LOW-LEVEL PROCESS MODELLING IN THE OPNET MODELER SIMULATION ENVIRONMENT

Karol Molnár, Petr Kovář

Department of Telecommunications, Faculty of Electronics and Communications, Brno University of Technology, Purkynova 118., 612 00 Brno, Czech Republic,

phone: +420 54114 9190, e-mail: molnar@feec.vutbr.cz, kovarpetr@phd.feec.vutbr.cz

Recently nearly all methods and algorithms designed to be used in telecommunication equipment have been evaluated and analysed in detail using simulation environments. These environments greatly differ in the level of abstraction they can offer. This paper describes an implementation of a low-level state—machine in the OPNET Modeler simulation environment. The state-machine presented simulates the basic behaviour of the Simple Network Management Protocol (SNMP), which is not implemented in the recent version of OPNET Modeler. The paper focuses on the interaction between standard OPNET Modeler modules and customized components, which is represented by a communication process using the Interface Control Information application programming interface.

Keywords: OPNET Modeler, Process Editor, state – machine

1. INTRODUCTION

Simulation environments represent an important tool used to test, analyse and evaluate methods and algorithms designed to be used in telecommunication environments. Since nowadays there is an extensive research and development effort in the field of telecommunication technologies there are also several simulation environments designed to help in this work. These tools very often differ in the level of abstraction the can offer to the user. The paper focuses on the lowest level of simulation environment represented by state-machines and corresponding program codes. In this context the implementation of a simple communication protocol will be presented. This communication protocol will be based on the Simple Network Management Protocol (SNMP), which is not directly implemented in the OPNET Modeler environment at the time of writing this paper.

The aim of our long-time effort is to design and implement a QoS control mechanism between the end-station and the network and to use the SNMP protocol for data exchange. In the first step we need to evaluate this control method in the OPNET Modeler simulation environment. OPNET Modeler is quite flexible and user-friendly for a relatively fast and convenient implementation of our method but it does not support the SNMP protocol in its current release. For this reason first a simplified model of the SNMP protocol was implemented. This implementation process and the results achieved are described in the following chapters.

2. OPNET MODELER

OPNET Modeler [1] is a discrete event simulation environment for designing protocols and technologies intended for use in telecommunication networks. The sophisticated simulation models also allow extensive testing and demonstrating designs in realistic scenarios. It features a user-friendly graphical user environment for fast and clear design of a simulation scenario. In the background the object-oriented modelling allows us to quickly and easily develop different versions and mutations of a simulation component or scenario. In addition, there is an open interface which allows external object files, libraries, and other components to be integrated into an OPNET simulation scenario.

There are several simulation levels available in OPNET Modeler. The basic level represents computer network elements (workstations, servers, switches, routers, firewalls, etc.) and communication links between them. In the lower layer each element is decomposed into modules. A special editor, called Node Editor, is used to work on this level. The modules very often represent a communication protocol or a part of a protocol stack. The default node model for a workstation is shown in **Fig. 1**.

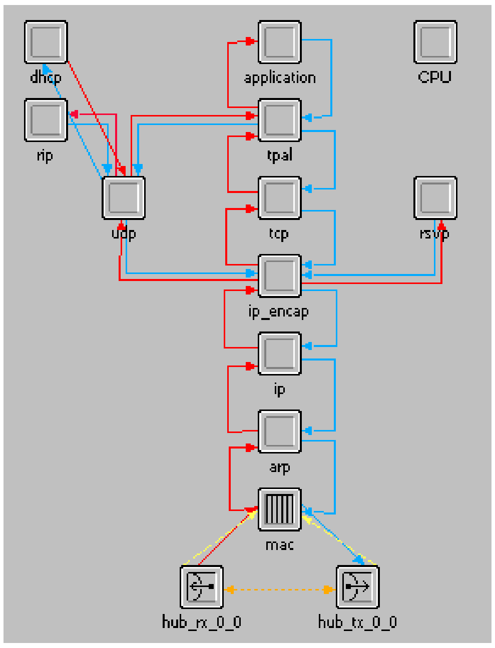


Fig. 1 Workstation node model

The most detailed level of abstraction is represented by a program code realised as a finite state-machine. A special tool, called Process editor, is available to simplify the implementation and evaluation of finite state-machines. In each state a list of executives are performed, which are defined as a C-like program code. These statements model actions executed by the simulation environment. There are two types of executives: the enter and the exit executive. The enter executive is preformed as soon as the finite state-machine enters to the given state. The exit executive is performed when the process leaves the current state for another one.

There are also two types of states available: forced and unforced. A forced state (green circle) provides the executives, both enter and exit executives, and then passes the control to another state. In the case of an unforced state (red circle) the enter executive is performed when the control is passed to the given state. Next, the system waits in this state for an event, which passes the control to another one. Before changing the state the exit executive is performed.

3. SIMPLIFIED SNMP MODEL

OPNET Modeler contains enough predefined models to build large, very sophisticated and complex simulation scenarios. There is also an extensive set of parameters for these models implemented directly in the system. On the other hand, there are some special situations when a new protocol that is not available in the system must be implemented. In our case a simple, request-response type communication protocol was required, which can be used for one-way data transmission from the access router to the workstations. We need this protocol to transmit selected configuration parameters from the access router to the workstation. Since in real systems the required information is usually stored in a system database, called Management Information Base (MIB), a standardized communication protocol has been selected which is directly designed to exchange such type of information. This protocol is called Simple Network Management Protocol.

The communication between the router and the workstation requires only a relatively small set of features from a full SNMP implementation. For this reason we decided to use a simplified model of this protocol. Our model does not support traps and uses only the GetRequest and the GetResponse messages.

In the first stage the MIB is modelled by a static data structure initialized when the simulation starts. This database consists of rows, each containing an object ID, object type and object value. An example of such a structure is shown in **Tab. 1**.

Tab. 1 Example of an MIB

row	Objec ID	Type	Value
0	1.3.6.1.2.1.2.2.2.1.1.1	integer	7
1	1.3.6.1.2.1.2.2.2.1.2.4	string	ATM0/0/3
2	1.3.6.1.2.1.2.2.2.1.5.1	gauge	15625000
3	1.3.6.1.2.1.2.2.2.1.6.1	string	00:D0:91:C9:4D:EA
4	1.3.6.1.2.1.2.2.2.1.9.1	time	674
5	1.3.6.1.2.1.2.2.2.1.10.1	counter	25175
6	1.3.6.1.2.1.2.2.2.1.22.1	oid	0.07

The state-machine of the implemented agent and a simplified manager are shown in Fig. 2. This figure also shows the process of packet exchange required by our system. The agent represents an edge router which stores the required information in its MIB. Workstations act as an SNMP manager, which uses the GetRequest message to ask the agent to send the required data. The manager answers by sending GetResponse messages.

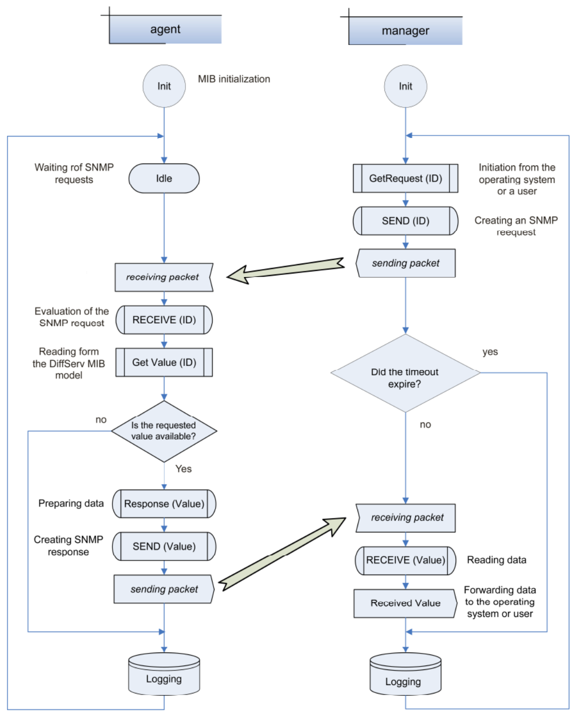
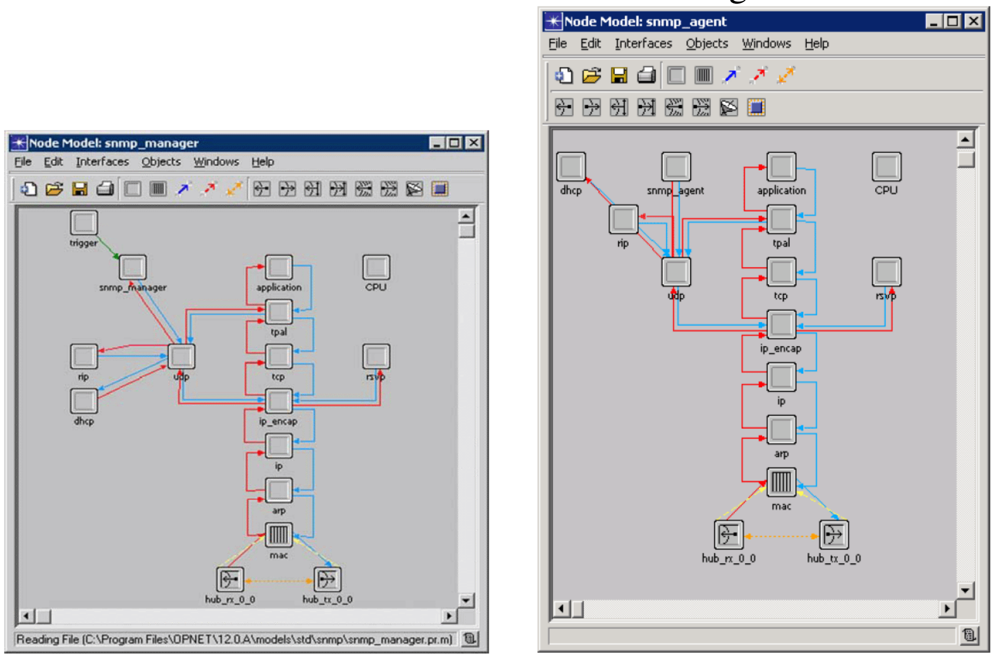


Fig. 2. The state-machine representing the manager, the agent and their interaction

Next, the finite state-machine from Fig. 2 has to be implemented in the OPNET Modeler simulation environment. For this reason the standard architecture of network components must be extended by additional modules. This extension can be done in the Node Editor. The modified architectures are shown in Fig. 3.



Reading File (C:\Program Files\OPNET\12.0.A\models\std\snmp\snmp_manager.pr.m)

Fig. 3 SNMP manager and SNMP agent node models

The workstation (manager) is extended by modules called *snmp_manager* and *trigger*. The node model of the router is extended by the module *snmp_agent*. Both the *snmp_manager* and the *snmp_agent* modules are connected to the *udp* module. To control periodic polling at the workstation the *trigger* module generates interruptions in constant intervals. The *snmp_manager* and the *snmp_agent* modules provide the required functionalities, which are represented by a finite state-machine configured in the Process Editor. The finite state-machines for the manager and for the agent are shown in Fig. 4.

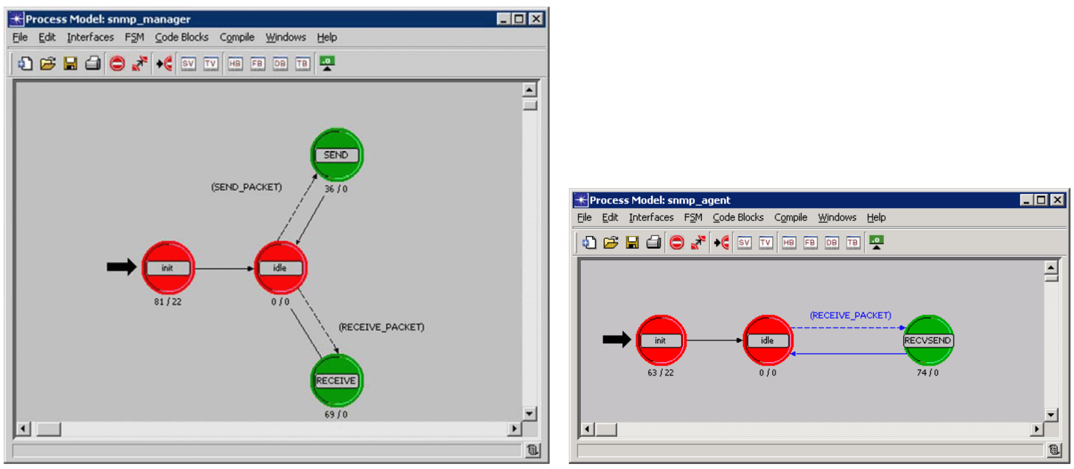

Process Model: snmp_manager
File Edit Interfaces FSM Code Blocks Compile Windows Help
SEND
(SEND_PACKET)
36 / 0
init
81 / 22
idle
0 / 0
(RECEIVE_PACKET)
RECEIVE
69 / 0
Process Model: snmp_agent
File Edit Interfaces FSM Code Blocks Compile Windows Help
(RECEIVE_PACKET)
init
63 / 22
idle
0 / 0
RECVSEND
74 / 0

Fig. 4 The SNMP manager and SNMP agent finite state-machines

As shown in Fig. 4, both modules are first initialized and then they wait for a triggering event. The *snmp_manager* processes two of such events. The first is an interruption from the *trigger* module, which evokes processes related to sending a request. The arrival of the response represents the second event. The *snmp_agent* module reacts only to one event, which is represented by the arrival of a request.

The implementation of the MIB look-up functions is quite straightforward and it is not necessary to describe it in detail. The main implementation steps are evident from the state-machine shown in Fig. 2. Substantially more interesting is the data transmission between modules and consequently between network components.

We choose the Interface Control Information (ICI) application programming interface to exchange packets between network components. Since the source code of the simulation model is quite long, only the most important states the *SEND* and the *RECEIVE* states are described in detail. These two states realize the backbone of the communication process.

The *SEND* state for the *snmp_manager* contains the following steps (the description of each step is included in the source code):

```
// Creating a new packet
send_paket = op_pk_create_fmt ("snmp_paket");
// Allocating memory for the new message
sendID = ( SPkt *) op_prg_mem_alloc ( sizeof (SPkt) );
// Determining the required OID
strcpy (sendID->text, MIB_manager[change][0]->text);
// Filling the packet with data
```

```
op_pk_nfd_set (send_paket, "ID", sendID);
// New ICI
ici_ptr = op_ici_create ("udp_command_v3");
// Configuring ICI
op_ici_attr_set (ici_ptr, "local_port", loc_port); // Manager port
op_ici_attr_set (ici_ptr, "rem_addr", rem_addr); // Agent IP adresa
op_ici_attr_set (ici_ptr, "rem_port", rem_port); // Agent port
// Installing ICI
op_ici_install (ici_ptr);
// Sending the packet to the UDP module
op_pk_send (send_paket, UDPSTRM);
```

The *RECEIVE* state for the *snmp_manager* realizes complementary operations:

```
// Obtaining the ICI pointer
ici_ptr = op_intrpt_ici();
// Reading the packet
recv_paket = op_pk_get (op_intrpt_strm () );
recvID = (SPkt *) op_prg_mem_alloc (sizeof (SPkt));
recvDATA = (SPkt *) op_prg_mem_alloc (sizeof (SPkt));
// Obtaining data from the packet
op_pk_nfd_get (recv_paket, "ID", &recvID);
op_pk_nfd_get (recv_paket, "Value", &recvDATA);
// Forwarding data to the MIB Manager
strcpy (MIB_manager[zmena][2]->text, recvDATA->text);
// Discarding packet
op_pk_destroy (recv_paket);
```

The communication process in the *agent* is very similar to the previous procedures. The difference is only in how the corresponding data are obtained and further processed. More details about the implemented code can be found in [3].

4. CONCLUSION

OPNET Modeler is a powerful tool used to model, simulate and evaluate communication network scenarios and their components. It contains a large set of predefined simulation elements, but it also makes it possible to build user-defined network elements with proprietary functions. This paper presents a simple example how to implement a customized function and how to link it to the standard components of this simulation environment.

5. REFERENCES

[1] OPNET Technologies, Inc., OPNET Modeler Release 12 Product documentation, 2007.
[2] Case, J., Fedor, M., Schoffstall, M., Davin, A. A Simple Network Management Protocol (SNMP), RFC-1157 [on-line], May 1990. http://rfc.net/rfc1157.html.
[3] Bednarik, J. Modelování komunikace proprietárním protokolem, urceným pro výmenu informací o podporované technologii QoS, v prostredí OPNET Modeler, Brno, Brno University of Technology, 2007.

Acknowledgement

This paper has been supported by the Grant Agency of the Czech Republic (Grants No. 102/06/1569 and 102/05/P585) and the Ministry of Education of the Czech Republic (project No. MSM0021630513).