Performability Analysis of an Avionics-Interface

Lutz Twele, Holger Schlingloff and Helena Szczerbicka
TZI-BISS/ University of Bremen
P.O. Box 330440
28334 Bremen, Germany

ABSTRACT

This paper reports on a case study in the quantitative analysis of safety-critical systems. Although formal methods are becoming more and more accepted in the development of such systems, usually they are used in the verification of *qualitative* properties. However, in many cases system safety also depends on the fact that certain *quantitative* requirements are met. Therefore we are interested in statements about quantitative properties, which can be achieved by a rigorous formal method. Our approach is to create a Generalized Stochastic Petri Net (GSPN) model of the system and use it for the analysis of the system.

The object of this case study is a Fault Tolerant Computer (FTC) constructed by Daimler Benz Aerospace[1] (DASA) for the International Space Station (ISS). One part of the FTC is the Avionics Interface (AVI) which connects the FTC with a bus-system. We want to determine the data throughput that can be reached by the AVI and obtain informations about bus-usage-profiles which can cause the rejection of messages. Although such rejections are allowed according to the specification, they can cause a significant deterioration in the overall bus performance.

In this article we describe a GSPN model of the AVI software and its environment. This model is used to make predictions about the AVI performability. Since a complete analytical solution of the model is not possible due to its complexity and the infinite state space, a simulation is used to analyse the crucial AVI behavior for several bus-usage-profiles.

INTRODUCTION

For several systems safety is an essential design property. This holds especially for domains in which human life depends on the correct functioning of the system, e.g. in the control of a manned space station.

[1]this work was done in cooperation with Daimler Benz Aerospace and JP Software Consulting

The *Fault-Tolerant-Computer* (FTC) [9] of the Daimler-Benz Aerospace (DASA), Bremen, is constructed to make sojourns in the space station as safe as possible. The FTC is used for controlling the assembly of the station and scientific experiments. Its highly redundant structure allows an accurate operation even if some components malfunction. One part of the FTC is the *Avionics Interface* (AVI) which is responsible for the connection of the FTC with the *Avionics Bus System* (ABS) of the station.

We are interested in the throughput that can be reached by a single AVI under different conditions, to determine the performance of the FTC. In this case the performance is part of the safety specification of the system, because control of the station must be performed under hard real-time requirements.

Therefore it is necessary to use a method which allows a quantitative analysis and modelling of time. One formal method that offers these features is the concept of *Generalized Stochastic Petri Nets* (GSPN) [1]. Other concepts (e.g. queueing networks) turned out to be not powerful enough to model the AVI. GSPNs allow construction of models for complex concurrent systems that represent the structure of the model in an intelligible graphical way. This property is very useful as an interface between model developer and application experts. Nonetheless GSPNs are a formal method that allows analysis of different quantitative and qualitative properties of the model (e.g. analysis of the underlying Markov-chain or P-invariants). Especially for the AVI the possibility of simulating the GSPN is important since the state space that results by the model of the AVI environment is infinite. Another advantage is that GSPN are an approved specification method with numerous GSPN-Tools available. We decided to use the DSPN-Express [6] and TimeNet [4] tools because of their compatibility of the external net representation, the possibility of numerous methods for structural analysis and the fast simulator of TimeNet.

SYSTEM DESCRIPTION

The FTC has four *Lanes*, were each lane consists of an *Application Layer* (AL), *Fault Management Layer* (FML) and the *Avionics Interface Layer* (AVI). The structure of the FTC is displayed in fig. 1.

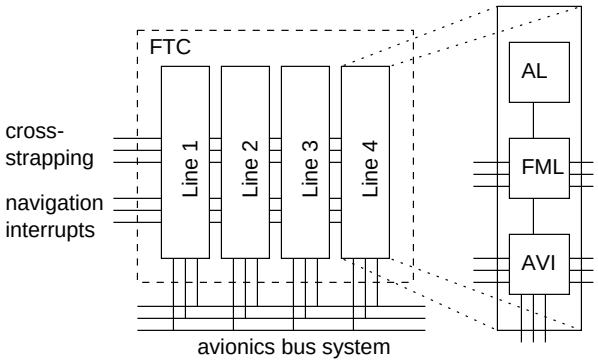

Figure 1: Four-lane FTC

The FTC can be configured to access the ABS as bus-master. The applications are running on the AL. If an application communicates with the station environment this is done via the AVI, which is connected to the *Avionics Bus System* (ABS). For this the AVI has to receive asynchronous messages from the AL and send them to the synchronous working ABS. On the other side the AVI receives messages from the ABS and sends them to the AL. Between AL and AVI resides the FML, which compares the messages that are exchanged between its AL and AV with the other FMLs for error detection. Therefore all FMLs are connected among each other.
The maximal throughput of the system is primally determined by the AVI for the requirements of communication with the ABS. The AVI is implemented in OCCAM and for every lane of the FTC one Transputer T805 is reserved just for the AVI.

The FTC communicates with its environment by *Messages*. These Messages consist of twelve *Boxcars* were each consist of a header and 64 Byte of data. Every Boxcar can be set as *Reading Boxcar* (RB) or as *Writing Boxcar* (WB), these notation expresses the action that is performed on the ABS by the Boxcar. The Messages and their composition of RB and WB are generated by the AL. The task of the AVI is to send the Messages to the bus. The ABS consists of 8 *MIL 1553*-busses were each bus can process one message at every clock cycle of 12.5ms. The main flow of messages through the AVI is shown in fig. 2 (taken from [3]).
The AVI has to put the data of WBs of a Message into the *Hybrid* at a given time and to get the RBs during the following two bus cycles. The Hybrid is a dual-ported ram which is used by the ABS to receive and send Messages. So the processing of one Message takes three bus cycles after getting it from the *Input-Output-Table* (IOT).
Process *p1.1* receives Messages from the AL via FML and writes them into the IOT. For every Message there is a designated bus cycle during which it is to be processed on the ABS. The Message stays in the IOT until its bus cycle starts.
One cycle ahead of the designated cycle, *p3.1* reads the Message from the IOT and sends it to *p0.2*, from where it is send to *p0.1*. Process *p0.1* writes the message into the Hybrid.
In the following cycle the Messages previously written into the Hybrid are processed on the ABS. This means sending and receiving the Boxcars of the Messages.
During the remaining time of this and the next cycle *p0.2* reads the RB of the Message from the Hybrid and sends them to the FML via *p4.1* and *p4.2*.

Since every Message is processed in three cycles, the Hybrid is organized as a triplex buffer. The Hybrid contains buffers for writing, processing on the bus and reading. The states of each single buffer changes automatically with every bus cycle.

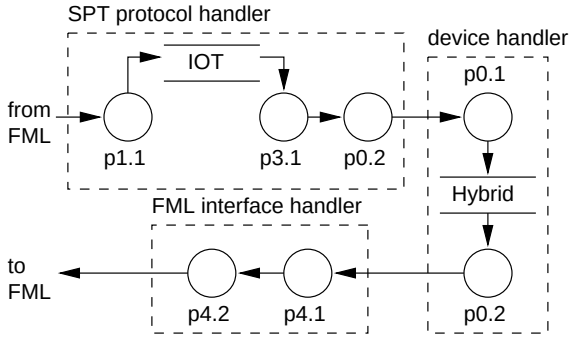

Figure 2: Main data flow within the AVI

It is important that WBs of Messages which were read from the IOT are written into the Hybrid before the next ABS bus cycle starts, otherwise the Message is rejected because the buffer is used for the processing on the bus. RBs of Messages that are received from the ABS must be ready while the next bus cycle is processed, otherwise the Message can not be received correctly.

A bus error watchdog is contained in the AVI, which supervises that Messages are received by the ABS in time. If the delay of the watchdog is expired, the Message is considered to be late and the watchdog initiates an exception. This exception causes an empty

cessing in *p0.2*, *p4.1* and *p4.2*.

For further sections it is necessary to mention that the AVI can handle two different types of Messages, (normal) Messages and *Flexblocks* which differ in the way they are written into the IOT. Normal Messages which enter the AVI contain all necessary informations for further processing. For Flexblocks the AVI has to determine some of this information while writing it into the IOT, therefore Flexblocks need about 20% more time to be processed in *p1.1* than normal Messages.

In addition to these tasks, the AVI has to react on several different interrupts that can be generated by the navigation system.

THE MODEL

In building a formal model of the AVI (see [8]) we used a top-down system specification provided by the developer [3]. It describes the system in pseudo-code at a low level. Furthermore we used the OCCAM-Code which implemented this description and a CSP-abstraction that was derived from the OCCAM-Code for other purposes [2]. These descriptions contain sufficient information about the AVI environment for our model.

In [7] a method of transforming CSP-code into generic Petri nets is proposed. For our problem such a transformation is not useful. Firstly, the complexity of the AVI CSP-code would result in a Petri net which is too large for an analysis. Secondly, the timing information would be missing in the automatically generated net.

The base of our model was the main data flow described in fig. 2 above. The essential structures that were necessary to construct a GSPN model of the AVI which allows the desired analysis are:

- models of single processes,
- dataflow of messages,
- scheduling of the processes,
- triplex-buffering,
- message generation unit as the interface to FML,
- behavior of the Hybrid as interface to the ABS and
- rejection of messages

Since no informations about timing of processes and probabilities of message distribution are contained in our starting specification, this information was obtained by estimation on the length of the code and measurement with the actual FTC. The formal timeless specification and the timing measurements were put together to build a Petri net model.

Figure 3 shows the structure and main data flow of the modelled system. The layout of the figure corresponds to the structure of the Petri net in fig. 6 below.

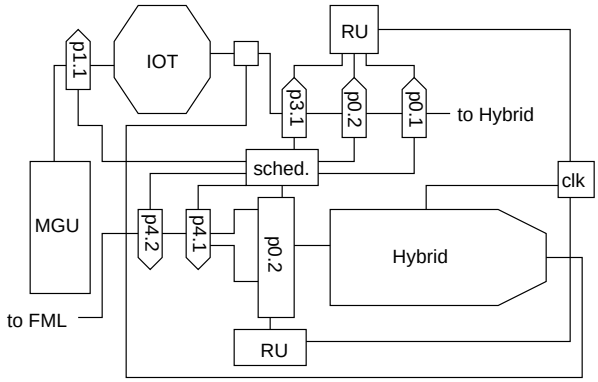

Figure 3: Overview of components

The OCCAM-processes *px.y* displayed in fig. 2 and 3 are all modelled with a similar structure. This structure implements the data flow of Messages through the processes, the scheduling and the possibility of interruption of the processes. Fig. 4 shows the structure of such processes.

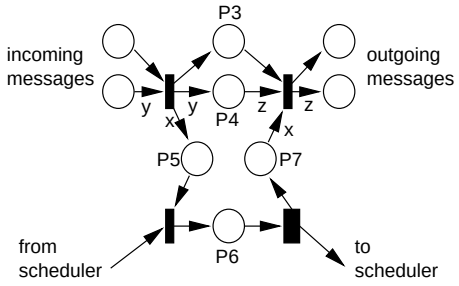

Figure 4: Structure of OCCAM process

Incoming Messages are stored stored in *P3* and *P4*, at the same time x token were put into *P5*, where x is the number of steps which are necessary to process one Message. When the scheduler provides the scheduling-token and a Message is to be processed, a token can be put into place *P6*. Now one step of the Message can be processed, a token is put in place *P7* to mark one more processed step, and the Scheduler

Message is finished and sent to *outgoing Messages*. For some OCCAM processes extensions were made to implement different properties (e.g. only one Message is allowed to be processed at a time).

Process *p0.2* is the one which differs most from this generic scheme. It implements one part of the triplex buffer. First the Messages enter a similar structure as in fig. 4 and are processed as usual. With every tick of the bus clock the state of this first process is completely put into a second one which represents the access to the third buffer of the triplex buffer.

The task of the *Scheduler* is to guarantee that at most one OCCAM process is active at one time. Therefore the scheduler contains a single token which is made available to the OCCAM processes. As described above, the OCCAM process concur for this token. The process which receives it can continue its calculation. After returning the token from the OCCAM processes it is delayed for a determined time before becoming available again. The Scheduler introduces a global dependency between all (potentially parallel) processes. Although this global dependency complicates the analysis, it is necessary to model a fair scheduling because it influences the timing of the system.

The *Message Generation Unit* (MGU) (fig. 5) creates Messages which should be processed by the AVI. The MGU is part of the AVI environment. It emulates the AL and FML of the actual system. The MGU produces messages with a deterministic rather than an exponential rate. The deterministic rate is an adequate choice, because the FTC partly synchronises with the ABS. This is caused by a frequently appearing scheme of information exchange between FTC and the station: the FTS is sending requests to the station components and then awaits the answers for further processing.

The MGU is constructed to generate up to two different message types in one simulation pass. The message types are generated by transitions *T3* and *T4*. The structure of the messages depends on the arc values *a,b* resp. *c,d* which determine the number of RB and WB for the message. The transition that is enabled to produce messages is determined by a token which is toggled between the places *P1* and *P2*, where a token on the place inhibits the enabling of the corresponding transition to generate its message type.

The duration a message type is generated depends on the delay of the transitions *T1* and *T2* which toggle the token determinining the message type. With exponential distributed delay at high rate (compared to the message generation rate) a nondeterministic change of the message types can be simulated.

The task of the *Hybrid* in our GSPN model is to be

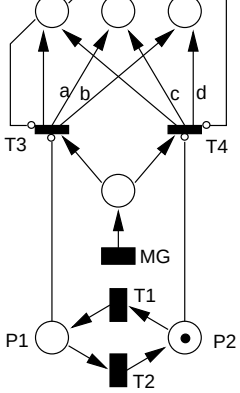

Figure 5: Structure of MGU

one part of the triplex buffer. A Message which enters the preprocessing phase by process *p3.1* is simultaneously put into the Hybrid. The Message is delayed until the next tick of the bus clock. During this bus cycle the Message is delayed for a time which depends on the number of RB, then it is made available to *p0.2* for postprocessing.

The rejection of Messages is implemented in two *Rejection Units* (RU). If a Message resides in a critical part of the data flow when a tick of the bus clock occors the RU removes this Message and puts it into special places for rejected Messages.

A graphical representation of the complete model is shown in fig. 6. Main data flow, OCCAM-processes, message generation and rejection can be identified by comparison with fig. 3.

SIMULATION

The main goal of this project was to determine a lower bound for the throughput of the system under different message types (normal Messages and Flexblocks), message structures (number of reading/writing Boxcars) and message traces. This lower bound is necessary to be able to guarantee certain system properties for all possible load distributions. Since the actual load is generated by software modules which were not available, no average distribution or bounds could be predicted. Therefore it is important to deliver exact simulation results together with the system, such that certain timing requirements of the application software can be verified.

Furthermore, we wanted to determine an optimal value for the delay of the bus error watchdog.

Validation of the model

Before starting the analysis we had to test and validate our model for consistency with the actual system

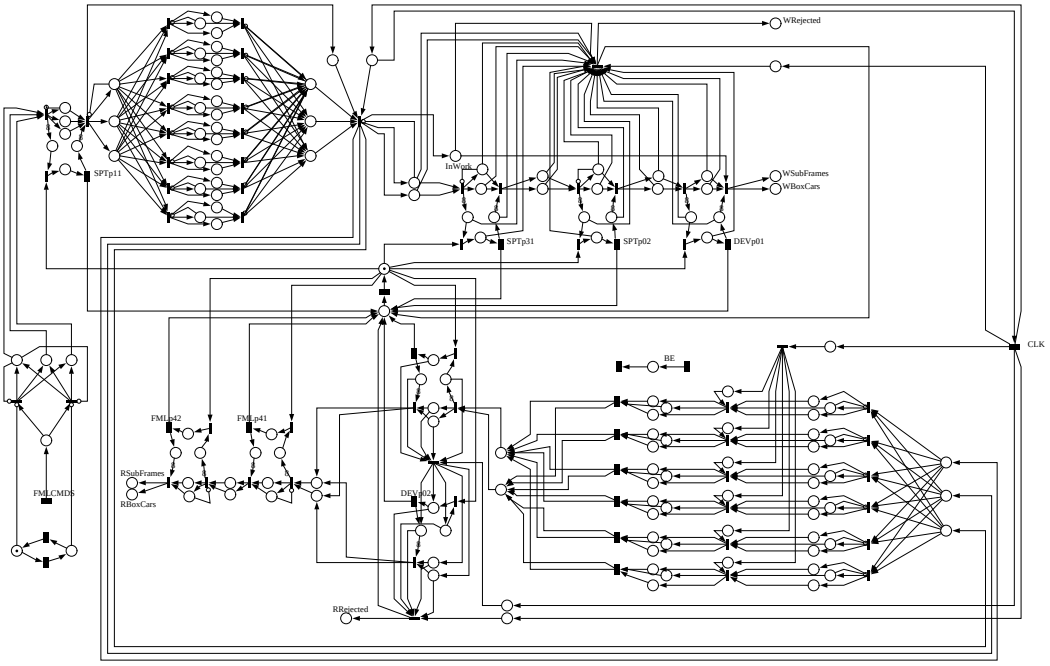

Figure 6: Model for the complete AVI-module

[5]. First the main data flow was tested using the token game of TimeNet. This was important especially for the marking dependent labels that were used in the model. Then the token game was used to test the scheduling and rejection of messages. Several properties could be shown by the analysis of submodels (e.g. the correct functioning of the message buffer), for others the whole model was needed (e.g. the correct functioning of the scheduler). Structural analysis was used to show that the postprocessing of messages does not change the message structures.
After validation of these qualitative aspects we tested the timed behavior of the Petri net model with the simulation component of TimeNet. First we sent single messages through the system, later on we use more complex message traces whose behavior was critical. Then we compared our simulation results with the expected results from the OCCAM-implementation and the measurements in the actual system.
The main problem of the validation was to locate model inconsistencies with the actual system. Sometimes the token game turned out to be insufficient due to the model complexity. In these cases, it was necessary to define additional reward measures which allowed to analyse the critical behavior of the model.

Simulation objectives and methods
The following values were to be determined:

- Maximal throughput with normal Messages.
- Maximal throughput for Flexblocks.
- Maximal throughput if messages with only reading or only writing Boxcars enter the AVI in an indefinite order. This situation is called *Load change*.
- Capable delay for the bus error watchdog.

Since the rejection of Messages leads to an eventual retransmission by software layers above we were only interested in throughput values where no messages are rejected.

Most of the results were obtained by the stationary simulation component of TimeNet. The main paramters of the simulation were the confidence level for the results and the length of time for one simulation pass. We decided to use a confidence level of 95% as an optimal compromise between computation time and accuracy. We wanted to use 1 sec. of system time to get our results and added 10% of this time for the transient part at the beginning of the simulation. We observed the places which contain the number of sent and received Messages and Boxcars and the number of rejected Messages. With the number of token in this places we could determine the desired values.

for throughput analysis the model was set with parameters for the different conditions. Then in several passes the message generation rate was increased until the threshold value was reached at which messages were rejected. The throughput could then be calculated from the number of sent and received Boxcars.

The watchdog delay was determined by setting the parameters of the message generator and increasing the delay until messages were rejected.

Achieved Simulation Results

For the simulation of about 1 sec. system time each single run used up to half an hour of CPU time on an average Sun workstation. Since the simulation does not require to build the complete state space of the model, the memory requirements could be neglected.

Our analysis resulted in throughput values for the different conditions which are accurate enough for the developer and user of the AVI. All results are within a given tolerance, compared to the actual system. The investigation particularly showed that the throughput of the system is only partly influenced by the distribution of RB and WB, but mainly depends on the dynamic change of load.

CONCLUSION

The results show that performance and performability analysis even of complex software is feasible with GSPNs. We found applicable methods for the construction of a GSPN model for a large software system that is implemented in OCCAM.

The advantage of using a simulation with GSPNs instead of measuring the actual system is the possibility to obtain data from points in the model which could not be accessed within the actual system. This facilitates the search for weak points of a system. Since the model is represented in a graphical way it could be used as a basis for discussions with the developers.

Besides the quantitative simulation results other properties of the system were exhibited and analysed with our model. To enable an efficient simulation it was necessary to choose a high abstraction level for the model. For this it was necessary to obtain detailled insight into the structure of the system. With this insight a complex but still intelligible model has been built.

Nevertheless, the question of finding an appropriate abstraction level is an important further research topic. For many complex systems, a complete analysis on a detailled level is not feasible. With a coarse abstraction, however, important system properties may be lost. Therefore, it is advantageous to model a system on several levels, and to validate each level in a compositional way.

ACKNOWLEDGEMENT

We would like to thank G. Urban, DASA AG Bremen, and Prof. Dr. J. Peleska, JP Software Consulting, for the continuing support during this study.

REFERENCES

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis: *Modelling with Generalized Stochastic Petri Nets*, Chichester, John Wiley and Sons, 1995

[2] Bettina Buth, Michel Kouvaras, Jan Peleska, Hui Shi: *Deadlock Analysis for a Fault-Tolerant System*, In Michael Johnson (Ed.): Algebraic Methodology and Software Technology. Proceedings of the AMAST'97, Sidney, Australia, December 1997, Springer LNCS 1349, 1997, pp. 60-75.

[3] DASA Daimler Benz Aerospace AG, *DMS-R FTC SW DDD vol. 1,2*, Doc. DMS-R-RIBRE-DDD-0001, 1996.

[4] R. German, Ch. Kelling, A. Zimmermann, G. Hommel: *TimeNET - A Toolkit for Evaluating Stochastic Petri Nets with Non-Exponential Firing Times.* Journal of Performance Evaluation, Elsevier, The Netherlands, Vol. 24, 1995, pp. 69-87.

[5] Jack P.C. Kleijnen: *Verification and Validation of Simulation Models*, European Journal of Operational Research 82, 1995, pp. 145-162

[6] Christoph Lindemann: *Performance Modelling with Deterministic and Stochastic Petri Nets*, Chichester, John Wiley and Sons, 1998.

[7] Ernst-Rüdiger Olderog: *Operational Petri Net Semantics for CCSP*, In Grzegorz Rozenberg (Ed.): Advances in Petri Nets, Berlin, Springer LNCS 266, 1987, pp. 196-223.

[8] Holger Schlingloff, Lutz Twele: *DASA Project FTC - Stochastic Load Analysis of AVI*, University Bremen, TZI-BISS, Internal Report, 1998.

[9] Gerd Urban, Hans-Joachim Kolinowitz, Jan Peleska: *A Survivable Avionics System for Space Applications.* To appear in: FTCS-28, 28th Annual Symposium on Fault-Tolerant Computing, Munich, Germany, 1998.