Tuning Development of Distributed Real-Time Systems with SDL and MSC: Current Experience and Future Issues

R. Gerlich

BSSE System and Software Engineering, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany
Phone: +49/7545/91.12.58, Fax: +49/7545/91.12.40, e-mail: gerlich@t-online.de

Keywords: System verification and validation, interdependence between behaviour and performance, system architecture, performance modelling, performance evaluation, exhaustive simulation, automated code generation, SDL, MSC

Abstract:

SDL provides powerful capabilities for verification[1] and validation[2] of a system's behaviour and for automated code generation. This allows to perform system validation at a higher level of abstraction and earlier in the development life cycle. However, one needs to be carefully to really gain advantage of such capabilities, especially when applying SDL to a broader class of applications which may be called "decision-making, distributed systems". Firstly, state explosion may prevent to get any benefit from exhaustive simulation or much effort is required to limit the number of states thereby loosing most of the advantages of automated testing. Secondly, the current means of SDL and of SDL tools may not be sufficient to identify all bugs of a system's specification and design. Even when exhaustive simulation does not report any error, the system may not run correctly on the target, or vice versa, the optimum practical solution may be rejected as erroneous. This paper will analyse the situation, provide with a solution for tuning of system development which is based on an additional layer, called EaSySimII, on top of the ObjectGEODE tool, and will identify future issues.

1. Introduction

Compared with other languages a major advantage of SDL and MSC is their capability to provide on an abstract and formal level the means for definition of (a) information exchange between a system's components by MSC's (Message Sequence Charts) and (b) a system's behaviour by FSM's (Finite State Machines). This allows to automate verification of information exchange and of behaviour. In consequence, verification can be performed at a higher level of abstraction and earlier in the development life cycle. This helps to save costs and to reduce risks.

---

[1] "Verification" means to check if the system is built right.

[2] "Validation" means to confirm that the right system is built. Hence, validation refers to all system properties, while verification may only refer to some properties.

Due to these advantages SDL was selected during the project OMBSIM [1] which was execeuted for the European Space Agency ESA/ESTEC in order to define an alternative system life cycle [2,3,4,5]. As it was already known by previous activities [6] that consideration of performance impacts is a "must" for system validation the SDL tool ObjectGEODE [7] was complemented by performance analysis and simulation capabilities provided by the SES/workbench tool [8]. The resulting tool environment was called "EaSySim" (Early System Validation Simulation" environment). This environment has been improved significantly in mean time by BSSE and a completely new implementation "EaSySim II" [9] is now available which overcomes all the weakness of the first environment and provides new capabilities for system validation. It is based on ObjectGEODE, the actual version of GEODE used, and provides the performance simulation capabilities by SDL means and additional support functions which are implemented as operators in C. EaSySim II still provides access to SES/workbench, but also to other tools and (user) software in a transparent manner.

A number of activities have been executed since 1992 when the first ESTEC project HRDMS [10] on system validation started, which especially concentrated on performance matters. Since 1992 the development approach has been continuously improved and the productivity of development steadily increased, mainly based on the powerful capabilities of SDL, but also by proper organisation of the development steps [5].

This experience now allows to give recommendations for tuning of system development, how to obtain correct verification results or how to increase system quality. Although extending the verification process to performance properties, verification becomes much simpler because notion of time and shared resources introduce an ordering scheme which reduces the number of system states. This helps to avoid state explosion and to master exhaustive simulation in such cases when it is not possible otherwise. Moreover, it was recognised that the number of system states may be taken as a measure for system quality indicating how well defined a system really is.

To conclude: when extending the verification process towards performance matters this does not only lead to more reliable results, but it also simplifies system validation and hence allows to tackle more complex systems.

While SDL is more addressing an abstract, mathematical system, the EaSySim II environment concentrates on a real, physical system and its properties. This extension of the scope is needed when dealing with a more general class of distributed systems with SDL.

2. The Impact of Performance on System Validation

This section identifies the risks which arise when the SDL capabilities for behavioural verification are applied to a larger class of distributed systems without considering all relevant system aspects such as performance. Otherwise errors in a system may remain hidden and may cause sporadic or permanent faults during later system operation.

2.1 An Extended Application Domain Requires An Extended Scope of Validation

In the past, SDL was mainly applied to telecommunication applications. Such applications form a sub-class of "distributed applications" which often can be characterised as a sequence of "one-point-to-one-point" communications. Many such communications may occur at the same time and they may compete for resources. But they do not disturb each other during execution of a protocol sequence, because there is no signal exchange between them.

Usually, no such communication request has a higher priority than any other, and the next action will not start before the previous action has been completed. This makes it reasonable to ignore time and to assume that a state transition does not take any significant time at all. Performance aspects may be important but they only impact the consumption of resources and the duration of activities, and not the system behaviour.

However when taking into account a more general class of distributed systems for which "n-point-to-m-point" connections ("anybody can communicate with anybody else at any time") are allowed, time plays a more important role: performance of the (real) distributed system may impact the validation process and the results may not match the (physical) system architecture.

The reason is: signals may not propagate with the same (average) transmission rate through a network. When they take different paths (1) the transmission rate may depend on the path, (2) the number of processing steps may be different. If transmission rate is assumed to be infinite (zero propagation time) this dependency is not recognised. Also, in case processing time is ignored the number of processing steps do not impact the final arrival time of a signal. But consumption of time makes the difference between "ideal" and physical systems. And this difference makes validation harder in case of distributed systems with arbitrary communications.

Due to zero-propagation time signals arrive in a order in a process queue which may be different from the order in the physical system. Consequently, the real sequence on the physical architecture may never occur during simulation. Hence, successful verification by simulation with SDL and SDL tools does not necessarily mean that the system will work correctly on teh architecture because the impact by performance is not known: whether it invalidates the result or not.

As it is shown in section 2.2 even in case of a synchronous master-slave protocol which is run on two uni-directional lines, time consumption of transmission and data processing cannot be neglected.

Hence, in order to obtain results which are compliant with the real, physical system we need to consider performance matters already during system validation by simulation.

Several activities are known which introduce notion of time in SDL [11,12,13,14], but they only concentrate on aspects like channel delays and response times or violation of performance constraints, but do not consider that time may impact system behaviour. They analyse time delays e.g. in the queues of application processes although the signals may not have to wait there, but e.g. in the network or the on the processors. Behaviour remains the same when time consumption in physical resources added.

As SDL tools already provide the capability of exhaustive simulation and support a priori distributed systems it is possible to extend tool capabilities such that a more general class of distributed systems is covered. The EaSySim II environment provides such enhanced capabilities on top of the ObjectGEODE tool: consequently, a user can exploit the performance of a certain system architecture and can validate such a distributed system under realistic conditions.

2.2 A Protocol Example: Succeeding with Validation of an Erroneous System

The protocol shown by Fig. 2a has been used during the project HRDMS [10] and in mean time it turned out that it is a very good example to demonstrate (a) violation of validated behaviour when introducing timing aspects, (b) the weakness of validation of system properties under artificial (simplified) operational conditions, (c) the interaction between system tuning and correctness of results of exhaustive simulation, (d) the significant reduction of system states when introducing performance aspects into exhaustive simulation.

The protocol is completely deterministic and synchronous from a logical point of view, which is the reason that people believe that performance matters can really be ignored for its validation. When taking exactly the sequence of signals as shown by Fig. 2a the protocol will *never* run free of errors on the system architecture of Fig. 2b. And this is the good point of its determinism. When starting to remove the bug (by varying the sequence of the signals and playing with timing) the protocol may loose its determinism due to performance and environmental impacts and it becomes even harder to identify the bug.

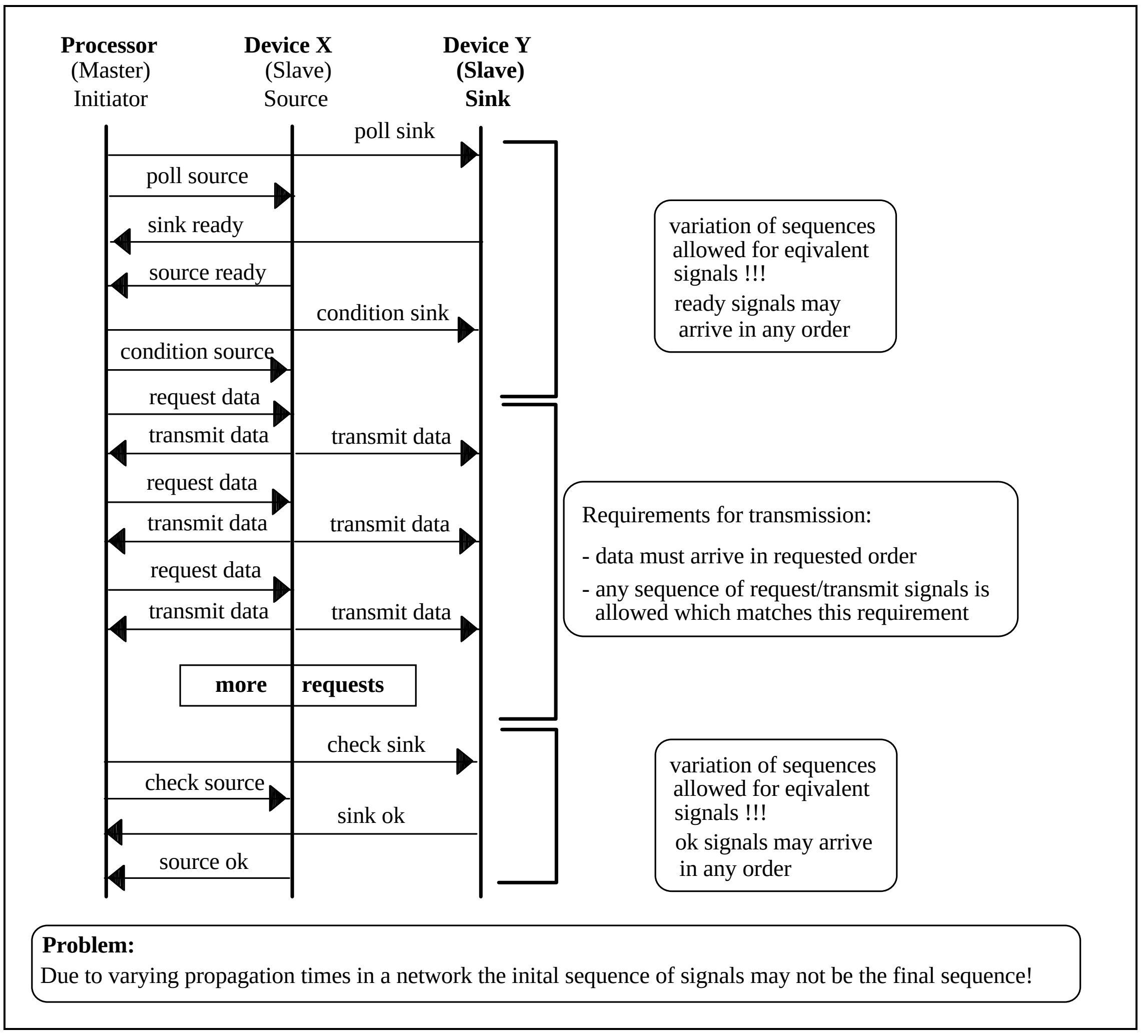

Processor
(Master)
Initiator
Device X
(Slave)
Source
Device Y
(Slave)
Sink
poll sink
poll source
sink ready
source ready
condition sink
condition source
request data
transmit data
transmit data
request data
transmit data
transmit data
request data
transmit data
transmit data
more requests
check sink
check source
sink ok
source ok
variation of sequences allowed for eqivalent signals !!!
ready signals may arrive in any order
Requirements for transmission:
- data must arrive in requested order
- any sequence of request/transmit signals is allowed which matches this requirement
variation of sequences allowed for eqivalent signals !!!
ok signals may arrive in any order
Problem:
Due to varying propagation times in a network the inital sequence of signals may not be the final sequence!

Fig. 2a: A Sample Protocol

From a logical point of view this protocol seems to be free of conflicts. However, conflicts arise from the given architecture: one needs to execute it under real conditions to identify the conflict. In consequence, one can never be sure that no error will occur in the real environment when performance aspects are ignored during system validation by simulation.

The goal of the protocol shown by Fig. 2a is to exchange data between the source and sink devices. This transfer is initiated and supervised by the processor (master) for each data request. As it is a typical "master-slave" protocol it should be free of conflicts. But this is not true.

The processor polls the source and sink devices whether they can provide data or whether they are ready to accept data, respectively. If both respond with 'ready' the processor conditions the devices and requests data, cycle for cycle. At the end of data transmission the processor checks the source and sink devices whether all data have been transmitted correctly. And this final checking sequence causes the problem (Fig. 2c)

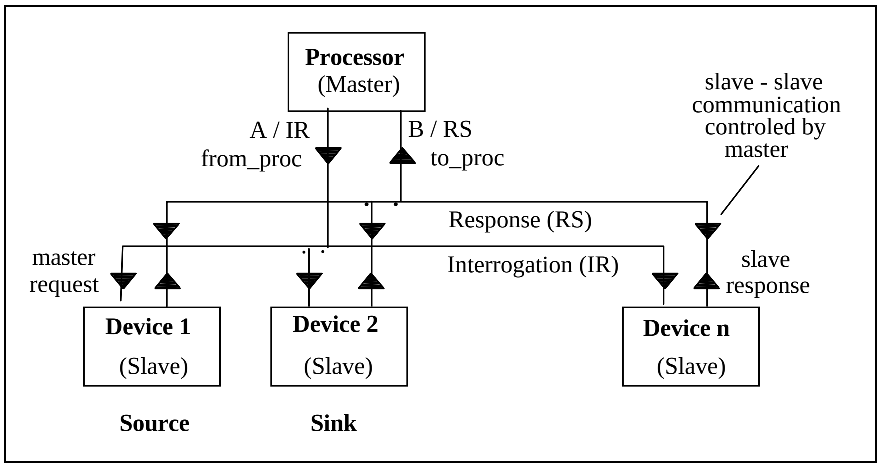


Fig. 2b: System Architecture

When ignoring transmission time all transmitted data arrive before the final checks. This is different when shared resources like bus or processor cycles are consumed on a real architecture like the one shown by Fig. 2b. Then the signal sent from the source device to the sink device is synchronised with the bus cycles. This causes the last data sent from the source to the sink to arrive later at the sink device than the check signal issued by the processor directly to the sink device. This confirms (a).

The reason is that data coming from the source are delayed by two bus cycles before they arrive at the sink. During the first bus cycle the request signal is processed by the source device and the data are written to the output registers. During the second cycle the data is transmitted to the sink device. However, the check signal is directly sent from the processor to the sink device and is therefore one cycle faster.

To solve this problem an idea could be to change the sequence of the check signals (check sink, then check source, see Fig. 2a) and to add a further cycle between the last data request and transmission of the check signal for the device, because this will delay the check sink signal by two cycles which are needed according to Fig. 2c.

However, as mentioned already above this will make life even harder. For a certain test the protocol may run correctly, but not in all cases. If several transmissions are initiated on the processor, the delay of the check signal will provide an empty bus slot which may be used by another protocol sequence running in parallel. If this sequence addresses the same sink device, the sink will again identify an error due to incompatibility of its state with the incoming data (in best case) or it will accept the wrong data (in worst case). This confirms hypotheses (b) and (c). In consequence, exhaustive simulation executed with filter conditions will deliver wrong results because such side effects may not be detected due to filtering of side effects.

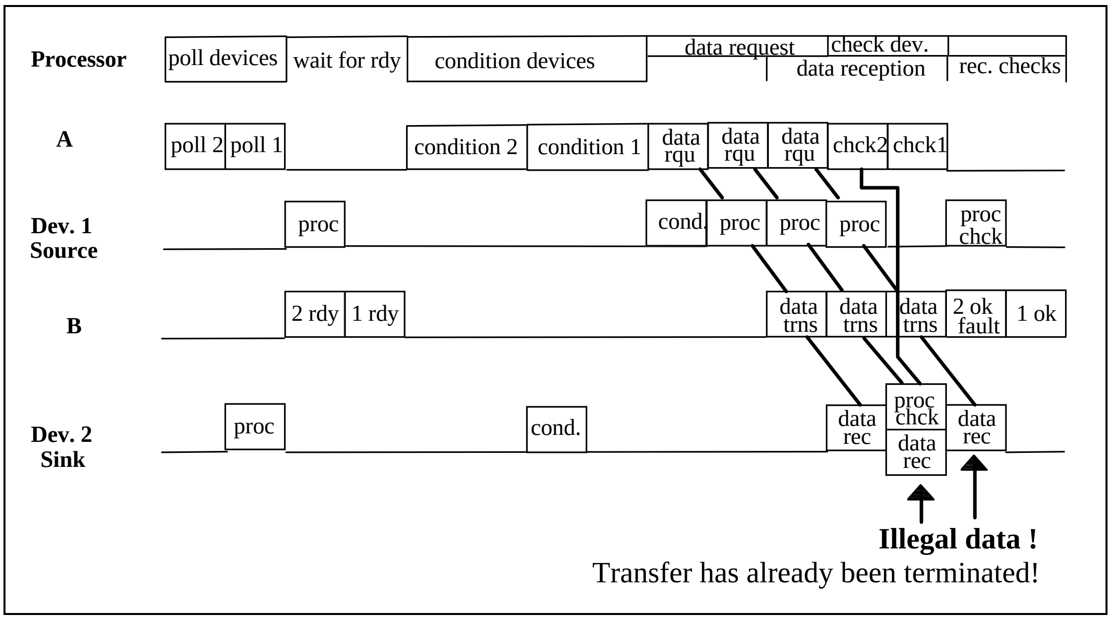


Fig. 2c: Timing Aspects

Hence, the behaviour of the protocol changes from "completely deterministic" to "non-deterministic" due to performance impacts and potential side effects from other data transfers. Only the two constraints (1) "two cycles delay between request of last data and transmission of the check signal for the sink device" and (2) "no further signal must be transferred to the sink device between the last data request and transmission of the check signal" solves the problem and the protocol can always be executed completely deterministic and correct.

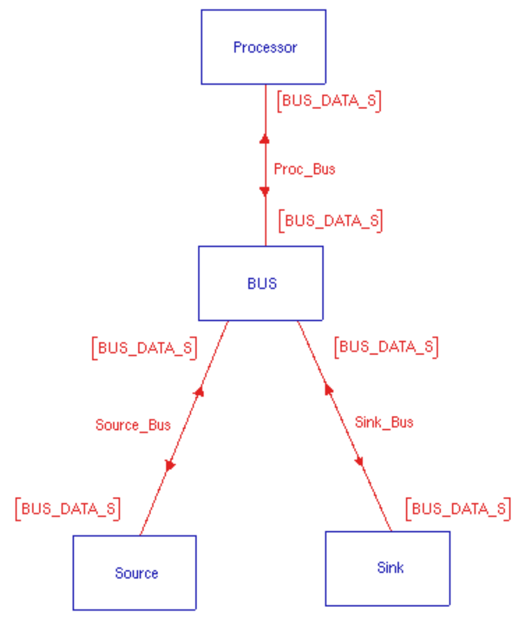


Fig. 3: Principal System Structure

The protocol of Figs. 2a-2c has been implemented in SDL for several implementations of the bus as shown by Figs. 3 - 7. The principal system structure (Fig. 3) has always been kept and mainly the bus implementation was changed. For optimisation of system performance the timing of data requests in the processor was also varied, e.g. the next request may be sent before the data of the previous request arrives.

If processing time and transmission time are ignored system performance does not matter at all. So the simplest solution is to send the next data request (or the final check) only when the requested data have been received. This ensures execution of the protocol free of errors as long as no other data transfer is running in parallel which may use the empty bus slots and may cause a state mismatch in the source or sink devices. However, when introducing time consumption this way of protocol processing becomes very inefficient because three cycles are always needed per data request yielding a bus utilisation of only 33% at most.

To achieve a higher bus utilisation, all signals following the ready signals could be issued immediately by the processor because it should be a matter of the bus interface to queue all such signals. But with SDL one cannot always proceed in this manner. If transmitting all the signals by a single burst they will be stored in a SDL queue and selected from this queue depending on which mode is used for simulation in a tool. In case "random simulation mode" is applied the sequence of the signals will be changed and the SDL simulator will detect errors in the protocol, although in practice they will not occur.

Knowing about this problem the bus clock can be used to transmit request by request synchronously from the processor not waiting for a response. Then bus utilisation remains as efficient as in case of a burst.

Such processing also reduces the number of system (SDL) states: the queue lengths are reduced to one element only at a certain time and this simplifies significantly exhaustive simulation because instead of n! mutations only 1! are considered by the tool.

Figs. 4 - 7 (Figures 5b-7 and the tables follow on the next pages) show several representations of the bus, changing from a very simple bus representation to the real bus architecture consisting of two uni-directional bus lines and a bus clock. Table 1a gives the results of verification by exhaustive simulation for the simple bus models (Figs. 4, 5a-c) which do not take into account timing aspects. Table 1b shows the results for the full bus architecture based on synchronous transfer of the data related to Figs. 6a-c and 7.

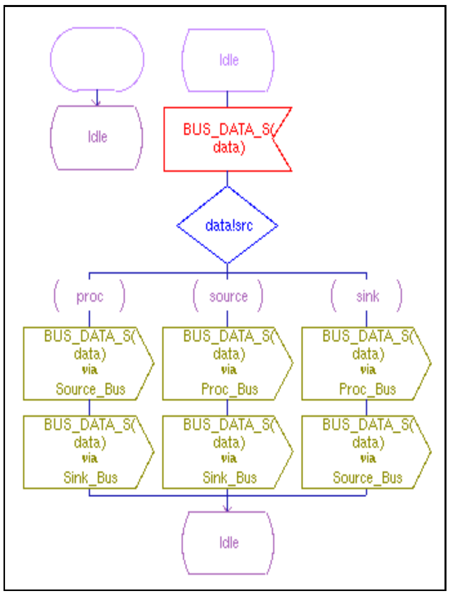

Idle
Idle
BUS_DATA_S(data)
data!src
proc
source
sink
BUS_DATA_S(data) via Source_Bus
BUS_DATA_S(data) via Proc_Bus
BUS_DATA_S(data) via Proc_Bus
BUS_DATA_S(data) via Sink_Bus
BUS_DATA_S(data) via Sink_Bus
BUS_DATA_S(data) via Source_Bus
Idle

Fig. 4: Simple Bi-Directional Bus with Broadcasting

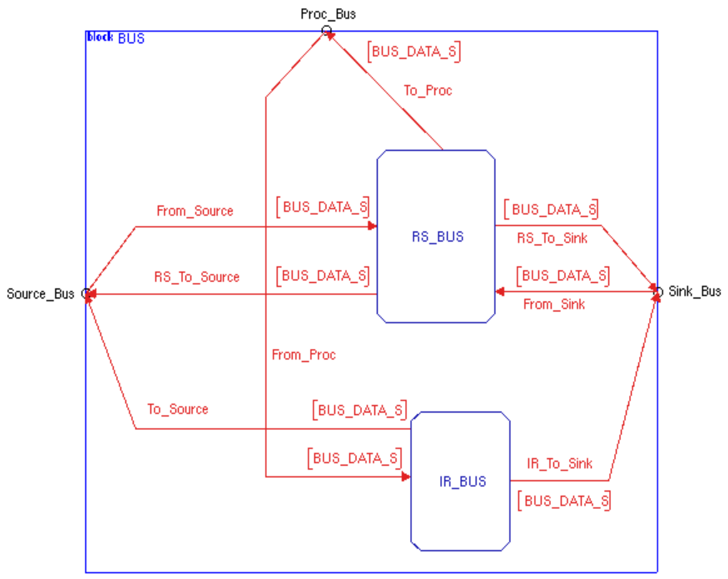


Fig. 5a: Bus with Two Uni-Directional Bus Lines

Two cases have been investigated for the full bus architecture of Fig. 6: (1) the period of the bus clock has been set to a non-zero value (the expected case) or (1) it has been set to a zero value in order to ignore transmission time.

In case of Fig. 7 an equivalent "functional" bus architecture without clock is used in order to get a comparison between non-zero bus time slots and equivalent transmission steps with zero transmission time. This means that in case of Fig. 7 a signal is immediately transferred after its reception without waiting for the bus clock.

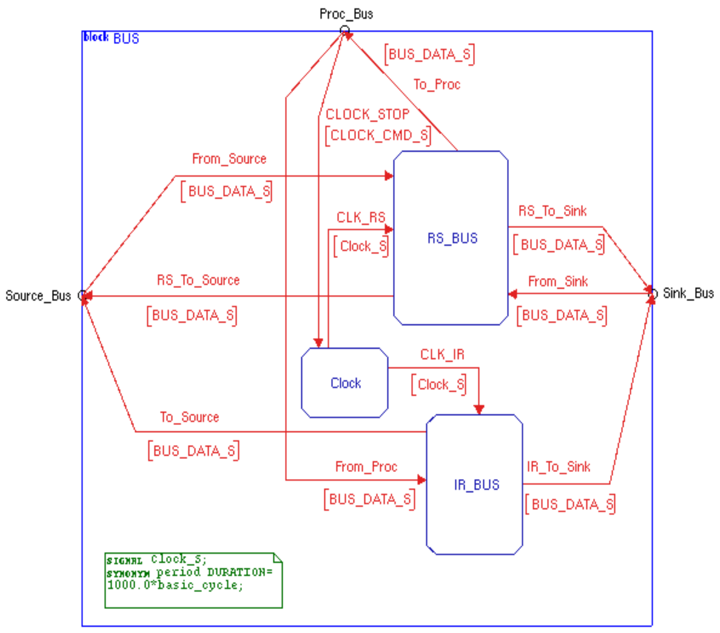
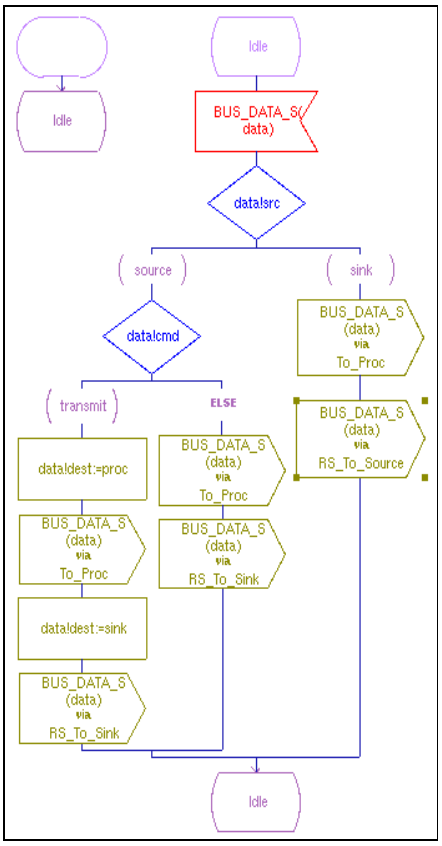
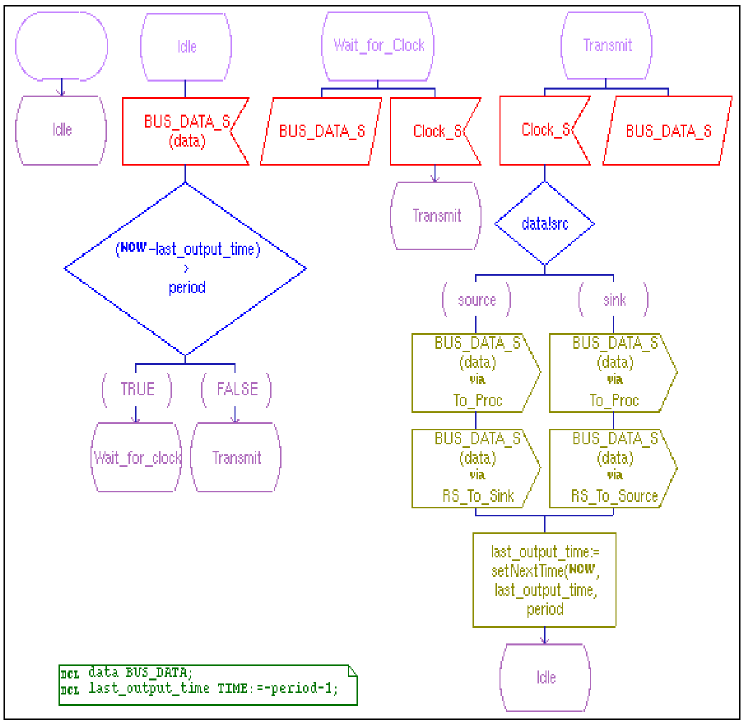


Fig. 6a: Fully Representative Bus



Idle
Idle
BUS_DATA_S(data)
data!src
source
sink
data!cmd
BUS_DATA_S (data) via To_Proc
transmit
ELSE
BUS_DATA_S (data) via RS_To_Source
data!dest:=proc
BUS_DATA_S (data) via To_Proc
BUS_DATA_S (data) via To_Proc
BUS_DATA_S (data) via RS_To_Sink
data!dest:=sink
BUS_DATA_S (data) via RS_To_Sink
Idle

Fig. 5b: Device-to-Processor Line (RS bus)



Idle
Wait_for_Clock
Transmit
Idle
BUS_DATA_S (data)
BUS_DATA_S
Clock_S
Clock_S
BUS_DATA_S
Transmit
data!src
(NOW-last_output_time) > period
source
sink
BUS_DATA_S (data) via To_Proc
BUS_DATA_S (data) via To_Proc
TRUE
FALSE
BUS_DATA_S (data) via RS_To_Sink
BUS_DATA_S (data) via RS_To_Source
Wait_for_clock
Transmit
last_output_time:= setNextTime(NOW, last_output_time, period
Idle
DCL data BUS_DATA;
DCL last_output_time TIME:=-period-1;

Fig. 6b: Uni-Directional Bus Line with Clock (Device-to-Processor Line)

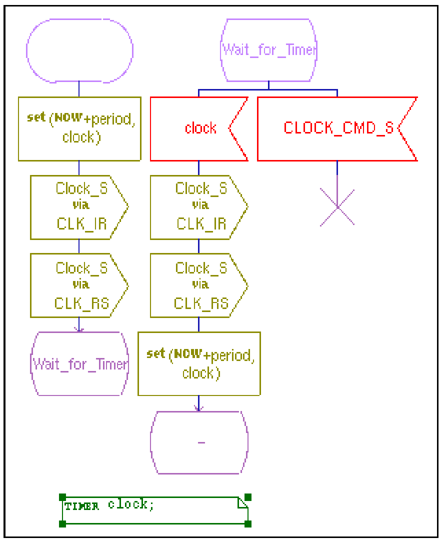
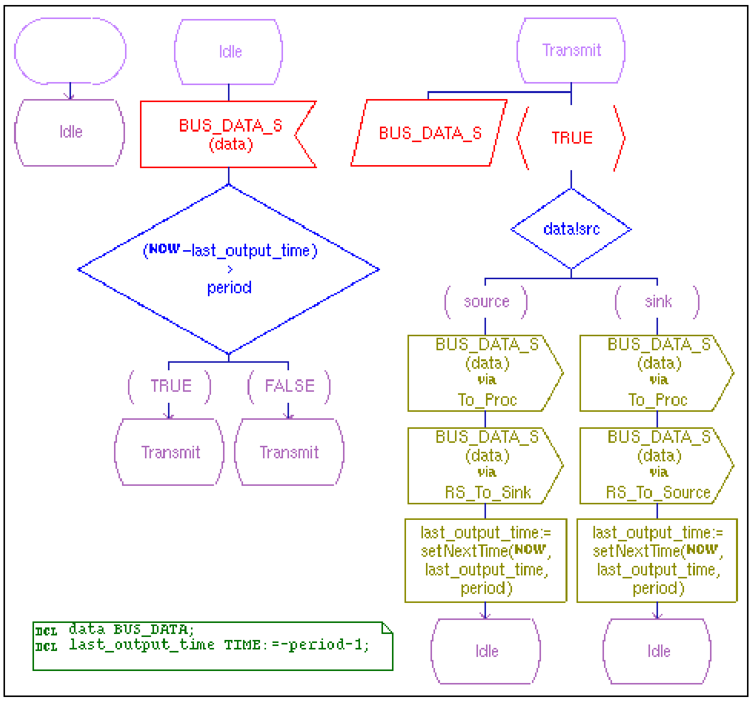


Fig. 6c: Bus Clock

Fig. 4 shows the simplest implementation of the bus: the bus just distributes the received signal to all connected devices (the sending device excepted) and acts as a bi-directional bus with one bus line. The bus shown by Figs. 5a - 5b introduce two uni-directional bus lines. So the bus of Fig. 5 already represents the real architecture but it still ignores time consumption.



Idle
Transmit
Idle
BUS_DATA_S (data)
BUS_DATA_S
TRUE
(NOW-last_output_time) > period
data!src
source
sink
TRUE
FALSE
Transmit
Transmit
BUS_DATA_S (data) via To_Proc
BUS_DATA_S (data) via To_Proc
BUS_DATA_S (data) via RS_To_Sink
BUS_DATA_S (data) via RS_To_Source
last_output_time:= setNextTime(NOW, last_output_time, period)
last_output_time:= setNextTime(NOW, last_output_time, period)
DCL data BUS_DATA;
DCL last_output_time TIME:=-period-1;
Idle
Idle

Fig. 7: Uni-Directional Bus Line (w/o clock)
(Device-to-Processor Line)

In the next step shown by Figs. 6a - 6c a bus clock is introduced driving both bus lines. This bus is a fully representative model of the real bus w.r.t. the required degree of detail. The IR bus (bus A in Fig. 2b) transfers the signals from the processor to the devices and the RS bus (bus B in Fig. 2b) takes the opposite direction. Fig. 7 shows a representation which is functionally equivalent to the one of Fig. 6b when the bus period is set to zero, but gives different simulation results.

Table 1a shows the results of simulation for three simple bus types. No error is detected, but the system has low performance. The low utilisation of the bus will be recognised later when the software is executed on the real hardware. However, then it is very expensive to change the processing algorithm for the protocol.

Although system representation for test 3 is more complex due to the two bus lines, less system states were generated. This confirms that more complex systems do not necessarily have a higher number of system states. It is a matter of possible paths through the system, and obviously the bus of Fig. 5 is more accurately defined than the one of Fig. 4. This tuning aspect is discussed in more detail in section 3. Table 1b gives the results for six different timing approaches.

In case of test 4 the same algorithm as for tests 1 - 3 was used. As the next data request is only issued when the response for the previous request arrives no error was observed, but bus utilisation is still poor. It is surprising that the number of system states is again much lower compared to tests 2 and 3 although the bus is more complex. The reduction of system states occurs because the clock synchronises the processing steps and ambiguities in system behaviour are removed. This confirms hypothesis (d) given at the beginning of this section.

A burst of data requests is issued for test 5. The SDL specific queueing mechanism caused an error during random and exhaustive simulation. Although the algorithm is correct (it includes the required delay between data requests and checks), SDL simulation will reject this algorithm because the way the SDL tool is simulating the system is the same as in real world.

Test 6 adjusted the algorithm to the needs of the SDL tool and transferred the data requests synchronously with the bus clock. The missing delay between data requests and checks was detected. For test 7 the bus period was set to zero. This increased the number of system states and exhaustive simulation did not terminate because computer resources were exhausted.

#	Functionality	Ref. to Fig.	Bus Period	# Bus Cycles	States	Trans.	# Errors Random Sim.	# Errors Exhaust. Sim.	Bus Util. %	Correct Result
1	simple bus next data request or check when previous data received 3 data requests	-	n/a	n/a	184	329	0	0	33	yes but low performance protection needed against side effects
2	simple bus with broadcasting next data request or check when previous data received 3 data requests	4	n/a	n/a	830	1910	0	0	33	yes but low performance protection needed against side effects
3	simple bus with broadcasting and two uni-directional lines next data request or check when previous data received 3 data requests	5	n/a	n/a	617	1320	0	0	33	yes but low performance protection needed against side effects

Table 1a: Results of Protocol Validation for Functional Bus Representations

Exhaustive simulation was aborted and the error in protocol processing was not yet identified.

Test 8 repeated test 7 with the equivalent bus implementation of Fig. 7 which does not use the bus clock. Again, the number of system states is significantly higher when performance aspects are ignored, bit exhaustive simulation terminates. The need for the additional delay was not detected. Test 9 runs the correct algorithm under real timing conditions and the correct result is obtained.

3. Tuning of System Development

In the previous chapter the risks and chances for system development were discussed: (1) if not all system properties (like performacne) are subject of verification and validation the system will not run correctly in the real environment although no errors have been identified during the verification and validation process, (2) if inappropriate verification and validation procedures are applied correct implementations may be rejected, (3) consideration of performance aspects simplifies the verification and validation steps because a real, physical system behaves much simpler than an abstract, mathematical system.

Although SDL provides already powerful capabilities for verification and validation of distributed systems, means are missing which allow for detailed and representative modelling of timing. In consequence, SDL has to be enhanced such that the needs of verification and validation of a real (distributed) system will be met.

According to above conclusions (1) - (3) capabilities for performance analysis and simulation and the scheduling policies like priority-based, pre-emptive scheduling need to be added. EaSySim II does it on top of ObjectGEODE.

#	Functionality	Ref. to Fig.	Bus Period	# Bus Cycles	States	Trans.	# Errors Random Sim.	# Errors Exhaust. Sim.	Bus Util. %	Correct Result
4	real bus architecture next data request or check when previous data received 3 data requests	6	non-zero	24	470	990	0	0	33	yes but low performance protection needed against side effects
5	real bus architecture burst of 3 data requests, delay before checks	6	non-zero	n/a (due to error)	715	1556	1	1	→ 100	no data requests are not processed in the right order
6	real bus architecture synchronous transfer of requests and checks, no delay between last request and checks	6	non-zero	n/a (due to error)	558	1153	1	1	→ 100	yes missing delay before checks caused an error
7	same as above	6	zero	n/a	2097150	8018550	0	0	→ 100	no missing delay not identified exh. sim. does not terminate
8	same as above but with equivalent architecture, no clock	7	n/a	n/a	18130	56760	0	0	→ 100	no missing delay not identified
9	real bus architecture burst of 3 data requests, 2 bus cycles delay between last data request and first check (check sink)	6	non-zero	17					→ 100	yes protection needed against side effects

Tab. 1b: Results of Protocol Validation for Fully Representative Bus

But tuning of system development can also be done in view of quality and feasibility. It was recognised that the number of system states provides a feedback on ambiguities left open in system definition: if undesired paths through a system's FSM's are removed then the number of system states decreases. Reduction of number of system states by more accurate implementation and by consideration of performance matters allows to master verification and validation of systems for which exhaustive simulation would not terminate otherwise.

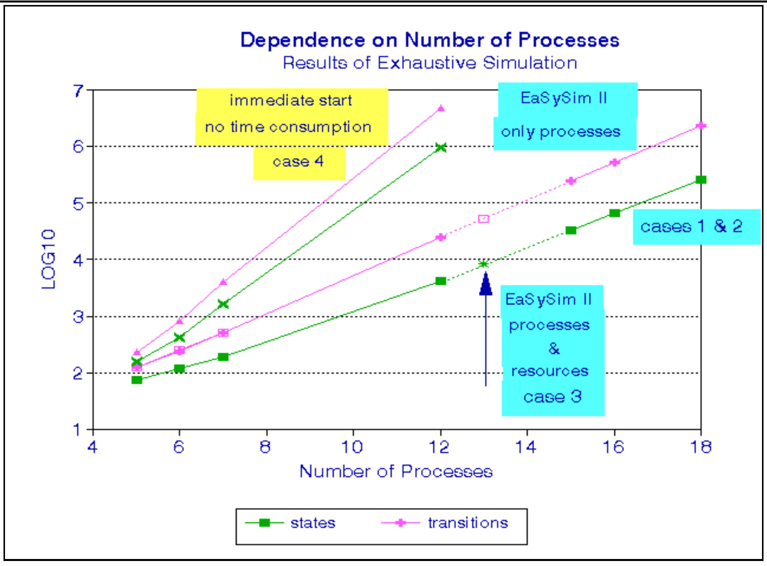


Fig. 8: Dependence of System States on Number of Processes

In fact, state explosion is a critical point of exhaustive simulation. It does not help at all to have such a powerful capability, but to fail for practical cases.

As was mentioned in section 2 filtering is not a good idea to make exhaustive simulation feasible because it may exclude critical cases, especially interaction with parallel activities. To take real advantage of exhaustive simulation one needs to reduce system states by other measures.

Fig. 8 shows the dependency of number of system states w.r.t. to number of processes for different implementations of an application. For case 4 performance aspects have not been considered. In this case systems of up to 15 processes may be validated by exhaustive simulation. It is believed that EaSySim II represents already an approach which creates a minimum number of system states (cases 1-3). But even in case of EaSySim II the number of processes are limited to about 20 - 25 processes[3]. The feasibility or unfeasibility of exhaustive simulation is also impacted by the simulation time which is about 8 hours for case 4 and 12 processes and about 1 hour for cases 1-3 and 18 processes.

3.1 Means for Tuning Verification and Validation

For tuning of verification and validation with SDL four different measures have been identified which reduce the number of system states and increase quality of a system:

1. use of a subset of SDL,
   e.g. avoid to distribute timers all over the SDL model, to use VIEW/REVEAL and EXPORT/IMPORT because such constructs may generate a lot of background traffic and a number of system states you can't control
2. enhancement of SDL tools by means which are adequate for the system under development
   EaSySim II provides optimised time managment and supports scheduling policies
3. consideration of performance aspects as described in the previous sections,
4. unambigous (and error free) definition of a system's behaviour by FSM's by evaluating the feedback from exhaustive simulation.

[3] In section 3.2 the means will be described which are provided by EaSySim II to escape from this limitation:transparent partitioning of a SDL system.

A reduction by a about four orders of magnitude was achieved for a sample application. The reduction at the beginning was obtained by means (1) and (2), while means (3) and (4) contributed to "fine tuning" at the end.

This experience is reflected by the EaSySim II Δ-approach© which is shown by Fig. 9.

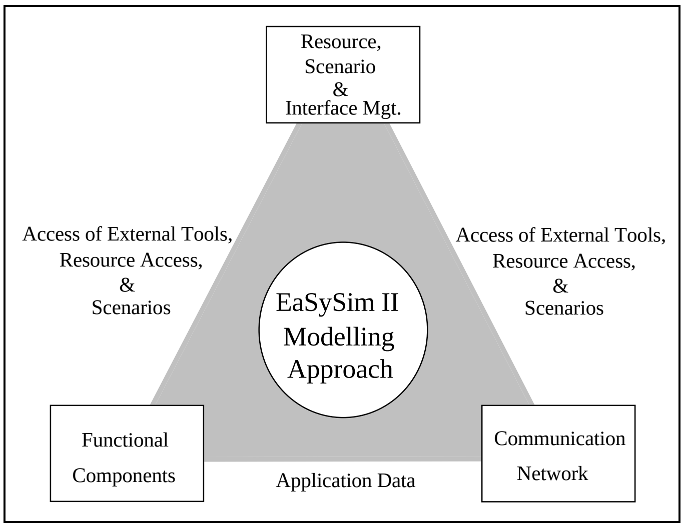

Resource, Scenario & Interface Mgt.
Access of External Tools, Resource Access, & Scenarios
EaSySim II Modelling Approach
Access of External Tools, Resource Access, & Scenarios
Functional Components
Application Data
Communication Network

Fig. 9: The EaSySim II Δ-Approach©

EaSySim II extends ObjectGEODE. It provides support functions which help to organise a SDL system such that the number of system states can be reduced and resources can be consumed. Also, by this organisation a system can easily be distributed in a transparent manner. Moreover, this mechanism allows to communicate with other simulation tools such as SES/workbench or other software like window managers, database systems and other EaSySim II environments on the same or on remote processors.

EaSySim II divides a system into three principal parts: (1) a management part which drives the tests and system operation and provides an interface to the outside world, (2) a "functional" part which covers functional, behavioural and performance aspects of the application, and (3) a "communication" part which represents the network of the application through which the functional components are communicating with each other under consideration of performance.

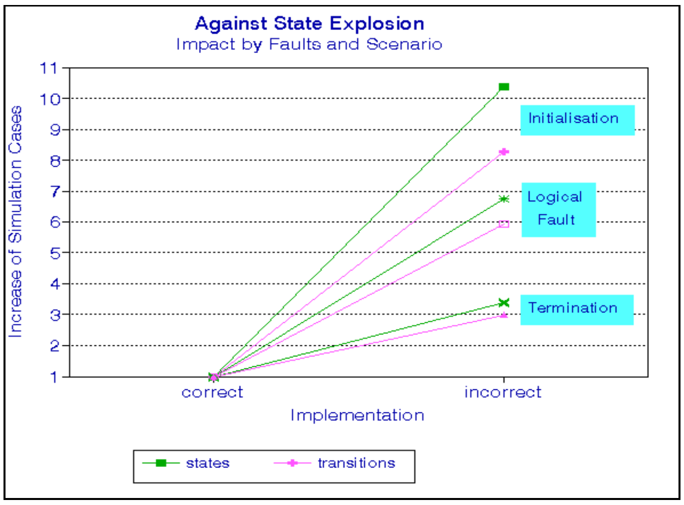


Fig. 10: Impact by Logical Faults and Performance Aspects

It is also possible to change the communication topology at run-time. This allows to perform redundancy switching for a fault-tolerant system after occurence of a fault. Hence the Fault Identification and Recovery Procedure (FDIR) can be subject of exhaustive simulation.

Fig. 10 demonstrates the impacts of (3) and (4) on number of system states during exhaustive simulation. A real, physical system is activated by switching on its components one after the other. In SDL all start transitions are executed at time "T_0". This causes a lot of additional system states as is shown by Fig. 10. The same happens when terminating a system immediately. EaSySim II provides the means to activate or de-activate a SDL system like a physical system and this reduces significantly the number of system states. The mechanisms provided by EaSySim II act like a capacitor connected across the poles of an electrical switch.

Moreover, it was observed that number of system states is also reduced when a logical fault is removed like an illegal path through the FSM's, a deadlock or an exception.

3.2 Means to Tune Efficiency of System Development

It was already pointed out that SDL provides powerful capabilities to make system development more efficient e.g. by means of formal description of behaviour, exhaustive simulation and automated code generation. Such capabilities have to be complemented when applying SDL to a broader class of distributed systems as discussed above.

Early transition to the target system will confirm the simulation results and hence is recommended. Fig. 11 shows the steps through such a life cycle and how the implementation approaches more and more the final system in the host and target environment.

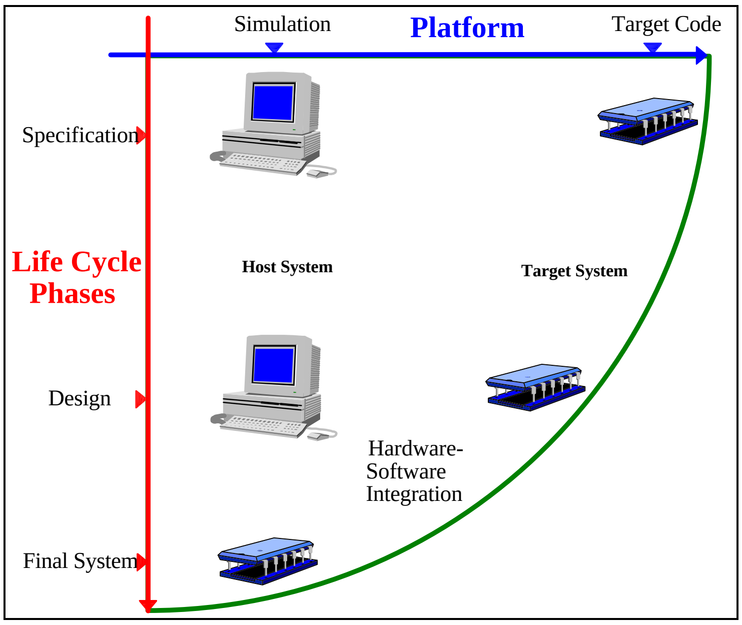


Fig. 11: Two-Dimensional Life Cycle

EaSySim II provides such means which simplify the transition between host and target environments, between simulation and generated code. It provides an automated procedure for code generation from SDL code including automated creation of the libraries speeding up the whole process to 15 mintes only. Moreover, EaSySim II tools remove performance instrumentation automatically.

Not all details of an implementation do contribute to a system's behaviour. To avoid state explosion one either needs to follow above suggestions or one also may export (or needs to export) some details to other languages such as C or Ada[4].

This applies to the set of (large) data structures, which may also be divided into a SDL part if relevant for behaviour and a C/Ada part if not relevant. If e.g. a variable x defined in SDL is only needed to make a decision like "x>0" then (the real variable) x should not be declared in SDL but the boolean SDL operator *testx* could be used which imports the result of the comparison "x>0" from C/Ada to SDL. In this case not all values of x are made visible to the exhaustive simulator but only such values which impact behaviour. This yields a significant reduction of SDL system space.

EaSySim II supports transparent partition of a system (not impacting its architecture): system processes my be included in more than one EaSySim II environment. So the scope of exhaustive simulation can be limited to some parts only, and exhaustive simulation can sequentially be applied to all system parts one after the other.

[4] The author has already successfully implemented SDL operators in Ada [15].

4. Future Issues

Currently, exhaustive simulation concentrates more on the combinatorial exploration of state space. When introducing performance, shared resources and more sophisticated scheduling policies multiple occurence of events nearly disappears and hence the need for combinatorial exploration becomes less important. What needs more detailed investigation is the timed execution of events and its consideration by exhaustive simulation, preferably in a more formal manner. When considering performance, interference between parallel system activities becomes much more important for system verification and validation.

Also, in order to master state explosion a more abstract interface should be introduced for implementation of behaviour. The object-oriented paradigm seems to be appropriate to cover this point: information hiding should be applied to such parts of an (SDL) implementation which do not impact a system's behaviour. More principal considerations are needed in this context.

5. Conclusions

Potential risks have been discussed which occur when SDL is used for development of a more general type of distributed systems and means were identified to master such risks. It was described how wrong conclusions may be derived on the correctness of an SDL implementation: a SDL system may be considered as correct after verification by SDL simulation, although it will never run correctly under real conditions. Vice versa, a correct implementation may be rejected by the SDL simulator. This is very dissatisfying and dangerous in case of safety-critical systems.

The EaSySim II environment was presented as a solution for most of the problems which are currently not solved by SDL and for optimisation of development of distributed systems.

Performance matters were identified as the main reason for such weakness of verification. Hence, verification and validation without consideration of performance is meaningless, because one does not know about its potential impact. Implementations which are considered as completely deterministic and correct may turn out as non-deterministic and incorrect in practice, even when people believe to be sure that the implementation is correct and performance cannot cause any problem.

Moreover, it was shown that consideration of performance aspects do simplify verification and validation significantly and help to master state explosion. So the extension of verification and validation from purely behavioural to performance aspects does not complicate system development, but eases it and reduces risks at early life cycle phases.

The number of system states and state transitions should be considered as a representative figure for quality of an implementation. A high figure may indicate that the system is not yet well defined.

The capabilities needed to complement SDL and SDL tools by performance evaluation issues can be provided as add-on's. However, more formal consideration of performance aspects and better support by the language to master state explosion are needed.

References:

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

[2] 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

[3] R.Gerlich, C. Joergensen: An Alternative Lifecycle Based on Object-Oriented Strategies, International Symposium on "On-Board Real-Time Software" , November 13-15, 1995, Noordwijk, The Netherlands

[4] 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

[5a] R.Gerlich: From CASE to CIVE: A Future Challenge, 'DASIA 96' - Data Systems in Aerospace, May 20-23, 1996, Rome, Italy

[5b] R.Gerlich: Experience with Simulation, Automated Code Generation and Integration, 'DASIA 97' - Data Systems in Aerospace, May 26 - 29, 1997, Sevilla, Spain

[6] 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

[7] ObjectGEODE SDL-Tool, Verilog, 150 rue Vauquelin, F-31081 Toulouse Cedex, France

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

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

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

[11] L.Braga, R.Manione, P.Renditore: A Formal Description Language for the Modelling and Simulation of Timed Interaction Diagrams, FORTE/PSTV'96 Conference, Kaiserslautern, October 8-11, 1996

[12] M. Buetow, M.Mestern, C.Schapiro, P.S.Kritzinger: Performance Modelling with the Formal Specification Language SDL, FORTE/PSTV'96 Conference, Kaiserslautern, October 8-11, 1996

[13] S.Fischer: Implementation of multi-media systems based on real-time extension of Estelle, FORTE/PSTV'96 Conference, Kaiserslautern, October 8-11, 1996

[14] M.Diefenbruch, J.Hintelmann, B.Mueller-Clostermann: The QUEST-Approach for the Performance Evaluation of SDL-Systems, FORTE/PSTV'96 Conference, Kaiserslautern, October 8-11, 1996

[15] R.Gerlich, Y.Lejeune: How to use ObjectGEODE with Ada, January 1997, internal communication, unpublished