS). Essas tecnologias são utilizadas para direcionar individualmente os comprimentos de onda para as portas de saída desejadas.

Na Figura 2 apresenta-se um diagrama para o funcionamento de um nó ROADM de grau quatro, pois existem quatro pares de entradas e saídas do nó.

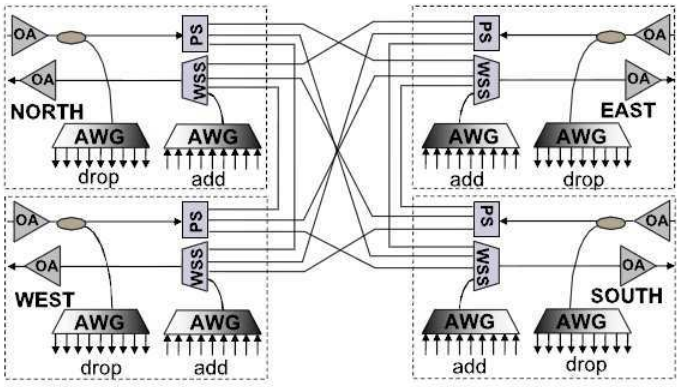

Figura 2 - Diagrama de um nó ROADM de grau 4 com chaves WSS

Uma cópia dos canais de entrada é demultiplexado, permitindo que todos os canais de entrada possam ser recebidos localmente. O WSS está posicionado em cada saída. Os canais que entram em cada grau são divididos em várias fibras com uma cópia de cada conjunto de canais direcionados para cada uma das portas N de cada WSS de todos os outros graus (exceto o grau em que os canais entraram no nó) (COLLINGS, B. C.; ROORDA, P., 2011).

2.2 SISTEMA OPERACIONAL LINUX EMBARCADO

O sistema operacional Linux foi primeiramente lançado no verão de 1991. Inicialmente como um hobby de um cientista da computação chamado Linus Torvalds. Inicialmente Linux era apenas acessível através de seu código fonte para aqueles que possuíam conhecimento suficiente para compilar e instala-lo (YAGHMOUR, MASTERS, 2008).

Apesar da pouca pretensão no momento do lançamento do núcleo do Linux em 1991, este sistema pode ser encontrado em uma grande variedade de dispositivos eletrônicos que possuem uma capacidade de processamento suficiente para operar um sistema operacional. Alguns exemplos de dispositivos que utilizam Linux podem ser citados como smartphones Android, receptores de TV digital como TiVo, e até mesmo em Hollywood, Linux é utilizado por super computadores na renderização de filmes que utilizam computação gráfica. Hoje em dia o CPqD utiliza largamente sistemas Linux que são embarcados nos mais diversos tipos de dispositivos de rede, como por exemplo chaves WSS.

"Linux embarcado refere-se tipicamente a um sistema completo, ou no contexto de um representante de Linux embarcado, refere-se a uma distribuição para dispositivos embarcados. [...] não existe nenhuma forma especial do núcleo direcionada a aplicações embarcadas. Ao invés disto o mesmo código fonte do núcleo é construído com o propósito de ser utilizado na maior quantidade possível de dispositivos" (YAGHMOUR, MASTERS, 2008).

Existe uma diferença entre sistemas Linux embarcados, que são sistemas que utilizam o núcleo do Linux para realizar a comunicação e gerenciamento com o hardware do dispositivo, e *distribuições* Linux para aplicações embarcadas direcionadas ao desenvolvimento de tais sistemas. De forma geral os fornecedores destas distribuições disponibilizam um conjunto de ferramentas que possibilitam o desenvolvimento de aplicações, como por exemplo, compiladores cruzados, ambientes de desenvolvimento, imagens de inicialização do sistema entre outros.

Apesar da similaridade de sistemas Linux embarcado com sistemas que utilizam o núcleo do Linux em maquinas com mais recursos computacionais, o processo de desenvolvimento para tais sistemas pode variar. Em sua grande maioria, os sistemas embarcados apresentam uma pouca flexibilidade para alteração e reconfiguração devido às limitações do hardware. Cabe então ao desenvolvedor projetar e construir o sistema levando este aspecto em conta para obter um melhor desempenho.

2.3 PROTOCOLO NETCONF E A LINGUAGEM YANG

2.3.1 PROTOCOLO NETCONF

Netconf é um protocolo desenvolvido pela empresa americana Juniper Networks, fundada em 1996, que possibilita uma linguagem comum de configuração de dispositivos de redes que se utiliza de mensagens no formato XML (*Extensible Markup Language*) para a manipulação das configurações.

O protocolo Netconf usa o paradigma de chamadas procedurais remotas (*Remote Procedure Call - RPC*). Um cliente codifica uma chamada RPC em um XML e o envia para um servidor em uma sessão segura e orientada a conexão. O servidor responde também em formato XML. Os conteúdos da requisição e da resposta são descritos em XML, possibilitando que tanto servidor como cliente consigam reconhecer as limitações da sintaxe na troca de informações (Enns, Bjorklund, 2011).

Netconf permite que clientes descubram quais os recursos disponíveis em um servidor através das mensagens em formato XML. Desta forma, clientes que implementam o protocolo Netconf são capazes de se adaptarem as capacidades de cada servidor. Estas capacidades podem ser facilmente estendidas no lado do servidor de uma maneira não centralizada.

Tipicamente o servidor possui uma aplicação rodando em segundo plano capaz de processar mensagens XML que utilizam o protocolo Netconf. Existem no mercado várias soluções que implementam este protocolo e que fornecem ferramentas de desenvolvimento para aplicações do lado cliente, onde um script ou uma aplicação é utilizado para o usuário final elaborar uma mensagem de configuração e envia-lo ao servidor. Tipicamente estas aplicações do

lado cliente utilizam como entrada de informações a linha de comando, o termo para este tipo de entrada é *Command Line Input* (CLI).

A Listagem 1 exemplifica uma mensagem no protocolo Netconf em que o cliente deseja configurar um dispositivo de rede definindo alguns parâmetros de configuração para uma determinada interface de rede.

```
<rpc message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <edit-config>
         <target>
           <running/>
         </target>
         <config>
           <top xmlns="http://example.com/schema/1.2/config">
             <interface>
               <name>Ethernet0/0</name>
               <mtu>1500</mtu>
             </interface>
           </top>
         </config>
       </edit-config>
</rpc>
<rpc-reply message-id="101" xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
       <ok/>
</rpc-reply>
```

Listagem 1 – Exemplo de mensagem Netconf

2.3.2 LINGUAGEM YANG

YANG é uma linguagem de modelagem de dados usada na modelagem de configurações e estados de dados manipulados pelo protocolo de configuração de redes (*Network Configuration Protocol – Netconf*), chamadas procedurais e notificações (Bjorklund, 2011).

A linguagem YANG é utilizada para definir modelos de configurações de dispositivos de redes capazes de se comunicarem utilizando o protocolo Netconf. A linguagem é definida em uma hierarquia de dados que podem descrever configurações, estados, chamadas e notificações

capazes de serem traduzidas em formato XML e se comunicarem via Netconf. O equivalente XML de um modelo YANG é chamado de YIN. A tradução de YANG para YIN acontece sem perdas, portanto é possível a tradução reversa de YIN para YANG.

Modelos YANG são definidos em módulos e sub-módulos. Estes módulos podem ser utilizados e estendidos em outros módulos garantindo uma grande flexibilidade no momento da modelagem de sistemas. Outra vantagem da utilização de modelos YANG para a modelagem de dispositivos é que é possível definir limitações a valores de elementos do modelo, podendo estas limitações estar dependentes de outros elementos.

A linguagem YANG define vários elementos e tipos de variáveis que se assemelham a tipos de dados utilizados em linguagens de programação como *int*, *float*, *double* entre outros. Os elementos básicos da linguagem YANG são:

i. *Leaf*
ii. *Leaf-list*
iii. *Container*
iv. *List*

Sendo leaf o elemento básico da linguagem YANG. Os outros elementos são capazes de manipular e encapsular elementos do tipo *leaf*. No modelo YANG da Listagem 2 está descrito as configurações de uma interface rede, bem como demonstra como é possível restringir valores de elementos do modelo.

```
container interface {
         leaf ifType {
             type enumeration {
                 enum ethernet;
                 enum atm;
             }
         }
         leaf ifMTU {
             type uint32;
         }
         must "ifType != 'ethernet' or " +
```

```
            "(ifType = 'ethernet' and ifMTU = 1500)" {
            error-message "An ethernet MTU must be 1500";
        }
        must "ifType != 'atm' or " +
            "(ifType = 'atm' and ifMTU <= 17966 and ifMTU >= 64)" {
            error-message "An atm MTU must be  64 .. 17966";
        }
    }
```

Listagem 2 – Exemplo de modelo YANG

O CPqD utiliza em alguns projetos soluções do mercado que implementam o protocolo Netconf para a gerência de alguns dispositivos de rede, bem como usa amplamente a linguagem YANG para modelar as configurações e estados destes equipamentos.

2.4 TESTES DE UNIDADE

2.4.1 TDD

Test-Driven Development ou TDD é um processo de desenvolvimento de software baseado em pequenos ciclos de desenvolvimento, em que antes de se escrever a primeira linha de uma função, o desenvolvedor deve antes escrever uma função de teste que inicialmente irá falhar. Em seguida deve-se implementar a funcionalidade de modo que o teste escrito anteriormente seja aceito. O passo seguinte então é desenvolver a funcionalidade real da função e executar o teste novamente, verificando se a função está se comportando da maneira esperada.

Na Figura 3 apresenta-se o ciclo de desenvolvimento utilizando testes de unidade. Primeiro o teste de unidade para a funcionalidade que se deseja codificar deve ser projetado, em seguida o teste deve ser executado e inicialmente irá falhar. Deve-se refatorar o código até que o teste seja satisfeito. Este ciclo deve ser repetido até que a funcionalidade esteja finalizada.

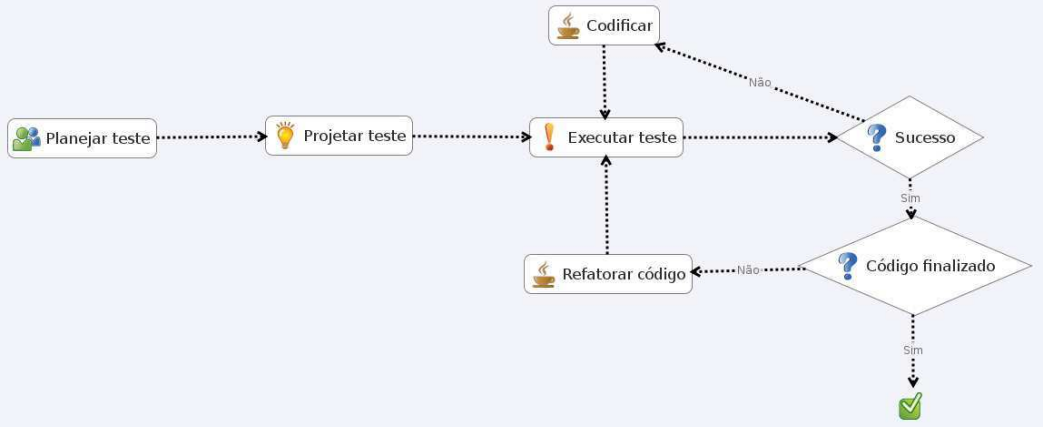

Figura 3 - Ciclo de desenvolvimento TDD

Testes de unidade são tão importantes para a qualidade do código quanto o código final em si. Talvez sejam até mais importantes, pois testes de unidade preservam e aperfeiçoam a flexibilidade, manutenção e reusabilidade do código final (Martin, 2008).

A utilização de testes de unidade no processo de desenvolvimento de software ajuda na validação do código. É bastante comum que após grandes modificações em uma parte do código, outro componente do sistema venha a falhar, isto pode causar uma grande perda de tempo e longas seções de depuração de código para se encontrar a falha. Com a validação e segurança que testes unitários garantem, o desenvolvedor pode realizar modificações no código de maneira mais segura, pois os testes vão indicar que possivelmente todos os sistemas estão funcionando após uma grande modificação, caso algum componente venha a falhar, os testes irão dizer o local em que a falha ocorreu no relatório de saída.

2.4.2 GOOGLE C++ TESTING FRAMEWORK

O CPqD adota em alguns projetos o desenvolvimento com testes de unidade e utiliza o *Google C++ Testing Framework* ou gTest, para a implementação dos testes unitários. O gTest é um *framework* completo de testes para a linguagem C++ desenvolvido pela Google, de fácil uso e com uma pequena curva de aprendizagem. É possível realizar vários tipos de testes de forma simples e direta. O gTest pode ser utilizado em várias plataformas (Linux, Mac OS X, Windows, Cygwin, Windows CE e Symbian), como o CPqD trabalha com múltiplas plataformas, o gTest é um framework que acomoda bem a demanda de desenvolvimento dentro da empresa.

A Listagem 3 mostra como é fácil a implementação de testes de unidade em C++ utilizando o Google C++ Testing Framework. Supondo que se deseja testar uma função que calcula o fatorial de um número, é possível realizar testes sem a necessidade de escrever uma função "main" que execute os testes.

```
1.    int Factorial(int n); // Calcula o fatorial de n
2.
3.    // Testa fatorial de 0
4.    TEST(FactorialTest, HandlesZeroInput) {
5.      EXPECT_EQ(1, Factorial(0));
6.    }
7.
8.    // Testa fatorial de números positívos
9.    TEST(FactorialTest, HandlesPositiveInput) {
10.     EXPECT_EQ(1, Factorial(1));
11.     EXPECT_EQ(2, Factorial(2));
12.     EXPECT_EQ(6, Factorial(3));
13.     EXPECT_EQ(40320, Factorial(8));
14.   }
```

Listagem 3 – Testes unitários com gTest

2.5 MEMÓRIA EEPROM

Um dos principais problemas no desenvolvimento de sistemas embarcados são as limitações do hardware. Para diminuir as dimensões e o custo de um dispositivo, fabricantes limitam o hardware o que acarreta também no aumento na dificuldade do desenvolvimento de aplicativos para aquela plataforma. Algumas das placas utilizadas em projetos dentro CPqD possuíam estas limitações, mais especificamente na capacidade de armazenamento do dispositivo. Fez-se necessário então o uso da memória EEPROM disponível no sistema para armazenar informações de configurações e estados dos dispositivos de rede.

EEPROM que vem do inglês *Electrically Erasable Programmable Read-Only Memory*, é um tipo de memória não volátil, ou seja, mantém sua informação mesmo sem estar energizado, capazes de guardar pequenas quantidades de dados. Diferente da memória EPROM, que requer a remoção do chip para a sua escrita, a memória EEPROM pode ser programada ainda na placa através de sinais especiais programados. As memórias *Flash* são um tipo de memória EEPROM.

Uma limitação deste tipo de memória é a quantidade limitada de operações de escrita. Este numero varia entre mil e um milhão. Sistemas embarcados são projetados para serem utilizados por um longo período de tempo, portanto a limitação na quantidade de escritas na memória EEPROM é um fator que deve ser considerado no momento do planejamento e desenvolvimento do projeto.

3 Ferramentas utilizadas

3.1 AUTOTOOLS

Autotools é um conjunto de ferramentas open-source que auxilia na criação e distribuição de pacotes para plataformas baseadas em Unix. As ferramentas disponibilizadas pelo Autotools estão sob uma licença *open source* com algumas exceções que permitem o uso para a criação de código proprietário.

Os dois principais componentes do Autotools, também conhecido como GNU build system, são os programas autoconf e automake. O autoconf é responsável pela geração de um script de configuração chamado *configure*, como também é responsável pela criação de um *Makefile* que será utilizado no momento de compilação do pacote. O automake é responsável pela criação de um arquivo chamado *Makefile.in* que servirá de input para o autoconf. Após a geração destes artefatos, o usuário final poderá chamar a sequência de comandos *./configure*, *make* e *make install* para compilar e instalar um pacote gerado com Autotools. Esta forma de criação de pacotes é amplamente utilizada por diversos projetos *open source*.

Para a criação de pacotes utilizando Autotools é necessário configurar os seguintes arquivos que servirão de entrada para o autoconf e automake.

i. *configure.ac*

ii. *Makefile.am*

iii. *src/Makefile.am* (caso exista uma subpasta contendo o código fonte)

Na Figura 4 apresenta-se o fluxo da criação do pacote e a sequência de artefatos que são gerados durante o processo.

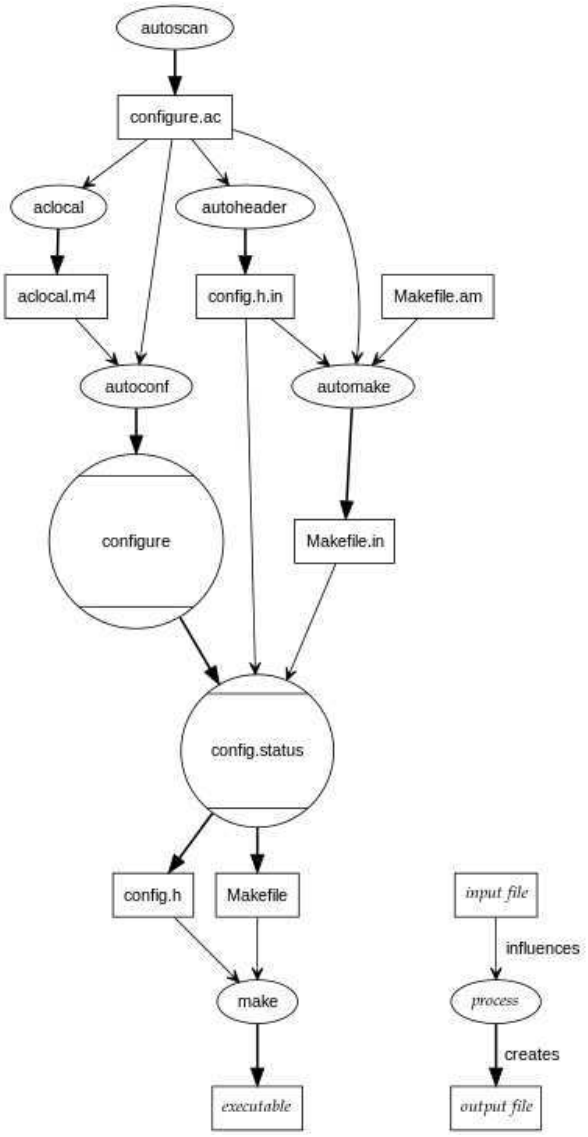

Figura 4 - Fluxo da geração de pacote utilizando automake/autoconf

3.2 JENKINS BUILD SERVER

Integração contínua, também conhecida como CI (*Continuous Integration*), é um dos pináculos do desenvolvimento de software moderno. No momento em que integração contínua é introduzida em uma organização, modifica radicalmente o modo como a equipe pensa em todo o processo de desenvolvimento (Smart, 2011).

Em um ambiente corporativo onde projetos são desenvolvidos para varias plataformas, um problema constante é o ambiente em que o código escrito é compilado. Principalmente quando se está cross-compilando um projeto, se faz necessário instalar várias bibliotecas, cross-compiladores e outras ferramentas. Muitas vezes a configuração deste ambiente não é realizada de forma simples, o que dificulta a compilação do mesmo código em outras maquinas, atrasando o processo de desenvolvimento. O Jenkins auxilia na solução deste problema. Com esta ferramenta é possível criar trabalhos de forma simples, em que o Jenkins compila o código baixado a partir de um repositório online como SVN ou git. Com isto, o desenvolvedor não precisa mais se preocupar com a configuração de ambientes de desenvolvimento para múltiplas plataformas.

Além de realizar compilações de projetos, o Jenkins pode ser configurado para realizar testes unitários após a compilação de um projeto, emitir relatórios, executar compilações noturnas entre outras opções. Uma grande vantagem que esta ferramenta proporciona é que o desenvolvedor é incentivado a enviar o seu código para o repositório mais frequentemente, além do que boas práticas de desenvolvimento como TDD podem ser automatizadas e incentivadas dentro do ambiente da empresa.

4 Atividades desenvolvidas

4.1 NETCONF/YANG TESTING FRAMEWORK

O CPqD utiliza em alguns projetos o protocolo Netconf juntamente com a linguagem YANG para a configuração e gerenciamento dos dispositivos de redes desenvolvidos. Uma gama de aplicações são desenvolvidas utilizando um *framework* que implementa estes protocolos. Para garantir a confiabilidade deste framework, foi requisitado o desenvolvimento de um framework de testes unitários capazes de validar a ferramenta escolhida.

Foi desenvolvida uma suíte de testes utilizando o *Google C++ Testing Framework* capaz de testar mais especificamente os padrões de modelagem descritos na RFC6020. Para cada elemento descrito na linguagem YANG como, por exemplo, *leaf, leaf-list, container* entre

outros, um conjunto de testes foi projetado para testar se estes modelos se comportavam de forma correta em conjunto com a ferramenta escolhida pelo CPqD que implementa o protocolo Netconf e a linguagem YANG.

Analisando o relatório de erros gerados pelo gTest pôde-se constatar que alguns modelos não funcionavam de forma correta, com estes relatórios em mãos é possível elaborar um relatório de erros para os desenvolvedores da ferramenta para a correção destes erros em versões futuras da ferramenta.

4.2 CRIAÇÃO DE PACOTES

O modelo de desenvolvimento de software do CPqD incentiva a reutilização de código através de pacotes. Pequenas bibliotecas são desenvolvidas e armazenadas em um repositório na forma de pacotes gerados utilizando a ferramenta Autotools. Estes pacotes são compilados em uma maquina que possui o Jenkins build-server a fim de garantir uma uniformidade no ambiente em que estes pacotes são gerados. Para cada pacote, um job é criado no Jenkins. Vários pacotes foram gerados para várias bibliotecas. Foram criados pacotes para o gTest, gMock, libPoco entre outros. Cada pacote gerado foi compilado para todas as plataformas utilizadas no CPqD, contribuindo para o reuso e facilidade de uso de software em sistemas embarcados.

4.3 LIBEEPROM

Sistemas embarcados muitas vezes não possuem uma grande quantidade de memória disponível, ou por motivos de especificações do sistema, a memória existente é utilizada por outras aplicações e funcionalidades. Chaves WSS utilizadas no sistema ROADM apresentaram este cenário, fez-se necessário encontrar uma forma alternativa de guardar informações de configurações do sistema de forma permanente.

Para um funcionamento continuo do sistema, é necessário guardar o estado da chave WSS de forma que quando a chave sofra alguma ação de reinicio, o estado anterior seja restaurado, a fim de manter o sistema em um estado válido. Foi realizada uma análise das capacidades de memória do sistema e foi decidido que estas configurações seriam armazenadas em uma memória EEPROM disponível na placa.

A memória EEPROM apresenta um tempo de gravação relativamente lento, foi preciso definir uma estratégia para se armazenar apenas dados novos na memória, exigindo então um mecanismo para verificar os dados já existentes. Outro problema que a memória em questão apresentou é que a memória EEPROM possui uma quantidade limitada de operações de escritas, portanto foi necessário o desenvolvimento de uma estratégia de gravação de informação que não ultrapassasse o número máximo de operações de escrita de forma a maximizar o tempo de uso da placa, que foi estipulado em no mínimo 25 anos. Por fim, a memória EEPROM possui apenas 1 Mega byte de espaço de armazenamento, logo as configurações devem ser compactadas antes da escrita.

O trabalho proposto foi o de projetar e desenvolver uma biblioteca em C++ capaz de compactar arquivos ou diretórios de configurações e em seguida armazena-los na memória EEPROM da chave WSS utilizada no sistema de transmissão. A biblioteca foi desenvolvida nos modelos de desenvolvimento contínuo adotados pelo CPqD, necessitando então estar sob a forma de um pacote gerado pela ferramenta Autotools, que por sua vez seria compilado para várias plataformas em uma maquina com o Jenkins build-server.

Por motivos de sigilo este relatório não possui nenhum trecho de código ou formato dos arquivos de configurações utilizados no sistema.

4.3.1 REQUISITOS

A biblioteca deverá ser capaz de atender os seguintes requisitos:

i. Escrita na linguagem C++;
ii. Compactar um arquivo de configuração ou diretórios contendo mais de um arquivo de configuração em um arquivo .zip;
iii. Escritos na forma de um pacote utilizando Autotools e armazenado em um repositório para facilitar o uso da biblioteca e incentivar o reuso de código;
iv. Possuir uma API (*Application Programming Interface*) que receba caminhos de diretórios, arquivos ou smart pointers para arquivos de configuração a serem armazenados;
v. Possuir um mecanismo de verificação de erros, a fim de validar os dados salvos na memória EEPROM.

Durante o desenvolvimento do sistema, foi decidido que o sistema ROADM deveria ser capaz de funcionar com sistemas legado, que também armazenavam arquivos de configurações na memória EEPROM. Contudo, estes arquivos eram armazenados em um formato diferente do utilizado no novo sistema. A biblioteca então deveria ser capaz de extrair as configurações de um sistema legado e exportar os dados em um formato XML com uma estrutura previamente definida, a fim de proporcionar uma compatibilidade com o novo sistema.

4.3.2 TECNOLOGIAS UTILIZADAS

Para facilitar o desenvolvimento da biblioteca algumas tecnologias foram utilizadas durante a implementação.

i. libPoco: Biblioteca escrita em C++, portável para varias plataformas possuindo uma API limpa com várias funcionalidades bem estabelecidas voltadas para aplicações de rede;
ii. Autotools: Sistema de geração de makefiles e scripts de configuração. Foi utilizado na geração e configuração do pacote da biblioteca;

iii. Jenkins build-server: Build server utilizado para compilar a biblioteca para várias plataformas;

iv. Gtest/Gmock: Framework desenvolvido pela Google capaz de facilitar a implementação de testes de unidade. Utilizado em todas as suítes de teste da biblioteca.

4.3.3 DIAGRAMAS UML

Os diagramas UML presentes nesta seção demonstram a arquitetura geral da biblioteca e seus componentes.

4.3.3.1 DIAGRAMA DE CLASSES

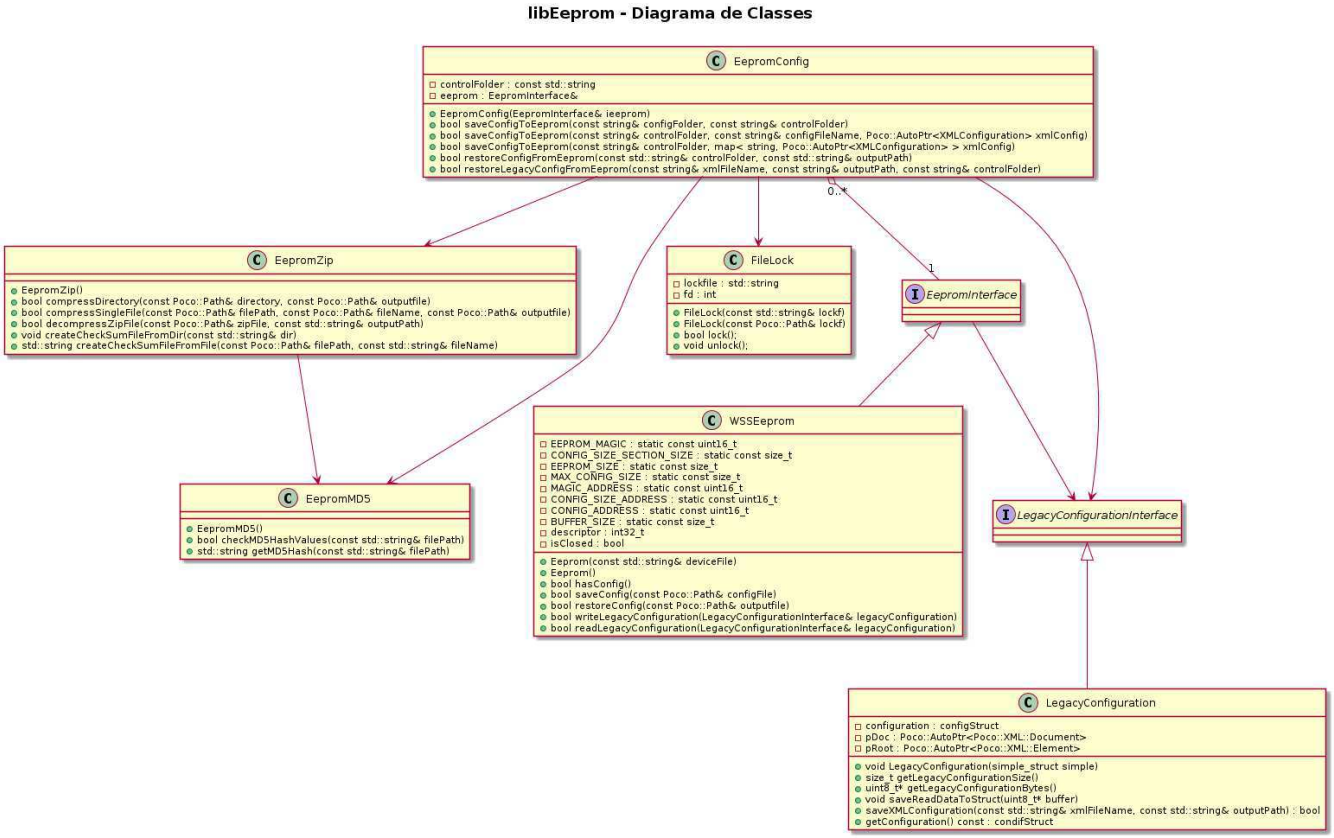

Figura 5 - Diagrama de classes descrevendo todas as classes implementadas e suas interações

A classe *EepromConfig* implementa as chamadas de alto nível do sistema. Com esta classe é possível com apenas uma chamada realizar as operações de escrita e leitura de configurações.

As classes *EepromZip*, *EepromMD5* e *FileLock* são utilizadas para as funcionalidades de compactação, calculo de *hash* MD5 e lógica de compartilhamento de recursos.

A interface *EepromInterface* é uma camada de abstração que permite a utilização da libEeprom em outras plataformas. O usuário final deve estender esta interface e implementar seus métodos com chamadas para a memória EEPROM da plataforma utilizada. A classe *WSSEeprom* implementa esta interface com chamadas de escrita e leitura da memória EEPROM contidas na chave WSS.

A interface *LegacyConfigurationInterface* é uma camada de abstração que permite o usuário final definir o tipo de configuração legado de acordo com o sistema em que a biblioteca será inserida. A classe *LegacyConfiguration* implementa esta interface de modo que o sistema funcionará com o tipo específico de configuração armazenada em chaves WSS legado.

4.3.3.2 CASOS DE USO

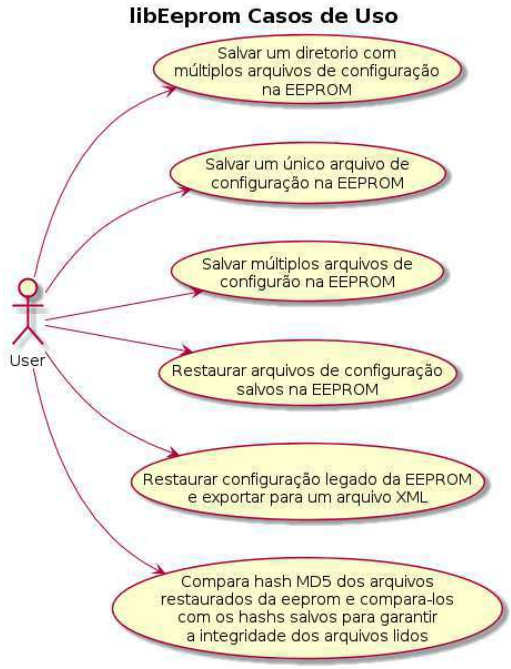

Figura 6 - Diagrama de casos de uso

4.3.3.3 DIAGRAMAS DE SEQUÊNCIA

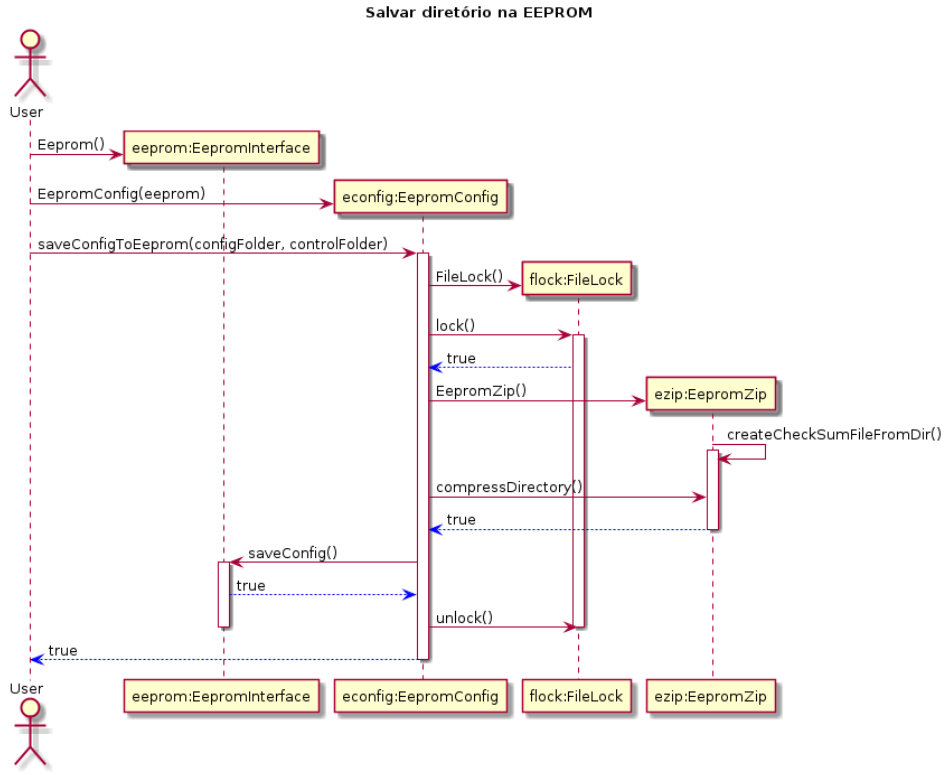

Figura 7 - Diagrama de sequência da operação de escrita de um diretório na EEPROM

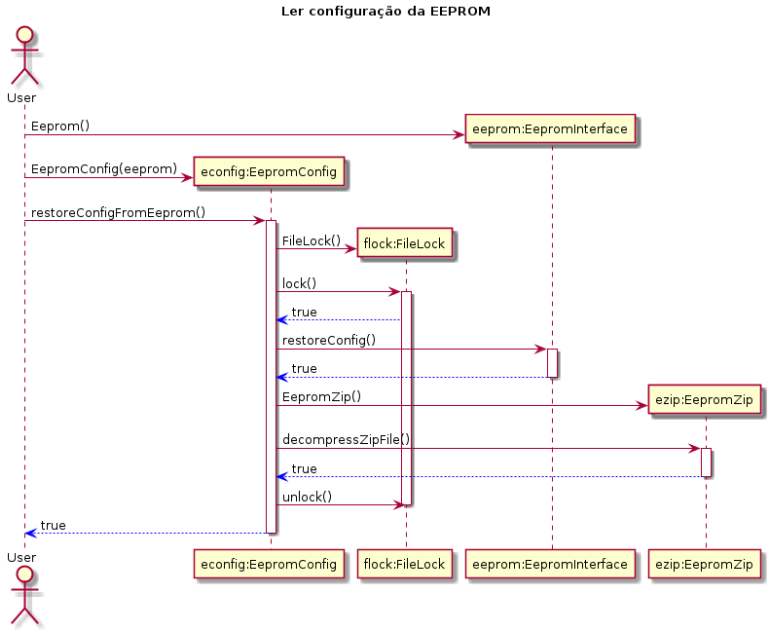

Figura 8 - Diagrama de sequência da operação de leitura das configurações salvas na EEPROM

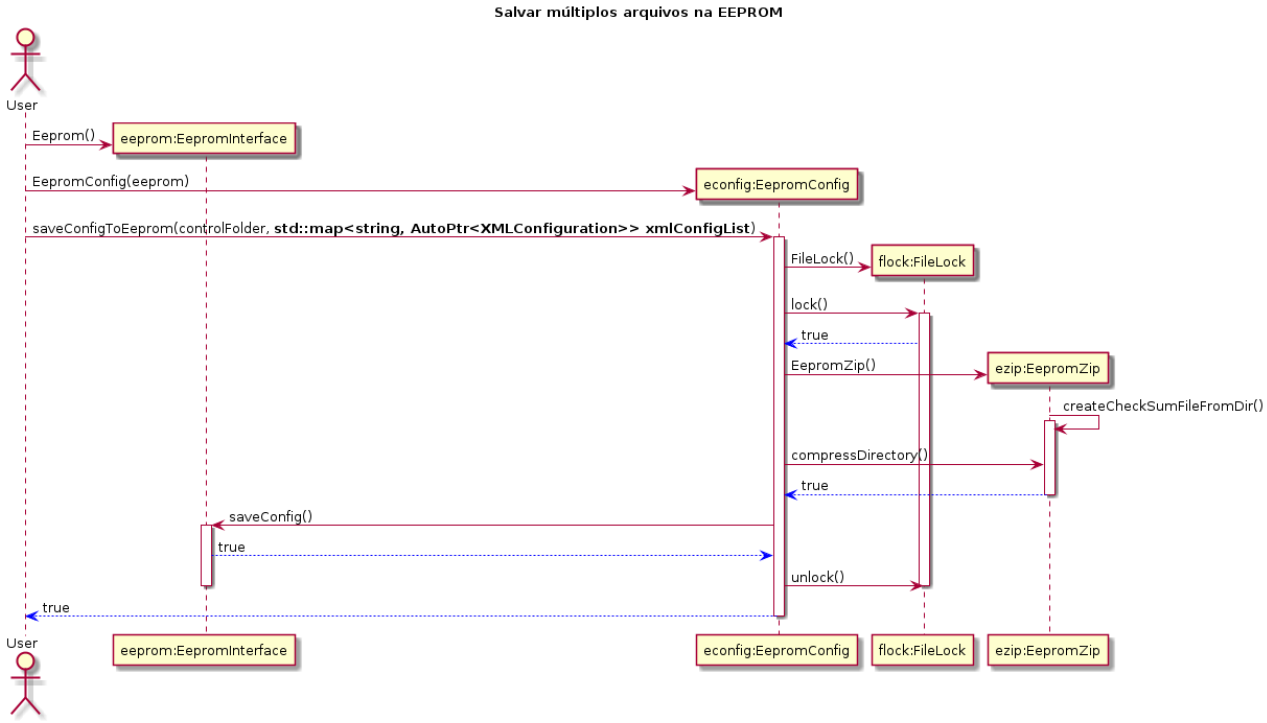

Figura 9 - Diagrama de sequência da operação de escrita de vários arquivos na EEPROM

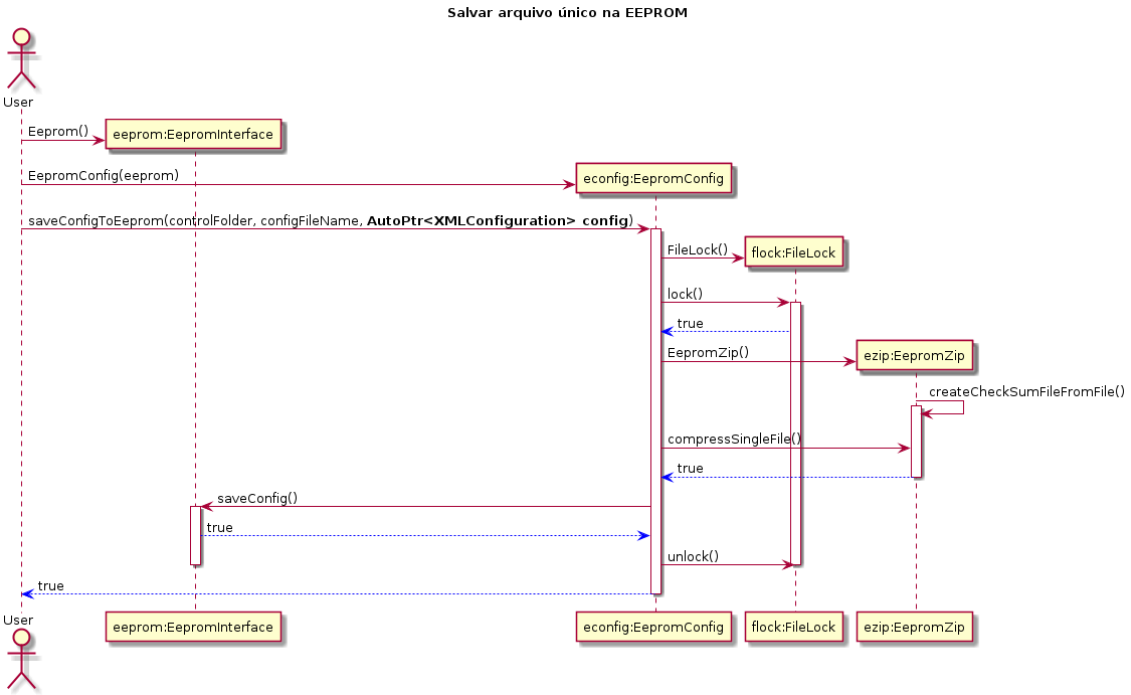

Figura 10 - Diagrama de sequência da operação de escrita de um arquivo na EEPROM

4.3.4 ALGORITMO DE DADOS PREVIAMENTE ARMAZENADOS

A fim de acelerar o processo de escrita dos arquivos de configuração na memória EEPROM, foi desenvolvida uma estratégia para realizar uma leitura antes da escrita, para verificar se os dados a serem escritos já estão presentes na memória.

O algoritmo armazena os dados a serem escritos em um buffer de quatro kilo bytes. Em seguida a biblioteca realiza uma operação de leitura na memória e armazena os dados lidos em outro buffer de quatro kilo bytes. Após isto são comparados os dois buffers preenchidos, verificando se cada byte dos buffers possui a mesma informação. O diagrama da figura 11 descreve o algoritmo implementado.

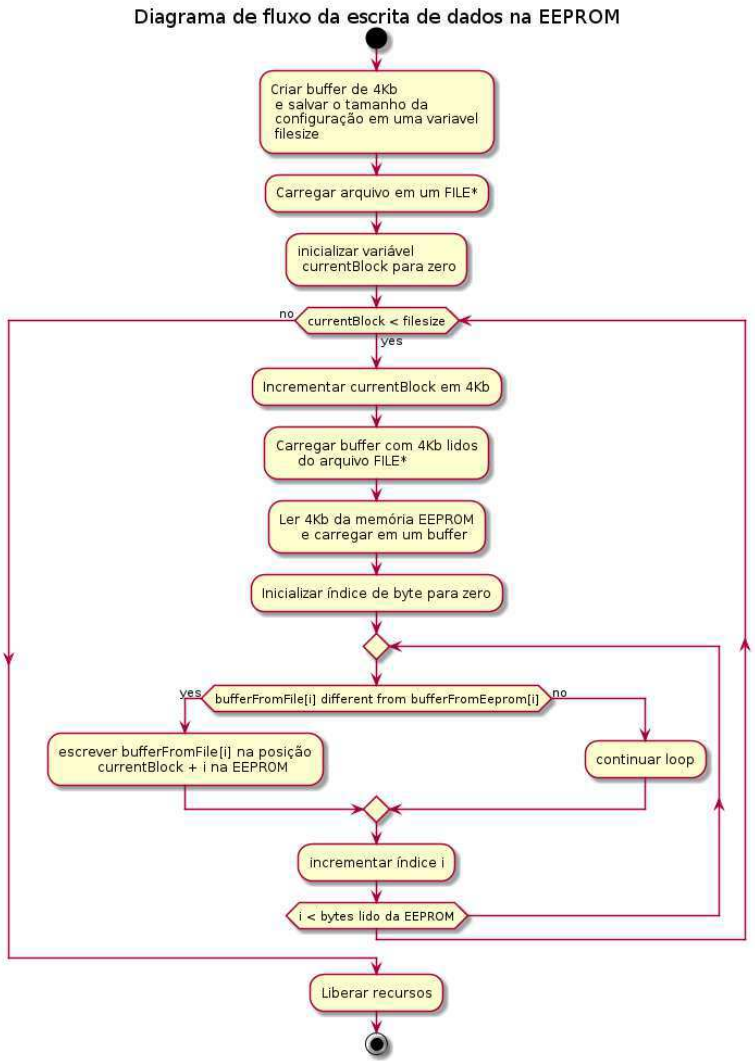

Figura 11 - Diagrama descrevendo o algoritmo para a escrita de dados na memória EEPROM

4.3.5 VERIFICAÇÃO DE ERROS

Para garantir a consistência dos dados escritos e lidos da memória EEPROM, foi adotado uma estratégia de verificação de erros. A solução adotada utiliza o algoritmo de *hash* MD5 (*Message-Digest 5*). Por ser um algoritmo unidirecional, é impossível a restauração dos arquivos originais a partir de seu *hash* MD5, garantindo a segurança dos dados.

O *hash* MD5 é calculado para cada arquivo salvo na memória EEPROM. Em seguida estes *hashs* são escritos em um arquivo "md5check.txt", que também será compactado e armazenado junto com as configurações. Quando uma operação de leitura é realizada, a biblioteca é então capaz de garantir a integridade dos arquivos, calculando o *hash* MD5 de cada arquivo lido e comparando com os *hashs* que foram previamente armazenados na operação de gravação.

Foi decidido que as informações de hash MD5 dos arquivos de configurações devem seguir o seguinte padrão:

<nome do arquivo>;<hash MD5>

No caso em que se deseja salvar um diretório contendo múltiplos arquivos de configuração, cada arquivo contido no diretório deverá possuir uma entrada no formato descrito acima. Portanto no caso de um diretório *configurations* conter 3 arquivos de configurações *file1.xml*, *file2.xml* e *file3.xml*, o arquivo md5check.txt deverá conter as seguintes entradas a fim de facilitar a leitura das informações no momento da verificação de erros.

file1.txt; F67C2BCBFCFA30FCCB36F72DCA22A817
file2.txt; F4A924C70F8E4FD4C64CD29ABE9B341F
file3.txt; C916D1C6FF019A10EFE7B8E579A278F2

4.3.6 TESTES DE UNIDADE

O CPqD adota uma cultura de testes de unidade para realizar a validação dos software desenvolvidos. Portanto antes mesmo da implementação de cada módulo e classe da biblioteca, um conjunto de testes foi desenvolvido. Um aspecto importante a se destacar é que é necessário

realizar o projeto do sistema com os testes de unidade em mente, utilizando conceitos como dependecy injection e a utilização de interfaces de classe entre outros.

Para os testes das classes EepromZip, FileLock e EepromConfig foram utilizados objetos mock da classe Eeprom, pois como a classe Eeprom realiza a leitura e escrita na memória, o processo de cross-compilar a biblioteca e em seguida transferir o binário gerado pelo gTest para a chave WSS se torna muito lento, desacelerando o processo de desenvolvimento e depuração de erros.

4.3.7 APLICATIVOS DE *BOOT*

O objetivo final do desenvolvimento da biblioteca é o do desenvolvimento de três aplicativos capazes de realizar operações de escrita e leitura na memória EEPROM. O primeiro aplicativo é responsável por escrever arquivos de configuração na memória EEPROM de vinte em vinte minutos, a fim de manter a vida útil da memória EEPROM por um longo período de tempo.

O segundo aplicativo é responsável pela leitura das configurações armazenadas na memória EEPROM. Este aplicativo será executado sempre que a chave WSS sofrer alguma operação de reinicio de sistema, como quando a chave é desligada e ligada ou quando a chave WSS for reinserida em um sistema. O terceiro aplicativo é responsável por manter a compatibilidade do novo sistema com chaves WSS legado. O aplicativo é capaz de realizar a leitura de uma configuração legado e em seguida exportar esta configuração em um arquivo XML nos padrões do novo sistema. De maneira análoga ao segundo aplicativo, este programa será executado em operações de *boot* da chave.

5 Conclusão

Após o término do período de estágio no CPqD pode-se destacar os vários conhecimentos adquiridos como o estudo prático de redes ópticas, desenvolvimento de software para dispositivos embarcados, gerenciamento de pacotes em um projeto de grande porte e sua validação através de testes de unidade.

Com a conclusão do estágio pode-se verificar a importância no crescimento profissional que uma experiência no ambiente de uma empresa pode proporcionar a formação do estudante. O dia a dia com pessoas experientes foi um fator determinante para um maior aprendizado em diversas áreas em que o CPqD atua, indo desde o estudo e pesquisa de redes ópticas ao desenvolvimento de software para dispositivos embarcados.

Constatou-se também que o CPqD está atuando com tecnologias no estado da arte nos sistemas em que está desenvolvendo, bem como a atualidade da grade curricular do curso de engenharia elétrica da UFCG, que proporcionou uma ótima base teórica e prática para as atividades que foram realizadas no decorrer do estágio.

6 Bibliografia

Wright, J. T., Da Silva, A. T., & Spers, R. G. (2010). **Internet Banda Larga: um Estudo Prospectivo Exploratório sobre a sua Penetração, Tecnologias de Conexão e Impactos no Brasil em 2020**. XIII SemeAd - Seminários em Administração.

COLLINGS, B. C.; ROORDA, P. ROADM-Based Networks. In: ZYSKIND, J.; SRIVASTAVA, A. **Optically Amplified WDM Networks**. San Diego: Academic Press, 2011. p. 23-45.

YAGHMOUR, K. et al. **Building Embedded Linux Systems**. 2ª Edição. ed. [S.l.]: O'Reilly Media, 2008.

SURHONE, L. M.; TIMPLEDON, M. T.; MARSEKEN, S. F. **Netconf**. 1ª Edição. ed. [S.l.]: Betascript Publishing, 2010.

ENNS, R. et al. Netconf RFC6241. **Netconf RFC6241**, 2011. Disponivel em: <https://tools.ietf.org/html/rfc6241>. Acesso em: 10 Outubro 2013.

BJORKLUND, M. YANG RPC6020. **YANG RFC6020**, 2010. Disponivel em: <http://tools.ietf.org/html/rfc6020>. Acesso em: 10 Outubro 2013.

MARTIN, R. C. **Clean Code**. 1ª Edição. ed. [S.l.]: Prentice Hall, 2008.

SMART, J. F. **Jenkins The Definitive Guide**. 1ª Edição. ed. [S.l.]: O´Reilly Media, 2011.