SystemC and SDL Co-Modelling Implementation

Alexander Stepanov, Irina Lavrovskaya, Valentin Olenev, Alexey Rabin
Saint-Petersburg State University of Aerospace Instrumentation
Saint-Petersburg, Russia
Alexander.Stepanov@guap.ru, Irina.Lavrovskaya@guap.ru, Valentin.Olenev@guap.ru, Alexey.Rabin@guap.ru

Sergey Balandin, Michel Gillet
Nokia Research Center and Nokia Devices
Helsinki , Finland
Sergey.Balandin@nokia.com. Michel.Gillet@nokia.com

Abstract

Nowadays SDL and SystemC are two very popular languages for modelling embedded systems. Each of them provides some specific features, which are not supported by another language. So the hypothetical possibility of combined use of these two languages promises a number of benefits for researchers. There are several approaches for co-modelling on SDL and SystemC. This paper specifically addresses and discusses one of them. The considered way is the most applicable from the point of view of implementation and makes strong integration of SDL and SystemC languages. Moreover nowadays this approach is successfully used in practice.

INDEX TERMS: CO-MODELLING, EMBEDDED NETWORKS, SYSTEMC, SDL, MODELLING ALGORITHMS.

I. INTRODUCTION

The embedded system can be seen as a special kind of computer system where the processing logic is built-in to the device on which it operates. Such systems are developed to carry out one or several specialized functions and usually have strict real-time computational restrictions.

The embedded systems can be implemented for the single-chip microcontrollers and specialized or universal microprocessors. Examples of such embedded systems are cash dispenses, payment terminals, handheld computers, telecommunication equipment; similar systems are built in even such devices, as traffic lights [1].

The embedded systems design encounters a number of difficulties caused by increasing complexity of projects, increasing requirements to products reliability, power consumption and demand to maximum speed-up the project design phase. The traditional approach to the system design based on “top-down” principles is not longer capable to fulfill these requirements. In the design of modern systems that contain one or several processor kernels, so-called “spiral” design procedure is used.

II. Main Part

A. Overview of embedded systems modelling

The main objectives of the project specification process are definition and specification of the basic functions of the system and development of executable system model. This system model is used to verify correctness of the system operation from the functional point of view, estimate required hardware resources and define the system architecture. It is used also to check various mechanisms and approaches defined in the specification for the internal correctness and compatibility.

The important stage of the embedded systems design is testing of the target systems by using the exact computer model of the standard as it is defined in the specification. This modelling is used for validation of the specification itself and for testing used algorithms and major system modules.

For developing of system models different languages are used (such as SDL, SystemC, VHDL, Verilog, etc.). Let's consider first two languages. SDL and SystemC are much different and also they give different abilities for developers and have various benefits.

B. Brief comparison of SDL and SystemC

The SystemC modelling becomes one of the most efficient and widely used methods for studying, analysis and constructing multi-component systems, such as stacks of protocols, embedded networks of a large number of nodes, systems-on-chip, networks-on-chip, etc. The SystemC is C++ library that is specifically designed for modelling parallel systems. This library allows describing multi-component systems and program components and modelling their operation [2].

The SDL (Specification and Description Language) is a language for unambiguous specification and description of the telecommunication systems behavior [3]. It is intended for description of structure and operation of the distributed real-time systems. SDL structuring features are its key advantage as they allow simplifying the description of the large and compound systems [4].

Selection of the most suitable modelling language is the key factor of success and should be carefully thought before the system modelling starts. This choice depends on a large number of factors, which primary driven by the vision of what kind of problems need be solved by the developed model. Let's now discuss the applicability difference of SDL and SystemC languages for creation of protocol models. Both languages could be potentially used for solving any modelling problem, but the efficient use requires a number of features to be taken into account in each concrete situation.

There are two main approaches to modelling the data transmission protocol. The first one suggests per layer modelling of the protocol stack [5]. SDL is the most reasonable solution in the case, when description of system modules and interactions between them are most important [4]. If the main purpose of per layers protocol modelling is to model internal logic of the protocol, then SystemC model is more applicable. The second approach to protocol modelling assumes modelling of a system of devices [5]. Using SDL in this case is not efficient because all interactions between instances should be specified completely manually. In its turn, modelling of large networks in SystemC requires to describe classes for devices and to create interconnection between instances of these classes by binding all ports. So there will be no need to describe each connection individually.

Another factor to be considered is a possibility to dynamically determine parameters of the system in the second modelling approach. Assume there is a need to create a model of switch to simulate a network of devices. SDL language does not allow giving number of switch ports as a parameter. Designer has to describe switches with different numbers of ports as completely types. While in SystemC the programmer can define the switch class and set the number of ports as a parameter. Thus, in this case SystemC is better fulfilling the system of devices description.

Possibilities for testing of the models are the last point of our comparative analysis. Thus SDL is more suitable for the testing of models for which inter-module interactions are significant, i.e. there is no need to test separate parts of the model. Whereas SystemC models are better suitable for testing in all other types of cases. However, the role of proper competent code writing is also stronger increasing in this case.

Summarizing the above we can say that the choice of modelling language in each concrete case primary depends on the modelling purposes. In addition, it is usually useful to specify main stages of work and ways of dealing with them before actual development starts. For instance, a number of modeled objects, timing, and further testing's implementation way are among the most important factors. Each of these factors can have a significant impact on the decision to favor one or another language [6].

C. SDL/SystemC co-modelling organization

In this paper we want to discuss the approach for solving the task of co-modelling organization. This way assumes to have a SystemC project, which corresponds to the whole model. The whole model contains the SDL and SystemC parts. Consequently, this approach uses the C/C++ representation of the SDL system [7].

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

Let's consider some abstract SDL Tool. This SDL Tool should meet the following requirements:

- provide possibility to generate C code from the implemented model, which will be the equivalent of the SDL;
- the generated C code operation should be controlled by some kind of manager engine (*SDL_kernel*);
- *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 SDL system, so that one SDL transition is executed during each call of this function. One SDL transition is a change of system state from one to another.

It should be pointed out that such kinds of SDL Tool exist, for example, all these features are provided by IBM Rational SDL Suite.

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

- preparation of SDL system as the part of the whole model;
- generation of C/C++ code on the basis of created SDL system, after it SDL code is not used;
- insertion of this C/C++ code to the SDL kernel;
- preparation of SystemC part of the model;
- integration of SDL kernel with the generated C code into the whole model.

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

This approach has several disadvantages. To make any structural change in SDL model designer has to make necessary modifications in SDL code and then rebuild it to C/C++. Another disadvantage is a need to create interface to connect C/C++ analogue of SDL and SystemC models.

But on the other hand this way of co-modelling has a number of advantages. Thanks to this approach it is possible to work with both SDL and SystemC parts at the same time. Moreover, this co-modelling solution allows using of SystemC scheduling because the SystemC part is a master [7].

Co-modelling organization starts after implementing SDL and 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 SDL kernel function *SDL_Simulate()*. Sensitivity for *SDL_thread* can be specified by any acceptable way. The choice of this way depends on the requirements to the modeled system. The initialization of SDL part of the model requires call of *SDL_Init()* function.

Fig. 1 shows a simple example of SDL and SystemC co-modelling. 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 are in SystemC. This system will be used as an example to describe realization of discussing approach.

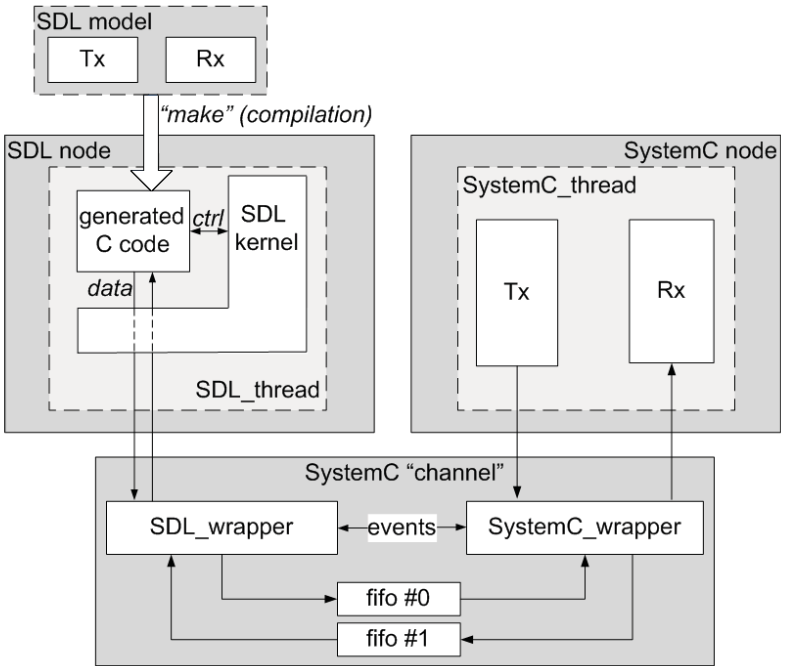

Fig. 1. SDL/SystemC co-modelling example

D. Notion of scheduling

One of the key aspects for discussing modelling principles of different languages is scheduling as it is very important for elaboration of co-modelling approaches.

To discuss this topic the notion of delta-cycles should be introduced. The first solution for organization of delta-cycles is provided in SystemC and requires declaration of a notion for delta-delay (Δ), which is used to define cause-effect relation between different events. In terms of modelling time the delta-delay triggers in zero time. If two events occur at the same time, but the first event causes the second one then it is considered as there is "delta-delay" between these events. From the viewpoint of modelling time one clock-cycle contains a large number of delta-cycles. The modelling time increments after each event, referred to that moment as executed [2].

Fig. 2 shows the cause-effect relation between events in SystemC.

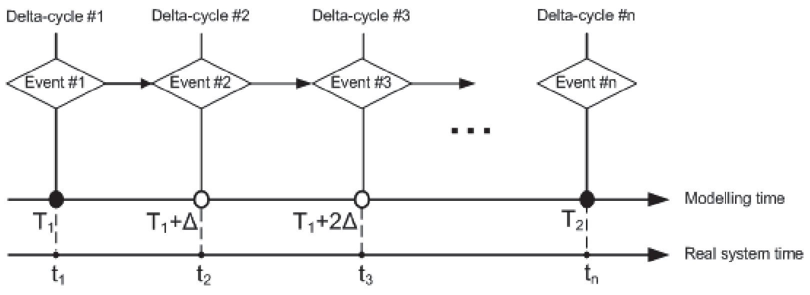

Fig. 2. Cause-effect relation of events in SystemC

Implementation of the delta-cycle is defined in the several steps. They are task scheduling, value evaluation, value update and processing of notifies. The typical flow chart for handling delta-cycles is shown on Fig. 3.

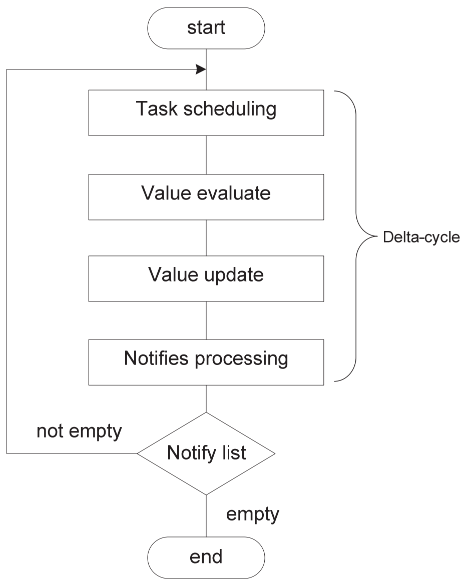

Fig. 3. Algorithm of handling delta-cycles

SDL specification doesn’t define notions of the delta-cycles and delta-delays. However, we have to introduce these structures to describe the principles of SDL modelling for elaboration of co-modelling approaches.

Let’s take certain moment of modelling time, when there is a number of scheduled events, where each SDL event represents transition of the process from one state to another. Each delta-cycle contains one execution of SDL transition and consequently during SDL delta-cycle a set of tasks is performed, one of which is scheduling of the new events. There are two ways for events scheduling – signals and timers.

Using of signals means that the event should be performed at the current moment of modelling time. Such event is processed during the next delta-cycles after delta-delay and this delta-delay triggers in a zero time.

The timer expiration is scheduled at the different moment of modelling time. So it causes new event, which is processed when all current time events will be performed.

During *SDL_Simulate()* call a SystemC event should be scheduled. If this event is scheduled by a signal, then SystemC function *notify(SC_ZERO_TIME)* should be called. Thus, *SDL_Simulate()* function is called at the next delta-cycle. Otherwise, if timer setting is used for scheduling, then SystemC function *notify(delay > 0)* is called. To sum up, for SDL/SystemC co-modelling a common scheduler of events is used.

E. Way to address the co-modelling approach

Interaction between SDL and SystemC parts (look at the system on Fig. 1) is organized in the following way:

- each node generates data and sends it (as *data.request*) to the remote one via channel;
- local *data.request* and *data.response* become *data.indication* and *data.confirm* for the remote node correspondingly;
- when node gets the *data.indication*, it generates and sends *data.response*;
- when node gets the data.confirm, it generates and sends new data (*data.request*).
- reading and writing operations to SystemC channel is processed at different delta-cycles [2].

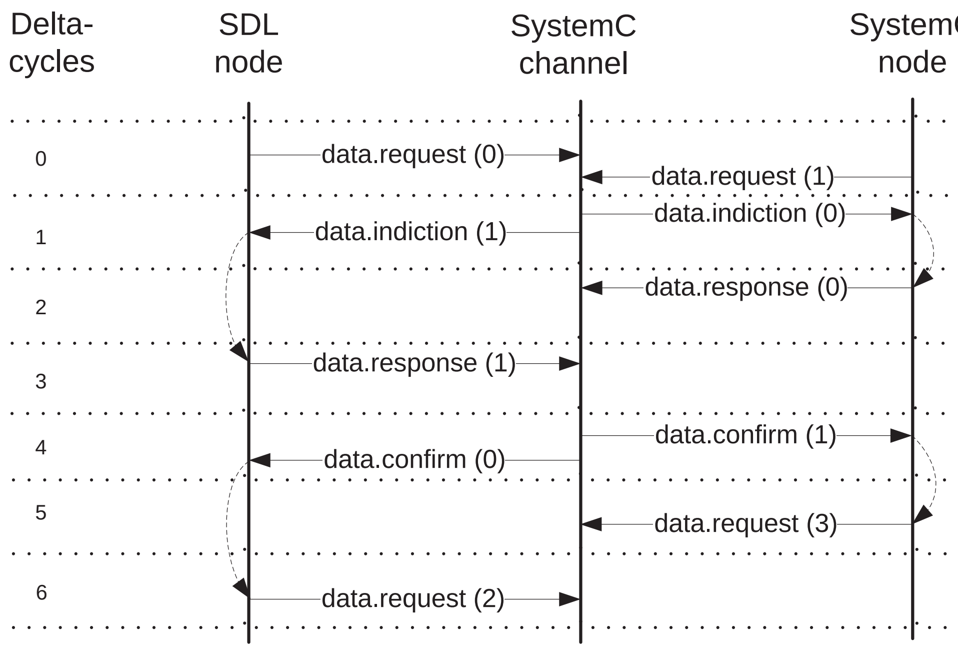

Fig. 4. Interaction between SDL and SystemC parts

A description of interaction between SDL and SystemC parts according to the example above is presented in Table 1.

III. Conclusion

The paper gives analysis of the way how SDL and SystemC languages can be used together for co-modelling of the embedded systems.

The paper provides brief comparison of SDL and SystemC and shows advantages and disadvantages of these languages for modelling.

The main purpose of this paper is how to organize co-modelling of SystemC and SDL in order to take the best from both languages. We propose the approach for organization of SDL/SystemC co-modelling, which combines all advantages of SystemC and SDL modelling. For implementation of this way additional specified SDL Tool features are required and a common event scheduler should be used. We consider using scheduling mechanisms of SystemC for it. Thus SDL Tool should be able to use SystemC functions for event triggering. For demonstration of proposed approach a simple example of SDL/SystemC co-modelling is given.

TABLE 1.
DESCRIPTION OF SDL AND SYSTEMC INTERACTION FOR THE EXAMPLE

Delta -cycle	**SDL part work**	**SystemC part work**
0	SDL thread calls *SDL_Simulate()* function. During this call SDL Tx generates *data.request* and writes it to SystemC channel. After this function *SDL_Simulate()* is finished.	SystemC Tx generates *data.request* and writes it to channel.
1	SDL thread calls *SDL_Simulate()* function. During this call SDL Rx reads *data.indication* from channel and sends SDL signal to intermediate block between Tx and Rx. So a new event is scheduled at the next delta-cycle.	SystemC Rx reads *data.indication* from channel and sends incoming data to the Tx part. So a new event is scheduled at the next delta-cycle.
2	According to the scheduled event, SDL thread calls *SDL_Simulate()*. During this call SDL intermediate block sends signal to Tx. So a new event is scheduled at the next delta-cycle.	According to the scheduled event SystemC Tx receives data from Rx and generates new *data.response* and writes it to channel.
3	According to the scheduled event, SDL thread calls *SDL_Simulate()*. During this call SDL Tx receives signal from intermediate block and generates new *data.response* and writes it to channel.	
4	SDL thread calls *SDL_Simulate()* function. During this call SDL Rx reads *data.confirm* from channel and sends SDL signal to intermediate block between Tx and Rx. So a new event is scheduled at the next delta-cycle.	SystemC Rx reads *data.confirm* from channel and sends incoming data to the Tx part. So a new event is scheduled at the next delta-cycle.
5	According to the scheduled event, SDL thread calls SDL*_Simulate()*. During this call SDL intermediate block sends signal to Tx. So a new event is scheduled at the next delta-cycle.	According to the scheduled event SystemC Tx receives data from Rx and generates new *data.request* and writes it to channel.
6	According to the scheduled event, SDL thread calls *SDL_Simulate()*. During this call SDL Tx receives signal from intermediate block and generates new *data.request* and writes it to channel.	

References

[1] Wikipedia, free online encyclopedia. (2008). Embedded Systems. Retrieved June 29, 2008, from http://en.wikipedia.org/wiki/Embedded_system.

[2] Black, D., & Donovan, J. (2004). *SystemC: From the Ground Up.* New-York: Springer Science+Buisness Media, Inc.

[3] International Telecommunication Union. (2002). *Recommendation Z.100. Specification and Description Language (SDL).* Geneva.

[4] Karabegov, A., & Ter-Mikaelyan, T. (1993). *Introduction to the SDL language*. Moscow: Radio and communication.

[5] Olenev, V. (2009). Different approaches for the stacks of protocols SystemC modelling analysis. Proceedings of the Saint-Petersburg University of Aerospace Instrumentation scientific conference (pp. 112-113). Saint-Petersburg: Saint-Petersburg University of Aerospace Instrumentation (SUAI).

[6] Stepanov, A. (2009). Comparison of SDL and SystemC Languages applicability for the protocol stack modelling. Proceedings of the Saint-Petersburg University of Aerospace Instrumentation scientific student's conference (pp. 76-80). Saint-Petersburg: Saint-Petersburg University of Aerospace Instrumentation (SUAI).

[7] Olenev, V., Rabin, A., Stepanov, A., & Lavrovskaya, I. (2009). 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).