Simulation of a System's Behaviour and Its Physical Environment: How to Master Tool Integration

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/

Abstract: This paper describes why and how two different types of simulation need to be integrated: event-driven simulation (applied to the system's behaviour and asynchronous environmental parts) on one side and time-discrete or multi-rate simulation (applied to the system's control part and synchronous environmental parts) on the other side.

This work has been carried out in the course of the ESPRIT project CRISYS [1] which deals with critical distributed automation systems. Simulation is used to verify and validate such systems already in an early development phase.

The example application includes asynchronous parts related to overall, high-level system control and supervision, and synchronous parts at the lower levels. Based on this example the benefits of an integrated simulation environment are discussed together with organisational measures which ease tool integration.

Keywords: behavioural simulation, multi-rate simulation, tool integration, event-driven simulation, continous simulation, time-discrete simulation, synchronous languages, Lustre, SDL, CRISYS

1. INTRODUCTION

Verification and validation of a system in an early development phase [2,3,4,5] requires representative system simulation and an adequate tool envrionment. Available simulation tools mainly focus on one of the following simulation types only: behavioural/discrete event simulation [6,7], continuous/time-discrete/multi-rate simulation (CDM)[1] [8,9], performance simulation [10,11,12].

In the past simulation was performed separately for each type. Possibly, simulation was restricted to such applications or parts of it for which appropriate tools existed. Hence, simulation could not be applied to verify and validate a complete system.

When we look on automotive applications in (most of) the current cars no sophisticated high level system control exists which needs to be simulated. Coordination of breaks, gear and engine was/is only controlled by the driver. This will change for the near future.

Also, in the space area simulation was only applied to parts of subsystems, e.g. to attitude and orbit control, guidance and navigation control, thermal and power supply systems based on CDM simulation.

With the increasing number of low level components (LLC) and the demand for a higher degree of automation the need for a "high level" control system arises which coordinates the lower levels. To verify and validate such a system correctly a broader simulation environment is needed which covers all three simulation types: behavioural/event-driven, CDM and performance.

Such a simulation environment allows to develop and test a new version of a hardware-software combined system without the need to manufacture the hardware before the design is settled. This way the hardware and software may be changed until an optimum design is achieved. This is of importance not only for space area, but e.g. also in the area of automation.

Integration of event-driven and performance simulation tools [2,3,13] and of event-driven and CDM tools [14] have already been performed in the past. Now, in the course of the CRISYS project an integration of event-driven, performance and CDM simulation has been done.

This paper explains the theoretical background and the techniques which were applied to tool integration, the example application and the capabilities of the tools and the integrated environment.

In chapter 2 the example application is introduced and the relevant problems w.r.t. simulation are identified. Chapter 3 analyses the integration, verification and validation issues from a principal point of view. The tool environment is described by chapter 4. Chapter 5 discusses organisational issues of tool integration and chapter 6 describes the system's partitioning in the integrated tool envrionment. Finally, chapter 7 concludes on the performed work.

2. THE APPLICATION

In the CRISYS project we derived from a complex application a representative, simplified distributed application which shall be subject of our investigations: it is a system which transports, sorts and distributes parts. The system consists of three belt sections and a switch for sorting. Fig. 2-1 shows in the upper part the principal elements: the belt sections, the switch, the part feeder and the overall monitoring and control processor

[1] In the following the abbreviation "CDM" is used for continuous, time-discrete and multi-rate simulation.

(MCP). The components of the belt sections are shown by the lower part of Fig. 2-1. The environment is added to each of the components: the feeder on top level, the motor drives on belt level.

For our further considerations it is essential to mention that this application is distributed which means the system includes several independent clocks. Of course, we could try to synchronise all clocks, but the question is if this is always possible under normal and anomalous conditions. As we will see later, continuous synchronisation is not possible in case of our application due to environmental or operational impacts.

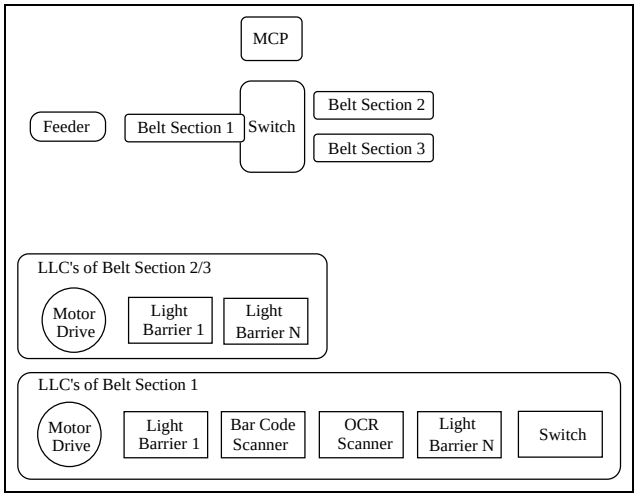

Fig. 2-1: The Part Sorting and Distribution System

The parts are put on the initial belt section 1 and identified by an OCR system and a bar code reader. Their positions relative to the environment are monitored by light barriers. The light barriers are intelligent and do not only provide a dark/light signal, but perform plausibility checks on a part's dimensions and hence inlcude a control part. Also, the scanners, the switch and the motor drive have own controllers.

Depending on the information received from the OCR system and/or the bar code reader the switch directs a part to final belt section 2 or 3. The drive of each belt is independently controlled by a speedometer. The operator may stop each belt or all belts by an emergency button. Such a stop and the following restart disturbs synchronisation of the three belt sections: the different mechanical and physical properties of each belt section require independent control of speed during start and stop.

The monitoring and control processor (MCP) supervises the low level components (LLC) (internals of the three belt sections and the switch) and responds to their service requests. It has typically an asynchronous behaviour: the sequence of the parts is not equidistant, the belt may even be empty for some time, hence requests concerning such parts are not predictable. Also, exceptions occur randomly from the LLC's e.g. in case of a jam or removal of parts from the belt. The event rate of the MCP is (mainly) related to the flow of the parts on the belt.

The LLC's are of synchronous nature: they continuously sample sensor values (light barriers, tachometer) and derive actions which are either input to the MCP, the belt drive, the switch or a following LLC. Their sampling frequency depends on the speed of belt movement and the required resolution of a part's position. If the (average) length of a part is 300mm and the required resolution is 3 mm, the sampling frequency of the LLC's is 2 kHz.

The physical environment requires CDM simulation in case of the motor drive and event-driven simulation for the feeder. It is assumed that the drive's speed is correlated with the local clock of each belt section.

Following exceptions concerning transportation and part's identification may occur: two parts are not clearly separated and hence identified as one part, a part may get lost, the position of the parts on the belt changes relative to the other parts during transportation, the distance between two consecutive parts is too small to change the switch position, the belts run at different speed. A belt may even be stopped while the other ones are still moving.

The clocks of each section may differ in frequency and may have a non-zero phase shift.

There are similarities between this application and a space application which also includes a mission and monitoring system at high level with asynchronous behaviour like MCP and sensors and actuators of AOCS/GNC and TCS which are of synchronous nature and are running at different and independent clock frequencies. Hence, the integration issues as discussed for the example application are also of relevance for a distributed space application.

3. INTEGRATION ISSUES

When discussing about integration of simulation environments of different types, first it should be clarified if such an integration is meaningful or not. After discussion of the characteristics of event-driven/behavioural and CDM simulation, this chapter will discuss about such integration issues.

3.1 CDM and Synchronous Simulation

Typically, CDM simulation is applied to problems which necessarily require intermediate steps in time to move from instant t1 to t2: if t1 and t2 are arbitrarily selected it is not possible in general to derive the system state at t2 directly from the state at t1. Usually, the system function is non-linear (e.g. differential equations need to be solved) and requires intrapolation to get the state at t2 at sufficient accuracy. Such systems need time-discrete simulation.

Moreover, the numerical methods usually require equidistant steps in time, which leads to periodic

sampling of sensor values, calculation of system states and actuator commands. Some parts of the system may execute at larger periods (which are fixed over time) being multiples of the basic (smallest) period.

If the period changes from to time during simulation and operation, this type is called multi-rate simulation. A singular case is an infinite period, e.g. when the operator stops a belt drive. In a periodic or multi-rate approach all system steps are executed synchronously to a clock.

Lustre [15] supports implementation and verification of synchronous systems. System properties may formally be proved or checked during simulation.

In the synchronous approach inputs are always read at the beginning of a period and execution is/must be completed until the end of the period. In fact, an absolute notion of time is not needed: everything happens within a clock period, the system is completely driven by the clock, all elements are synchronously progressing with the clock, even if the clock's period may vary over time.

A belt section (as part of the environment) of our application is a typical example of a synchronous system: the belt, the parts on the belt, the signals from light barriers and scanners are synchronously following the clock of the drive.

Typically, the controllers of the drive, the scanner and the light barriers are also driven by a belt clock: hence, a belt's motor drive and its other internals can be completely modeled by a CDM simulation environment.

3.2 Event-Driven Simulation

In case of event-driven simulation two following events may be separated by an arbitrary time interval, no intrapolation is needed. The next system state can be directly derived from the previous state or is possibly independent of it. E.g. at t1 a timeout may be set which suddenly changes the system state if it occurs at t2. No intermediate instants in time need to be entered.

From a simulation point of view the advantage of event-driven simulation is that it can rapidly progress in time as required by the occurence of events, not being constrained by accuray problems.

In our application the mean time between two events triggered by a sensor along the belt is $\Delta T=L/v$, where L is the average distance between the beginning of two consecutive parts on the belt and v is the speed of the belt. For L=300mm and v=3m/s we get $\Delta T=100$ ms.

The period of a sensor needs to be at most half (according to Shannon's theorem) of $\Delta t'=\Delta s/v$ where Δs =3mm is the required resolution. Hence we get $\Delta t=1/2 \bullet \Delta t'=500$ µs which is smaller than ΔT by a factor of 200.

3.3 Why to Keep Things in Their Specific Environment

We now analyse whether it is possible to cover all parts of a distributed system by one simulation type and tool. As the synchronous / periodic approach is the best approach from a reliability and safety point of view we start to analyse whether or not to take the synchronous approach for the whole distributed system.

3.3.1 Performance Issues

The large difference in rates of R=1/(500µs /100ms)=200 looks like a good reason to separate event-driven from CDM simulation because then we do not need to run the behavioural part at 2 kHz. However, as we will see this is not completely true in any case and we need further arguments to keep things separate.

What is ignored in above comparison is that a number of sensors may generate events which need to be processed by the behavioural part of a system. Now, if we have N sensors per belt, each sensor may generate an event. This decreases the mean time between (asynchronous) events down to $\Delta\tau=\Delta T/N$. For large N $\Delta\tau$ may become smaller than Δt, and hence it may be better to poll the sensors for events at the rate of the synchronous clock. The break even point is $N=2\ L/\Delta s$ which is 200 in our case.

This result may be interpreted as: if the mean density of sensors per part is higher than the required resolution of position, it is better to poll the sensors in order to limit the overhead by processing of asynchronous events.

Above limitation of the reaction time of asynchronous events does not lead to degradation of system performance, because the period $\Delta\tau$ represents the accuracy limit as given by the resolution Δs. Hence, a shorter reaction time $\Delta\tau<\Delta t$ is really not required. Consequently, in such a case polling of asynchronous events can coexist with cyclic execution.

So it really depends on the actual system configuration whether separate processing of asynchronous and synchronous events brings an advantage of performance or not.

However, in our case asynchronous processing of events will be of advantage because we have only a few LLC's in the whole system.

3.3.2 Modelling Issues

We have seen that each belt section can be considered as a closed system concerning the low-level components and the environment. But each belt section may have its own characteristics w.r.t. to the other ones: the local belt clocks may differ in phase and period. If a belt is stopped while the others are still running the correlation of events is even lost.

Such deviations are a consequence of a distributed system where each system part is more or less independent of the other ones by the environmental or operational constraints even if the goal of system design is to synchronise all parts as far as possible.

From a simulation point of view a master clock may be introduced which drives the local clocks of the belts. This way different phases between the local clocks can be generated.

The synchronous approach does not know progress of time, but only progress in terms of clock periods. This is not a problem as far as we model a "closed" system for which everything progresses according to the clock: inputs are read at the beginning of a period, execution is terminated before the end of the clock cycle. Even timeouts could be expressed by clock cycles.

However, we get a problem if the clock periods are different (e.g. if a belt is stopped) or vary (slowly) over time. Then the local time scale may not be refered to for exchange of information with other (remote) parts of the system.

For modelling this means that we need a central time manager which drives the synchronous parts (with possibly varying clock periods) and a monitor which is capable to execute according to the global, universal time scale and is independent of any of the local clocks. Obviously, the time manager and the monitor are of asynchronous nature and cannot be modeled by a synchronous environment.

Consequently, a pure synchronous environment cannot cover all modelling issues and we need to integrate the asynchronous and the synchronous world.

In case of our example application the time manager, the MCP and the feeder are modeled in the asynchronous modelling environment, the LLC's and the motor drive in the synchronous environment.

The task of the time manager is to synchronise both worlds. It processes asynchronous time events and clocks. In both environments time progresses by timer requests: in case of the asynchronous modelling part the time manger is informed about a future event and the time of its occurence (e.g. a timeout), for the synchronous part the scheduling request for the next clock cycle is sent to the time manager.

This way the asynchronous and the synchronous parts can coherently progress in time[2].

3.3.3 Verification and Validation Issues

Due to its deterministic nature verification and validation (V&V) of a synchronous system (or parts of it) is less critical than of the asynchronous part: in a synchronous system no potential conflicts due to concurrency exist. Therefore the synchronous approach should be applied as far as possible.

However, we have seen that a distributed system and its environment canot completely be implemented in a synchronous approach because of the ever-occuring asynchronous events (e.g. exceptions) and potential loss of synchronisation.

In case of our application V&V of two synchronous system parts which run at the same period, but encounter a shift of the clocks may be more difficult, but still possible in a synchronous simulation environment.

If the clock periods are different and vary over time the specific assumptions on synchronicity cannot be applied: it may happen that for one (or a number of) period(s) no input is received while for another period two (or more) inputs are acquired. This violates the assumptions about synchronicity and invalidates the V&V results obtained for a pure synchronous system.

Hence, for representative system simulation we need (a) to cover synchronous and asynchronous properties and (b) to provide appropriate verification and validation means.

A synchronous simulation and V&V environment takes advantage of the deterministic execution and data exchange. It will not support V&V of asynchronous behaviour. Vice versa, an asynchronous environment cannot take into account simplification due to synchronous execution. Consequently, an asynchronous and a synchronous environment need to be integrated for representative system verfication and valdiation by simulation, so that the advantages of each environment are fully available for V&V of each type.

4. THE TOOL ENVIRONMENT

In the CRISYS project the ObjectGEODE tool [6] is applied to the asynchronous system part and the SCADE tool [9] to the synchronous part of the distributed system. Both tools are supplied by Verilog.

4.1 ObjectGEODE, EaSySim II and SCADE

ObjectGEODE is based on SDL, which is a broadly applied formal method (and language) in the area of telecommunication. SCADE is based on Lustre [15] and is an outcome of projects in the area of nuclear power plants and aircraft manufacturing.

In SDL a system is hierarchically decomposed into blocks (nodes of the system tree) which again include processes (leaves of the system tree). Processes are excutable units. Their behaviour is defined by Finite State Machines (FSM).

A process may have a number of states. Upon reception of an input the received data and the status data are interpreted and processed during a "state transition". At

[2] We do not want ot discuss here the problems related to interactive debugging and its impact on synchronisation of both environments.

the end of a state transition the process either remains in the same state or enters another state where it again waits for an input.

EaSySim II extends SDL/ObjectGEODE towards performance simulation. Therefore it provides a global time manager (which is also accessable from SDL-external processes) into which the SCADE scheduler can easily be integrated.

The EaSySim II performance extension to ObjectGEODE allows to assign time to data transmission and execution of state transitions during simulation. It also provides means to track the data flow for such data channels which are not directly supported by SDL/ObjectGEODE like TCP/IP, ISDN/WAN or RS232. The related instrumentation is automatically removed by an EaSySim II tool when code is generated for the target system.

In SCADE and Lustre a system is also hierarchically decomposed into "operators" which may include other operators. So an operator is the generic unit of modelling. At the lowest modelling level basic entitities like logical gates, comparators, mathematical operators etc. are provided which allow to graphically synthesize an operator out of other existing atomic or non-atomic components. For data exchange between operators specific routines need to be added by the user for which templates are generated by SCADE.

ObjectGEODE and SCADE allow for simulation, verification (according to SDL and Lustre methodology) and code generation.

4.2 The Integration Approach

Integration of the asynchronous and the synchronous part is based on EaSySim II capabilities which are needed to communicate with SCADE. EaSySim II provides the global time management and the functions needed for routing of data between SCADE and EaSySim II / ObjectGEODE.

From the asynchronous SDL part the synchronous SCADE part is just considered as another process and the same communication means are applied as used internally by EaSySim II.

This SCADE process consists of a shell which organises the communication with EaSySim II, decodes and encodes the messages and schedules the synchronous program(s) (top level operators) as generated by SCADE.

5. ORGANISATIONAL ISSUES

Having identified the need for integration of asynchronous and synchronous simulation environments the next question is how easy is it to merge them. This depends on the openness of the environments (tools), but also on the architecture and standards of the software implementation.

5.1 Time Management

Execution in the synchronous and asynchronous part must be synchronised due to the interaction by data flow and events. When an event is raised at a certain instant in one part it must be received at (nearly[3]) the same instant in the other part. Consequently, synchronisation points must be defined between the two worlds and the time scales must be harmonised.

Therefore a time manager is introduced which manages events from both worlds: it inserts time requests in one single queue and notifies the clients at the desired instant.

The synchronous part will ask for clock events, while the asynchronous part will request for any events related to decisions according to the actual system state and consumption of resources.

It is not needed that the synchronous part will ask for every clock event: it may be sufficient that only each n. clock period a scheduling request is issued. This will allow to proceed at a higher rate in the synchronous part if required for accuracy reasons without exposing the time manger to this high rate.

Then the n cycles are executed sequentially by a non-synchronised loop and after completion of the loop the synchronous part waits for the next synchronisation point (assuming that the execution time in the real system does not exceed the estimated time of n loop executions during simulation). How large n can be depends on the accuracy of interaction: during a duration of n*<clock period> an exchange of information between the asynchronous and the synchronous part is not possible. Hence, n is determinned by <the smallest time interval during which no interaction is allowed> / <clock period>.

In the synchronous part data from synchronous elements are feed in at the beginning of a cycle. Data to be exchanged with other elements which are procuded during a cycle are stored when they produced and processed by the next cycle. This way no notion of time is needed in the synchronous part.

For exchange of data with the asynchronous system part the same mechanism is applied: data coming from the asynchronous part will be feeded into a synchronous element at the beginning of its next cycle. Vice versa, data to be transmitted from the synchronous part to the asynchronous part will be transmitted at the end of each cycle or beginning of next cycle respectively.

The approach for synchronous data communication implies that two synchronous SCADE operators O1 and O2 running at different clock periods T and m*T, but completely synchronous to one clock, can only exchange data with a period of m*T. This also applies to the communication with the asynchronous part. It is a matter

[3] "nearly" addresses the required accuracy.

of system design whether such a potential large delay is acceptable or not.

5.2 Interfaces

For integration of both simulation environments (EaSySim II / ObjectGEODE and SCADE) introduction of generic input and output layers is very important. This has to be considered by the overall system design. Experience with generic architectures shows that such generic layers are not only of advantage for integration of simulation environments. They bring in a significant increase of flexibility to react on architectural changes or extensions.

The approach as supported by EaSySim II is described below. It addresses mainly the communication within the asynchronous part (ObjectGEODE, EaSySim II) and between the asynchronous and the synchronous part as a matter of inter-process communication (different address spaces). The communication within the synchronous part occurs within one process only (intra-process comunication) and can easily be managed by moving data between data structures.

5.2.1 Data Exchange

A generic approach requires a standardisation of the data exchange format. This standard format is used for all communication channels. Its main components are: an identifier ("command") describing the purpose of the message, its source and destination, and the message type. Two principal types are used: a short and an extended type. The extended type allows to add arbitrary information by a message trailer (up to a certain maximum length[4]) like ASCII text or binary records as indicated by the message type field.

Data transmission and reception is based on a generic input / output approach which provides a common entry point for data output and another common entry point for data input (Fig. 5-1).

The output messages are routed through logical channels. Logical channels define a set of redundant and/or non-redundant (or a combination of both) physical channels like OBDH bus, Ethernet, ISDN (Integrated Services Digital Network), RS232. Specific routines of the EaSySim II library are forwarding the messages through the right physical channel.

Vice versa, on reception of such messages the system architecture allows to feed in data coming from different physical channels to one single entry point which is the starting point of data processing.

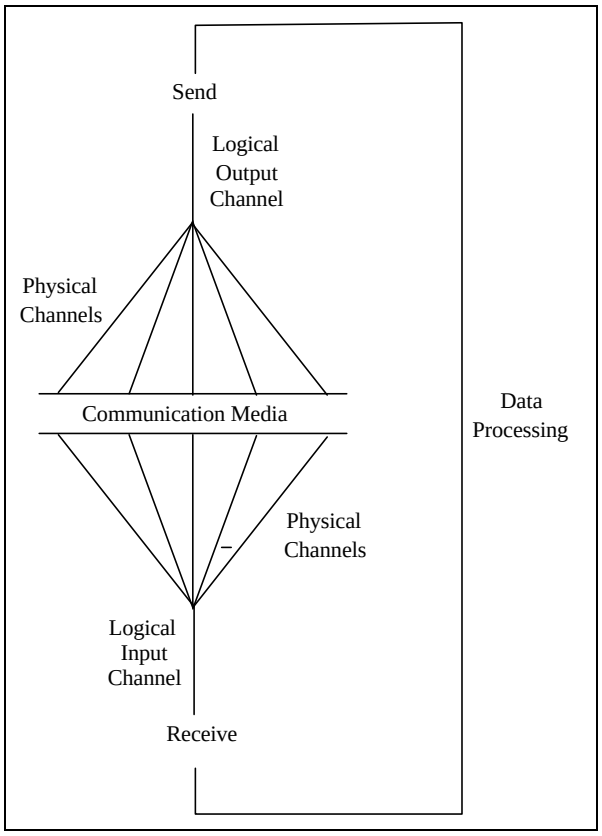

Fig. 5-1: I/O Interface Standardisation

This mechanism makes it easy to integrate the two different simulation environments. Data can be distributed arbitrarily in a system, and the communication channels may be changed (even at run-time) without effecting other parts of the application.

This communication approach is similar to CORBA [16]. But it does not need general encoding or decoding of data like by IDL (Interface Description Language), because the transmission of data is transparent to the distributing functions: only the sender and the receiver need to know about the used format of the message trailer.

The whole communication is uniquely based on this standard format. In principle only one structured (subprogram) parameter is alwasy exchanged inside and between the processes instead of a number of subprogram parameters[5].

5.2.2 Traceability Needs

The EaSySim II approach described above extends the basic mechanisms of SDL and ObjectGEODE. However, such extensions are not considered by the V&V means of SDL which only know about the normal SDL communication channels.

[4] Message trailers which exeeced the maximum length need to be partitioned

[5] This approach has been influenced by the SDL communication mechanism between processes based on messages.

To be able to track the data flow completely, so that the V&V tools of ObjectGEODE can also consider the SDL non-standard communication channels, functions were added which generate an output of the complete data flow which can be processed for verification and validation.

This extension is possible because (1) the ObjectGEODE verifier uses the standardised textual MSC (Message Sequence Charts) syntax and (2) EaSySim II applies the standardised I/O approach of Fig. 5-1.

It is even possible (and intended) to track the data flow of both parts, the asynchronous and the synchronous one, by a global MSC

6. MODELLING RESULTS

All parts of the example application were assigned to an asynchronous and synchronous part in the following manner (Fig. 6-1):

- asynchronous part
  -- MCP (system)
  -- feeder (environment)
- synchronous part
  -- belt control (system)
  including LLC's and their controllers
  3 instances in total
  -- belt motor drive (environment)
  3 instances in total

The asynchronous parts are modelled within the EaSySim II / ObjectGEODE environment.

A belt section (control and environmental part) is represented by a SCADE operator and decomposed into a control and environmental operator and so on.

Each of the three synchronous sets consisting of belt control, belt motor drive and sensors are represented as an independent synchronous program. Each belt section program is executed according to an own timeline defining the period and the phase w.r.t. the other belt programs. Each timeline is managed by the time manager of EaSySim II.

The current system model is already executable in the simulation environment, but its verification and validation is not started yet. It is planned to apply the SCADE/Lustre verification means to the belt sections separately and then to apply the SDL verification and validation means to the whole system. Each of the synchronous belt sections is considered as a black box which generates and consumes data and events. The interaction of the three belt sections is monitored, verified and validated by the asynchronous part of the simulation environment.

Faults will be injected by EaSySim II covering generation of illegal messages and loss of messages. Also, variation of phases and of clock periods of the synchronous parts shall be initiated by the asynchronous part of the system.

By recording of the complete data flow in terms of MSC files the verification capabilities of ObjectGEODE can even be applied to the whole system taking the belt section programs as white boxes.

Due to representative modelling of time in the EaSySim II environment and the capability for model execution the application cannot not only be verified, but also be validated.

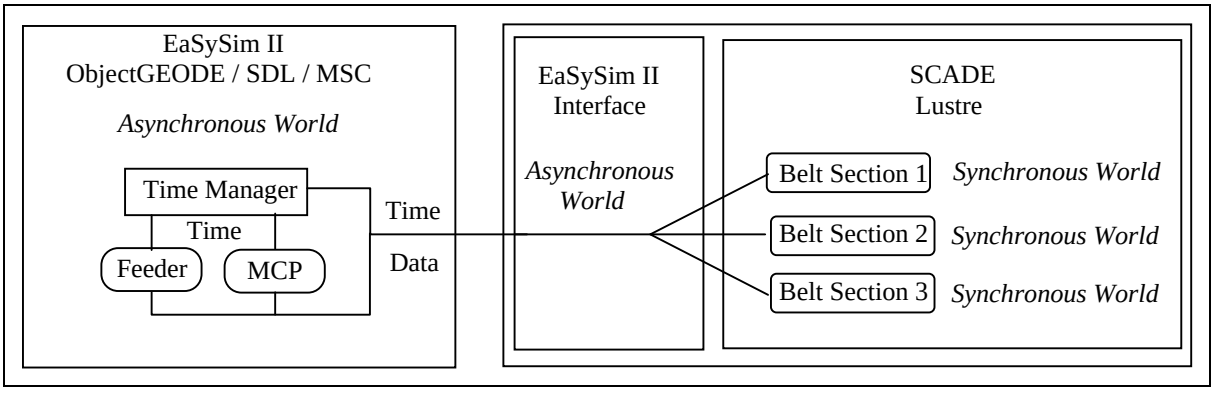

Fig. 6-1: Asynchronous - Synchronous Partitioning

7. CONCLUSIONS

The intention of this paper is twofold: (1) to explain why separate modelling and verification needs to be performed on the asynchronous and synchronous part of a system and its envrionment, (2) to demonstrate the feasibility of simple integration of asynchronous with synchronous tools.

It has been outlined that distribution (separation) and independent operation of synchronous system parts bring in an asynchronous behaviour. This requires asynchronous capabilities for simulation, verification and validation (including fault injection) which cannot be provided by synchronous tool environments. Consequently, this leads to integration of existing methods and tools to efficiently reuse of what is

available on the market. This approach not only yields a cost-effective, but also a time-efficient solution.

Integration was significantly eased by the open architecture of EaSySim II, ObjectGEODE and SCADE, EaSySim II standardisation of data exchange procedures, its global time management and communication support library.

Although an extension of SDL is needed, the interface to the ObjectGEODE verification tool could completely be kept by providing some glueing functions in addition thanks to the standardisation and openness of SDL and MSC languages.

Also, the SCADE open interface eased significantly integration: just the top level operators needed to be called from the generic process as provided by EaSySim II.

The current status allows for a first step of verification and validation of the example application which shall be done during the next months. It is expected that the results will encourage to refine the system so that future versions of the real system can be modelled, analysed, verified and validated in such an integrated environment.

8. REFERENCES

[1] CRISYS (Critical Instrumentation and Control System) ESPRIT project EP 25514
Team members: Schneider Electronics (prime), Aerospatiale, BSSE, CEA, Elf, GMD, NP Technology, Siemens Electrocom, University of Grenoble (UJF), Verilog, Verimag

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

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

[4] E.Conquet, G. Touer: Design and validate embedded SW with the formal language SDL,
Proceedings DASIA'98 - Data Systems in Aerospace, May 25-28, 1998, Athens, Greece

[5] J.-L.Terraillon: The benefits of formal description techniques for space on-board systems and their integration in an on-board architecture,
Proceedings DASIA'97 - Data Systems in Aerospace, May 26-29, 1997, Sevilla, Spain

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

[7a] D. Harel: Statecharts: "A visual formalism for complex systems", Sci. of Comput. Prog., vol 8, pp. 231-274, 1987

[7b] i-LOGIX Inc., 3 Riverside Drive, Anover research Park, Andover, MA 01810, USA

[8] Integrated Systems Inc. (ISI), Headquartesrs, 201 Moffet Park Drive, Sunneyvale, CA 94089, USA

[9] SCADE tool, Verilog, 150 rue Vauquelin, F-31081 Toulouse Cedex, France

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

[11] OPNET, MIL3 Inc., 3400 International Drive, NW-Washington, DC 20008, USA

[12] BONeS, Alta Group of Cadence Design Systems Inc., Corporate HQ, 555 River Oaks Parkway, San Jose, CA 95134, USA

[13] EaSySim II environment, Rainer Gerlich BSSE, Auf dem Ruhbuehl 181, D-88090 Immenstaad, Germany

[14a] Y.Tanurhan, S.Schmerler, K.D. Mueller-Glaser: "Integrated Design Process with MESA/MERLAN", IEEE International Conference on Control Applications 96

[14b] S.Schmerler: "Coupling Statemate and MatrixX by Code Integration", 4.Dt. Statemate Anwenderforum 96

[15] N.Halbwachs: Lustre Language Reference Manual, V5, September 1997

[16] The Common Object Request Broker Architecture and Specification, Rev. 2.0, Object Mangement Group (OMG), July 1995, updated in July 1996

[17] ITU, Recommendation Z.100, Specification and Description Language, SDL, 1989, Geneva. Blue Book, Vol. X.1, and appendices A, B, C, D, F1, F2, F3

The work described above on integration of asynchronous and synchronous methods and tools has been executed in the course of the ESPRIT project CRISYS (EP 25514). The paper expresses the ideas and conclusions of the author, not necessarily the opinion of the team.

Acknowledgement: The author wants to thank Mr. Axel Poigne and Mr. Reinhard Budde from GMD (Bonn) and Mr. Paul Caspi from Verimag (Grenoble) for their valuable discussions about the synchronous approach which helped to clarify the separation into asynchronous and synchronous system parts of the application. He furhter thanks Mr. Christian Hotte from Verilog for his support for the SCADE tool.