

# Mutual Cooperation of External Application and OPNET Modeler Simulation Environment

K. Molnár, J. Hošek, L. Růčka

*Faculty of Electrical Engineering and Communication, Brno University of Technology*

**Abstract**—the OPNET Modeler is a simulation environment enables modeling of large-scale networks with in detail defined parameters. This paper is focused on one of the important functions of this software tool which is the ability to interconnect it with a real external system. Thanks to predefined modules and interfaces it is possible to carry out the data exchange between the OPNET Modeler environment and an external system. The client-server model created in the OPNET Modeler is defined in this paper. On the server side the external application was linked over the special interface. This application performed the user data generation and their forwarding into the OPNET Modeler. In the simulation environment received data were put together and sent to the client side where they were displayed and checked.

**Index Terms**— Co-simulation, ESD, esys, OPNET Modeler.

## I. INTRODUCTION

THE simulations and simulation tools, in any field of human activity, represent one of the possible ways how to find a solution of certain problem. In simulation the proposed solution and achieved results can be tested and analyzed before their realization in the real environment. On the other hand it is necessary to point out that the accuracy of achieved simulation results depends on the reality of the simulation model. It means that the model should describe the reality as precisely as possible, but in case of the excessive abstraction the results can be significantly different.

Currently there are many simulation tools in the network and data communication field. They differ from each other by the complexity, abstraction degree, technical support and the licence type under which they are provided. However, most of these simulation tools have one common feature - they use the simulations based on the discrete events when the events are placed into the event list and then processed in corresponding sequence.

The NCTUns, the community project OMNeT++, the Network Simulator developing on the University of Southern California or the OPNET Modeler by OPNET Technologies, Inc. belong among the most widely known simulation tools.

## II. OPNET MODELER

The last mentioned, OPNET Modeler (OM), is a part of

software package OPNET (Optimum Network Performance). The OM was introduced in 1987 as the first commercial simulator created in the MIT (Massachusetts Institute of Technology). Due to the long development the OPNET Modeler comes into the state of a big complexity. It currently enables to model the wide area networks with precisely defined attributes. Due to this feature very accurate results can be achieved [1].

The OPNET Modeler is a developing environment suitable for design, simulation and analysis of data networks. Due to the large number of already created Ethernet, IP (Internet Protocol), TCP (Transmission Control Protocol) and ATM (Asynchronous Transfer Mode) device models and due to possibility of their modification the OM enables true simulation and flexible analysis of the behaviour of all parts and components of the data network. The OM is hierarchical and object oriented. The graphic interface mirrors the real location of network components. The first level behaviour of the particular components is described by the source code in programming language C/C++. The network devices source codes are accessible which means that the users can modify the code and add their own functions if required. This feature extends the utilization of the OPNET Modeler [1].

The complexities of the simulation statistics definition are the important feature of this simulation environment. The charts or reports can be generated from the resulting statistics. The results can be also exported in form of a table. The mere simulation is accelerated and its length depends on the created network scenario complexity, number of collected statistics and on the computer performance as well.

## III. CO-SIMULATION IN OPNET MODELER ENVIRONMENT

The possibility of an interconnection between the OPNET Modeler and external system is the great advantage of this simulation environment. This interconnection is called co-simulation. In order to proper function of the co-simulation the special interface called External System Interface (esys) is defined in the OM. This interface ensures the control and data exchange between both systems. The esys interface is part of the complex system ESD (External System Domain / Definition) [2].

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 [2].

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. 1 [2].

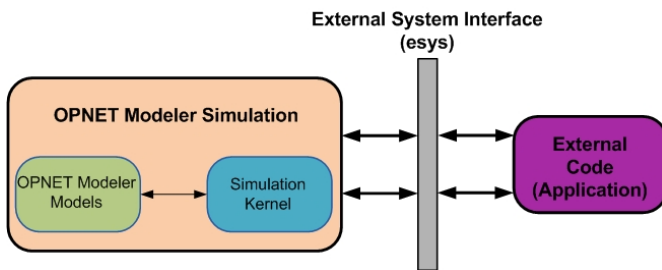


Fig. 1. Basic co-simulation scheme in OPNET Modeler environment

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.

#### A. Basic Co-simulation Components

The co-simulation process is composed from the following components: Simulation description, External System Definitions (ESD) model, External System (esys) interface, co-simulation code (ESA API) and external system code.

#### Co-simulation Approaches

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 [3]. The first option is that the OM is part of the larger program. In this case the OPNET models code is linked into the external system. The second approach option, which is shown in Fig. 2, 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.

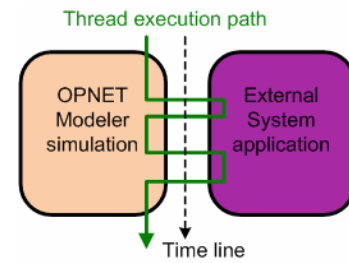


Fig. 2. OPNET Modeler as a control application

#### 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 [4].

#### 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 [1].

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 [1].

#### 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 [1].

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, which are essentially arrays with a variable number of elements, directly to the esys interfaces.

### B. OM 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 [1].

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" [1].

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.

### C. Data Exchange through esys Interface

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. 3). 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 [1].

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. 3). 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()* [1], [3].

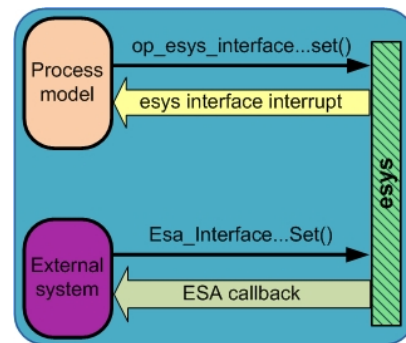


Fig. 3. Invoking and transfer of data entry interruption and callback function

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.

## IV. REALIZATION OF DATA EXCHANGE BETWEEN EXTERNAL APPLICATION AND OPNET MODELER

The client-server communication model was created in the simulation environment OPNET Modeler. The external application generating user data is connected to the server. The data are transferred through the esys interface to the server in the OM. On the server side the data are built into the UDP (User Datagram Protocol) datagram and sent to the client. The client processes the datagram received and displays user data. By the comparison of data received by the client with the data sent by the server the whole co-simulation

process can be verify. The architecture of the system designed is shown in Fig. 4.

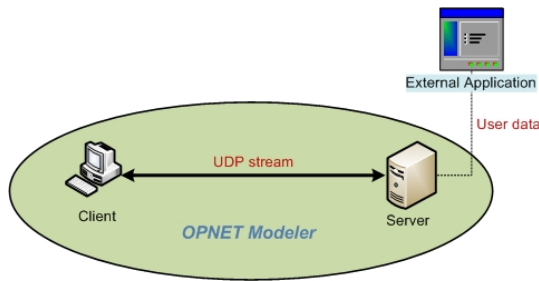


Fig. 4. Designed system architecture

The following programming steps had to be done to ensure the data exchange between the external application and the OPNET Modeler [1]:

- extend the server model by the required interfaces,
- define the ESD model,
- create simulation description file,
- implement ESA API functions.

A. Extension of Server Model

It was necessary to extend the internal server model by two additional modules. The first one, *udp\_interface*, represents the UDP interface. The second one, *esys* module with name *external\_interface*, was connected to the *udp\_interface* module. The *external\_interface* acts as the interface for the data interchange between the external application and the OM. The states and processes ensuring the commands, interruptions and data exchange were defined in the *external\_interface* module. The extended server model is shown in Fig. 5.

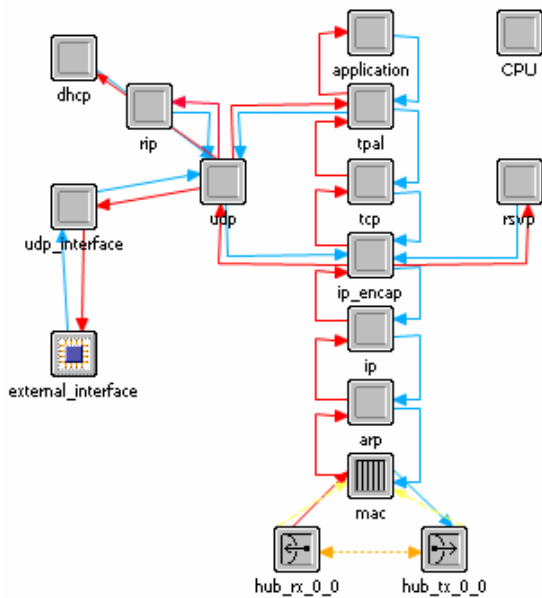


Fig. 5. Extended server model

The *udp\_interface* module is connected with the *udp* module that represents standard UDP layer in the server node. The *udp\_interface* module receives data from the external interface and modifies it to the form suitable for a transmission through the OPNET Modeler network model. Then the data are sent to the *udp* module that ensures the UDP units transmission [5].

B. External System Definition Model

New ESD model was created and linked to the *external\_interface* module. The ESD model shown in Fig 6 describes the attributes of the *esys* interface.

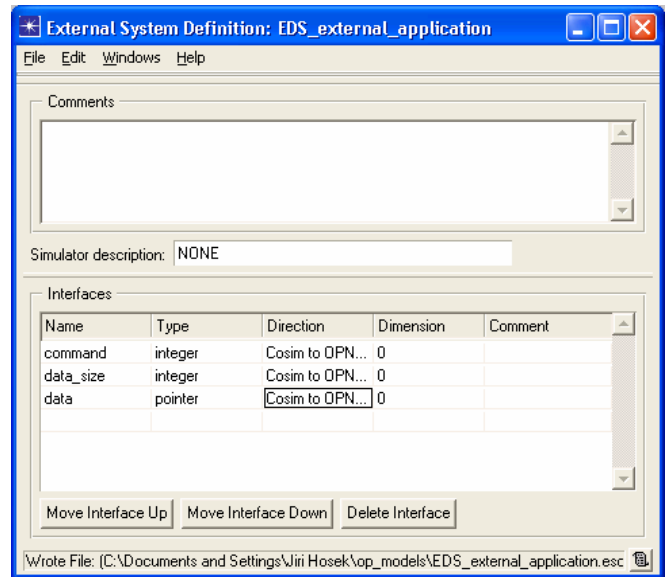


Fig. 6. ESD model

Three interfaces were created within the ESD model. Each interface has defined co-simulation direction from the external application to the OPNET Modeler. The interface *command* implements the command handover between the external application and OM. Next interface *data\_size* specifies the size of transferred data. The data type of both interfaces is integer. The last interface *data* with the data type pointer refers directly to the data value.

C. Simulation Description

The next step of the co-simulation setting was to create the simulation description file. The co-simulation system was designed in such way where the OPNET Modeler acts as the control element. Therefore the OM invokes the external application starting that was linked to the OM as a dynamic library. The name of the dynamic library file was defined in the created simulation description that is shown in Fig. 7.

As shown in the figure, the co-simulation was running under the operating system Microsoft Windows. The “yes” option in the *use\_esa\_main* attribute determines that the external application is linked into the OM as the dynamically

loaded library. In the case of the “no” option the OPNET Modeler would be embedded in larger simulation system [3].

```
# Simulation Description file for Cosimulation

start_definition
  platform:      windows
  use_esa_main:  yes
  kernel:       development
  bind_lib:     ext_data.dll
end_definition
```

Fig. 7. Simulation description file

The OM simulation kernel was set to a development mode that enables the using of the console for the co-simulation controlling and debugging. The *bind\_lib* attribute is the most important one because it specifies the file name of the external application dynamic library linked into the OPNET Modeler. The “.dll” file path must be specified in the OM preferences.

#### D. Implementation of ESA API Functions

It was necessary to implement the ESA API functions into the external application code and the OM code as well [3]. The control function *main()* was implemented into the OPNET Modeler because the OM was determined as the control element of the co-simulation process. Within this function the OM invokes the pre-initializing routine *Esa\_Main()* that supports the basic initialization of primary OM services. In consequence, the function *esa\_main()* implemented in the external application code is invoked. The *esa\_main()* contains two initializing functions *Esa\_Init()* and *Esa\_Load()*.

The function *Esa\_Init()* initializes the ESA library. It sets the OM environment and loads module files according to the arguments list. This function must be called before the other ESA API functions are used. After the ESA library initialization the simulated network components with the associated files (models, objects) are set by the function *Esa\_Load()*.

When the initialization process is finished the co-simulation environment is ready for performing of the user defined events – in our case the data exchange between the OM and external application. The data were placed on the esys interface by the function *Esa\_Interface\_Value\_Set()* [3]. The defined co-simulation execution path is shown in Fig. 8.

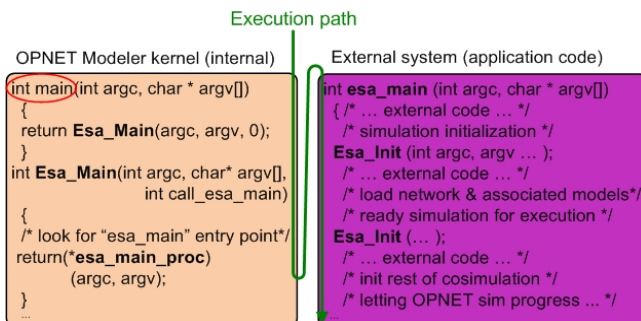


Fig. 8. Co-simulation execution path

## V. CONCLUSION

The development of modern technologies is very fast and the simulation programs are very powerful tools especially for the testing of new technologies before their integration into the real environment. The OPNET Modeler is one of the most known simulation environment that enables creating of the complex scenarios and modelling of various network architectures and protocols. Very important for the simulation of new technologies is the connection with the real existing systems. That is why the simulation software should be able to work with the real data. The OPNET Modeler enables the mutual cooperation with the external systems such as the user applications or hardware components.

In the OPNET Modeler the ESD system is defined. This system is composed from the specialized interface esys and several ESA functions. This whole system assures the interconnection of the simulation model with the real environment. The various data and information can be transferred through the esys interface by both directions. The co-simulation results are affected by the real conditions and they have higher significance for the following research.

## ACKNOWLEDGMENT

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] Opnet Technologies, *Opnet Modeler Product Documentation Release 14.5*, Opnet Technologies Inc., 2008.
- [2] J. Hošek, K. Molnár, L. Růčka, „DiffServ Extension Allowing User Applications to Effect QoS Control“ in *Proceedings of the 13th WSEAS International Conference on Communications*, Rhodes, 2009, pp. 1–5.
- [3] Opnet Technologies, *Interfacing Multiple Simulators Using OPNET Co-Simulation Technologies*, OPNETWORK 2007, Opnet Technologies Inc., 2007.
- [4] 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.
- [5] M. Bartl, *Aplikace zpracující reálný síťový provoz v prostředí OPNET Modeler*, VUT v Brně, Brno, 2009.