EaSySim II SDL Extensions for Performance Simulation

Rainer Gerlich
BSSE System and Software Engineering

Auf dem Ruhbuehl 181
D-88090 Immenstaad, Germany

Phone +49/7545/91.12.58 Mobile: +49/171/80.20.659 Fax +49/7545/91.12.40
e-mail: gerlich@t-online.de www: http://home.t-online.de/home/gerlich/

Keywords: SDL, performance, simulation, system validation, SDL extension, EaSySim II

1. INTRODUCTION

In 1992 ESA/ESTEC started activities which aimed to support system validation by simulation already in an early development phase in order to reduce the development risks, time and costs. A first study called HRDMS (Highly Reliable Data Mangement System and Simulation) [1] concentrated on performance aspects of system validation. During this step the performance simulation tool SES/workbench [2] was applied. In order to cope with validation of a system's behaviour SDL was selected during the following study OMBSIM (On-Board Management System Behavioural Simulation) [3] due to its capabilities to specifiy a system's behaviour in a formal manner, to simulate and verify a system's model and to generate code from the model.

The HRDMS actitivities showed that an approach for system validation needs to cover both, behavioural and performance validation. Therefore for OMBSIM GEODE [4a] from Verilog was complemented by the performance simulation tool SES/workbench. The resulting environment was applied in the following ESA/ESTEC project DDV (Data Management System Design Validation) [5].

Due to the limitations of SDL88 concerning procedures the coupling between both tools could only established at code generation level, not at simulator level. Also, problems were encountered with stimulation of the system by operational and test data, mainly related to their distribution through the (SDL) system. This caused an unacceptable high instrumentation of SDL code which was difficult to be removed automatically.

The coupling at code generation level allowed only to investigate performance matters for a number of test and operational scenarios as defined by the user, but not to run exhaustive tests as supported by the simulator. This indicated a need for performance extensions of the simulator. On the other side, it was obvious that tool coupling and the specific dynamic organisation of performance simulation tools which require big resources (space and time) would not allow to execute the large number of test cases usually occuring during exhaustive simulation. This lead to the decision to extend SDL itself towards performance aspects.

Based on the experience gained during the previous activities BSSE established a new environment EaSySim II [6] which extends SDL and ObjectGEODE [4b] (the latest version of the GEODE tool) by features needed to validate a system for functional, behavioural and performance properties. This extension was possible due to the new features provided with SDL92, especially for procedures.

EaSySim II supports performance modelling with SDL in a general manner: it provides functionality by which resources such as processor time or bus cycles can be consumed when

a state transition is executed. Pre-emptive and non-pre-emptive consumption of resources is possible.

2. THE SDL EXTENSION OF EASYSIM II

By EaSySim II a set of SDL procedures and additioal operators (implemented in C) are provided which allow to model a certain strategy for access and consumption of resources. When calling such a SDL procedure the resource and a logical identifier are passed by which the resulting consumption and the access priorities are identified. A process activity consuming a resource is suspended when it asks for the resource and it is resumed when it has consumed the resource.

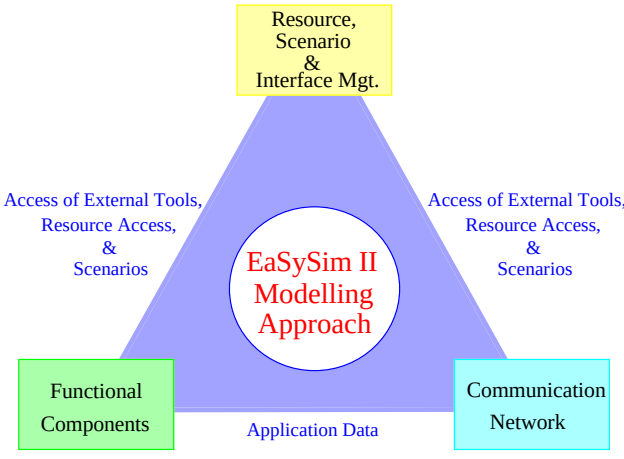

Fig. 1: The Δ©-Approach of EaSySim II

The amount of consumption is specified by resource specific units e.g. in terms of processor or bus cycles at user level and at hardware interface by hardware related units which allow to consider different power of processors or bus rates.

Mapping of SDL processes onto resources usually breaks the process and communication hierarchy established in SDL. This is a constraint for propagation of "resource" signals imposed by SDL which makes it difficult to distribute the information related to resource consumption directly from the resource management to a SDL process.

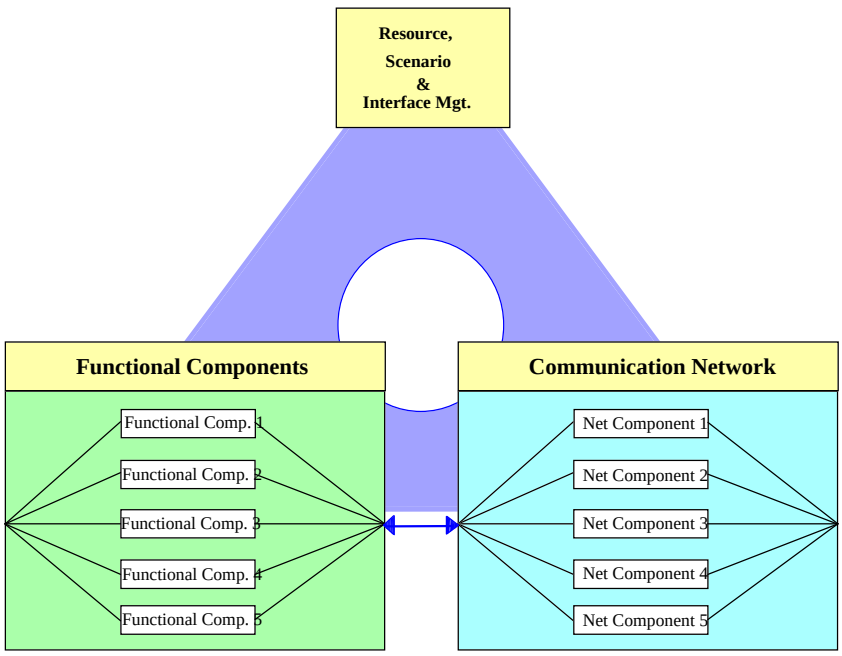

Fig. 2-: Refined View on the Δ-Approach

In the EaSySim II environment this problem has been solved by the Δ© approach (Fig. 1). Each of the application components allocated at the "bottom corners" of the triangle can access directly the resource manager at the top. Each of the two (SDL) application blocks contains again a flat nested hierarchy of blocks as shown by Fig. 2. So each block and at the lowest level each process can directly communicate with the "Resource Manager" at the top. The problem with propagation of resource signals disappears.

This solution has been introduced into EaSySim II to cover fault-tolerance and system reconfiguration, but now it serves well also for the performance extension of SDL. By organisation of a SDL system like shown by Figs. 1 and 2 it is possible to implement a system's topology by data, which means the data flow is not determinned in a "hard-coded" manner by the SDL channels, but dynamically established by data at run-time which are read from a file, at least once when the system is started.

Both capabilities (resource management and dynamic topology) allow to evaluate efficiently performance of different system configurations and topologies as shown in chapter 4.

3. MORE DETAILS ABOUT THE EASYSIM II IMPLEMENTATION

EaSySim II provides a user with a SDL skeleton and templates for blocks, processes and signals which allow him to plug-in his application and to execute it in a combined behavioural/functional and performance modelling environment: the intention of EaSySim II is to support an engineer such that he can more concentrate on the issues of the application rather than on modelling issues.

The support is given by SDL procedures and by operators. Data for resource consumption and topology are provided by files which contain application specific information.

3.1 Description of the Extensions

3.1.1 Consumption of Resources

Consumption of resources is specified on a logical level in each process at SDL statement level. Following information has to be provided:

1. at system level globally for each process a resource (e.g. processor or bus) has to be specified,
2. at process level logical references are used per process, instance and resource; they define the application specific resource consumption in terms of cycles of the resource and are inserted as procedure calls,
3. when a resource is accessed the number of cycles to be consumed is derived from three parameters: the user units, the basic system cycle (expressed as duration) and a weight (multiplicator) which allows to introduce different power of resources in order to make them comparable.

By (2) specific resource consumption can be asked for within a process depending on the type of the executed SDL statement or a number of statements. Such a request may be inserted once or several times into a state transition or into other sequences of SDL statements whenever a (global) procedure call is allowed. (1) and (3) are determinning the actual value of resource consumption when the procedure is called.

The global procedure CONSUME is provided with EaSySim II to cover resource consumption. It expects as parameters the process id (logical name of the process, called "device"), its logical (EaSySim II) instance identifier (which is different from a SDL instance), the resource and the logical units of resource consumption. The procedure calls which instrument the SDL code may be automatically removed by an EaSySim II tool when code is generated for the target system.

In case of a bus the length of a data record is taken to compute the resource consumption of the bus because the number of bytes are determining the number of bus cycles. Also, a priority may be given to allow for priority-based scheduling.

Different weights (see 3 above) for resources can be assigned by data included in file *cycles.in*. This allows to model different processor performances without changing the logical resource figures. In this manner the data rate of a bus or the performance of a processor may be changed. The logical entity of resource consumption can be a figure for number of instructions and the cycles would refer to the bus clock period.

The logical unit (see 1 above) is converted to a numerical value by data given by the files *timecons.in* and *cycles.in*.

The provsion of such performance data by files allow for fast, comfortable and flexible performance modelling and easy analysis and comparison of performance properties.

In order to reduce the number of SDL processes within a model it is possible to declare a "dumb" process (like a bus) which only serves as (SDL) place holder for a resource as "EaSySim II resource". Then a number of such "EaSySim II resources" may be represented by only one physical SDL process.

3.1.2 Time Management

Time is centrally managed by the RSIM (Resource, Scenario and Interface Manager) part of EaSySim II for optimisation reasons. To consume time e.g. to start a timer and to wait for a timeout a user has to call the global procedure TIMEMGT and to specify the expiration time and an identifier for the logical timer. When the EaSySim II timer expires the resource manager will send back a signal including the timer id to the SDL process so that it can process this event.

The procedure RESETTIMER allows to cancel a timer event requested for by TIMEMGT.

TIMEMGT corresponds to the SDL primitive "set" and RESETTIMER replaces "reset". Both SDL primitives still can be used if a user wants to do so.

3.1.3 Pre-Emptive Resource Consumption and Scheduling

A model can consume resources and time in two different ways: it may be blocked until the resource has been consumed or it may remain ready for other activities during resource consumption. E.g. in case it is waiting for a timeout a model should be capable to execute other activities. Or a model should remain ready to accept more SDL signals when it is consuming a resource during execution of a state transition. This can be compared with a piece of software which can be interrupted by an external event while executing.

When resource consumption is non-blocking, a model is still capable to process incoming SDL signals immediately, although it has been suspended during execution of a state transition. Such later signals may compete with already queued signals according to their priority. This allows realistic modelling of performance impacts.

In the other case, when the model is blocked during consumption of the resource, incoming SDL signals are saved (if SAVE is applied by the user) or discarded (if not saved). Then a process can continue only when the resource has been consumed. EaSySim II provides support for both types of resource consumption.

The procedure CONSUME supports non-pre-emptive modelling, the procedure COMSUMECC (CC=concurrent) is provided for pre-emptive resource consumption.

In case of pre-emptive consumption the delayed signal or data are buffered by EaSySim II and sent back to the model after consumption. In order to continue at the right location within the process, the receiving process needs to know to which event the consumption was related to. This is indicated by an additional parameter (entry point) which has to be specified when CONSUMECC or TIMEMGT are called. On reception of an expiration signal a process determines by the returned parameter via a DECISION where to continue.

3.2 The Provided Procedures

Following SDL (global) procedures are currently available. They are using internally operators implemented in C.

- **CONSUME**(dev,inst,data,resource,cycles)

  It consumes <cycles> units of <resource>. <dev> and <inst> identifiy the consuming device (process and EaSySim II instance). <data> (defined by a standard type) defines the data to be processed, e.g. to be transmitted on a bus, so that the resource consumption on a bus can be derived automatically from the length of the transmitted data.

  The calling process is blocked until end of resource consumption, i.e. the procedure CONSUME is not left before end of consumption.

- **CONSUMCC**(dev,inst,data,resource,cycles,EntryId)

  CONSUMECC (CC=concurrent consumption) allows to consume more than once a resource in non-blocking mode.

  When calling CONSUMECC the process is not blocked. Control is immediately given back from CONSUMECC to the process. At the end of resource consumption the resource manager will send a signal including the delayed data and the specified entry back to the process which will continue at the returned entry point.

  By EntryId a process can distinguish for which event a resource has been consumed. Using EntryId together with a decision will allow a user to continue at the right location in the process after resource consumption.

- **TIMEMGT**(dev,inst,TimerId,expiration time)

  Management of time is supported by procedure TIMEMGT. It is called instead of the SDL "set" (expiration,timer). At <expiration time> a signal including the timer id and a reference time <timeref> is sent back to the calling process. By the timer id a user can distinguish which one of his logical timer has expired. <timeref> provides information on when the timer has expired.

  This information may be of interest in case a process is blocked when the timer signal arrives. Then processing of this signal may be delayed (hopefully not discarded, therefore SAVE should be used) until the process is ready again. By comparison of actual reception time with the reference time a user can identify such an undesired delay.

  TIMEMGT can be used for initiation of activities like cyclic processing or for generation of timeout signals.

- **RESETTIMER**(dev,inst,TimerId)

  An event defined by TIMEMGT for device dev, instance inst and timer TimerId is canceled.

3.3 The Data Files

Data files provide the information needed to derive the actual value of time consumption. Following files are related to resource consumption:

1. devset.in

   This file defines for a logical process (device "dev") the resource it is mapped on.

2. timecons.in

   This file relates the logical reference used by a device "dev" to consume cycles of a resource. To allow for variation of time consumption low and high limits need to be given.

3. cycles.in

   This file expresses the cycle of a certain resource by basic cycles. Hence, a logical cycle can be expressed by absolute values. This allows to model processors and buses of different power leaving the logical value of resource consumption unchanged.

The contents of the files is described below: For devset.in only such fields are listed which are related to resource consumption.

- **devset.in**

  device name
  : defines the process which consumes the resource

  resource
  : defines which (shared) resource is consumed by the process. This allows for a flexible allocation of resources without any need for recompilation.

- **timecons.in**

  device name
  : defines the name of the process by which this reference is used

  consRef
  : defines the logical reference for resource consumption.

  CyclesLow
  : defines the lower limit for resource consumption in logical units of the resource (which is a product of basic cycle and resource cycles (see cycles.in)).

  CyclesHigh
  : defines the upper limit for resource consumption in units of the resource

  At run-time a value between CyclesLow and CyclesHigh is randomly chosen.

- **cycles.in**

  This file allows a user to define the power of each resource in terms of "basic" cycles. The effective number of cycles consumed by a resource per logical unit is

  <cycles of the resource> * <basic cycle>

  resource
  specifies the resource for which the number of cycles are given.

  cycles
  defines the number of basic cycles consumed by a resource per logical unit.

4. ANALYSIS OF RESOURCE UTILISATION: AN EXAMPLE

Fig. 3 shows an application which was implemented with EaSySim II. It is a rather simple application but it serves well to identify the support which EaSySim II gives to system analysis by simulation. The application consists of two senders and a receiver which communicate through a network of four buses. Each sender sends a signal to the receiver and

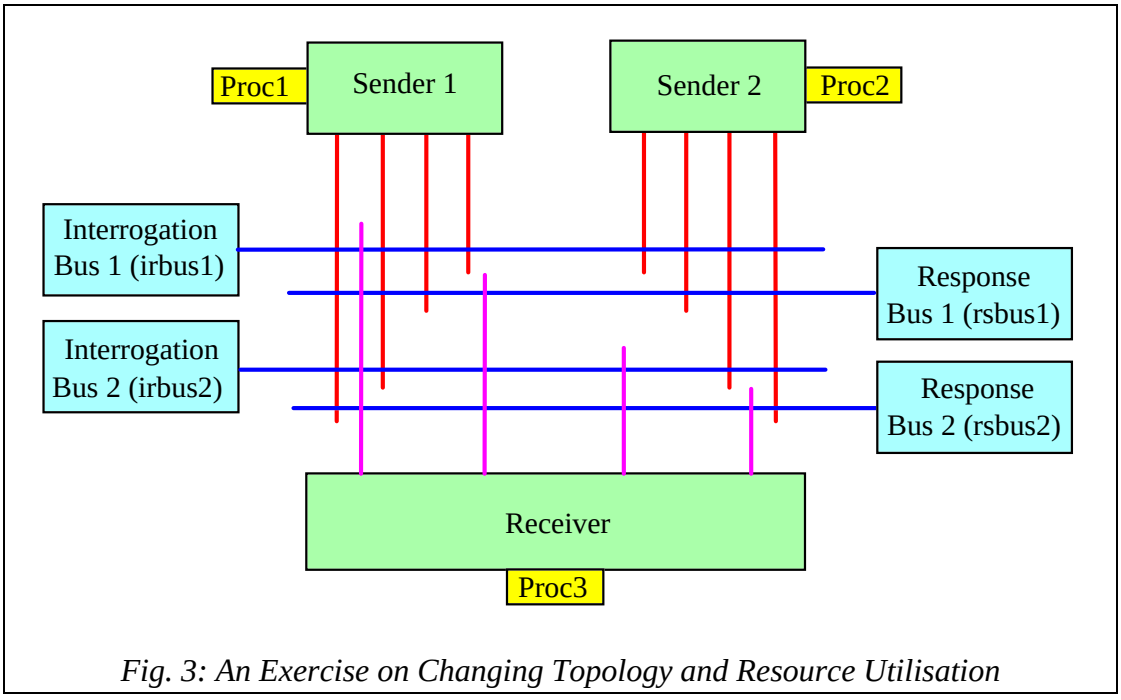

Fig. 3: An Exercise on Changing Topology and Resource Utilisation

the receiver just sends it back to the originator.

Now, we can play with resources which are: the three processors Proc1, Proc2 and Proc3 and the four buses irbus1, irbus2, rsbus1, rsbus2. We can try to let all three processes (sender1, sender2 and receiver) to run on one processor only or each process on an own processor, we can use only one bus for all signals or all four buses each carrying only forward or backward signals for each sender and so on.

There are a lot of combinations for resource usage, each may have advantages and disadvantages. By changing the communication topology (i.e. connecting processors and buses at the interconnecting lines) we can investigate its impact on overall performance, we can change the time consumption of processes on the processors and so on. These performance investigations can either be done by specific scenarios (random simulation) or we can apply exhaustive simulation to all such configurations.

With EaSySim II no recompilation of the SDL source code is needed to change the system topology. Only the description files have to be changed defining (a) which device is connected to which bus, (b) which channel shall carry which signal, (c) on which processor time shall be consumed, (d) what the amount of time consumption for each process and bus shall be.

As examples we define the following configurations:

configuration 1:

irbus1 carries the signals from *sender1* and *sender2* to the *receiver*, and *rsbus1* carries the response signals. *Sender1* and *sender2* share the same processor, *receive*r runs on its own processor.

configuration 2:

all three processes share the same resources: they all run on the same processor and all signals are transmitted via *irbus1*.

Message sequence charts (MSC) derived for these two configurations are shown by Figs. 4 and 5. These MSC's were obtained without recompilation, but just by changing data files. So it is very easy and fast to evaluate different topologies for their performance and to analyse a system's behaviour under realistic conditions.

The MSC's include timing information like start and departure or arrival time which allows to evaluate e.g. data acquisition time or latencies.

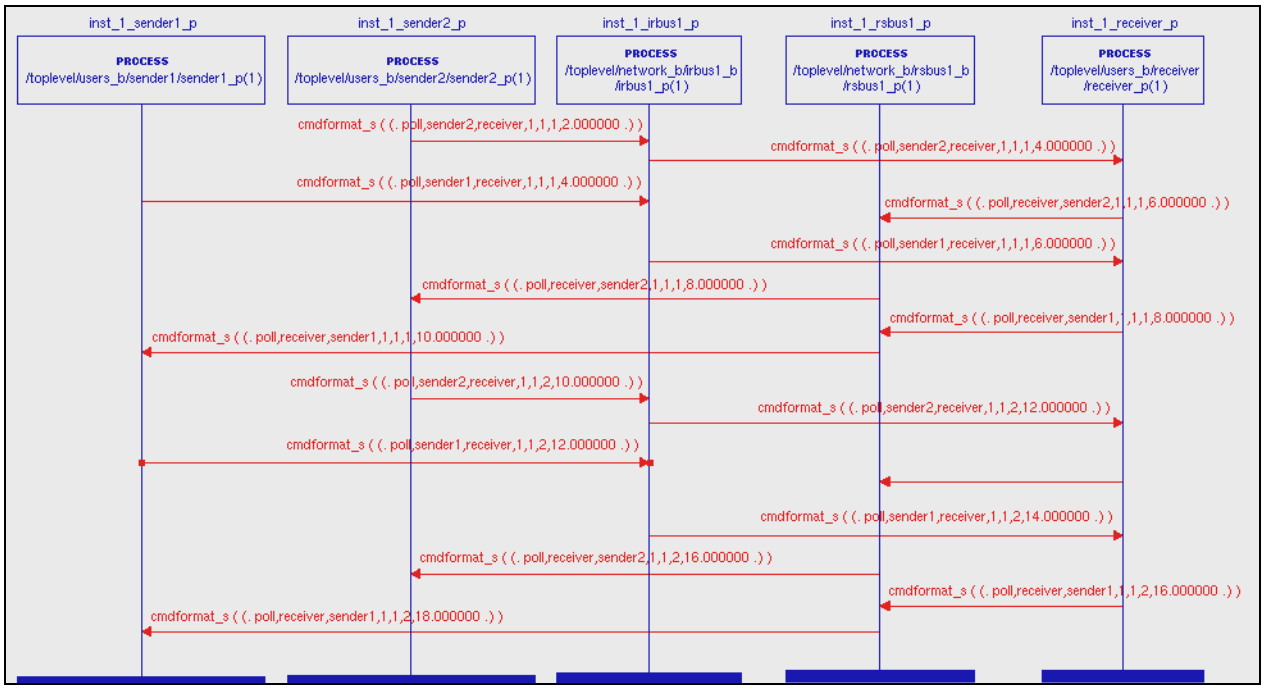

Fig. 4: MSC for Configuration 1 Using Two Buses

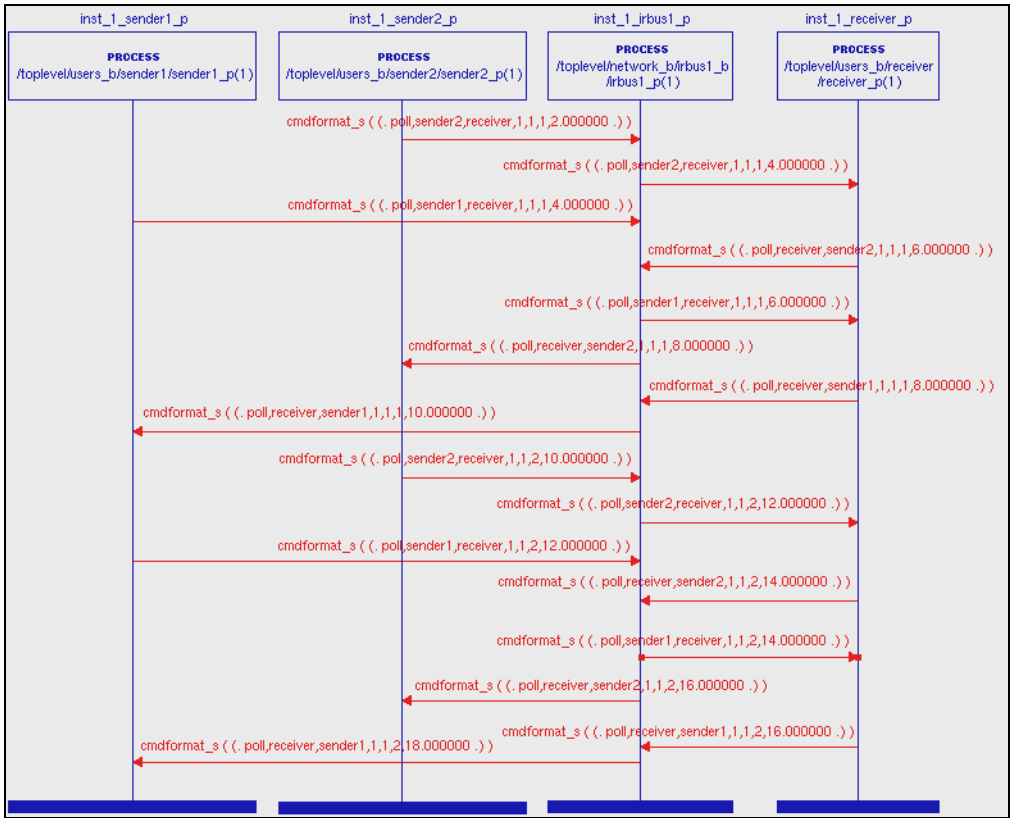

Fig. 5: MSC for Configuration 2 Using One Bus Only

5. CONCLUSIONS

It has been shown that performance extensions are possible in SDL92 which allow a fast evaluation of different system topologies and analysis of a system's behaviour under realistic conditions. Moreover, resources, especially time, is consumed during execution of a state transition.

A specific approach for distribution of SDL signals was applied which solves the problem related to mapping of the hierarchy of SDL proceses onto resources. This approach was originally defined for flexible and dynamic reconfiguration of a fault-tolerant system at run-time and to save time in case of updates of the SDL model. But it serves well to solve the problems related to performance extension of SDL.

In case SDL user's do not want to apply this specific modelling approach a change of the language may be needed and/or corresponding support by the tools.

6. REFERENCES

[1a] HRDMS (Highly Reliable DMS and Simulation), ESTEC contract no. 9882/92/NL/JG(SC), Final Report, 1994, Noordwijk, The Netherlands

[1b] R.Gerlich, N.Schäfer, A.Schäferhoff: Early Validation of DMS Design by a Reusable Environment, EUROSPACE On-Board Data Management Symposium on "Technology and Applications for Space Data Management Systems", January 25-27, 1994, Rome, Italy

[2] SES/workbench, Scientific and Engineering Software Inc., Building A, 4301 Westbank Drive, Austin, Texas, 78746-6564, USA

[3a] OMBSIM (On-Board Management System Behavioural Simulation, ESTEC contract no. 10430/93/NL/FM(SC), Final Report Nov. 1995, Noordwijk, The Netherlands

[3b] R.Gerlich, Ch.Schaffer, Y.Tanurhan, V.Debus: EaSyVaDe / EaSySim: "Early System Validation of Design by Behavioural Simulation", ESTEC 3rd Workshop on "Simulators for European Space Programmes", November 15-17, 1994, Noordwijk, The Netherlands

[3c] R.Gerlich, Th. Stingl, Ch. Schaffer, F. Teston, G. Martinelli, Y. Tanurhan: Use of an Extended SDL Environment for Specification and Design of On-Board Operations, Systems Engineering Workshop, November 28-30, 1995, ESTEC, Noordwijk, The Netherlands

[4a] GEODE SDL-Tool, V2.1, Verilog, 150 rue Vauquelin, F-31081 Toulouse Cedex, France

[4b] ObjectGEODE SDL-Tool, V3.01(1.01), Verilog, 150 rue Vauquelin, F-31081 Toulouse Cedex, France

[5] DDV (DMS Design Validation), ESTEC contract no. 9558/91/NL/JG(SC), Final Report Dec. 1996, Noordwijk, The Netherlands

[6a] EaSySim II: The Enhanced Environment for System Validation, R. Gerlich BSSE, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[6b] EaSySim II User's Manual, R. Gerlich, BSSE, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany