

# Integration of Real Network Components into OPNET Modeler Co-simulation Process

MILAN BARTL, JIRI HOSEK, TOMAS MATOCHA, KAROL MOLNAR, LUKAS RUCKA

Department of Telecommunications

Faculty of Electrical Engineering and Communication, Brno University of Technology

Purkynova 118, 612 00 Brno

CZECH REPUBLIC

xbartl02@stud.feec.vutbr.cz, hosek@feec.vutbr.cz, xmatoc00@stud.feec.vutbr.cz, molnar@feec.vutbr.cz,  
rucka.lukas@phd.feec.vutbr.cz

*Abstract:* - The OPNET Modeler is a simulation environment enables modeling of large-scale networks with in detail defined parameters. This paper is focused on the possibility to communicate with external systems and applications that this environment offers. Due to this we are able to interconnect simulation environment with real network system and gain the simulation results more accurate and in accordance with the reality. We created communication model composed from network scenario created in the OPNET Modeler, C/C++ external application and Cisco router. For this purpose we implemented the SNMP protocol into the OPNET Modeler. This model enables data exchange between external network component and work station in OPNET Modeler scenario. This model is very useful especially in the situation when you need to include a real network device into simulation process, because the pure simulation environment is not quite sufficient.

*Key-Words:* BER, Esys interface, MIB, OPNET Modeler, SNMP.

## 1 Introduction

The complexity of today's networks necessitates the need for network management to optimize network performance. There are several methods to manage network equipment. The first possibility is to manage network devices locally. This method requires direct access to network devices and is time demanding. This type of management is very complicated, especially if the network consists of many devices or the network is a geographically vast. Generally, local management is not able to ensure optimal network performance and fault detection.

Remote network management represents another solution. This method does not require direct access to the network devices because it uses a remote application for the network management. The prerequisite for this type of management is that this functionality must be implemented in each network device to be managed.

An important part of the network management is network monitoring. Network monitoring allows surveying of the current state and behaviour of the network equipment. Due to network monitoring it is possible to respond to different events in the network more quickly and solve non-standard behaviour of the network devices.

The development of a monitoring and management system for network equipment is quite a complex task. Often different simulation tools can be used to improve the development process by modelling specific events

rarely appearing in real networks. The simulation environment allows the developers to evaluate several alternative solutions before their implementation into real systems [1]. Recently there are many simulation tools available to cover different needs from the field of network and data communications. The essential difference between these tools lies in the complexity and degree of abstraction [2].

One of the leaders between simulation environments specialized for complex modelling and simulation of communications networks, devices and protocols is the OPNET Modeler (OM) simulation tool [3]. This program is able to create and simulate behaviour of any network architecture. The friendly Graphic User Interface (GUI) and many specific editors (e.g. network editor, node editor, process editor etc.) are the main advantages of this application. The GUI is showing real layout of network elements and is able to generate various statistics too. In this way, for example, can be simulate any network states and then prevent from unwanted events or mistakes before the network is implemented to the real environment.

The OM is an open source application which means that the users can modify source codes of every network nodes and create their own functions. The source codes are written in programming language C. This feature extends the flexibility of this simulation environment [3]. The results of simulations can be saved into the many file formats, for example into the XML (eXtensible Markup Language) or HTML (Hypertext

Markup Language), or save in form of tables. The OM can make backward load input data from this file formats. The OPNET Modeler contains animation viewer and the OPNET Debugger tool. By the help of these components user can keep track of simulation process in detail and detect possible mistakes. The simulation is running with the acceleration which depends on the simulation model complexity. Therefore it can be simulate the month behaviour of network in order of several hours [3].

All components in the OM are modeled in the object oriented approach which provides intuitive mapping to real systems [4]. Particular components in the OM are described by C/C++ source codes and are user accessible. It allows the users to modify the source code and to add new custom functionalities if required. A unique feature of the the OM is the possibility of an interaction between the OM and external systems. This possibility is ensured by a complex set of functions called External System Domain/Definition (ESD) [3], [5]. The external system might represent almost anything from a general algorithm to a specific model of a hardware entity (e.g. user designed hardware, real network devices). It gives a flexible platform to test new ideas and solutions with low cost and brings more accurate results to the entire research process.

## 2 Implementation of SNMP Protocol into OPNET Modeler

### 2.1 SNMP Protocol

The Simple Network Management Protocol (SNMP) is one of the most often used solutions for remote network management. SNMP was introduced in 1988 [6] to meet the growing need for a standard for network devices management. SNMP was developed as a temporary tool and was intended to be replaced by a solution based on the Common Management Information Service/Protocol (CMIS/CMIP) architecture. Today SNMP is still the most popular method of network management because it can be easy implemented and disposes with great interoperability [7].

SNMP is a communication platform between the SNMP agent and the SNMP manager. The agent is located at the managed device and makes accessible the configuration information and statistics of this devices. The SNMP manager is placed on the device that manages the network nodes. The manager sends requests, encapsulated into the SNMP operations, to the agent. As a response the agent usually sends back the requested data. In the case of critical events the agents can inform the manager without any previous polling

using a special message called trap [7]. The communication model of SNMP is shown in Fig. 1.



Fig. 1. SNMP communication model

### 2.1.1 SNMP Operations

There are three version of the SNMP protocol – SNMPv1, SNMPv2 and SNMPv3. Currently the most used version is the SNMPv2. For gathering and changing management information in network devices following operations can be used [7]:

- Get,
- Get-Next,
- Get-Bulk,
- Set,
- Get-Response,
- Trap,
- Notification,
- Inform,
- Report.

Each of these operations has a standardized Protocol Data Unit (PDU) format that is used by managers and agents to send and received information. The structure and detailed description of above mentioned SNMP operations can be found in [7].

### 2.2 Management Information Base

The Management Information Base (MIB) is a structured collection of configuration and measurement information of a given network device. The MIB entries are organized into a treelike hierarchy that enables to divide the MIB database into several independent branches. The existing branches can be extended by other branches or objects corresponding to the supported functions and requirements of network component manufacturers [7], [8].

#### 2.2.1 MIB Management

The statistics and configuration values stored in the MIB are accessible through Object Identifiers (OIDs) [8]. The value corresponding to OID represents the current state of the object. The above mentioned SNMP operations Get, Get-Next, Get-Response are used to gather and modify management information in the MIB of the SNMP agent.

When the manager sends an SNMP request to the agent it has to know the OID of the MIB's entry that wants to discover/modify. In the case when the manager

needs just one entry from the agent it uses the Get-Request SNMP operation. After the agent receives this SNMP request, the value bound to the corresponding OID is obtained from the MIB. Then the retrieved information is encapsulated into the Get-Response SNMP message and sent back to the manager.

Another way to obtain the required information from the MIB is the Get-Next SNMP operation that enables to retrieve a block of values from the MIB. The Get-Next message contains only the starting OID from which the SNMP agent starts to look-up the MIB. When the manager receives the Get-Next Response from the agent it generates another Get-Next Request command repeatedly until the agent returns an error, signaling that the end of the MIB has been reached and there are no more objects left to retrieve [7].

The above described SNMP operations are used for reading from the MIB database. When the manager needs to write a specific value into the MIB it has to use the Set SNMP operation. This operation contains the specific OID and the corresponding value that should be entered.

### 2.3 SNMP Model in OPNET Modeler

The majority of application protocols available in the OM are represented by a parametric traffic generator producing a mathematically described pattern of network traffic. It means that the communication on the application level is simulated by dummy data units with application specific traffic parameters [9]. On top of that, the SNMP protocol is not implicitly implemented in the current version of the OM, but the OM enables the functional modelling of custom network technologies and protocols. We used this feature to implement the SNMP communication model into the OM in the C programming language.

As a result of this work a simulation model with agent – manager architecture had been created where the components communicated with each other using SNMPv2 messages. The SNMP agent and manager were modelled by finite-state machines realized by C/C++ functions. These functions support receiving and sending of SNMP messages. The process model of the SNMP manager created in the OM is shown in Fig. 2.

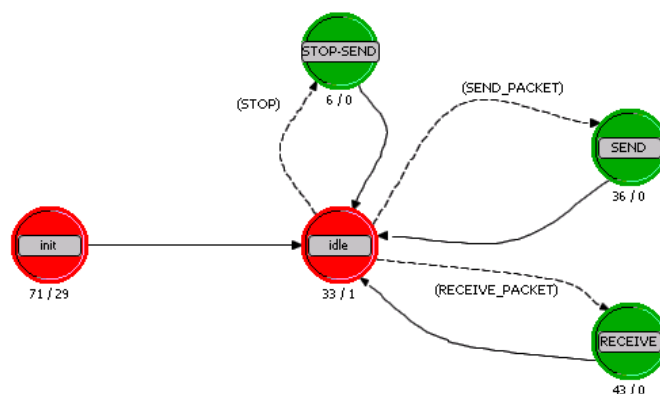


Fig. 2. Process model of SNMP manager

According to [7] the SNMPv2 message has the following standardized structure:

```

/* Variable Bindings*/
typedef struct variablebin{
    char *ObjectID;
    char *Value;
} VARIABLEBIN;

/* SNMP PDU */
typedef struct snmppdu{
    int PDUtype;
    int RequestId;
    int ErrorStatus;
    int ErrorIndex;
    VARIABLEBIN *VariableBin;
} SNMPPDU;

/* SNMP Packet */
typedef struct snmppaket{
    int Version;
    char *Community;
    SNMPPDU *SNMPPdu;
} SNMPPACKET;

```

Each SNMP message is created dynamically. First, the SNMP operation type (PDUtype) is defined and then the message is filled up with the OID (ObjectID) or specific MIB value (Value) according to the operation type.

In order to transmit the SNMP message in the way described in RFC 1067 [6], it was necessary to encode and decode every message using Basic Encoding Rules (BER). BER is one of the encoding formats defined as the part of ASN.1 (Abstract Syntax Notation One) and specified by the ITU (International Telecommunication Union) in X.690 recommendation [10]. A single instance of an SNMP message has to be encoded into a string of octets. BER defines how the objects are encoded and decoded so that they can be transmitted over a transport medium such as Ethernet [11].

Since BER is not available in current version of the OM we have to implement it in C language. The C functions programmed perform encoding and decoding

of every field of SNMP message as a data type identifier, the length description in bytes and the actual data. By this way each SNMP message is encoded into an octet sequence. Subsequently, this sequence is transmitted over the communication link to the destination node where it is decoded back into the original SNMP message.

The implementation of the SNMP protocol and BER encoding was the first step to develop a functional communication system between real network components and the OM which is described in next chapter.

### 3 Communication with Real Network Component

#### 3.1 Possibilities of Communication between OPNET Modeler and Real Systems

There are two possibilities how to interconnect the OM with real network equipment. The first possibility is the already mentioned ESD system. The second is called System In The Loop (SITL). Naturally, both of them have their advantages and disadvantages.

SITL is a separately distributed library for the OM which provides an interface to link real network hardware or software applications to the OM discrete event simulation. External devices are connected to the simulation loop over SITL gateways operated as a bridge interface between the simulation environment and the network interface of the host computer. Packets transmitted between the simulated and real networks are converted between real and simulation formats. The SITL module is mainly focused on real-time communication with devices based on Ethernet technology while the use of the ESD system is much more versatile [3].

As mentioned before, the ESD system allows an interconnection between the OM and external system. This interconnection is called co-simulation [3]. The ESD system consists of several components. These components are Simulation description, External System Definitions model, External System Interface, co-simulation code and code implemented in external system or application [12].

The external system is represented in the OPNET Modeler as a model whose behavior is determined by an external code. Such model can be represented by anything from microchip till user application. The OM passes and receives data from the external system without having any implicit knowledge of the method of processing these data [12].

The basic advantage of this feature is that there is no need to intricately define the new simulation model of

the existing system we want to include into the simulation. There is only need to modify existing system to implement the esys interface by the library functions. These functions ensure the data exchange through the esys interface. The whole architecture is based on the control exchanging between the OPNET Modeler and external system. The basic principle of co-simulation in the OPNET Modeler is shown in Fig. 3 [12].

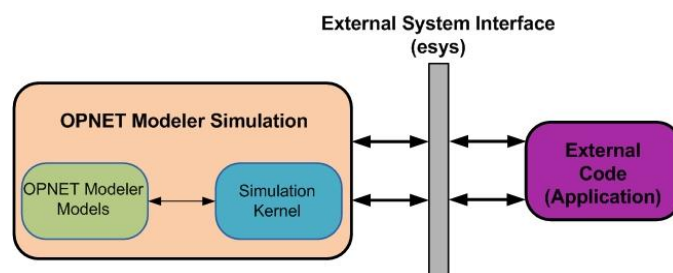


Fig. 3. Basic co-simulation scheme in OPNET Modeler environment

The co-simulation is available only in the sequential version of the OPNET Modeler kernel where only main thread of process should invoke the OPNET APIs (Application Programming Interface).

The co-simulation has two basic requirements. There is a need to implement code of the esys interface on the OPNET Modeler side and also in the external system. The second requirement relates to the computing memory, because more complicated simulations are more memory-intensive.

The co-simulation is established by using a special interface of the OM called External System Interface (esys). The esys interface is represented in the OM by a process module. Thus it can be implemented anywhere in the model structure. The esys interface ensures the control and data exchange between the OM and external systems [3]. The whole ESD system is more complex and more universal than SITL but on the other hand it does not provide such a high speed of the communication and simulation. It is because the control of simulation is shared between the OM and an external system which can lead to slower event processing.

#### 3.2 Co-simulation Approaches and Components

There are two basic co-simulation approaches in the OM. The difference of these approaches lies in the determination which co-simulation side will be the controlling unit [13]. The first option is that the OM is part of the larger program. In this case the OPNET Modeler models code is linked into the external system. The second approach option, which is shown in Fig. 4, is to dynamically link the external system code in the form of library into the co-simulation in the OM. In this case,

the OPNET Modeler is controlling element. We use the second option in our project so the paper is focused on this co-simulation approach.

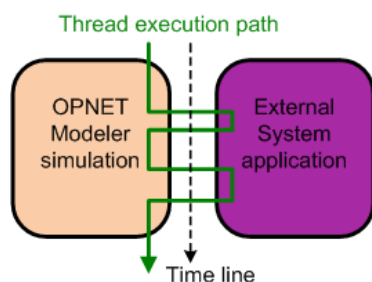


Fig. 4. OPNET Modeler as a control application

The whole co-simulation process is based on three following components that have to be properly defined.

### 3.2.1 Simulation Description

The simulation description is a text file with the “.sd” suffix that defines the list of object files, libraries and other attributes that are required by the OM during co-simulation. Thanks to this description the OPNET Modeler is able to find and invoke files and objects of the external system. Furthermore the OM finds attributes such as compilation option, object file names and the type of operating system. The simulation description consists of one or more blocks of statements set off by the identifiers *start\_definition* and *end\_definition*. The simulation description file is not used if the user defines own makefiles [14].

### 3.2.2 External System Definitions (ESD)

The ESD specifies the number and attributes of external system interfaces. Through the defined esys interfaces the external system can communicate with the co-simulation code implemented in the OM. There is the External System Editor in the OPNET Modeler that gives a way to build and edit the ESD model. The first ESD attribute that has to be defined is the esys interface name. It can be used to refer to the interface during co-simulation. The next attribute is a data type which defines the type of data that are transferred over the esys interface. One esys interface can handle only one selected data type, so it is very important to define right data type for both direction of co-simulation. Most of data type available in the OPNET Modeler responds to the C/C++ data types [3].

The next ESD attribute specifies the direction in which information flows through the esys interface. There are three possible directions – from OM to external system, from external system to OM and bidirectional. The data value passed over the esys interface is valid until the old data are overwritten with new ones. The last ESD attribute is dimension which

specifies the number of elements for the esys interface. The default dimension value of 0 identifies a simple esys interface. A value of 1 or greater indicates that the interface is a vector interface with the specified number of elements. The ESD with the simulation description make up the ESD module [3].

### 3.2.3 External System Interface

The esys interface is the only physical component in the OM which represents the communication instrument with the external system. The esys interface is a process module and thanks to this it can be implemented into a model structure of any network device along other processes where it is exactly needed. The properties of the esys interface process module are defined by the associated ESD module. Only esys interfaces specified in this way can be used for communication with the external system. Inside of the esys process module process model (algorithm) is created. The main task of this algorithm is to interact with the external system. The structure and setting of this process model (inner logic) reflect the way how the OM executes the information exchange with the external system. A variety of kernel procedures let you control data transmission as well as data transformation between the OPNET Modeler and external system [3].

Although the data transformations needed depend on specific circumstances, it will be often needed to transform objects (such as packets) to a form usable in the external system's domain. Conversely, it will be needed to take values received from the external system and convert them back into objects that Modeler can use. There are general mechanisms that can assist you with these conversions. For example, application of value vectors that are essentially arrays with a variable number of elements, directly to the esys interfaces.

## 3.3 OPNET Modeler Functions for esys Interface Support

The co-simulation code binds the OM kernel and external system code together. In order to this binding the OM contains kernel procedures that enable data reading and writing on the esys interface. These procedures allow the transfer of one particular value or the array of values. The OPNET Modeler kernel procedures are closely described in the OPNET Modeler External System package documentation [3].

There is the External Simulation Access (ESA) API that provides a support for the esys interface on the external system side. The ESA API is a set of C/C++ functions that are defined in the header file “esa.h”. This file is part of the OM system library. The ESA API functions perform the co-simulation initialization, application flow controlling between the OM and

external system and the data exchange through the esys interface.

The co-simulation initialization can be described by the following steps. After the co-simulation start-up the basic OPNET Modeler kernel services are initialized. The OM simulation subsystem is loaded consequently. During this operation, the processing preferences are initialized and then the simulation environment settings are specified. After that the network model with associated components are loaded and the co-simulation time is set to "0" [3].

In order to proper operation of all co-simulation process the external code has to enable the OPNET Modeler to perform the simulation events starting or stopping.

The data insertion on the esys interface is executed by the OM kernel procedures. In order to know that the external system wrote the data on the esys interface the OPNET Modeler uses the principle of event interruptions to announce this event. When the data are placed on the interface the data entry interruption is invoked and delivered to the esys process model (see Fig. 5). The kernel functions enable the process model to get the identifier of the interface with written data. This identifier is saved in the esys interface interruption. If the process model knows the interface identifier, it can read the data from the interface. The data reading is realized also by the OM kernel procedures. After all, data are processed by the internal logic of the esys model [3].

In the external system the *callback* function is used to manipulate with the external interface. The callback function is part of the ESA API function set. The external system does not have access to the OM kernel procedures that is why it can use only defined ESA API functions. After the data are placed on the esys interface the callback function is invoked in the external system (see Fig. 5). The callback function is responsible for the data processing sent from the OM. Therefore the callback function must be properly implemented in the external system. The decision which function of the external system becomes the callback function is defined by the ESA API function called *Esa\_Interface\_Callback\_Register()* [3], [12], [13].

There are two ways of data writing on the esys interface. The delayed data writing on the esys interface performed by specific ESA API function is the first way. This ESA function notifies the OM about new data placing with a delay. The delay duration is one of parameters of this function. The data writing with immediate announcement is the second way.

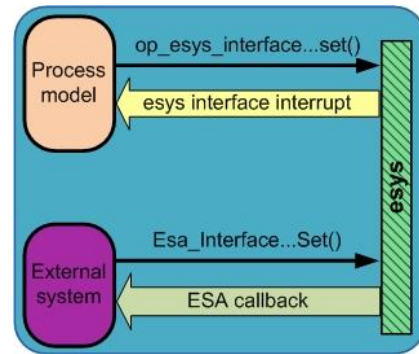


Fig. 5. Invoking and transfer of data entry interruption and callback function

### 3.4 Configuration of Node Model snmp\_manager\_esys

Into the node model of the SNMP manager we implemented a new process model esys. This model enables communication with the environment outside the OM and it is connected to User Datagram Protocol (UDP) process model, because it needs to work with IP address and port. the modified topology of the workstation node model is shown in Fig. 6.

Attributes of this process model have one special item called the ESD Model. This model is accessible through button labelled Edit ESD Model in the Attributes window (see Fig. 7).

The first part contains one row with the text box to input name of a special data file called Simulator description. This file will be described further.

The second part defines the esys interface/s connected to this esys process model. These interfaces can transmit a data from the simulation to the external application or in opposite direction.

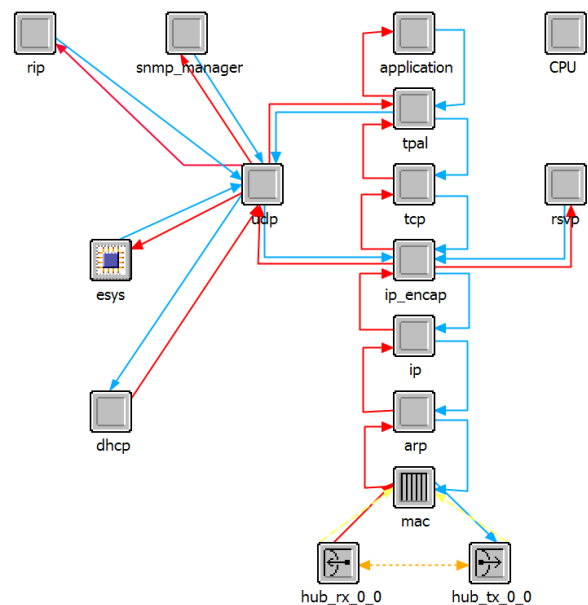
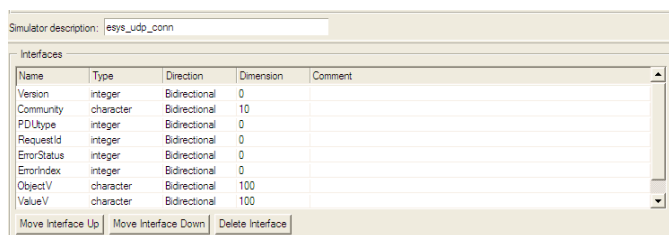


Fig. 6. Modified topology of SNMP manager node model

The field *Name* specifies the name through which the interface is accessible. The following field specifies a data type of the interface. These types can be common C/C++ data types (integer, character, pointer etc.) or special types of string or bit (for special Value Vector data types in the OM).

In the field *Direction* is set the direction of the transmission, from the OM to external code, external code to the OM or bidirectional interface.

The last important field is named *Dimension*. It specifies the dimension of the array for storing the values. If set on zero, only one value can be set on this interface. Setting another value on this interface would overwrite the previous value.



Name	Type	Direction	Dimension	Comment
Version	integer	Bidirectional	0	
Community	character	Bidirectional	10	
PDUtype	integer	Bidirectional	0	
RequestId	integer	Bidirectional	0	
ErrorStatus	integer	Bidirectional	0	
ErrorIndex	integer	Bidirectional	0	
ObjectV	character	Bidirectional	100	
ValueV	character	Bidirectional	100	

Fig. 7. Configuration of ESD model

### 3.5 Configuration of Process Model `snmp_manager_esys`

Starting with the header block of the `snmp_manager_esys` process model, we need to define conditions of the state transitions, included header files and structure for packet type of SNMP. This process model contains four states (see Fig. 8).

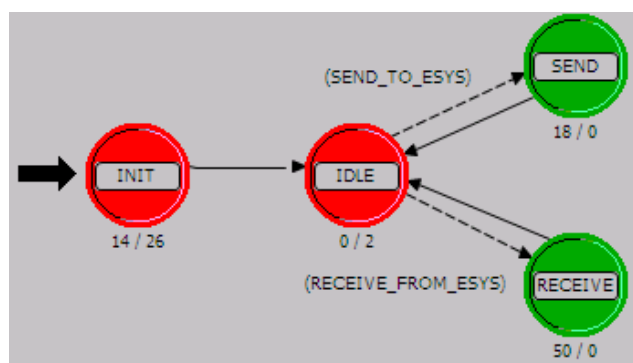


Fig. 8. Process model `snmp_man_esys`

#### 3.5.1 INIT State

Its function is to obtain pointers on objects surrounding the process (like UDP process model) and on the process itself. Then it creates an interruption for itself to register the UDP port. It registers the port number 162 and in the end is allocated a memory for error messages and for the IP address string.

#### 3.5.2 IDLE State

After initialization comes the process to this state. It remains there until some interruption comes from outside. It has got no code but in the Exit executives, where is obtained pointer on the ICI interface that produced the interruption. ICI interfaces are commonly used to store additional information for the interruption.

#### 3.5.3 SEND State

Whenever are received data coming from the UDP process model (i.e. data to send out of the simulation), the process transits into this state. Together with the data to send is obtained the IP address of the target SNMP agent. Then, the child process is invoked and the data are written on the esys interfaces. Necessity of using child process will be explained in following chapter.

#### 3.5.4 RECEIVE State

The reverse state to SEND is RECEIVE state. It reads a data from esys interfaces and stores it in the pre-allocated structure. Then is the structure wrapped into the packet and the packet is send to the UDP process model.

### 3.6 Child Process

The system of co-simulation is following: whenever are a data written on the esys interface with condition `OPC_ESYS_NOTIFY_IMMEDIATELY`, an immediate interruption is invoked in the external code (on the other side of esys interface). When this happens, the process on the OM side that invoked the interruption has not returned from the sending state, because the simulation in the OM is paused [3]. However, we need the main esys process to be unblocked because of manipulation with incoming data from the external code.

For this purpose we can create a new process that will send the data in behalf of the main process. At first, in the parent process model must be declared, which process/es will be available to become "child". This declaration is accessible in the menu item File – Declare Child Process Models, where is a list of all process models saved in the model directories (add/remove model directory in the menu item File - Manage Model Files).

The child process contains only two states, START and EXEC. Immediately after invocation, the process transits to EXEC state that contains the whole code.

At first we need to obtain a structure with the parent data (SNMP data and IP address). Then we write these data on the esys interface and with the last value written on the interface we create an interruption for the external code. In the end is de-allocated the memory with the SNMP data. The last step is performed after the co-simulation returns from the external code.

### 3.7 Configuration of Simulation Descriptor

This file contains a information for the co-simulation builder and linker. Structure of this file is strictly defined. All definitions must be wrapped in the block starting with a `start_definition` and ending with a `end_definition`.

```
start_definition
platform:      windows
use_esa_main:  yes
kernel:       development
bitness:      32bit
dll_lib:      esys_udp_conn.dll
end_definition
```

We run the co-simulation under 32-bit Microsoft Windows operating system. We use the OM Debugger Console for debugging the simulation, so we need a development kernel.

In our case, the OM side should be “in charge”. Setting `use_esa_main` on yes means that we use the external dynamic loaded library (DLL file) in the OM simulation. The co-simulation is started from the OM GUI like common simulations.

With `use_esa_main` set on yes, we need to define a name of the DLL file with the external code. This name is a parameter of item `dll_lib`.

### 3.8 Definition of External Application

In order to implement a communication between the OM and a real network node we created an external application that is used as middleware between the OM model and the network interface of the local workstation. The application reads data from the OM, translates them into SNMP requests and sends them to a network node. Then it waits for the SNMP response from an agent implemented in the destination node. When SNMP response arrives, it is translated back into a data structure compatible with the OM and it is entered into the simulation model.

We used the functions of the OM API for sending/reading data to/from the OM. These functions are declared in the `esa.h` header file, which also contains the functions to initialize the co-simulation and register callback functions.

The external application developed is compiled as a dynamically loaded library. All the functions that should be available to the co-simulation coordinator have to be exported with code extern "C" DLLEXPORT. There are two functions of this type `esa_main` and `callback`.

The `esa_main` function completes the basic initialization procedures of the co-simulation and includes the Winsock library initialization (for communication through the network interface) and

registration of the callback function to the ESYS interface.

The *callback* function is called, whenever data are written on this interface [13]. The data are read from the interface, encapsulated to the SNMP request and sent to the SNMP agent. After obtaining the response, the corresponding data is converted and forwarded to the simulation model through the ESYS interface.

For sending the SNMP request there is an internal function (without an export) called *snmp*. This function realises the SNMP communication using the functions and structures from the *snmp.h* and *Mgmtapi.h* windows header files [15]. It can create an SNMP Get-Request and Get-Next-Request to obtain values from the MIB database of the SNMP agent. When the SNMP structure is filled with data, the request can be sent via the *SnmpMgrRequest* function. If this call succeeds, the *SnmpVarBind* structure is filled with the value of the requested object by the SNMP agent.

The data returned by real equipment is stored in a data type, which is incompatible with the OM, so a conversion is necessary [16]. With respect to the type of the returned value it is converted and saved in a special structure. This structure contains a constant used to specify the value type and the ID of the returned object (saved as string). The conversion of complex data types to standard string is executed by functions *readAsnOid*, *readAsnAddress* and *readAsnString*.

After processing the data obtained, the corresponding structures are deallocated by calling the *SnmpUtilMemFree* function. Finally, the *SnmpMgrClose* function destroys the SNMP manager instance and the pointer on the structure for the converted values.

## 4 Simulation Scenario

In real network conditions the SNMP manager creates the SNMP message and sends it toward to the SNMP agent. By default, the SNMP message is delivered through the network to the SNMP agent using the UDP transport protocol.

Our evaluation test-bed consists of a real network represented by a Cisco router C1841 and a simulation model in the OM environment. Because of this combination of real and simulated systems, this solution has more complex architecture than the majority of real infrastructures, but at the same time, it offers several interesting research opportunities. The whole evaluation test-bed is composed from the SNMP manager created in OM, esys interface, external application and the SNMP agent implemented in a real router with SNMP support. The architecture of the test-bed is shown in Fig. 9.

The communication process of our simulation scenario is described in following text. The SNMP manager creates the SNMP message in the OM and



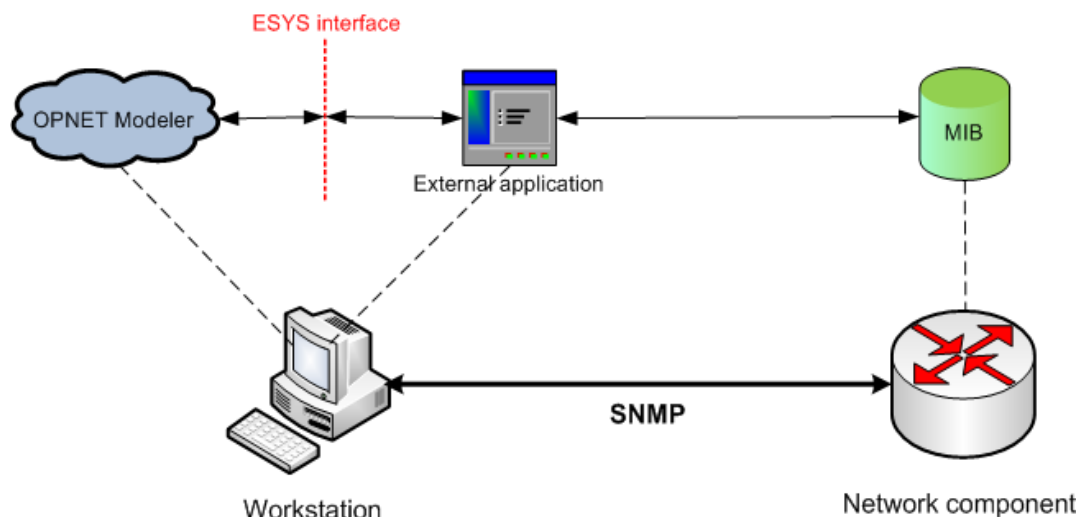


Fig. 9. Architecture of the evaluation test-bed

sends it to the esys interface. Through the esys interface the SNMP message is passed to the external application running on local workstation. During the transmission through the esys interface data conversion must be provided. This conversion is necessary because the OM and the external application use incompatible data types. The external application places the converted data into an SNMP message and sends it to the SNMP agent in UDP datagrams. The SNMP agent is located in a hardware device. After processing, the SNMP agent sends the answer in as an SNMP message back to the external application. The external application extracts data from the response and forwards them to the esys interface. In the esys interface, data is converted and sent into OM. In the OM, the data received is processed by the model of the SNMP manager.

## 5 Conclusion

We have created a communication system able mutually exchange information between real network components and the the OM simulation environment. This system uses SNMP messages to communicate with each other. It can create an realistic SNMP message inside the simulation environment of the OM, encode it using the BER algorithm and transmit to a real network device through the esys interface. The destination network device can search for and read the required value from the MIB database and send it back to the OM. To achieve this functionality was necessary to modify the source code of external applications and the model created in the OM. The modification is based on the implementation and configuration of esys interface, which is part of the ESD system. The possibility of interconnection of simulations running in the OM with a real environment is the result of modifications.

The interconnection of real and simulating environments opens the way towards complex simulation scenarios that can be use e. g. within the development of complex communication systems for network managing or quality of service assurance.

### Acknowledgement:

This paper has been supported by the Grant Agency of the Czech Republic (Grant No. GA102/09/1130) and the Ministry of Education of the Czech Republic (Project No. MSM0021630513).

### References:

- [1] Koutny, M., Mlynek, P., Krajsa, O., Modelling of PLC Communication for Supply Networks. *Proceedings of the 13th WSEAS International Conference on Communications*, 2009, pp. 185-189.
- [2] Bojkovic, Z., Bakmaz, B., Bakmaz, M., Multimedia Traffic in New Generation Networks: Requirements, Control and Modeling. *Proceedings of the 13th WSEAS International Conference on Communications*, 2009, pp. 124-130.
- [3] Opnet Technologies, *OPNET Modeler Product Documentation Release 15.0*, 2009.
- [4] Skorpil, V., Novak, D., Network Elements Controlled by Artificial Neural Network. *Proceedings of the 14th WSEAS International Conference on Communications*, 2010, pp. 145-148.
- [5] Hosek, J., Rucka, L., Molnar, K., DiffServ Extension Allowing User Applications to Effect QoS Control. *Proceedings of the 13th WSEAS International Conference on Communications*, 2009, pp. 39-43.

- [6] Case, J., Fedor, M., Davin, J., *Simple Network Management Protocol*, RFC1067, 1988.
- [7] Mauro, D., Schmidt, K., *Essentials SNMP, Second Edition*, O'Reilly Media, 2005.
- [8] Barker F., Chan, K., Smith, A., *Management Information Base for the Differentiated Services Architecture*, RFC3289, 2002.
- [9] Wessing, Y., Berger, Y., Berger, M., Simulation Based Analysis on Dynamic Resource Provisioning in Optical Networks Using GMPLS Technologies. *WSEAS Transaction on Communications*, Vol. 5, No. 1, 2006, pp. 185-189.
- [10] *ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, ITU-T Standard X.690, 2002.
- [11] Larmouth, J., *ASN.1 Complete*, Academic Press, 1990.
- [12] Hosek, J., Rucka, L., Molnar, K., Mutual Cooperation of external application and OPNET Modeler simulation environment. *Proceedings of the International Workshop RTT 2009 Research in Telecommunication Technology*, 2009, pp. 1-5.
- [13] Opnet Technologies, Using Modeler's Co-simulation Capabilities to Integrate with External Systems, *Proceedings of the OPNETWORK 2007*, 2007.
- [14] K. Fujita, *Extending Opnet Modeler with Client Profiles for Selecting Data Sources in WAN*, project, Department of Information Science and Telecommunications, School of Information Science, University of Pittsburgh, Pittsburgh, 2003.
- [15] Microsoft Corporation (2009). Simple Network Management Protocol. *Microsoft Development Library* [Online]. Available: <http://msdn.microsoft.com/en-us/library/aa377993>
- [16] Lucio, G. F., Ferrera, P. M., Jammeh, E., Fleury, M., Reed, J. M., OPNET Modeler and NS-2: comparing the accuracy of network simulators for packet-level analysis using a network testbed. *WSEAS Transaction on Computers*, Vol. 2, No. 3, 2003, pp. 700-707.