



UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

MESTRADO INTEGRADO EM ENGENHARIA DE TELECOMUNICAÇÕES E INFORMÁTICA

GESTÃO DE REDES

Trabalho Prático Nº3

LUÍS FILIPE FREITAS FERREIRA A70016



EDUARDO MIGUEL MENDES DA SILVA A70216



10 de Fevereiro de 2017

Índice

	Página
1 Introdução	5
2 Objetivos	6
3 Requisitos	7
4 Servidor de Números Aleatórios	8
4.1 Fase A	9
4.2 Fase B	12
4.3 Fase C	16
4.4 Fase E	19
5 Conclusão	21

Lista de Figuras

	Página
1 Grupo <i>unpredictableParam.</i>	9
2 Grupo <i>unpredictableTable.</i>	10
3 Teste realizados ao sistema na fase B.	15
4 Teste realizados ao sistema na fase B.	18
5 Grupo <i>unpredictableTable.</i>	20

Lista de Tabelas

Página

Lista de exemplos

	Página
1 Exemplo código JAVA <i>unpredictable</i> MIB	10
2 Método utilizado para ler do ficheiro de configuração.	12
3 Método utilizado para guardar os parâmetros do ficheiro de configuração na MIB.	13
4 Método utilizado para da comunidade.	13
5 Método utilizado para adição da configuração VACM.	14
6 Método utilizado para registar os objetos de gestão.	14
7 Método utilizado para obtenção das <i>seeds</i>	16
8 Método utilizado para obtenção da matriz de números aleatórios. . .	16
9 Método utilizado para guardar a matriz na tabela da MIB.	17
10 Exemplo Comparação da chave de configuração e pausa das operações de refrescamento	19
11 Exemplo Classe <i>RefreshTimer</i>	20

1. Introdução

No âmbito da unidade curricular de Gestão de Redes, do curso de Engenharia de Telecomunicações e Informática, foi proposto aos alunos a realização de três trabalhos práticos. Este relatório diz respeito ao último trabalho prático, o número 3.

O principal objetivo deste trabalho prático, consiste em consolidar os conhecimentos adquiridos na unidade curricular, nomeadamente sobre a utilização prática do modelo de gestão preconizado pelo INMF (*Internet Standard Management Framework*) que engloba componentes como o protocolo SNMP (*Simple Network Management Protocol*) e as MIBs (*Management Information Bases*).

Foi pedido aos alunos a conceção de agente SNMP que seja um servidor de geração de números aleatórios cuja interface comunicacional seja através de SNMPv2c numa linguagem à escolha, no nosso caso foi usado a linguagem Java.

Para a resolução deste projeto foi necessário a utilização de APIs SNMP e diversas ferramentas adicionais que nos permitem a construção do que é requerido no enunciado, ou seja, a implementação de uma ferramenta de gestão.

"if you need motivation don't do it"

Elon Musk

2. Objetivos

Este trabalho prático tem como objetivo a consolidação da utilização prática do modelo de gestão preconizado pelo *Internet-standard Network Management Framework* (INMF), dando especial relevo ao *Simple Network Management Protocol* (SNMP) e às *Management Information Bases* (MIBs).

Um outro objetivo consiste na utilização de APIs SNMP para construção de ferramentas de gestão (agentes e gestores) e a investigação da aplicação do SNMP em sistemas de gestão nos mais variados ramos da engenharia aplicacional.

3. Requisitos

Como requisito para o desenvolvimento do projeto prático é necessário um sistema com um agente SNMPv2c instalado (preferencialmente o NET-SNMP) e um pacote de desenvolvimento numa linguagem de programação que disponibilize APIs para construção de gestores e agentes SNMPv2c (como por exemplo o SNMP4J).

Os requisitos funcionais deste trabalho passam pela geração de números aleatórios de acordo com as parametrizações definidas pelos utilizadores. Para isso é necessário o desenvolvimento de uma MIB que permita uma semântica e sintaxe adequadas à correta abstração dos requisitos funcionais. Por último, deve ser construído um agente que implemente o serviço SNMPv2c.

Este agente quando executado, deve consultar um ficheiro de configuração com os parâmetros de inicialização e construir uma matriz $M_{t,k}$ com as sementes do ficheiro indicado no ficheiro de configuração. A matriz $M_{t,k}$ será usada para gerar a tabela de números aleatórios a implementar como instância de MIB do agente, ou seja, a tabela da MIB conterá N linhas a que equivalem N números aleatórios (um número de D dígitos por cada linha).

Cada número aleatório i da tabela da MIB corresponde à concatenação circular de D elementos de cada linha i da sub-matriz $L_{N,D}(p,q,M)$, em que p e q são calculados dinamicamente a cada refrescamento da tabela de números aleatórios.

Como argumento do programa (na linha de comandos ou através dum interface dinâmico) deve ser indicada a chave de configuração para autorização da operação de *reset* do agente através do SNMP.

4. Servidor de Números Aleatórios

O foco principal deste projeto prático é o desenvolvimento de um agente SNMP que seja um servidor que gera números aleatórios. Este tipo de serviço remoto através da *Internet* está disponível a diferentes sistemas que necessitem de formas de obter números aleatórios não controlados por processos internos ao próprio sistema computacional, como por exemplo o serviço *random.org* cuja interface comunicacional é efetuada através de HTTP. O objetivo deste trabalho consiste na implementação de um serviço que possua uma interface comunicacional através de SNMPv2c e não através de HTTP.

O primeiro passo realizado foi a definição dos requisitos funcionais, ou seja, o tipo de resultados esperados tendo em conta as possíveis parametrizações que os utilizadores podem realizar.

Após o primeiro passo estar completo, o objetivo seguinte passou pela definição de uma MIB com os grupos de objetos com a semântica e sintaxe adequadas à correta abstração dos requisitos funcionais estabelecidos anteriormente.

Por fim, o foco de trabalho consistiu na construção e teste do software do agente que implementa o serviço num agente SNMPv2c, desenvolvido na linguagem de programação Java.

4.1 Fase A

Esta fase tinha como propósito a construção da *unpredictable MIB* que será utilizada para o agente SNMP guardar os parâmetros de configuração e os números aleatórios. Foram utilizados dois programas externos: o *MIB Designer* e o *AgenPro*.

O *MIB Designer* serve para construir e editar a MIB através de uma interface gráfica facilitando assim a sua conceção, evitando erros de sintaxe e semântica. O *AgenPro* foi utilizado para gerar o código Java da *unpredictable MIB* através do ficheiro criado pelo *MIB Designer*.

A MIB possui dois grupos: *unpredictableParam* e *unpredictableTable*. O grupo *unpredictableParam* possui os objetos escalares (R, N e D) que representem os parâmetros de funcionamento e ainda um objeto escalar especial (do tipo *string* e apenas com permissões de escrita) que servirá para verificar autorizações para a operação *reset* do agente como podemos ver na Figura 1.

```
unpredictableParam OBJECT-GROUP
    OBJECTS {
        unpredictableRefresh,
        unpredictableEntries,
        unpredictableDigits,
        unpredictableReset,
        unpredictableCommunity,
        unpredictablePort,
        unpredictableSeed
    }
    STATUS      current
    DESCRIPTION
        "A collection of scalar objects with the starting parameters (R,N,D). "
    -- 1.3.6.1.3.1.3.1.1
    ::= { uminhoMIBGroups 1 }
```

Figura 1: Grupo *unpredictableParam*.

O grupo *unpredictableTable* contém a tabela de N números aleatórios com apenas duas colunas, uma para o índice de entrada (chave da tabela) e outra para o número aleatório (sequência de D dígitos hexadecimais), como está exemplificado na Figura 2.

```
unpredictableTable OBJECT-GROUP
    OBJECTS {
        randomKey,
        randomNumber
    }
    STATUS current
    DESCRIPTION
        "Group with the random number table."
-- 1.3.6.1.3.1.3.1.2
::= { uminhoMIBGroups 2 }
```

Figura 2: Grupo *unpredictableTable*.

Com a *unpredictable MIB* definida obtém-se o código Java exemplificado no Exemplo 1.

Exemplo 1: Exemplo código JAVA *unpredictable MIB*

```
1      public static final OID oidUminhoGrMib =
2      new OID(new int[] { 1,3,6,1,3,1 });
3
4      // Identities
5      // Scalars
6      public static final OID oidUnpredictableRefresh =
7      new OID(new int[] { 1,3,6,1,3,1,1,1,0 });
8      public static final OID oidUnpredictableEntries =
9      new OID(new int[] { 1,3,6,1,3,1,1,2,0 });
10     public static final OID oidUnpredictableDigits =
11     new OID(new int[] { 1,3,6,1,3,1,1,3,0 });
12     public static final OID oidUnpredictableReset =
13     new OID(new int[] { 1,3,6,1,3,1,1,4,0 });
14     public static final OID oidUnpredictableCommunity =
15     new OID(new int[] { 1,3,6,1,3,1,1,6,0 });
16     public static final OID oidUnpredictablePort =
17     new OID(new int[] { 1,3,6,1,3,1,1,7,0 });
18     public static final OID oidUnpredictableSeed =
19     new OID(new int[] { 1,3,6,1,3,1,1,8,0 });
20     // Tables
21
22     // Notifications
23
24     // Enumerations
25
26     // TextualConventions
27
```

```

28 private static final String TC_MODULE_SNMPV2_TC = "SNMPv2-TC";
29 private static final String TC_DISPLAYSTRING = "DisplayString";
30
31 // Scalars
32 private MOScalar<Integer32> unpredictableRefresh;
33 private MOScalar<Integer32> unpredictableEntries;
34 private MOScalar<Integer32> unpredictableDigits;
35 private MOScalar<OctetString> unpredictableReset;
36 private MOScalar<OctetString> unpredictableCommunity;
37 private MOScalar<Integer32> unpredictablePort;
38 private MOScalar<OctetString> unpredictableSeed;
39
40 // Tables
41 public static final OID oidRandomEntry =
42     new OID(new int[] { 1,3,6,1,3,1,1,5,1 });
43
44 // Index OID definitions
45 public static final OID oidRandomKey =
46     new OID(new int[] { 1,3,6,1,3,1,1,5,1,1 });
47 public static final OID oidRandomNumber =
48     new OID(new int[] { 1,3,6,1,3,1,1,5,1,2 });

```

4.2 Fase B

Para a segunda fase foi necessário construir e testar um agente SNMP que lê do ficheiro de configuração e implemente apenas o grupo *unpredictableParam* da *Unpredictable MIB*, ignorando assim o ficheiro com as sementes iniciais indicado no ficheiro de configuração.

Desta forma, foi necessário a utilização de uma API que permite a implementação das funcionalidades requeridas para um agente SNMP, no nosso caso utilizámos o SNMP4J, que nos permitiu a implementação em linguagem Java.

Para a construção do agente, foi criada uma classe que estende a classe abstrata *BaseAgent* fornecida pela API SNMP4J-Agent. Esta classe define uma estrutura que permite escrever agentes SNMP oferecendo já os métodos necessários para o efeito.

Após o início da execução do agente, este primeiramente acede ao ficheiro de configuração, que necessita de estar na mesma diretoria que o projeto. Este ficheiro será lido, os valores dos parâmetros interpretados e armazenados para que posteriormente possam ser guardados na MIB como podemos ver nos Exemplos 2 e 3.

Exemplo 2: Método utilizado para ler do ficheiro de configuração.

```
1      private void configurationParameters() throws IOException {
2          /*
3           * ArrayList with the configuration parameters
4           * index
5           * 0 - porta udp
6           * 1 - community-string
7           * 2 - refreshRate
8           * 3 - table-size
9           * 4 - number-size
10          * 5 - first-seed path
11         */
12         File conf = new File("unpredictable-conf");
13         if (conf.exists()) { // if exists it's true
14             BufferedReader fileParameters;
15             String param;
16             fileParameters = new BufferedReader(new FileReader("unpredictable-conf
17                                                         "));
18             while ((param = fileParameters.readLine()) != null) {
19                 String[] splitParam = param.split(":");
20                 params.add(splitParam[1]);
21             }
22         } else {
```

```

22         System.out.println("Sem ficheiro de configuracao.\n Coloque na pasta
           respectiva!");
23         System.out.println("Pasta " + System.getProperty("user.dir"));
24         System.exit(0);
25     }
26 }

```

Exemplo 3: Método utilizado para guardar os parâmetros do ficheiro de configuração na MIB.

```

1     private void setParameters() {
2         String porta = params.get(0);
3         community = params.get(1);
4         refresh = params.get(2);
5         tableSize = params.get(3);
6         numberSize = params.get(4);
7         seedPat = params.get(5);
8         String seedPath = Normalizer.normalize(seedPat, Normalizer.Form.NFD);
9         seedPath = seedPath.replaceAll("[\\p{InCombiningDiacriticalMarks}]", "");
10
11         uminhogrmib.getUnpredictableCommunity().setValue(getOctetString(community)
            );
12         uminhogrmib.getUnpredictableSeed().setValue(getOctetString(seedPath));
13         uminhogrmib.getUnpredictableRefresh().setValue(getInteger(refresh));
14         uminhogrmib.getUnpredictableEntries().setValue(getInteger(tableSize));
15         uminhogrmib.getUnpredictableDigits().setValue(getInteger(numberSize));
16         uminhogrmib.getUnpredictablePort().setValue(getInteger(porta));
17     }

```

Após os valores serem corretamente armazenados na MIB, o sistema inicia a execução propriamente dita do agente. Para este efeito, são executados vários métodos que asseguram o correto funcionamento deste. Nos seguinte Exemplos 4 e 5, podemos ver a adição da comunidade aos mapeamentos de nomes de segurança necessários para o SNMPv2c e a adição inicial da configuração VACM, que consiste na implementação concreta do SNMP-VIEW-BASED-ACM-MIB, ou seja, a configuração do modelo de acesso à view.

Exemplo 4: Método utilizado para da comunidade.

```

1     protected void addCommunities(SnmpCommunityMIB communityMIB) {
2         Variable[] com2sec = new Variable[]{new OctetString(community),
3             new OctetString("c" + community), // security name
4             getAgent().getContextEngineID(), // local engine ID
5             new OctetString(community), // default context name
6             new OctetString(), // transport tag
7             new Integer32(StorageType.nonVolatile), // storage type
8             new Integer32(RowStatus.active) // row status
9     };

```

```

10     M0TableRow row = communityMIB.getSnmpCommunityEntry().createRow(new
        OctetString("public2public").toSubIndex(true), com2sec);
11     communityMIB.getSnmpCommunityEntry().addRow((SnmpCommunityMIB.
        SnmpCommunityEntryRow) row);
12 }

```

Exemplo 5: Método utilizado para adição da configuração VACM.

```

1     protected void addViews(VacmMIB vacm) {
2         vacm.addGroup(SecurityModel.SECURITY_MODEL_SNMPv2c,
3             new OctetString("c" + community),
4             new OctetString("v1v2group"),
5             StorageType.nonVolatile);
6
7         vacm.addAccess(new OctetString("v1v2group"),
8             new OctetString(community),
9             SecurityModel.SECURITY_MODEL_ANY, SecurityLevel.NOAUTH_NOPRIV,
10            MutableVACM.VACM_MATCH_EXACT, new OctetString("fullReadView"),
11            new OctetString("fullWriteView"), new OctetString("fullNotifyView"
12                ),
13            StorageType.nonVolatile);
14
15        vacm.addViewTreeFamily(new OctetString("fullWriteView"),
16            new OID("1.3"),
17            new OctetString(),
18            VacmMIB.vacmViewIncluded,
19            StorageType.nonVolatile);
20
21        vacm.addViewTreeFamily(new OctetString("fullReadView"),
22            new OID("1.3"),
23            new OctetString(),
24            VacmMIB.vacmViewIncluded,
25            StorageType.nonVolatile);
26    }

```

Por último e de forma a concluir a execução do agente para esta fase do projeto foi necessário também registar todos os objetos de gestão utilizados, para isso implementámos um método que executa outro fornecido pela classe da MIB, como podemos ver no Exemplo 6.

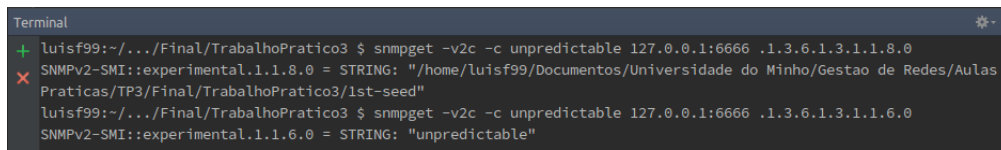
Exemplo 6: Método utilizado para registar os objetos de gestão.

```

1     private void registerManagedObject() {
2         try {
3             uminhogrmib.registerM0s(server, getDefaultContext());
4         } catch (DuplicateRegistrationException ex) {
5             throw new RuntimeException(ex);
6         }
7     }

```

Após esta implementação, elaborámos alguns teste ao sistema desenvolvido verificando e este estava a executar corretamente, como podemos verificar na Figura 3.

A terminal window titled "Terminal" with a dark background. It shows two successful execution of the `snmpget` command. The first command is `snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.8.0`, which returns the string `"/home/luisf99/Documents/Universidade do Minho/Gestao de Redes/Aulas Praticas/TP3/Final/TrabalhoPratico3/1st-seed"`. The second command is `snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.6.0`, which returns the string `"unpredictable"`. Both commands are preceded by a green plus sign icon, indicating success.

```
Terminal
+ luisf99:~/.../Final/TrabalhoPratico3 $ snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.8.0
+ SNMPv2-SMI::experimental.1.1.8.0 = STRING: "/home/luisf99/Documents/Universidade do Minho/Gestao de Redes/Aulas
Praticas/TP3/Final/TrabalhoPratico3/1st-seed"
+ luisf99:~/.../Final/TrabalhoPratico3 $ snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.6.0
+ SNMPv2-SMI::experimental.1.1.6.0 = STRING: "unpredictable"
```

Figura 3: Teste realizados ao sistema na fase B.

Estes testes consistiram na elaboração de vários *snmpget* após o programa do agente ser executado, sendo que resultados foram os esperados comprovando assim a boa implementação e a correta execução do sistema desenvolvido.

4.3 Fase C

A terceira fase foi necessário acrescentar a implementação do grupo *unpredictableTable* da *Unpredictable MIB*. Considerámos que os valores $N=T$, $D=K$ e $p=q=1$ e são constantes. Não foi pedido ainda a implementação da operação de refrescamento da tabela de números aleatórios nem a operação de *reset*, ou seja, a tabela construída no arranque do agente a partir da matriz $M_{t,k}$ do ficheiro de sementes foi mantida fixa e constante.

Para a conceção do que era requerido no enunciado do projeto para esta fase, foi necessário primeiramente aceder e ler o ficheiro que contém as *seeds*, como podemos ver no Exemplo 7, cuja diretoria foi obtida através do ficheiro de configuração previamente lido.

Exemplo 7: Método utilizado para obtenção das *seeds*.

```
1      private void getSeeds() throws IOException {
2          File conf = new File(seedPat);
3          if (conf.exists()) { // if exists it's true
4              BufferedReader fileParameters;
5              String param;
6              fileParameters = new BufferedReader(new FileReader("1st-seed"));
7              while ((param = fileParameters.readLine()) != null) {
8                  seeds.add(param);
9              }
10         } else {
11             System.out.println("Sem ficheiro de configuracao.\nColoque na pasta
12                                 respectiva!");
13             System.out.println("Pasta " + System.getProperty("user.dir"));
14             System.exit(0);
15         }
16     }
```

Com os valores das *seeds* armazenados pelo sistema, iniciámos a implementação da obtenção da matriz de números aleatórios. Como os valores dos parâmetros eram constantes, levando assim a que a matriz resultante fosse igual a $M_{t,k}$, desenvolvemos o método do Exemplo 8, que dos valores das *seeds* armazenados constrói a matriz resultante, no nosso caso de 8 colunas e 256 linhas.

Exemplo 8: Método utilizado para obtenção da matriz de números aleatórios.

```
1     private char[][] getMatrix() {
2         int numSize = Integer.parseInt(numberSize);
3         int tabSize = Integer.parseInt(tableSize);
4         char[][] matriz = new char[tabSize][numSize];
5         for (int i = 0; i < tabSize; i++) {
6             String aux = seeds.get(i);
7             for (int j = 0; j < numSize; j++) {
8                 char[] split = aux.toCharArray();
9                 matriz[i][j] = split[j];
10            }
11        }
12        return matriz;
13    }
```

Após a execução do método e respetiva obtenção da matriz, de seguida foi necessário guardar os valores correspondentes na MIB, mais propriamente na *unpredictableTable*. Para que isto fosse possível implementámos o método do Exemplo 9. Este método cria uma nova linha na tabela da MIB para cada linha da matriz original guardando o seu valor. É feita também a correspondência entre os valores guardados e uma chave na tabela, criando um novo OID para cada linha para que depois estes possam ser obtidos pela comando *snmpget*.

Exemplo 9: Método utilizado para guardar a matriz na tabela da MIB.

```
1     private void setValue(Object value) {
2         DefaultMOMutableTableModel model = (DefaultMOMutableTableModel)
            uminhogrmib.getRandomEntry().getModel();
3         Variable[] v = new Variable[]{getOctetString(value)};
4         model.addRow(model.createRow((new OID(new int[]{i})), v));
5         i++;
6     }
```

Depois da implementação estar concluída o grupo iniciou a fase de testes ao sistema desenvolvido até ao momento. Os testes elaborados consistiram na execução o agente e posterior obtenção dos valores da tabela através do comando *snmpget* como podemos visualizar na Figura 4.

```
Terminal
+ luisf99:~/.../Final/TrabalhoPratico3 $ snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.5.1.1.1
SNMPv2-SMI::experimental.1.1.5.1.1.1 = STRING: "f169b5f1"
✗ luisf99:~/.../Final/TrabalhoPratico3 $ snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.5.1.1.256
SNMPv2-SMI::experimental.1.1.5.1.1.256 = STRING: "3f8f63ab"
luisf99:~/.../Final/TrabalhoPratico3 $ snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.5.1.1.100
SNMPv2-SMI::experimental.1.1.5.1.1.100 = STRING: "70fc730a"
luisf99:~/.../Final/TrabalhoPratico3 $ snmpget -v2c -c unpredictable 127.0.0.1:6666 .1.3.6.1.3.1.1.5.1.1.258
SNMPv2-SMI::experimental.1.1.5.1.1.258 = No Such Instance currently exists at this OID
luisf99:~/.../Final/TrabalhoPratico3 $
```

Figura 4: Teste realizados ao sistema na fase B.

Como se pode verificar na figura e através da consulta dos valores presentes no ficheiro que contém as *seeds*, os valores correspondem comprovando assim que o serviço implementado funciona sem quaisquer problemas.

4.4 Fase E

Esta fase tem como objetivo a implementação da operação de *reset* da matriz $M_{t,k}$ e, por consequência da tabela de números aleatórios. operação de *reset* consiste em reiniciar a matriz $M_{t,k}$ com as sementes do ficheiro indicado no ficheiro de configuração. Esta operação só pode ser executada no agente quando é feita uma operação de *snmpset* à correspondente instância do objeto escalar do tipo string do grupo *unpredictableParam* e só é autorizada se o valor da *string* no pedido *snmpset* for igual chave de configuração. Após a execução da operação *reset*, o agente pausa as suas operações de refrescamento por $R * 10$ segundos e retoma a operação de refrescamento a cada $1/T$ segundos.

Para implementar a operação de *reset* a chave de configuração usada no arranque do agente é guardada na variável *cc* que é usada para comparar com a *string* colocada na instância *unpredicatableReset* quando executado um comando *snmpset* pelo utilizador. Se for igual pausa as operações de refrescamento senão continua, como exemplificado no Exemplo 10.

Exemplo 10: Exemplo Comparação da chave de configuração e pausa das operações de refrescamento

```
1
2      MOChangeListener l = new MOChangeListener() {
3          public void beforePrepareMOChange(MOChangeEvent moChangeEvent) {}
4          public void afterPrepareMOChange(MOChangeEvent moChangeEvent) {}
5          public void beforeMOChange(MOChangeEvent moChangeEvent) {}
6          public void afterMOChange(MOChangeEvent moChangeEvent) {
7              if (uminhogrmib.getUnpredictableReset().getValue().equals(
8                  getOctetString(cc))) {
9                  uminhogrmib.getUnpredictableReset().setValue(getOctetString(""));
10                 };
11                 u = 0;
12                 timers.cancelTimer();
13                 try {
14                     reset();
15                 } catch (IOException e) {
16                     e.printStackTrace();
17                 }
18                 try {
19                     timers.startTimer(delay, interval);
20                 } catch (IOException e) {
21                     e.printStackTrace();
22                 }
23             }
24         }
25     }
```

```

21         }
22     }
23 };

```

Para detetar um *set* à instância *unpredictableReset* foi implementado um *listener* descrito no exemplo 10 que é executado assim que esta instância for alterada. No caso de pausar as operações de refrescamento, implementou-se um *Timer* que só é executada após *delay* segundos e re-executada em *interval* segundos. A classe *RefreshTimer* encontra-se exemplificada no exemplo 11.

Exemplo 11: Exemplo Classe *RefreshTimer*

```

1
2     public class RefreshTimer {
3     private Timer timer;
4
5     public void startTimer(int delay, long interval) throws IOException{
6         this.timer= new Timer();
7         SNMPAgent agent = new SNMPAgent("0.0.0.0/6666");
8
9         timer.schedule(new TimerTask() {
10             @Override
11             public void run() {
12                 agent.refresh();
13             }
14         }, delay,interval);
15     }
16
17     public void cancelTimer(){
18         this.timer.cancel();
19     }
20 }

```

Para demonstrar o seu funcionamento, na função *refresh* uma variável está sempre a incrementar sempre que o método *refresh()* é executado e quando é realizada uma operação de *reset* a matriz $M_{t,k}$ é reiniciada, como podemos observar na Figura 5.

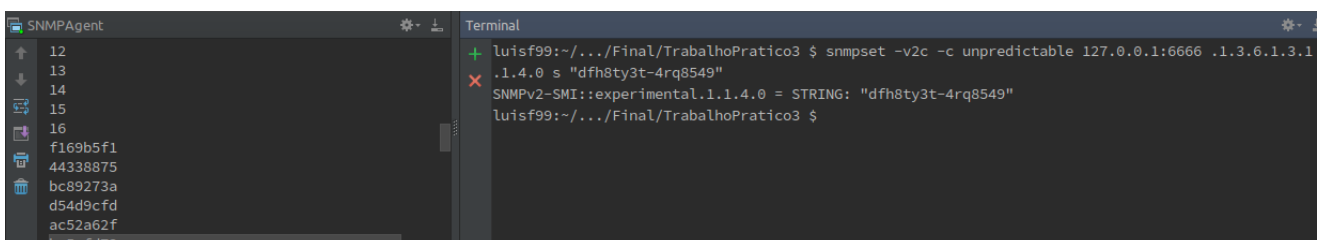


Figura 5: Grupo *unpredictableTable*.

5. Conclusão

Neste relatório é descrita a solução apresentada para resolução do terceiro trabalho prático proposto para a unidade curricular de Gestão de Redes.

De acordo os requisitos estabelecidos inicialmente, este trabalho foi implementado com sucesso, ou seja, tanto MIB (grupos e sintaxe) como o agente SNMP foram concessionados de forma correta e encontram-se a executar sem quaisquer problema. O agente foi construído com o serviço SNMPv2c e quando é executado utiliza os parâmetros do ficheiro de configuração e o ficheiro de sementes nele definido para gerar a matriz $M_{t,k}$. A tabela de números aleatórios é guardada no grupo *unpredictableTable* e os parâmetros de inicialização são guardados no grupo *unpredictableParam* da UMINHOGRMIB. O utilizador também pode ativar a ação de *reset* que reiniciará a matriz $M_{t,k}$ com as sementes do ficheiro indicado no ficheiro de configuração se a *string* utilizada no comando *snmpset* for igual à *string* utilizada como argumento na altura de execução do agente. Um aspeto menos conseguido foi o processo refrescamento da matriz $M_{t,k}$ e, por consequência, da tabela de números aleatórios, uma vez que não foi implementada devido a diversas dúvidas por parte do grupo sobre o que era requerido no enunciado do projeto prático.

Por último, foi um trabalho que abordou vários aspetos do protocolo SNMP que permitiu obter um maior conhecimento do seu funcionamento e a implementação de um agente com base neste neste serviço.