**Estudo Inicial Trabalho de Redes Convergentes**

**Estudo da especificação para definir escopos, entidades, atributos e métodos.**

**1) Aplicação deverá permitir o usuário inserir endereço IP e porta de conexão do**

**dispositivo remoto. Ex: IP 10.255.0.101 Porta – 22**

Entrada inicial do programa. Os valores de IP e Porta são passados para o objeto telnet que já está implementado. Precisam também passar um valor para a senha nesse passo.

**2) Aplicação deve permitir a configuração das portas WAN**

Comandos para configurar uma porta WAN:

(config)#interface s0/3/0

- Configurar qual interface da WAN

identificador do tipo (s, no exemplo) e identificador da porta (0/3/0)

(config-if)#ip address X.X.X.X X.X.X.X

-Configurar qual o ip da WAN

guardar o valor do ip

(config-if)#clock rate 128000

-Configurar o valor do clock (caso seja o master)

guardar o valor do clock

(config-if)#no shutdown

-Se está ou não ligado

no caso, seria uma bool mesmo

**3) Aplicação deve permitir a configuração das portas LAN**

Comandos para configurar uma porta LAN:

(config)#interface F0/0

- Configurar qual interface da LAN

identificador do tipo (F, no exemplo) e identificador da porta (0/0)

(config-if)#ip address X.X.X.X X.X.X.X

-Configurar qual o ip da LAN

guardar o valor do ip

(config-if)#no shutdown

-Se está ou não ligado

no caso, seria uma bool mesmo

**4) Aplicação deve permitir a configuração de senhas de Console/VTY e Enable**

Configurar a senha de VTY

(config)#line vty 0 4

verificar se são valores default, se forem, não precisam ser armazenados em variáveis e sim em constantes

(config-line)#password cisco

deve ser armazenado em uma String mesmo antes de ser enviada

Configurar a senha para o enable

(config)#enable password cisco

pode ser armazenado no mesmo objeto da senha de console/vty, mas usariam métodos diferentes

**5) Aplicação deve permitir a configuração de Rotas**

Inicialmente todos os dispositivos precisam usar roteamento estático ou dinâmico, logo não vai existir na mesma rede dispositivos com usando ambas as abordagens.

Estático

#show ip route

para ver a rota já estabelecida para o dispositivo

(config)#ip route 0.0.0.0 0.0.0.0 X.X.X.X

para adicionar a rota, os atributos ip e máscara são atributos certos

Dinâmico

#show ip route

para ver a rota já estabelecida para o dispositivo

(config)#router <protocolo>

para definir o protocolo

#version 2(só no caso do protocolo RIP)

(?)

#network X.X.X.X

definir o ip da rede

**6) Aplicação deve permitir a configuração do Nome do Dispositivo**

(config)#hostname CAUCAIA

trivial, só indica que deve ter um nome para o roteador e é interessante guardá-lo

**7) Aplicação deve permitir a configuração do horário do dispositivo**

(?)

Primeiro*:*

*clock timezone BRT -3 (dentro do config, clock timezone "zona de fuso horário" "diferença em relação ao GMT")*

Depois:

*clock set 10:50:00 Oct 26 2006 (dentro do enable, clock set "horário no padrão do exemplo ao lado")*

**8) Aplicação deve monitorar o status das interfaces**

Usando os mesmos atributos definidos para mudar as suas propriedades

**9) Aplicação deve fornecer visualização das rotas programadas**

Usando os comandos de show ip route dependendo do roteador, ou seja, mostrando as rotas estáticas ou dinâmicas.

**10) Aplicação deve fornecer visualização das configurações globais do dispositivo**

(?)

*Deve ser o "show run" a nível Enable.*

**11) Aplicação deve ser capaz de efetuar testes de conectividade para os dispositivos**

Enviando um simples comando de ping para os dispositivos que temos rotas definidas. Os dispositivos devem ter uma variável para indicar sua conectividade.

**12) Aplicação deve informar se o dispositivo é “Master” ou “Slave” no sincronismo**

(?)

*Dentro do enable,  
  
show controllers Serial0/1/0 ou show controllers Serial0/1/1*  
  
O que for Master vai mostrar DTE V.35 em uma das linhas, se for Slave, DCE V.35

**13) Aplicação deve informar a versão do sistema operacional instalado no dispositivo**

(?)

*show version (logo no nível inicial do router, o de usuário mesmo) ou show run (no nível de admin, depois do enable)*

**14) Aplicação deve apresentar um “Help” dos comandos existentes no modos do**

**dispositivo**

Texto hardcoded mesmo, na melhor das hipóteses escrito em um arquivo .txt que será exibido quando pedido.

Texto hardcoded mesmo, na melhor das hipóteses escrito em um arquivo .txt que será exibido quando pedido.

*Escrever ajuda no txt para todos os níveis e comandos? Precisa mesmo disso?  
  
Se for para ter ajuda, o VS deve ter suporte a caixas de comentário ao passar o mouse por cima de uma opção.*

**15) Aplicação deve informar de forma consolidada o status das interfaces e seus**

**respectivos endereçamentos IP.**

Uma tela mais detalhada mostrando os atributos das interfaces conectadas.

**16) Aplicação deve apresentar uma tela de monitoramento**

Parece redundante para mim, já seria a terceira tela com essas informações.

**Extras**

Os comandos de switch servem nesse router da cisco? Poderíamos adicionar para poder visualizar, editar e criar VLANs.

Também poderíamos gerar uma imagem da conectividade do router com os dispositivos mostrando as portas equivalentes.

**Classes Definidas:**

**IpModel**

string ip

string pass

int port

**InterfaceModel**

enum identificador

string porta. ex: 0/3/0

string ip

int clockrate

bool shutdown

**Configuration**

string senhavty

string senhaenable

string routername

enum hora

string sistemaOperacional

**Route**

enum tipo (estático/dinâmico)

string ip

string mascara

string rede

enum protocolo

Precisamos definir os enums do identificador de interface (1), da hora (2) e do protocolo (3).

(1): Pode ser um Enum **InterfaceType** que tem dois valores Serial e FastEthernet

(2): Pode nem ser um enum, que valores são possíveis para esse valor da “hora”?

(3): Também não lembro o que fica nesse valor, por favor, verifiquem e podemos ter um Enum do tipo **Protocol** com os valores possíveis.

**Classes de Controle:**

**ConfigutationHandler**

Responsável por lidar e fazer as alterações nas configurações. Para facilitar, usará singleton e sempre que for usada deve ser chamado estaticamente o método “getInstance()”. Ou melhor, talvez seja melhor que todos os Handlers sejam singletons.

**ConnectionHandler**

Responsável por lidar com os envios para conexão, ou seja, lida diretamente com a classe IpModel e envia as informações para o router, assim como trata caso dê algum erro do tipo “falta de senha” ou “senha inválida”. Usa a classe de ConfigurationHandler para ler os valores para as senhas. Essa classe que fará o teste de conectividade com todos os dispositivos.

**RouterHandler**

Responsável por manter a tabela de rotas do roteador, guardar as informações a respeito disso e sobre os status das interfaces do router. Lidará com as classes Route e InterfaceModel.

**RouterLevelHandler**

Classe responsável por lidar com a “posição” do programa no router, ou seja, garantir que estaremos no nível desejado para realizar dada operação. Ou seja, se vamos fazer uma operação que necessita do level enable, o RouterLevelHandler deve ir da posição que estamos até a posição do enable, descendo leveis ou subindo. Essa classe também guarda o estado atual.

**CommandHandler**

Classe específica de cada comando que faremos. Para cada comando deve haver um método específico com todos os parâmetros. Por exemplo, se vamos fazer um comando do tipo ip route 0.0.0.0 0.0.0.0 X.X.X.X então o CommandHandler deve ter um método do tipo:

void addStaticIpRoute(string ip, string mask)

Também, o commandHandler deve chamar o RouterLevelHandler e passar o nível desejado (o nível atual já está na sua configuração).

Explicando melhor as classes de controles:

Inicialmente, nós teremos uma caixa de diálogo para colocar o ip, porta e password, esses valores são passados para uma entidade do tipo IpModel. Esse objeto IpModel é passado como parâmetro para a classe ConnectionHandler que vai realizar a conexão e tratar caso dê algum erro, nesse momento, se der erro, a interface gráfica deve mostrar o erro e deve tratá-lo, pedindo uma nova senha, por exemplo, ou indicando que o ip de conexão é inválido. Digamos que a conexão foi bem sucedida.

Agora vamos supor que na aplicação eu queria verificar minha tabela de rotas. Eu vou no botão respectivo a isso e deve abrir uma tabela de rotas (esse processo é indiferente por enquanto e será mais bem definido quando tivermos as imagens das interfaces gráficas). Ao clicar no botão para ver as tabelas, chamaremos o CommandHandler chamando o método (1) que nos traz a tabela de rotas e vamos atualizar a tabela de rotas no RouterHandler, depois disso, usaremos a tabela de rotas do RouterHandler e mostraremos na tela.

(1): Nesse momento, o commandHandler deve chamar o RouterLevelHandler e passar que quer ir para o estado **config**, e ele deve verificar em que estado estamos e ir até o **config**. Esse passo é transparente para a aplicação. Se necessitar de alguma senha para acessar o **config**ou o **enable**, então o RouterLevelHandler deve pegar o valor da senha que fica armazenado no ConfigurationHandler.

Obviamente, o CommandHandler será a maior classe em termos de métodos, porque ela deve ter um método para cada comando que nossa aplicação realiza (não necessariamente para todos os comandos possíveis).

Eu pensei que a gente podia também ter um histórico na aplicação baseado nas rotas e tudo mais, seria trivial de fazer, depois escrevo mais sobre isso.

Espero que tenha ficado claro como a aplicação vai funcionar usando essa arquitetura.

Detalhe importante, **quando as senhas são salvas no ConfigurationHandler?** Quando nos conectamos e quando for verificado que a senha está vazia a aplicação deve pedi-la ao usuário em uma caixa de texto.