# Obtaining an SDL Entity Identifier Using SDL/SystemC Co-modeling

Pavel Morozkin
Saint-Petersburg State University of Aerospace Instrumentation
Saint-Petersburg, Russia
pavel.morozkin@guap.ru

**Abstract**

This paper describes a technical problem of obtaining a unique SDL entity identifier using SDL/SystemC co-modeling. The problem occurs during scientific research in co-modeling field as well as work under real projects. Solution consists in redefinition the standard function of SDL simulation kernel for getting access to entity name and then conversion its name to a unique identifier. The paper explains the situation where problem appears, demonstrates its application using SDL/SystemC co-simulation and describes in practice a solution that fully solved the problem and makes possible further research.

**Index Terms:** SDL, SystemC, Co-modeling.

## I. Introduction

It is a well-known [1] fact that SDL [2] language is most widely used FDT [3] in the telecommunication area. SystemC [4] is a system design language which successfully consolidated the design flow in use at many companies [5]. SDL and SystemC co-modeling [6] is based on model superposition, where each model is designed using different language. One argument in support of this technology — it focus on SDL model testing using co-simulation [7]. Obtaining an identifier of SDL entity is one of many technical problems, which often arise during co-simulation. Besides, in scientific work the problem also causes difficulties for understanding co-model behaviour. In spite of fact that the problem is local, it really complicates the analysis of the co-modeling results.

## II. Way to problem

### A. SDL model features

Understanding the problem requires an example of SDL test model of communication protocol layer. The test system diagram is shown in Fig. 1. The model consists of two nodes each of which has the same type and implements one data transfer protocol layer. During SDL simulation of this system for writing log file or display diagnostic messages a special SDL procedure *SDL_LogMsg* is used. Goal of this procedure — provide a mechanism of writing diagnostic messages in log file during SDL simulation. Also all messages can be of tree types. This is *INFO*, *WARNING* and *ERR* types. The property if this division is different behaviour of SDL simulator. If current message has *INFO* or *WARNING* type, simulation will be continued. But if message has *ERR* type, SDL simulator will be stopped. This mechanism helps to monitor and understand behaviour of the system, because communication protocol specification of industrial standard usually has complex structure. The specification contains many blocks, processes and states. Moreover, SDL system can contain several independent protocol layers which work in parallel. Especially for debugging such systems the procedure has been proposed. An example of using this procedure is shown in Fig. 2 and 3.

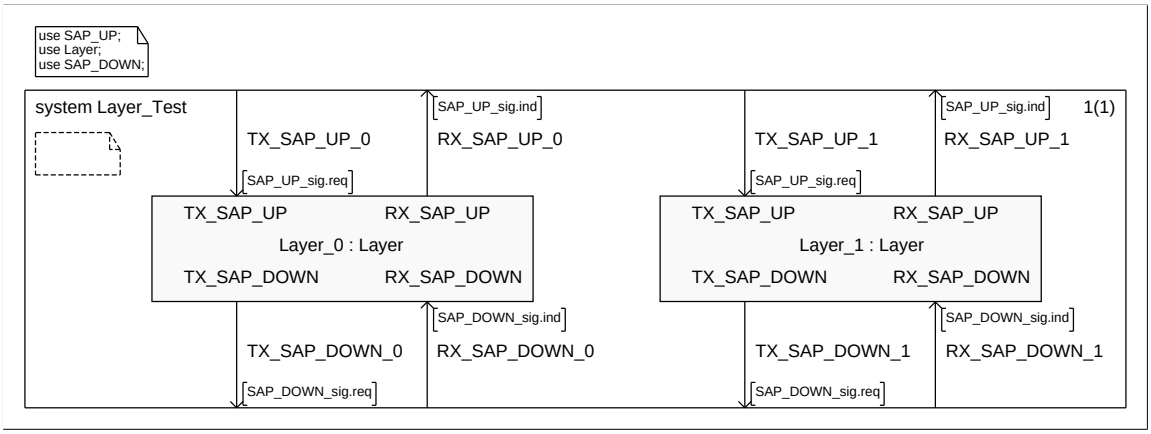


Fig. 1. Test for protocol layer system

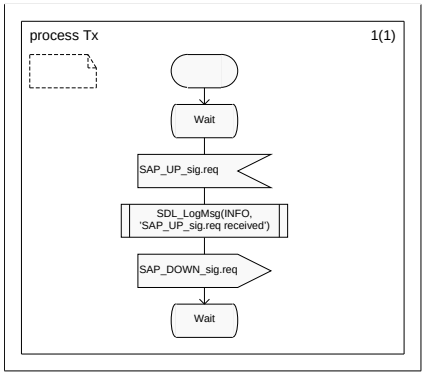


Fig. 2. Tx process

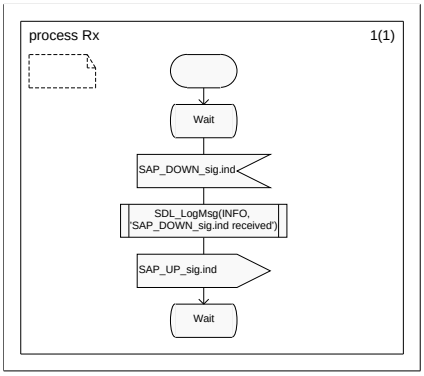


Fig. 3. Rx process

### *B. SDL/SystemC co-simulation features*

SDL/SystemC co-simulation — efficient technology, created as result in-depth research in SUAI university and used for SDL model testing and co-modeling with both SDL and SystemC using features of this languages in common. Technology has been successfully used for creation of communication co-model in such project as UniPro [8]. This model consists of SDL model of protocol layer and SystemC model of the same layer. Communication is performed using SystemC channel. Now proposed technology used in SpaceWire-RT [9] project especially for testing of SDL models of SpaceWire-RT protocol stack. In this case SDL model consists two instances of protocol layer or protocol stack which works independently. Communication is also performed using SystemC channels. For testing of different protocol layers special models of channels are used. They work using several algorithms of errors generation, algorithms of signal loss, etc. SystemC test environment is master component that fully controls of slave SDL system.

The technical organisation of described co-model consists as follows. Firstly performed translation from original graphic SDL model to functionality equivalent C-code. Next step — development of SystemC components which are used for conversion of SystemC primitives to SDL signals. After that SystemC channel created according channel requirements. Next stage is creation of SystemC test control components. Important thing is adaptation of SDL simulation kernel for correct work with both SDL model and SystemC environment. The final

stage is checking correctness of co-model architecture and writing a test cases. Test cases are implemented as SystemC components, which work in accordance with different algorithms. By switching between these components, developers can change test scenarios.

Architecture of a co-model is shown in Fig. 4. It includes three main parts — SystemC test control components, an SDL part (SDL model and SDL simulation kernel [11]) and SystemC channels. SDL model is represented by generated [12] C-code that implements original graphic model.

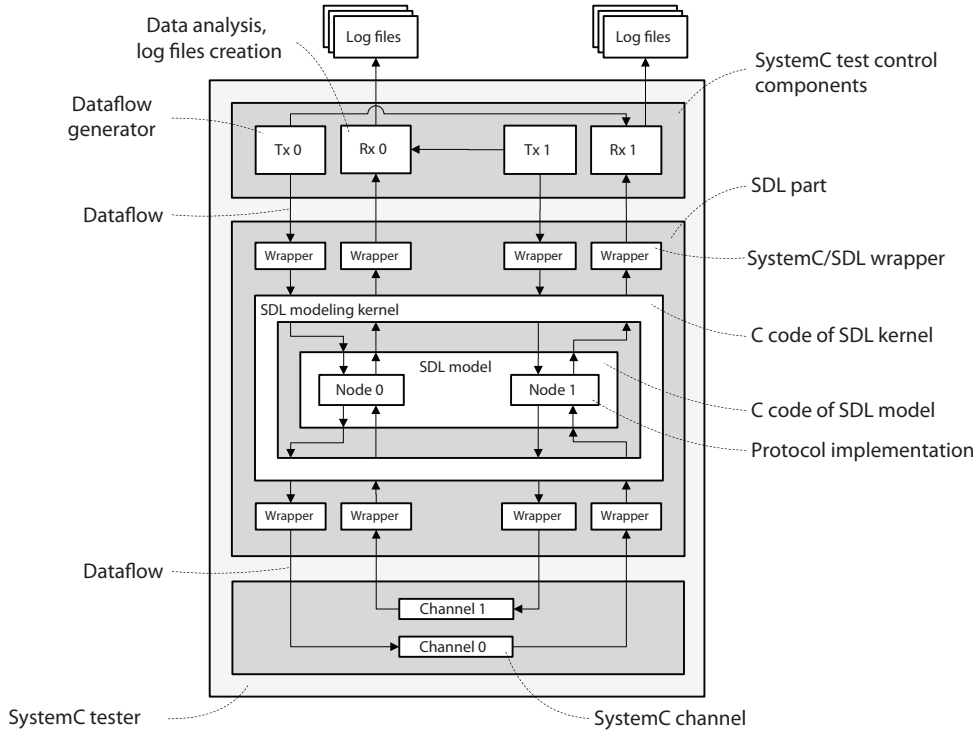


Fig. 4. Co-model architecture

Way of obtaining C-code has been described in detail in [13]. Also co-model includes wrappers that convert SystemC primitives to SDL signals and perform a reverse conversion.

During co-simulation with *SDL_LogMsg* procedure *xPrintString* function is used. The function originally defined in SDL simulation kernel [11] and used for output information about behaviour of model. But in co-model *xPrintString* has been redefined in the following manner as it shown in Fig. 5.

```
extern "C" void xPrintString (char* str)
{
      ostream & stream = cout;
      stream << str << endl;
}
```

Fig. 5. *xPrintString* function

### C. Log files writing features

Usually co-model is designed as such way that each component has its own stream to output diagnostic messages, but currently all streams are configured so that each message

is written in common log file for further behaviour analysis of SDL model. The example of a part of this log file is shown in Fig. 6. Acronym TE means Test Engine, SL — SDL Layer and CH — Channel. The significant problem is that information about number of a

```
           Node #0                                    Node #1
0 ns ..................................................................
-          TE: SAP_UP_sig.req sent
-                                                     TE: SAP_UP_sig.req sent
-    INFO. SL: SAP_UP_sig.req rcvd
-    INFO. SL: SAP_UP_sig.req rcvd
1 ns ..................................................................
-          CH: SAP_DOWN_sig.req rcvd
-                                                     CH: SAP_DOWN_sig.req rcvd
-    INFO. SL: SAP_UP_sig.req rcvd
-    INFO. SL: SAP_UP_sig.req rcvd
```

Fig. 6. Listing of part of common log file

SDL node generating diagnostic message is absent. Therefore, diagnostic messages from SDL model are not formatted. Ascertained only fact that message has been generated, but no more. Certainly, this situation causes difficulties for developers while analysing logs. Developers can only assume a number of SDL instance produced a message. In the case that SDL model is complex, the problem significance is increased. Summing it up, a method that solves the problem and allows make a new research in analysing of co-model behaviour, is needed.

## III. Proposed solution

On the face of it, solution is clearly simple. All that needed is get access to different SDL entity name and then convert its name to number. But during co-modeling SDL model that represented as C-code is similarity of black box. It includes an inputs and outputs for sending and receiving signals and mechanisms for start SDL simulation kernel every time that SystemC test environment has been sent the signal to SDL model. Understanding a C-representation of SDL model — model structure hierarchy, SDL kernel scheduling algorithms using queue of ready processes and queues of input signals, algorithms of communication between SDL components, construction and manipulation of behaviour tree [11], etc — is a complex [10] task. What is worse, SDL kernel is implemented using semi-obfuscated coding style. Nevertheless, after research of structure and functionality of SDL model behaviour tree the way of getting access to current entity name using SDL behaviour tree was found.

Research methodology of this way based on positivist research paradigm and focuses on empirical experiments with structures of generated behaviour tree and finding an efficient solution that solves the problem as well as satisfies the features of co-modeling. This means that new solution should not increase the simulation time, be acceptable to developers and should have high level of usability.

The solution consists in another redefinition of *xPrintString* function. The second argument has been added — *Self* [11] pointer to variable of *void* type. After addition of new functionality *xxPrintString* has been created. Part of its implementation is shown in Fig. 7.

During co-simulation SDL entity name of highest system level is obtained from behaviour tree of SDL model. Then the name is converted to an integer number using its last character, which used for identification. Therefore, a natural requirement for proposed solution is an SDL name format like *EntityName[identifier]*. Note that in this case identifier can be within the range 0-9. Fig. 8 illustrates the way to obtain name of SDL entity using behaviour tree.

```
extern "C" void xxPrintString (char* str, void* Self)
{
      short entity_id = 0;
      SDL_PId* P = (SDL_PId*)Self;
      xIdNode Unit = (xIdNode)P->LocalPId->PrsP->NameNode->Parent;
      Unit = Unit->Parent;
      ...
      entity_id = atoi (&(Unit->Name[strlen(Unit->Name)-1]));
      ...
}
```

Fig. 7. *xxPrintString* function

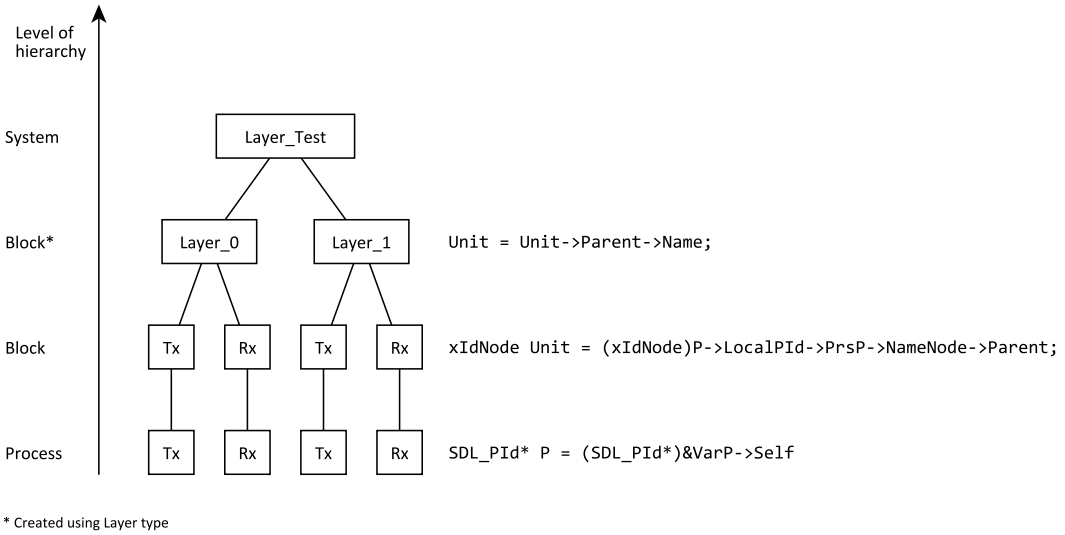


Fig. 8. Entity name obtaining steps

To use *xxPrintString* during co-simulation the developer should redefine original function using macro *XXPRINTSTRING* shown in Fig. 9. This macro should be included in the generated C-code of the SDL model.

```
#ifdef XXPRINTSTRING
#define xPrintString(str) xxPrintString(str, &VarP->Self)
#endif
```

Fig. 9. *XXPRINTSTRING* macro

As a second argument *xxPrintString* takes address of *Self* field of *VarP* [11] structure. This structure is a part of generated behaviour tree and it contains information about the current SDL process instance. After all redefinitions are completed, developer can obtain current SDL entity identifier during SDL/SystemC co-simulation and use it for different purposes, for example for logs formatting. A part of formatted version of the common log file is shown in Fig. 10. It's self-evident that in case where SDL model consists of complex communication protocol layers, formatted log file provides a clarity for scientific analysis of system.

```
          Node #0                                   Node #1
0 ns ..................................................................
-          TE: SAP_UP_sig.req sent
-                                                    TE: SAP_UP_sig.req sent
-    INFO. SL: SAP_UP_sig.req rcvd
-                                              INFO. SL: SAP_UP_sig.req rcvd
1 ns ..................................................................
-          CH: SAP_DOWN_sig.req rcvd
-                                                    CH: SAP_DOWN_sig.req rcvd
-    INFO. SL: SAP_UP_sig.req rcvd
-                                              INFO. SL: SAP_UP_sig.req rcvd
```

Fig. 10. Part of formatted common log file

## IV. Conclusion

The goal of this paper — to propose and explain the solution for an SDL entity identifier obtaining problem. The solution consists in using created *xxPrintString* function during co-simulation. As a result developer can monitor the behaviour of the SDL system controlled by SystemC test environment. Therefore, the problem is solved.

## Acknowledgment

The investigations and results presented in the current paper are performed under the financial support of the Ministry of Education and Science of the Russian Federation.

## References

[1] A. Mitschele-Thiel, "Systems Engineering with SDL: Developing Performance-Critical Communication Systems", *Wiley, 1St Edition*, pp. 3-5, 2001.
[2] Recommendation Z.100, "Specification and Description Language (SDL)", *International Telecommunication Union*, 2002.
[3] K. J. Turner, "Using Formal Description Techniques – An Introduction to Estelle, LOTOS and SDL", *John Wiley & Sons Inc*, 1993.
[4] D. Black, J. Donovan, "SystemC: From the Ground Up", *Springer; 1 edition*, 2004.
[5] T. Grötker, S. Liao, G. Martin, S. Swan, "System Design with SystemC", *Springer, 1 edition*, 2002.
[6] S. Balandin, M. Gillet, I. Lavrovskaya, V. Olenev, A. Rabin, A. Stepanov, "Co-Modeling of Embedded Networks Using SystemC and SDL", *IJERTCS journal, Volume 2, Issue 1*, 2011.
[7] M. Gillet, "Hardware/software co-simulation for conformance testing of embedded networks", *Finnish-Russian University Cooperation Program in Telecommunications (FRUCT) seminar*, Tampere, 2011.
[8] MIPI Alliance, http://mipi.org/specifications/unipro-specifications.
[9] SpaceWire-RT project, http://spacewire-rt.org.
[10] Daniel Dietterle, "Efficient Protocol Design Flow for Embedded Systems", 2009.
[11] Telelogic, "Telelogic Tau 4.5 Users Manual", Chapter 62, The Master Library, 2003.
[12] Telelogic, "Telelogic Tau 4.5 Users Manual", Chapter 58, The Cadvanced/Cbasic SDL to C Compiler, 2003.
[13] P. Morozkin, "Possibility of SystemC code generation from SDL specification", *Finnish-Russian University Cooperation Program in Telecommunications (FRUCT) seminar*, St. Petersburg, 2012.