

Gestão de Redes

TP3 - Agente SNMP para monitorização de datas de eventos

**Mestrado Integrado
em
Engenharia Informática**

Nome	Nº
Diogo Rocha	a79751
António Gomes	a67645

27 de fevereiro de 2021

1 Introdução

O seguinte relatório foi produzido na sequência do trabalho prático nº 3 da Unidade Curricular de **Gestão de Redes**. Foi pedido aos alunos pela equipa docente a criação de um agente **SNMP** que implemente uma pequena MIB para monitorização de datas de eventos e um gestor para obter informações da MIB.

2 Ferramentas utilizadas

Para o desenvolvimento do projeto foram utilizadas 3 ferramentas , *mib-designer*, utilizado para o desenvolvimento da **mib**, *agent-pro*, utilizado para criar um agente que implemente a **mib** criada, e sendo o código gerado pelo agente em linguagem *java* , foi utilizada esta mesma no restante desenvolvimento do projeto.

3 Criação da mib e do agente

Após debate em grupo decidimos que a mib necessária para responder aos problemas propostos necessitaria de apenas uma tabela de eventos com as seguintes colunas :

1. id - Identificador de um evento.
2. nome - Nome de um evento
3. duracao - Duração de um evento em minutos
4. deltaT - O intervalo de tempo de um evento em ano-mes-semana-dia-hora-minuto
5. dataLimite - A data limite de um evento antes de ser apagado da mib em ano-mes-semana-dia-hora-minuto
6. passou- Flag que indica se o evento já passou

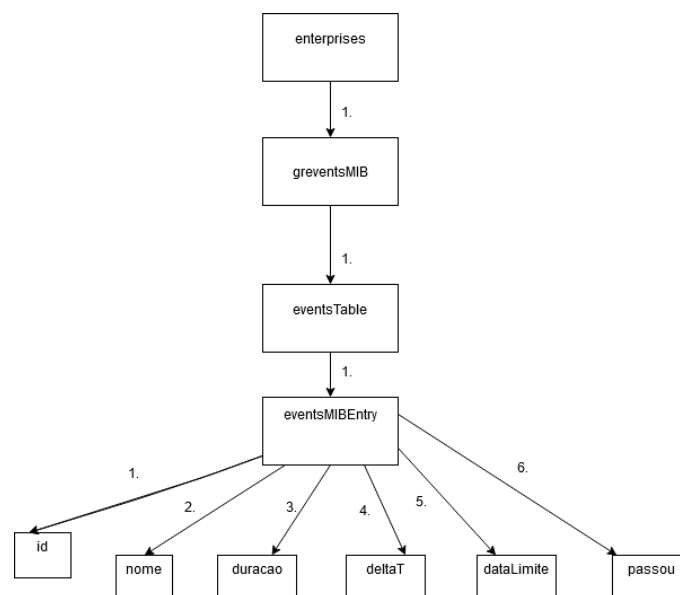


Figura 1: MIB

3.1 MIB-Designer

Após definida a estrutura acima simplesmente seguimos os tutoriais disponibilizados pela equipa docente e gerar a MIB foi relativamente simples, gerando assim um ficheiro txt.

```
GR-EVENTS-MIB DEFINITIONS ::= BEGIN
IMPORTS
    enterprises,
    MODULE-IDENTITY,
    OBJECT-TYPE,
    NOTIFICATION-TYPE
        FROM SNMPv2-SMI
    DisplayString,
    DateAndTime
        FROM SNMPv2-TC
    OBJECT-GROUP,
    NOTIFICATION-GROUP
        FROM SNMPv2-CONF;

greventsMIB MODULE-IDENTITY
    LAST-UPDATED "202102161438Z" -- Feb 16, 2021, 2:38:00 PM
    ORGANIZATION ""
    CONTACT-INFO
        ""
    DESCRIPTION
        ""
    REVISION "202102161438Z" -- Feb 16, 2021, 2:38:00 PM
    DESCRIPTION
        "Initial version."
    -- 1.3.6.1.4.1.1
    ::= { enterprises 1 }

eventsTable OBJECT-TYPE
    SYNTAX SEQUENCE OF EventsMIBEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        ""
    -- 1.3.6.1.4.1.1.1
    ::= { greventsMIB 1 }

eventsMIBEntry OBJECT-TYPE
    SYNTAX EventsMIBEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
        ""
    INDEX {
        id
    }
    -- 1.3.6.1.4.1.1.1.1
    ::= { eventsTable 1 }
```

Figura 2: Parte do ficheiro **txt** gerado

3.2 AgentPro

À semelhança da utilização do *mib-designer*, o *agentPro* também foi fácil de utilizar, a única dificuldade foi superada após uma reunião com o professor. Posto isto foram geradas 3 classes necessárias para implementar o agente : Modules , GREventsMIB e Agent.

```
public class Agent implements VariableProvider {

    static {
        LoggerFactory.setLogFactory(new JavaLogFactory());
    }

    private static final String DEFAULT_CL_PARAMETERS = "-c[s{=Agent.cfg}] -bc[s{=Agent.bc}]";
    private static final String DEFAULT_CL_COMMANDS = "#address[s{=udp:127.0.0.1/3003}<(udp|tcp):.*[/[0-9]+]";

    private LogAdapter logger = LoggerFactory.getLogger(Agent.class);

    protected AgentConfigManager agent;
    protected MDServer server;
    private String configFile;
    private File bootCounterFile;

    // supported MIBs
    protected Modules modules;

    public Agent(Map args) {
        configFile = (String)((List)args.get("c")).get(0);
        bootCounterFile = new File((String)((List)args.get("bc")).get(0));

        server = new DefaultMDServer();
```

Figura 3: Classe Agent

4 Implementação

Existe uma classe *AgentCon* como forma de encapsular as tarefas de população e atualização da MIB de eventos que caem sobre o agente. Dentro desta classe existem dois métodos, um para cada tarefa a realizar sobre a MIB de eventos.

4.1 Método de população da MIB: *insertEvents*

Neste método é usado um documento em formato json, *eventos.json* como base de dados de alimentação da MIB assim como forma de permanência dos dados da mesma. Este documento, num estado inicial, contém informação sobre eventos a serem introduzidos na MIB assim como toda a informação relevante à criação dos mesmos na MIB. O Conteúdo deste documento é processado para um objecto do do tipo *Eventos* criado com o intuito de mapear os eventos contidos no documento para um *ArrayList* com recurso à biblioteca jackson de java que permite o mapeamento direto e simples de objectos java para json e vice-versa.

```
public void loadEventos(){
    ObjectMapper mapper = new ObjectMapper();

    try{
        File json = new File( pathname: "/home/diogo/Desktop/GestãoDeRedes/TP3/GR_TP3/project/eventosDB/eventos.json");
        Eventos e = mapper.readValue(json, Eventos.class);

        eventos = new ArrayList<Evento>(e.getEventos());
    }
    catch (JsonGenerationException ge) {
        ge.printStackTrace();
    }
    catch (JsonMappingException me) {
        me.printStackTrace();
    }
    catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

Figura 4: Load Eventos

```
public void saveEventos(String nome){
    ObjectMapper mapper = new ObjectMapper();

    try{
        File json = new File( pathname: "/home/diogo/Desktop/GestãoDeRedes/TP3/GR_TP3/project/eventosDB/"+nome);
        Eventos e = new Eventos(eventos);
        mapper.defaultPrettyPrintingWriter().writeValue(json, e);
    }
    catch (JsonGenerationException ge) {
        ge.printStackTrace();
    }
    catch (JsonMappingException me) {
        me.printStackTrace();
    }
    catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

Figura 5: Save Eventos

Para cada evento mapeado é necessário verificar a sua existência prévia na MIB com o auxílio do método *containsRow()* que passado o identificador de uma linha verifica se a mesma existe na MIB. caso a mesma não exista esta é inserida com recurso a *addNewRow(oid, vars)*. Os dados do Evento a inserir são convertidos num array de Vars com recurso ao método *vars()* da classe Evento.

```
public Variable[] vars(){
    Variable[] variables = new Variable[6];
    variables[0] = new Integer32(id);
    variables[1] = new OctetString(nome);
    variables[2] = new Integer32(duracao);
    variables[3] = new OctetString(deltaT);
    variables[4] = new OctetString(dataLimite);
    variables[5] = new Integer32(passou);
    return variables;
}
```

Figura 6: Vars

```
public void insertEvents(GrEventsMib mib){ //fazer a população da mib
    Eventos evs = new Eventos();
    evs.loadEventos();

    //Percorre o arraylist de eventos e adiciona um evento de cada vez
    for(Evento e : evs.getEventos()){
        try {
            if(!mib.getEventsMIBEntry().getModel().containsRow(new OID(e.getId().toString()))){
                OID oidIndex = new OID(e.getId().toString());
                mib.getEventsMIBEntry().addNewRow(oidIndex, e.vars());
            }
        } catch (Exception ex){
            ex.printStackTrace();
        }
    }
}
```

Figura 7: Insert Eventos

4.2 Método de atualização da MIB: *updateEvents*

A cada minuto a MIB é atualizada, este processo é possível através do método *updateMIB(GrEventsMib mib)*. A MIB é percorrida como forma de actualizar cada linha da mesma e para facilitar este processo uma classe *Data* permite encapsular todos os procedimentos de actualização da data de um evento. O conteúdo do ficheiro json é carregado, de seguida cada evento contido no mesmo é atualizado segundo os métodos existentes na classe *Data*. A atualização de eventos rege-se pelas seguintes regras:

- Eventos com a flag *passou* a 0 são passado ou presente e o reverso para eventos futuros.
- O campo *DeltaT* representa o tempo que falta para o início do evento caso *passou* esteja a zero e representa o tempo passado após o fim do evento caso *passou* esteja a 1.
- Quando *DeltaT* alcança zero o evento teve início e a sua duração começa a ser decrementada.
- Quando a duração do evento chega a zero, o fim do evento ocorreu e por consequência *passou* torna-se 1 indicando que o evento já é passado e *DeltaT* começa a ser incrementado como forma de representar o intervalo de tempo passado entre o fim do evento e o momento atual e o valor de *DataLimite* passa a ser decrementado contando o tempo restante antes da eliminação do evento.

```
if(passou == 0 && !(dataDt.isZero())){
    dtRes = dataDt.decrementaData(deltaT);
    e.setDeltaT(dtRes);
}
if(passou == 0 && (dataDt.isZero())){
    e.setDuracao(duracao-1);
}
if(duracao <= 0){
    e.setDuracao(duracao);
    e.setPassou(1);
}
if (passou == 1) {
    dtRes = dataDt.incrementaData(deltaT);
    dlRes = dataDl.decrementaData(dataLimite);

    e.setDeltaT(dtRes);
    e.setdataLimite(dlRes);
}
if(dataDl.isZero()){

    listaApagar=new Evento(e);
    apagar=true;
}
```

Figura 8: Atualização de Valores de um Evento

```
if(apagar==true) {
    evs.removeEventos(listaApagar);
    evs.saveEventos( nome: "eventos.json");
    for (Integer j = 1; j <= mib.getEventsMIBEntry().getModel().getRowCount(); j++) {
        mib.getEventsMIBEntry().removeRow(new OID(j.toString()));
    }
    insertEvents(mib);
}
else{
    evs.saveEventos( nome: "eventos.json"); //guardar a table no ficheiro json
}
```

Figura 9: Eliminar eventos na MIB e ficheiro

- por fim quando *DataLimite* alcança zero e *passou* se encontra a 1 o evento é eliminado. Após a actualização dos valores do evento os mesmos são escritos na respectiva linha da MIB:

```
else {  
    //atualizar os valores na linha de cada coluna  
    mib.getEventsMIBEntry().getModel().getRow(  
        new OID (i.toString())).setId(new Integer32(id));  
  
    mib.getEventsMIBEntry().getModel().getRow(  
        new OID (i.toString())).setDuracao(new Integer32(duracaoI));  
  
    mib.getEventsMIBEntry().getModel().getRow(  
        new OID (i.toString())).setDeltaT(new OctetString(dtRes));  
    mib.getEventsMIBEntry().getModel().getRow(  
        new OID (i.toString())).setDataLimite(new OctetString(dlRes));  
    mib.getEventsMIBEntry().getModel().getRow(  
        new OID (i.toString())).setPassou(new Integer32(pI));  
  
    evs.setEvento(new Evento(e),id);  
}
```

Figura 10: Escrita de Valores na MIB e no ficheiro

5 Execução

5.1 Agente

Com o agente em execução as operações de inserção e remoção de eventos na mib podem agora ser realizadas. Em seguida é demonstrada a execução de uma simples aplicação gestora que procura e lista os eventos presentes na mib.

```
INFO: Agent state advanced to 40  
fev 27, 2021 4:31:21 DA TARDE org.snmp4j.log.JavaLogAdapter log  
INFO: Listening on socket 127.0.0.1/3003  
fev 27, 2021 4:31:21 DA TARDE org.snmp4j.log.JavaLogAdapter log  
INFO: Notification 1.3.6.1.6.3.1.1.5.1 reported with [] for context
```

Figura 11: Agente em execução

5.2 Manager

Como referido acima a aplicação gestora dispõe de uma menu de interação com o utilizador, será agora demonstrada a sua execução.

5.2.1 Menu inicial

```
Bem vindo ao Gestor de eventos!  
0 que deseja fazer?  
1 -> Procurar eventos por nome.  
2 -> Listar eventos já terminados.  
3 -> Listar eventos em curso.  
4 -> Listar eventos ainda por vir.  
5 -> Sair;
```

Figura 12: Menu inicial

5.2.2 Procura de um evento pelo nome

```
1  
Qual o nome do evento desejado?  
Fut champions  
Nome do evento a procurar: Fut champions  
Evento desejado : nome: Fut champions | Duracao: 1995 | Delta T: 0-0-0-0-0-0 | Data Limite: 1-1-1-1-0-0 | Passou: 0
```

Figura 13: Informação do evento escolhido

5.2.3 Listar os eventos já terminados

```
2  
Lista de eventos terminados:  
0 Fim do primeiro semestre já acabou há: 0 anos 0 meses 1 semanas 1 dias 1 horas 50 minutos  
0 Início do segundo semestre já acabou há: 0 anos 0 meses 1 semanas 1 dias 1 horas 50 minutos
```

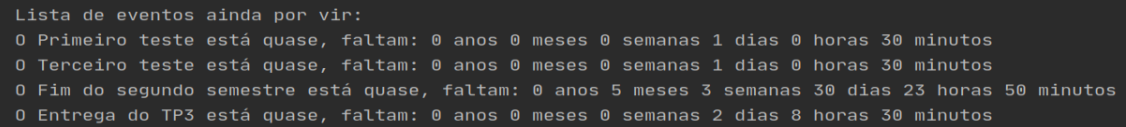
Figura 14: Lista de eventos já terminados

5.2.4 Listar os eventos em curso

```
3  
Lista de eventos em curso:  
0 evento Fut champions está a acontecer, faltam 1995minutos para o fim!
```

Figura 15: Lista de eventos em curso

5.2.5 Listar eventos ainda por vir



```
Lista de eventos ainda por vir:
0 Primeiro teste está quase, faltam: 0 anos 0 meses 0 semanas 1 dias 0 horas 30 minutos
0 Terceiro teste está quase, faltam: 0 anos 0 meses 0 semanas 1 dias 0 horas 30 minutos
0 Fim do segundo semestre está quase, faltam: 0 anos 5 meses 3 semanas 30 dias 23 horas 50 minutos
0 Entrega do TP3 está quase, faltam: 0 anos 0 meses 0 semanas 2 dias 8 horas 30 minutos
```

Figura 16: Lista de eventos ainda por vir

6 Conclusão

Para concluir pensamos que se atingiram os objetivos que foram propostos, toda a dinâmica do grupo, a divisão de tarefas e sucesso de execução das mesmas também deve ser referida dado a forma como foi desenvolvido o projeto. Como nota final gostaríamos de agradecer à equipa docente nomeadamente ao Professor João por toda a disponibilidade e prontidão de auxílio.