Universidade do Minho - Mestrado de Engenharia de Telecomunicações e Informática

Gestão e Virtualização de Redes

Módulo de Gestão de Redes - L-SNMPvS para Sistemas Domóticos

Grupo 3:

- João Pedro Costa Bastos - PG 57564
- Fernando João Santos Mendes - PG 55807
- Bruno Miguel Fernandes Araújo - PG 55806

Professores:

- Bruno Alexandre Fernandes Dias
- João Fernandes Pereira
- António Costa

Índice

1.Introdução

Este projeto tem como foco a implementação de um sistema de gestão domótico que visa suportar vários dispositivos, dispositivos estes, virtuais que procuram abstrair equipamentos de iluminação/temperatura como lâmpadas ou ar condicionados. O sistema segue uma estrutura simples ,sem necessidade de ligação a um sistema SNMP ou de inteligência artificial.

A comunicação entre o agente e o gestor é estabelecida utilizando uma versão simplificada do protocolo L-SNMP ou L-SNMPvS, este que utiliza uma codificação das mensagens em sequências de caracteres terminadas por '\0" (encapsulamento direto em datagramas UDP).

Desta forma, e em consonância com a matéria abordada na unidade curricular "Gestão e Virtualização de Redes" conseguimos pôr em prática sob as condições em cima descritas um programa que permite ao utilizador controlar remotamente todo o sistema sensores e de dispositivos através de uma interface gráfica intuitiva.

.

2.Implementação

2.1.Gestor

1. Classe Interface_completa:

Esta classe foi desenvolvida com o intuito de estabelecer uma conexão lógica entre a interface gráfica e o gestor. Dado que a interface gráfica corre sob o gestor, numa primeira análise a esta classe conseguimos identificar o bloco de código respetivo à criação da interface, sendo que para o sucedido foi utilizada a biblioteca *java.AWT* e explorada para obtermos o resultado que podemos averiguar na imagem 1.

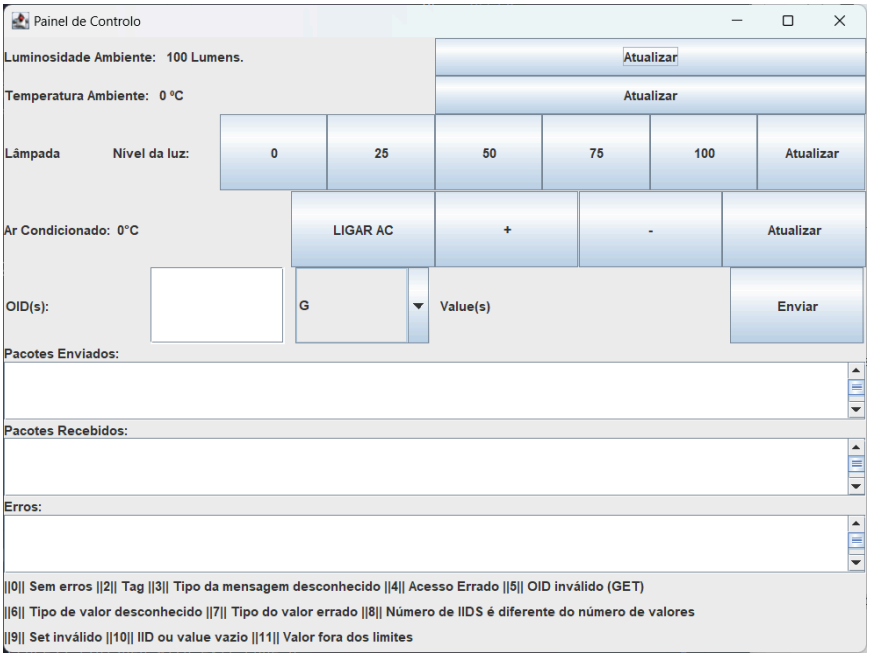

Imagem 1: interface gráfica do utilizador

Todos os botões e campos presentes na interface foram criados para dar resposta aos requisitos funcionais do projeto que dizem respeito às instruções feitas pelo cliente e são enviadas pelo gestor ao agente.
Estas instruções inserem-se em duas categorias, SET e GET, sendo SET a instrução ao agente de alterar um IID dado um valor escolhido pelo utilizador e GET a instrução ao agente de devolver o valor que está registado num dado IID. Todas as "queries" são controladas para evitar ambiguidades e reportar possíveis erros, cujos podemos analisar na figura 1 , todavia esta secção será explicada na classe do agente, visto que é no mesmo que tal acontece.

A interface é uma representação dos dispositivos abstraídos na L-MIB, como tal podemos verificar que para o nosso projeto a L-MIB foi populada com 2 atuadores e 2 sensores, sendo os valores dos sensores representados nas primeiras 2 linhas "Luminosidade Ambiente" e "Temperatura ambiente" e os 2 atuadores nas seguintes 2 linhas, "Lâmpada" e "Ar Condicionado". Cada um dos

dispositivos detém um botão de atualizar que permite realizar uma instrução do tipo GET com o intuito de ao receber o valor registado na L-MIB atualizar o mesmo na interface. O mesmo acontece para dispositivos que atuam de alguma forma, todavia estes detém mais funcionalidades como escolher a luminosidade da lâmpada ou até mesmo desligar (valor = 0) e ligar/ desligar o ar condicionado seguido do regulador de temperatura. Todos estes botões permitem ao gestor realizar instruções do tipo SET, ou seja, dado um IIDdefinido é atualizado o seu valor na L-MIB.

Além destes campos na interface, o utilizador tem a opção de fazer querys por extenso (imagem 2) especificas ao(s) IID(S) e valor(es) que o mesmo desejar.

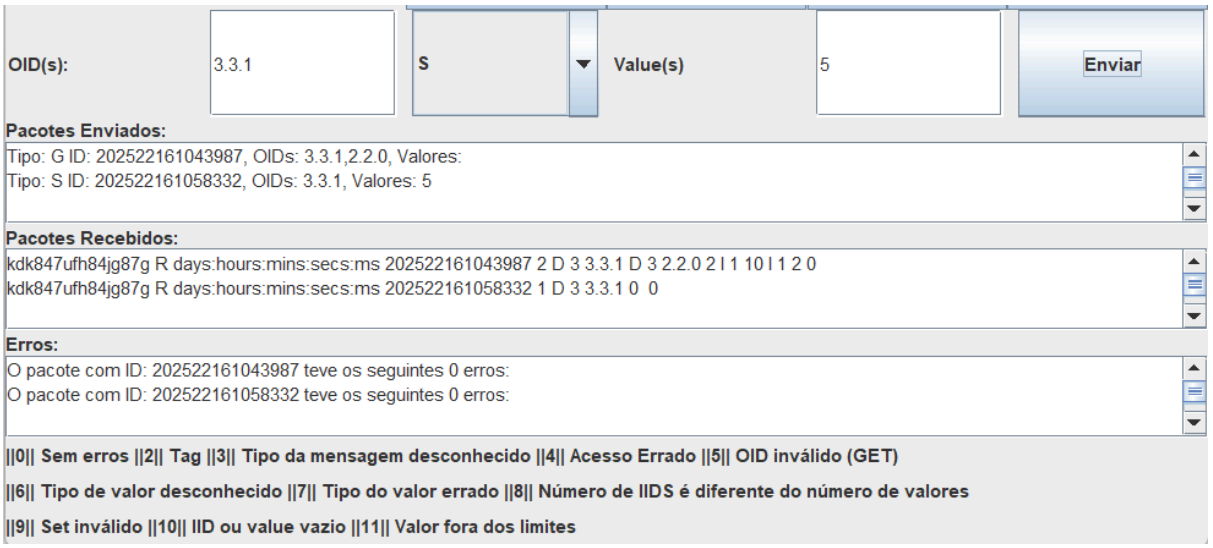

Imagem 2: Exemplo de instruções GET / SET

Nesta classe, numa segunda análise conseguimos identificar o papel do gestor responsável por gerar um pacote com todos os parâmetros que seguem o protocolo L-SNMP e encapsulados num socket segundo o protocolo UDP, para cada uma das funções da interface (botão de atualizar, regular temperatura, das instruções personalizadas, etc). Tal é feito através da função "SendPackageToAgent()". Nesta função é construído o pacote com todos os campos necessários (imagem 3) através da instância do tipo "Package_builder" especificada a seguir. Para que o pacote seja enviado num socket ,que é instanciado aquando a criação de uma interface, é utilizada a classe "Connection_manager".

Da mesma forma que esta função tem como propósito enviar os pacotes para o agente também os recebe e realiza o devido tratamento de erros e da divisão dos tópicso da mesma para o utilizador poder averiguar na interface o sucedido (na área dos pacotes recebidos e de erros).

2. Classe Package_builder:

Esta classe foi desenvolvida com o propósito de construir os pacotes segundo o formato da trama referenciado na imagem 1.

```
Tag | Type | Time-Stamp | Message-Identifier | IID-List | Value-List | Error-List
```

Imagem 3: Formato da trama

Esta classe é uma ferramenta ao dispor do gestor que é responsável por criar um pacote através da função “SNMP_Protocol”. Através desta função é possível retornar a informação a ser enviada para o agente encapsulado, ou seja, é devolvida a String da mensagem a ser enviada sendo os campos separados por “/0”.

3. Classe Connection_manager:

Esta classe foi desenvolvida com o intuito de criar a estrutura necessária para enviar juntamente com o socket a mensagem encapsulada pelo gestor como referenciado em cima, e receber o pacote.

4. Classe Main;

A classe "Main" tem como principal objetivo inicializar a interface gráfica para o utilizador e inicializar e popular uma estrutura de dados que representa uma MiniMIB, uma representação simbólica no gestor da L-MIB (Light Management Information Base), utilizada para armazenar e organizar informações sobre um dispositivo, os seus sensores, seus atuadores e os tipos de cada IID respetivo (utilizado na construção dos pacotes).

2.2.Agente

O Agente é um servidor equipado com uma L-MIB, capaz de receber simultaneamente pedidos de get e set de diversos Gestores. Para cada solicitação, o Agente cria uma thread específica, garantindo o tratamento individualizado de cada pedido.

A classe principal onde está este tratamento de cada pedido individualizado , chama-se "Connection_manager", esta recebe o socket e packet relacionado com o pedido do Gestor, bem como a L-MIB que foi criada na antes do while true, assim todos os pedidos dos diferentes Gestores estão a consulta/alterar a mesma L-MIB.

Estrutura da L-MIB:

Estruturamos a L-MIB de forma a que melhor representasse o proposto e que permitisse uma boa escalabilidade.

A **L-MIB** é constituída por 1 HashMap em que cada key é o id de cada "Structure", e os valores são um "Equipment", este é uma classe que contém 3 subclasses ," Devices","Sensors" e "Actuators" e 4 métodos abstratos, getValue, setValue, getnObjects e getInstances (Pelo nome é possível perceber o seu propósito.)
Cada uma destas tem uma lista de arrays de "MibObjects", o tamanho deste array depende do número de objetos presente em cada estrutura. (Lista pois queremos ter várias instâncias)
Por fim os MibObjects , representam ,como o próprio nome indica , os objetos da mib, estes têm 6 variáveis strings, correspondentes às suas características:

- ID -O índice correspondente a este.
- Name - O nome deste objeto.
- Type - O tipo deste objeto, se é Inteiro, String ou Timestamp.
- Acess - Se este objeto é read-only ou read-write.
- Description - Descrição do objeto.
- Value - O valor que se encontra neste objeto.

Além destas variáveis , esta classe tem um Lock de Read and Write, que mencionaremos com maior detalhe mais à frente, na questão da concorrência.

A melhor forma de explicar a estrutura , diríamos que é mostrando um get.
Na classe "Connection_manager" onde é tratado cada pedido individualmente, temos a seguinte chamada para dar get de um valor.

```
valor+=this.mib.getMib().get(ids[0]).getValue(ids[2],ids[1])+"\0";
```

Por exemplo, caso queiramos buscar o valor do objeto "reset" da estrutura Devices, teríamos o IID 1.10.1.
ids[0] será igual a 1 e com o método get dos hashmaps, buscamos o Equipment , device.
ids[1] será a instância, neste exemplo será 1.
ids[2] será o id do objeto que neste exemplo é 10.
Neste device temos também o método getValue que , recebendo o id do objeto e a instância ele faz a busca no objeto pretendido.

```
valor += this.InstanceOfDevices.get(Integer.parseInt(instance))[Integer.parseInt(idd)].getValue();
```

E claro por fim, o MibObject tem também um método getValue onde este retorna o valor.

```
valor += this.value;
```

Parse/Deparse:

É importante mencionar que os campos de Value-List e Error-List da mensagem de resposta ao pedido do Gestor são construídos ao longo desta função parse, tendo as variáveis nvalues, valor, nerros e error como a representação das componentes colocadas nesses campos.

Primeiramente a mensagem é separada por '\0', depois é feita uma verificação dos erros que possam acontecer antes da análise dos IID's e dos Values, erros como tag errada, tipo da mensagem desconhecido , número de IID's não coincidir com o número de Values, entre outros.
De seguida é feita uma distinção entre pedidos do tipo Get ou do tipo Set para terem o tratamento específico destes.

Parse de pedidos Get

É percorrido o campo dos IID's list tendo em conta o número destes, cada iteração do ciclo, é enviado o respetivo IID para uma função verifica as diferentes variações de IID's válidos que existem, se concluir que este não é válido, ela retorna um tuplo com 0 values e uma string com uma mensagem que afetará a variável de error, sendo o erro 5 adicionado a ela. Se este IID for válido , ela faz get do valor ou valores (depende do IID) e devolve um tuplo com o número de values e com a Value-List já no formato TLV para cada valor (no fim é adicionado 0 à variável error assim como incremento de nerros).

Parse de pedidos Set

É percorrido o campo dos IID's list tendo em conta o número destes, se não existir value para esse IID é adicionado o erro 9 á variável error, se existir é feita a chamada da função que verifica se o IID é válido se este for é feito um set desse value na L-MIB. (Existem vários níveis de verificação , se o objeto pode ser escrito, se o tipo deste value é o mesmo que o do objeto na L-MIB, etc..).
Ao contrário da função de get , esta função de set retorna apenas os erros, e a Value-List estará presente na mensagem de resposta mas estará vazia , assim como o número 0 para representar o número destes.

Deparse

O deparse é feito de uma forma muito simples, tendo em conta que a função parse já devolve o campo da Value-List juntamente com o da Error-List, precisamos apenas de reconstruir o pacote no formato L-SNMPvS para depois ser enviado.

Erros:

Dado que a comunicação não é síncrona é necessário utilizar um identificador único que identifique de forma unívoca os pacotes e dado que não existem correção de erros foi necessário implementar um campo referente aos mesmos que traduzisse ao utilizador os problemas de sintax, de processamento ou problemas relacionados com instruções. Como tal, podemos identificar na seguinte imagem os erros que são possíveis identificar.

```
||0|| Sem erros ||2|| Tag ||3|| Tipo da mensagem desconhecido ||4|| Acesso Errado ||5|| OID inválido (GET)
||6|| Tipo de valor desconhecido ||7|| Tipo do valor errado ||8|| Número de IIDS é diferente do número de valores
||9|| Set inválido ||10|| IID ou value vazio ||11|| Valor fora dos limites
```

Tratamento de Erros nos Gets e Sets

No funcionamento da L-MIB, os métodos de GET e SET lidam com os IIDs (Object Identifiers) de formas distintas, especialmente no que diz respeito ao tratamento de erros.

Operação de Get

Quando um GET é realizado, os valores dos IIDs solicitados são devolvidos apenas se estiverem corretos. Caso ocorra algum erro, nenhum valor é retornado. Em vez disso, é gerada uma ErrorList, onde:
- 0 indica que o IID é válido e seu valor foi retornado corretamente.
- 5 indica que o IID é inválido, ou seja, não existe na L-MIB.

Exemplo: Se for feita uma requisição de GET para três IIDs, e a ErrorList retornarn [3, 0, 5, 0], isso significa:
- O primeiro IID foi processado corretamente (0).
- O segundo IID não existe (5).
- O terceiro IID foi processado corretamente (0).

Neste caso, os valores dos IIDs válidos são retornados, enquanto os IIDs inválidos simplesmente não possuem valor associado na resposta.

Operação de Set

Na operação de SET, os IIDs são modificados para os valores fornecidos. No entanto, o comportamento diante de erros é diferente dos gets:
- Se um IID for válido, o seu valor é atualizado.
- Se um IID for inválido, ele ainda aparece na ErrorList, mas os outros IIDs válidos continuam sendo atualizados corretamente.
- Caso o número de IIDs seja maior que o número de valores fornecidos, o erro também é registrado.

Na ErrorList do SET:
- 0 significa que o IID foi atualizado com sucesso. .
- 9 significa que houve um erro porque não foi fornecido um valor correspondente ao IID.

Exemplo1: Se a ErrorList retornar [3, 0, 9, 0], isso indica:
1. O primeiro IID foi atualizado corretamente (0).

2. O segundo IID não tinha um tipo de valor correspondente para dar SET, então gerou erro (9).
3. O terceiro IID foi atualizado corretamente (0).

Apesar do erro no segundo IID, os restantes que são válidos foram processados corretamente nos SET (Foram colocados na L-MIB).

Concorrência:

Para assegurarmos a concorrência entre threads e evitar que ocorra um get de um valor que está a ser alterado, etc… decidimos trabalhar com locks de Read and Write, sendo que os read serão usados no método de get e write no método de set.
Assim o fizemos, na classe MibObjects colocamos um Readlock no método getValue e um writelock no método setValue que levarão unlock após efetuarem a leitura/escrita .
Uma thread que está num ReadLock pode efetuar a leitura quando outra se encontra também num ReadLock, esta só ficará bloqueada se houver uma Thread com um WriteLock. Assim como uma thread num WriteLock , fica bloqueada se houver outra num WriteLock.

Resumidamente,
1-RL 2- RL = Nenhuma thread bloqueada.
1-RL 2-WL = Thread 1 bloqueada até a Thread 2 acabar de fazer as alterações na L-Mib..
1-WL 2-WL = Uma delas fica bloqueada até a outra acabar de fazer as alterações na L-Mib (Depende de quem deu lock primeiro).

Desta forma apenas os objetos que estão envolvidos nos pedidos que estão a ser analisados ao mesmo tempo nas threads do agente, é que ficam bloqueados , sem haver necessidade de bloquear ou a L-MIB completa, ou os “Grupos” desta, bloqueamos ao menor nível possível.

Emulação dos dispositivos:

Temos presentes 1 grupo de Devices com 2 sensores e 2 atuadores.
Instância 1: Sensor de Temperatura e um Atuador que é um Ar condicionado (A/C)
Instância 2: Sensor de Luminosidade e um Atuador que representa uma Lâmpada.
Para a emulação do comportamento dos dispositivos, nós criamos um método chamado “emulacao” no construtor da L-MIB, neste temos uma thread que ficará sempre acordada, esta está encarregue de criar uma nova thread após o valor do objeto beaconRate do Device vezes * 1000 (beaconRate está em segundos e o Thread.sleep utiliza milissegundos então foi necessário converter multiplicando por 1000).
Esta nova thread fará a chamada do método “update”, neste temos a emulação destes dispositivos.
Para o Device, quando o valor do objeto reset é 1, é realizado um "reset", os objetos upTime e lastTimeUpdated são atualizados com o tempo atual, e o objeto reset é alterado para seu valor default, 0.
Para os Sensores/Atuadores analisamos o valor no objeto status dos Atuadores e efetuamos um incremento ou decremento no valor status dependendo se o status do sensor é menor ou maior que o do atuador, respetivamente. (Ps: A instância do atuador analisado e a do sensor cujo status foi alterado é a mesma).

Por fim temos também a questão dos atuadores estarem desligados, aí o seu respectivo sensor irá para um valor default de 10.

3. **Conclusão**

O desenvolvimento do projeto em questão permitiu uma aprofundada compreensão dos conceitos adquiridos na unidade curricular "Gestão e Virtualização de Redes", através da sua aplicação num ambiente domótico.A implementação do sistema com recurso ao protocolo simplificado L-SNMPvS mostrou-se uma abordagem eficaz para a comunicação entre o gestor e o agente, permitindo uma gestão remota intuitiva e eficiente dos dispositivos simulados. A estruturação da L-MIB em formato escalável e a implementação de mecanismos de concorrência garantem a integridade dos dados, mesmo em cenários com múltiplos gestores a interagir simultaneamente.Adicionalmente, o tratamento de erros nas operações GET e SET assegura o funcionamento robusto do sistema, permitindo a identificação e a resolução de falhas de forma clara. Muitas das dificuldades que surgiram neste projeto foram maioritariamente no formato das mensagens do protocolo L-SNMPvS, achamos que foi bastante abstrata e se calhar um pouco confusa, mas acabamos por entender o pretendido com ajuda do docente João Fernandes Pereira.

Em suma, este projeto demonstrou a viabilidade e eficiência de um sistema de gestão domótico baseado em L-SNMPvS, proporcionando uma base sólida para futuras expansões e aprimoramentos.