SDL and SystemC co-modelling: the protocol SDL models Tester

Alexander Stepanov, Irina Lavrovskaya, Valentin Olenev
St. Petersburg State University of Aerospace Instrumentation
190000, 67, Bolshaya Morskaya, St. Petersburg, Russia
Alexander.Stepanov@guap.ru, Irina.Lavrovskaya@guap.ru, Valentin.Olenev@guap.ru

Abstract

Testing of data transfer protocol can be performed by means of simulation and a variety of modeling languages may be used for it. Among them there are such languages as SDL and SystemC. Moreover an approach for SDL/SystemC co-modeling has been developed and already applied in practice. This paper discusses applicability of SDL/SystemC co-modeling for testing of data transfer protocols.

Index Terms: Co-modeling, SystemC, SDL, testing of SDL models.

I. INTRODUCTION

Most networks are organized as a stack of layers where each layer is built upon the one below. It is done to reduce design complexity. The purpose of each layer is to offer certain services for the upper layers, shielding those layers from the details of how the offered services are actually implemented.

Layer *X* on one machine carries on a conversation with appropriate layer *X* on another machine. The rules and conventions used in this conversation are collectively known as the layer *X* protocol. Basically, a protocol is an agreement between the communicating parties on how communication is to proceed [1].

Development of a new protocol specification usually requires a high-level language model. Benefits of this model consist in getting a state machine of a layer operated by the protocol and checking of the protocol correctness which can be achieved by simulation of the implemented model. During this step the model is inserted into a test environment which is responsible for sending/receiving test data and monitoring results of the simulation. Data generated by the tested model and a sequence of the model states which were reached during the test are required results.

To achieve the purposes pointed out above two distinct modeling languages can be applied. The first one is SDL which uses the concept of finite state machine for specification and description of modeled systems [2]. And the second one is SystemC which is based on C++ language and provides a number of features to implement the flexible and beneficial test environment [3]. Each of these two languages has a set of advantages and disadvantages and disadvantages of one language could be compensated by another's advantages in case of combined use. An SDL/SystemC comparative analysis and a brief description of SDL/SystemC co-modeling approach are given in the next chapters.

The main topic of this paper is applicability of pointed the SDL/SystemC co-modeling approach for data transfer protocol testing. In more details, the paper covers the following subtopics: the comparison of SDL and SystemC languages, the overview of the SDL/SystemC co-modeling, an implementation of the protocol SDL models Tester on the basis of the SDL/SystemC co-modeling and a description of the Tester modules.

Current paper is consecutive continuation of our previous papers in the field of SDL/SystemC co-modeling. The first paper "SystemC and SDL Co-Modelling Methods" concerns to the

foundation for discussed co-modeling, gives a number of methods for its organization, compares them and chooses the best one. Then, in the second paper “SystemC and SDL Co-Modelling Implementation” we make a more detailed analysis of the best method and discuss several ways for its implementation. In this paper we consider a practical applicability of the SDL/SystemC co-modeling on the basis of discussed implementations.

II. Comparison of SDL and SystemC from testing purposes viewpoint

Purposes of protocol specifications testing require a set of subtasks which should be solved by the chosen modeling language. So the choice of the using language depends on how a language fits these requirements. Let’s define the subtasks and consider the SDL and SystemC languages in accordance with them.

A. Data delivery model

Each data transmission protocol is either reliable or unreliable [1]. In first case the reliability of the data delivery has to be checked during testing step of the specification design flow. On the other hand, if the tested protocol is unreliable, estimation of the error rate can be required. So in both cases the following scheme of model testing can be applied: tested data is sent to one node, received from another and then they are compared to find any inconsistencies.

In case of the tested protocol provides an error recovery mechanism it is necessary to have a special error generator in the data transmission channel between the communicating nodes.

SDL language is primarily intended for specification and, consequently, does not contain enough data generation features. And SystemC based on C++ language provides all possibilities, which C++ can offer. Therefore, for checking of the data delivery model it is more preferable to use SystemC.

B. Behavioural model

Behaviour of the tested layer can be represented in terms of finite state machine. This approach provides an ability to monitor system’s states during operation. So for this task SDL is much more applicable modeling language in relation to its functionality based on the finite state machine approach. And SystemC is not an appropriate language in this case.

C. Modeling time

There are two different ways for modeling time managing. The first one is used when the purpose of testing is to check a layer operation. It does not require any knowledge of time and, moreover, the best solution is when the modeling time does not change during the whole test. The second way is used when there is a necessity to check default values of defined layer’s timeouts. Implementation of this task can be performed by the following method: definite delays should be inserted into particular points and other parts of the model have to operate without any delays.

SDL provides a possibility for model operation without delays and also contains a special timing mechanism. But, unfortunately, the SDL modeling kernel has poor abilities for modeling time management. As for SystemC it offers a flexible approach for time control but usually SystemC models does not work without delays. So the most preferable way is to combine both these languages in the following way: SDL is responsible for modeling time usage and SystemC is responsible for the modeling time management.

D. Comparison conclusions

To sum up, one of the best solutions for protocol model testing is:

- SystemC is used for implementation of test environment (data traffic generator, analyzer and channel between nodes) and SystemC modeling kernel is taken for simulation management;
- the whole protocol model (including description of how the model behaves from the time point of view) ought to be developed in SDL.

III. Overview of SDL/SystemC co-modeling approach

The approach for SDL/SystemC co-modeling concerned in this paper assumes to have a SystemC project, which corresponds to the whole considered model. The whole model contains the SDL and SystemC parts. Consequently, this approach uses C/C++ representation of the SDL system [4].

Before starting a description of the discussed approach, we need to introduce general principles of co-modeling with some requirements and important notions for modeling.

Consider some abstract SDL tool. This SDL tool should meet the following requirements:

- provide a possibility to generate C/C++ code from the implemented model, which will be the equivalent of the SDL;
- the generated C/C++ code operation should be controlled by some kind of a manager engine (*SDL_kernel*);
- the *SDL_kernel* should provide a number of functions for initialization and simulation of the SDL model. For the further discussion it is necessary to introduce declarations for two main functions: *SDL_Init()*, which is responsible for initialization, and *SDL_Simulate()*, which is responsible for simulation of the SDL system, so that one SDL transition is executed during each call of this function. One SDL transition is a system state change from one to another.

It should be pointed out that such kind of the SDL tool already exists. For example, all these features are provided by the IBM Rational SDL Suite [5].

The SDL and SystemC parts connection can be divided into the following stages:

- preparation of the SDL system as a part of the whole model.
- generation of C/C++ code on basis of the created SDL system. The SDL code is not used hereafter.
- insertion of this C/C++ code to the *SDL_kernel*.
- preparation of the SystemC part of the model.
- integration of the *SDL_kernel* with the generated C code into the whole model.

According to this approach, the SystemC model is a master and the SDL model is a slave. So SystemC provides the mechanisms for modeling.

Co-modelling organization starts after implementation of the SDL and the SystemC parts of the model. The SystemC project should contain a special thread, which is intended for the SDL part (*SDL_thread*). This thread calls the *SDL_kernel* function *SDL_Simulate()*. Sensitivity for *SDL_thread* can be specified by any acceptable way. The choice of this way depends on requirements to the modeled system. Initialization of the SDL part of the model requires a call of the *SDL_Init()* function.

Fig. 1 shows a simple example of the SDL and SystemC co-modeling. This is an example, when two nodes interact with each other through the channel, but one node is implemented in SDL, another node and channel – in SystemC [6].

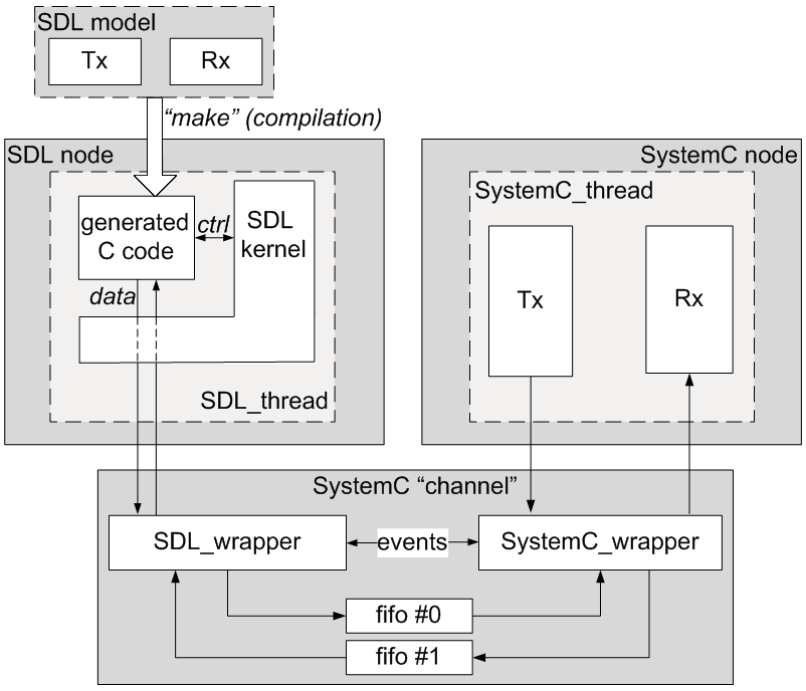

Figure 1. SDL/SystemC co-modeling example

IV. Overview of related studies

Unfortunately there are not many publications in the field of SystemC and SDL co-modeling, however a few good papers can be found. For example, there is a good publication by M. Haroud and L. Blazevic which gives superficial comparison of SDL and SystemC languages and proposes a method for SDL translation into SystemC code [7]. This method is used to translate the SDL implementation of protocol specification into the SystemC model. It allows using rigorous protocol modeling and verification provided by SDL and FPGA netlist synthesis enabled by SystemC. This paper gives a comparison and defines general use of SystemC and SDL for the protocols modeling. Another paper was published by T. Josawa et al. from Nokia Research Center [8]. This work is more closely correlated to this study and the proposed methods for the co-modeling have a lot in common with co-modeling approach described above. Nokia's method is illustrated by Fig. 2.

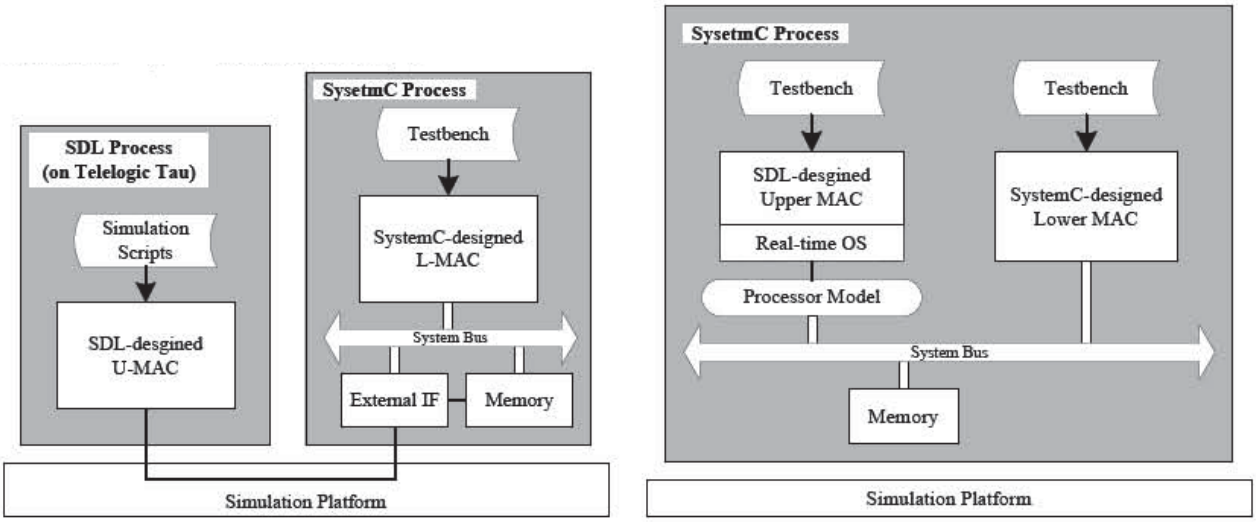

Figure 2. Two approaches for SDL and SystemC co-modeling proposed in Nokia

V. THE PROTOCOL SDL MODELS TESTER

Testing of data transfer protocol models can be organized in the following way. Model of the protocols or protocol stack are implemented according to the protocol specification in SDL language by using such SDL tool as described in the previous section. Then the necessary test environment is implemented in SystemC. And a communication between the SDL model and the SystemC test environment is established according to the SDL/SystemC co-modeling mechanism.

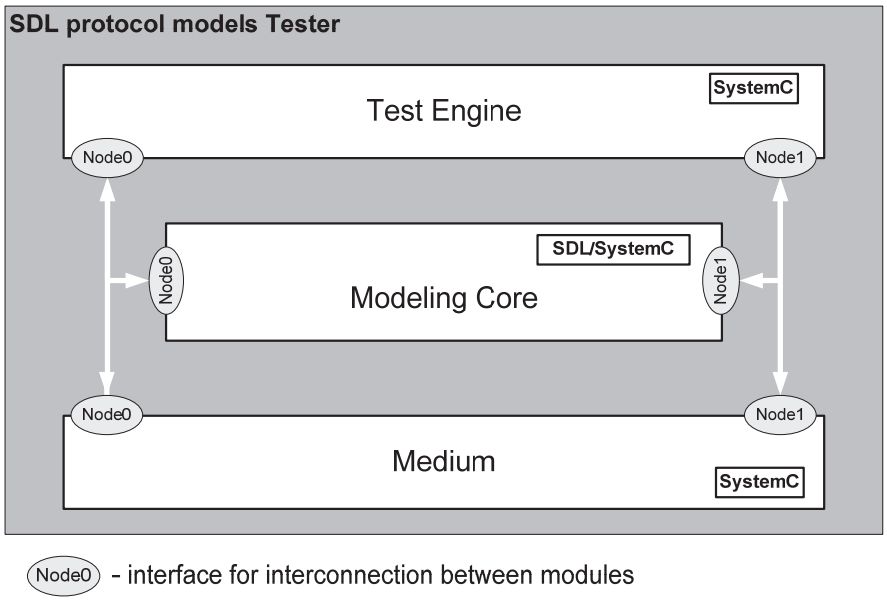

Figure 3. General scheme of the protocol SDL models Tester

A general scheme of the protocol SDL models Tester is depicted on the Fig. 3. The Tester contains three parts: the Test Engine, the Modeling Core and the Medium. Descriptions of all these modules and interconnections between them are discussed below.

VI. THE MODELING CORE DESCRIPTION

The Modeling Core consists of the testing SDL model, the SDL/SystemC Wrapper and the Communication Wrapper. Note that implementation of the Communication Wrapper depends on the SDL model. The features of the SDL model which should be taken into account are a number of layers of the node and interface descriptions of each layer.

The SDL model of a five-layer protocol stack is depicted on Fig. 4. This protocol stack contains five layers: Physical Layer (L1), Data Link Layer (L2), Network Layer (L3), Transport Layer (L4) and Application Layer (L5) [1]. Each Layer of the stack can communicate with adjacent layers through interfaces. The interface defines which primitive operations and services the lower layer makes available to the upper one [1]. For example, when the Network Layer uses a service of the Data Link Layer, in this case the Data Link Layer provides the service and defines an interface for access to it and the Network Layer is a user of the service.

Models of the Layers from first to fourth are implemented in SDL. The Test Engine can represent each of five layers described above in accordance with the following rule: when the Test Engine can communicate with a peer object via a Layer *X* protocol by consuming a Layer *X* service through the interface *X* it means that the Test Engine is a representation of the Layer *X+1* for current stack.

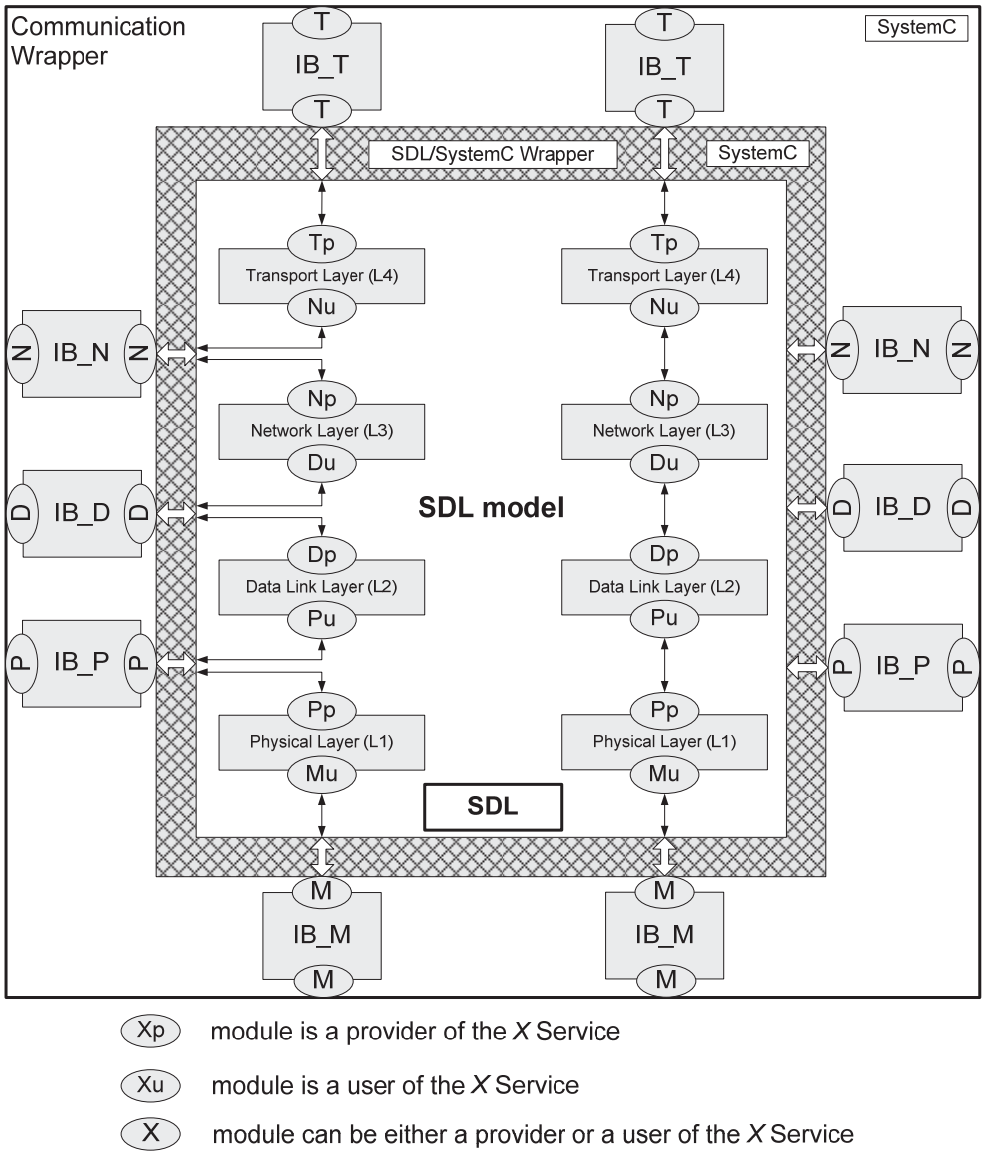

Figure 4. The SDL model of five-layer protocol stack and Communication and SDL/SystemC Wrappers

The discussed stack of protocols includes five interfaces provided by the following layers: the Medium, the Physical Layer, the Data Link Layer, the Network Layer and the Transport Layer. The Communication Wrapper can contain a number of special modules called Intermediate Blocks (IB). Each Intermediate Block addresses to a particular interface, so this IB can be used in the following cases:

- interconnection between the Test Engine and a layer providing the addressed interface;
- interconnection between a layer using the addressed interface and the Medium;
- interconnection between a user and a provider of the addressed interface.

The interconnection between the Test Engine and a layer providing the addressed interface should be established when the Test Engine is a consumer of the defined interface. An example of this case is shown on the Fig. 5. There is the Test Engine which represents the Transport Layer because it uses the *N* interface to access to a Network Layer service. It is implemented by introduction of the *IB_N* block. This block works as a service provider for the Test Engine and as a service user for the Network Layer. Note, that all Intermediate Blocks are implemented in SystemC. The approach of the SDL/SystemC co-modeling organization is discussed above.

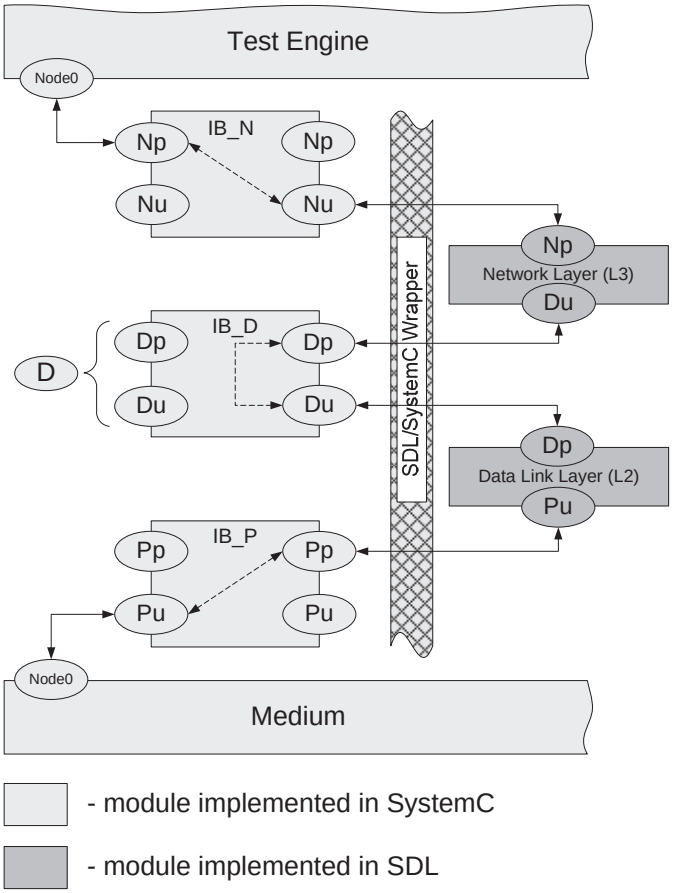

Figure 5. Example of the interconnection between the Test Engine, layers of the SDL model and the Medium through the Communication Wrapper

The interconnection between a layer using the addressed interface and the Medium should be established when the layer is the lowest layer of the stack and its protocol is used for data exchange between the nodes. The Medium implements a connection to the peer node and provides an interface for defined layer. On the Fig. 5 the Data Link Layer is the lowest layer of the stack and communicates with the Medium by the *P* interface through the *IB_P* block. In this case the data exchange between the nodes is performed by Data Link Layer data units. Consequently an object of the peer node connected to the Medium should also support the Data Link Layer protocol.

The interconnection between a user and a provider of the addressed interface leads to two sub cases:

- both user and provider are represented by layers of the stack;
- a user is represented by the Test Engine and a provider is represented by the Medium.

Two adjacent layers of the SDL stack can communicate with each other in two ways: via SDL channels or via an Intermediate Block corresponded to an interface of the lower layer. The last way can be chosen for getting features of an Intermediate Block defined below. This addresses to the first sub case which is depicted on the Fig. 5. There are the Network Layer and the Data Link Layer communicating through the *IB_D* block.

The second sub case is used when there is only one node in the SDL model and the Test Engine corresponds to the second node of the tested system. So, the Test Engine should send data to the first (SDL) node directly, i.e. through the Medium. In this case the Test Engine is a user of communication interface and the Medium is a provider.

Each Intermediate Block can provide the following features: managing data flows transmitted through the Communication Wrapper, parsing transmitted data, making logs and error injection. Conception of the data management in Intermediate Block is defined above. The parsing of transmitted data is introduced for getting information about data exchange between two adjacent layers. Obviously parsing mechanism of each Intermediate Block should be implemented in accordance with the communication protocol of the correspondent interface and purposes of the particular test case. And the making logs mechanism is necessary addition to data analysis, so it is used for monitoring of results. The error injection feature can be introduced for testing of error detection and/or error correction possibilities of a tested protocol.

A set of key issues are necessary to take into account for implementation of the SDL model. Firstly, according to the stack structure each layer shall be represented by one SDL block. Then all required layers should be joined to one SDL system. Each two adjacent layers of one node can be connected in two ways:

- if purposes of the test does not require getting of Intermediate Block features for considered connection, two SDL blocks can communicate through SDL channel only;
- else each block should be connected to the SDL system environment [2]. So in this case both layers will interconnect with an appropriate Intermediate Block.

Note that independently of test purposes the highest and lowest layers of both nodes have to connect to the SDL environment for communication with the Test Engine and the Medium correspondently.

VII. THE MEDIUM DESCRIPTION

The Medium module contains a set of channels and the Control block as depicted on the Fig. 6. Each channel is responsible for connection between two nodes by a particular data transfer protocol. So each channel provides a service for data transmission and an interface to access this service. The Control block sets the following features of the using channel:

- delay introduction – to block a transmitted data unit for a predefined time interval;
- error injection – to corrupt a transmitted data unit according to a predefined condition and value;
- making logs – to write information of particular actions into a specific out stream.

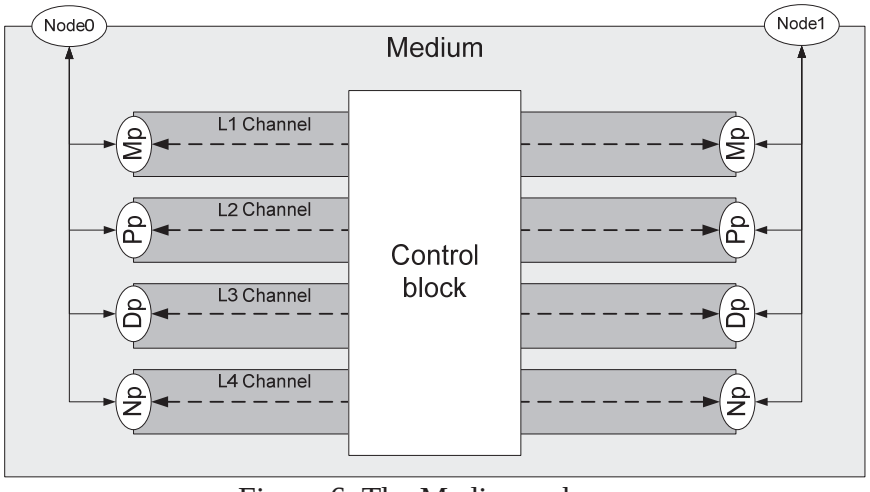

Figure 6. The Medium scheme

An example represented on the Fig. 6 corresponds to the five-layer protocol stack which was discussed above (see description of Fig. 4). Under requirements of this stack testing the Medium has to hold four channels to perform a data exchange by protocols of the following layers: the Physical Layer, the Data Link Layer, the Network Layer and the Transport Layer. So each of these layers can use the Medium for data transmission and access the Medium service through an interface correspondent the underlying layer.

VIII. The test engine description

The Test Engine module is responsible for control of the SDL model simulation. In terms of this requirement this module performs a number of specific tasks.

The first one is configuration of the SDL model Communication Wrapper before the start of test sequence. In accordance with the Intermediate Block features described above the Test Engine defines the transmission mode of each IB, enables/disables the data parsing, the making logs and the error injection.

The second task is configuration of the Medium. During this phase channel parameters such as the channel delay and the error injection are defined.

The main task is data exchange with the testing SDL model. If the tested model does not contain a special traffic generator module the Test Engine represents a certain layer of the tested stack. In the other words the Test Engine operates in accordance with a protocol of the chosen layer and uses services of the layer below. The implementation of these entire tasks is a core of each particular test.

IX. Conclusion

The paper gives the description of how to implement testing of data transfer protocols models. We start with introduction of the network protocols concept. It is based on such notions as layers, protocols and interfaces. The paper provides the proposal for implementation of the protocol models in SDL language and the necessary test environment in SystemC language and discusses the approach of the SDL/SystemC co-modeling.

The main purpose of the paper is the protocol SDL models Tester organization. The Tester consists of the following modules: the Test Engine, the Modeling Core and the Medium. The Test Engine is a driver which is responsible for Tester configuration, for launching a test, for sending and receiving data to/from the Modeling Core and for getting results. The Modeling Core is a combination of the SDL model of tested protocols and two wrappers. The first wrapper consists of Intermediate blocks used for access to the SDL model from the Test Engine and the Medium. The second wrapper is a module providing a communication between the SystemC part and the SDL part. Finally, the Medium block is a set of channels used by SDL nodes for interconnection.

Thereby in terms of simulation and modeling the discussed Tester gives the following abilities:

- representation of testing network layers by means of finite state machine;
- access not to the whole SDL model only but also to certain layers of stack through appropriate Intermediate Blocks;
- getting all necessary test results by SystemC implementation of the test environment.

The similar concept of the protocol SDL models testing has been already implemented in practice. Moreover the area of SDL testing leads to a number of advantageous research directions which can get additional benefits.

REFERENCES

[1] A. S. Tanenbaum, “Computer Networks, Fourth Edition”, Prentice Hall, 2003.

[2] International Telecommunication Union, “Recommendation Z.100. Specification and Description Language (SDL),” Geneva, 2002.

[3] D. Black, J. Donovan, “SystemC: From the Ground Up,” New-York: Springer Science+Buisness Media, Inc, 2004.

[4] V. Olenev, A. Rabin, A. Stepanov, I. Lavrovskaya, “SystemC and SDL Co-Modelling Methods,” *Proceedings of 6th Seminar of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program*, pp. 136-140, Saint-Petersburg: Saint-Petersburg University of Aerospace Instrumentation (SUAI), 2009.

[5] IBM, “SDL Suite and TTCN Suite Help,” *IBM Rational SDL and TTCN Suite*, 2009.

[6] A. Stepanov, I. Lavrovskaya, V. Olenev, A. Rabin, S. Balandin, M. Gillet, “SystemC and SDL Co-Modelling Implementation,” *Proceedings of 7th Conference of Finnish-Russian University Cooperation in Telecommunications (FRUCT) Program*, pp. 130-137, Saint-Petersburg: Saint-Petersburg University of Aerospace Instrumentation (SUAI), 2010.

[7] M. Haroud, L. Blazevic, “HW accelerated Ultra Wide Band MAC protocol using SDL and SystemC,” *Fourth IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)*, 2006,

http://fmv.jku.at/papers/HaroudBlazevicBiere-RAWCON04.pdf

[8] T. Jozawa, L. Huang, E. Sakai, S. Takeuchi, M. Kasslin, ”Heterogeneous Co-simulation with SDL and SystemC for Protocol Modeling,” *Nokia Research Center*, 2006, http://research.nokia.com/node/5789.