

Integrating Software Performance Evaluation in Software-Engineering

Matthias Becker, Lutz Twele, Helena Szczerbicka
University Hannover, Welfengarten 1, 30167 Hannover, Germany
xmb@informatik.uni-hannover.de

Keywords: performance and software engineering, modelling concept

Abstract

During a case study we encountered some problems, which could be solved manually this time, but pose a grand challenge regarding a generally applicable methodology.

The case study was about verification and performance evaluation of a Fault Tolerant Computer (FTC) System to be employed in the International Space Station (ISS). Four different specifications of the FTC had to be developed for different purposes (chronologically ordered): the informal specification (Data Flow Diagram), OCCAM code (implementation), a CSP model (deadlock-analysis), and a GSPN-model for performance evaluation. Each model has been derived from the predecessors, mostly manually.

This was an enduring and error-prone process, for which an integrated methodology should be developed. Why this is urgently necessary, and why this is a challenge and what steps might lead to a possible solution, these questions will be answered in this paper.

1 INTRODUCTION

The deployment of software systems in safety critical applications necessitates verification of the functional correctness of software. Thereto, also the timing specifications and requirements, that influence/are part of the correct behavior, have to be taken into account.

In the classical approach, system and software specification, implementation, behavioral and performance validation are separate activities, resulting in one model/formalism for each purpose. This may lead to inconsistencies and furthermore is a waste of manpower. Therefore it is desirable to develop a methodology that integrates specification, implementation, behavioral and performance validation.

In this paper we motivate the problem by a case study of a Fault Tolerant Computer system, we discuss essential features and outline a methodology. In section 2 we present the case study.

In section 3 we discuss some requirements that we

think are essential for a practically useful methodology. Especially, if performance evaluation is considered and included in the system development process from the beginning, the task of the modeller can be simplified to a great extent.

2 CASE STUDY

The case study presents throughput analysis of a *Fault Tolerant Computer* (FTC), that has been constructed by *DaimlerChrysler Aerospace* (DASA) (formerly Daimler Benz Aerospace) for usage in the *International Space Station* (ISS). The FTC consists of four redundant identical lanes that control each other and turn one lane off in case of different behaviour (fig. 1). This control is done

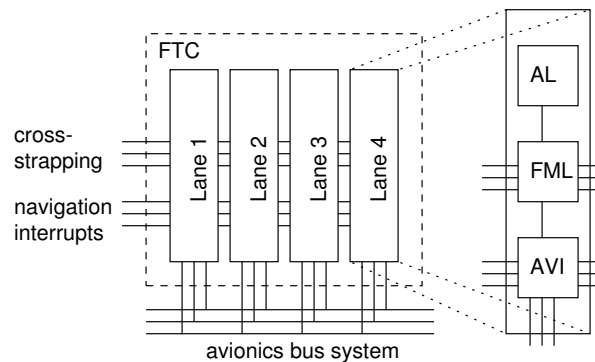


Figure 1: Four-lane FTC

with the Byzantine protocol [16]. Each lane consists of *Avionics Interface* (AVI), *Fault Management Layer* (FML) and *Application Layer* (AL). The communication with the rest of the station takes place via the *Avionics Bus System* (ABS). Application software on the AL demands for communication, the FML checks for failures and inconsistency, the AVI connects the upper layers to the bus. This study was concerned with the throughput of the bus system.

The task was to build a performance model of the AVI and its environment in order to determine requirements for traffic generation on the AL (rate of traffic, type of messages, load changes), that opti-

mize the performance of the ABS. This goal can be achieved by finding traffic generation characteristics, that guarantee a loss free communication over the bus system. These traffic characteristics serve as guideline for the future implementations of application software. However, the avoidance of losses is only a secondary goal, since the case of losses is considered in the specification of the system and does not pose a problem.

2.1 Approach

In order to develop a performance model of the FTC, we have decided to use a formal method that facilitates modelling of time and quantitative analysis. One method that offers these features is the concept of *Generalized Stochastic Petri Nets* (GSPN) [1]. Other concepts (e.g. queueing networks) have turned out to be not powerful enough to model the AVI. 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 analysis of the AVI the possibility of simulating the GSPN is important, since the state space of the AVI environment model is infinite. Nonetheless GSPNs represent the structure of the model in an intelligible graphical way. The *token game* (graphical execution of a GSPN) allows easy validation of the functionality of the model, and is a very useful interface between model developer and application experts. The problem is a missing concept for hierarchy, leading to large and complex models.

All available descriptions of the software system have been integrated in the development process of the GSPN model of the AVI. We have used the top-down system specification provided by the developer [4]. The system behaviour is specified on different levels, down to a description in pseudo-code at the lowest level. From the pseudo-code, OCCAM code already has been implemented. A CSP-abstraction has been derived from the OCCAM-Code for deadlock analysis [3]. These three specifications of the system (informal specification, OCCAM code, CSP) together with hardware defaults contain sufficient information about the AVI environment to build our GSPN model.

Automatic transformation of the CSP-code into a generic Petri net as proposed in [9] is not useful for our problem, the complexity of the AVI CSP-code would result in a Petri net being too large for an analysis. Furthermore the CSP code does not contain enough information for the performance model, the scheduling and message generation is missing for example.

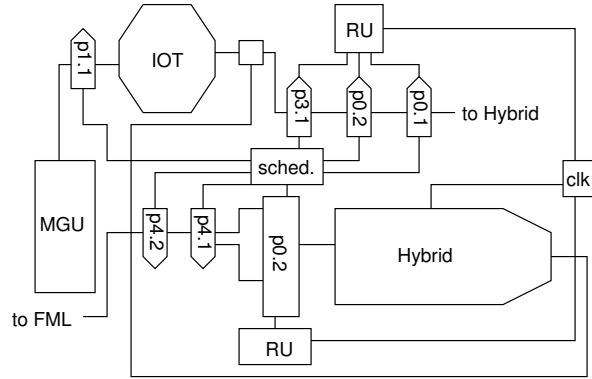


Figure 2: Overview of components

2.1.1 Building the GSPN Model

In order to build a GSPN model out of the specifications for the FTC, the first step is to understand the model (since we have not been involved in the development of the system specification) and find the crucial elements, that interact with the AVI and form its environment. In this phase the hierarchy in the informal specification is very helpful. On the upper levels we can find information about what elements of the ISS interact with the AVI at all, and on lower levels we find details of the interaction. In some fine points, it is also necessary to have a look at the OCCAM implementation to fully understand the detailed functionality. As result, we can identify the crucial components of the system that influence the throughput (cf. Fig. 2). As second step, GSPN models of the main components have been developed. For this task, the OCCAM and CSP code was used to a great extend. By identifying events and states of the components, a GSPN model was build for each component, not specifying firing times yet.

The timing information results from estimations and measurements of the OCCAM code, from hardware prerequisites and some estimations about software running on the FTC.

As last step, the GSPN models of the components are joined to one overall GSPN model. When merging the single components, special attention must be devoted to processes, that work in parallel by specification. These processes, specified as parallel, cannot work parallel in reality, where only one processor is reserved for the AVI. Therefore the scheduling of processes on the hardware processor must also be modelled.

2.2 Description of the GSPN Model

The basis of the overall GSPN model was the main data flow. We have chosen a level of abstraction

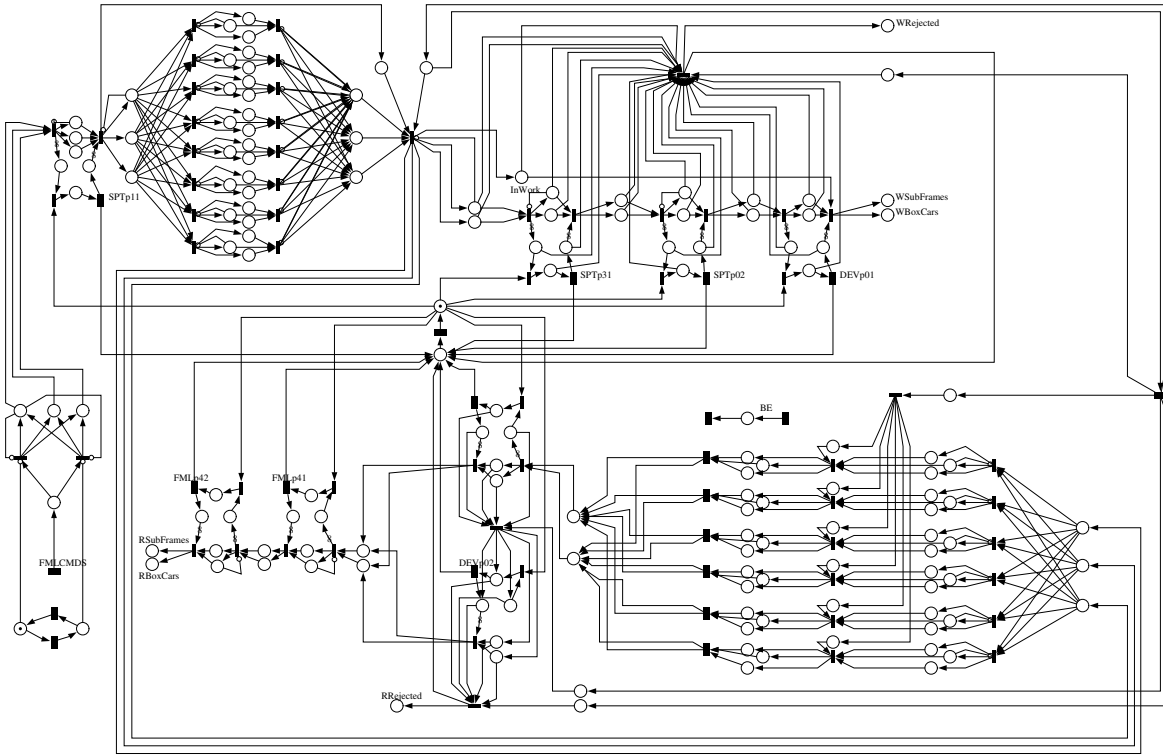


Figure 3: Model for the complete AVI-module

that results in a comprehensible model, that still allows to deduce properties of the behavior of the real system. This model contains only that information being important for the intended throughput analysis.

The essential structures that are necessary to construct the GSPN model of the AVI allowing the desired analysis are: (1) models of single processes (2) data-flow of messages (3) scheduling of the processes (4) triplex-buffering in the hybrid (messages of the current clock-cycle as well as one cycle before and after, are kept in the hybrid) (5) behavior of the hybrid as interface to the ABS (6) message generation unit as the interface to FML and (7) rejection of messages

The final GSPN model is shown in fig. 3. shows the structure and main data flow of the GSPN model in fig. 3 in an abstracted way, in order to illustrate the correspondences between components of the AVI and the GSPN model. As result, the GSPN model has been used to understand the influence of different load patterns on the bus throughput

and to suggest crucial factors for optimal usage of the bus capacity.

2.3 Conclusions from the Case Study

The results show that performance analysis even of complex software is feasible with GSPNs. We have 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.

One difficulty in this study has been the a posteriori development of the performance model. Since neither of the three existing specifications supports a time concept, a fourth model had to be developed. The timing information had to be extracted manually from several sources, because timing speci-

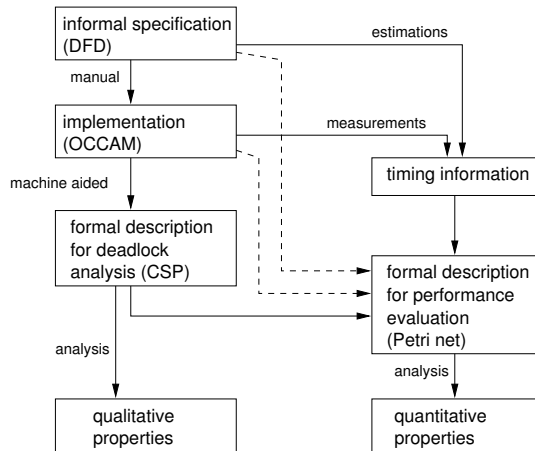


Figure 4:

cations and requirements are missing in the initial specifications.

If the GSPN model is built at the same time with the specification of the system, performance analysis would be less difficult. Then, the developer also would be enabled to make a crude analysis of the system performance during the development of the software. This would facilitate the improvement of performance of the system by using insight from early simulations and by testing different options for system structure.

If the building of a detailed GSPN model is integrated in the process of software development, then this detailed GSPN model could be used for deadlock analysis as well. The separate derivation of a CSP model would then be superfluous. A theoretically well-founded concept for hierarchical modelling with time *and* a user-friendly tool are essential requirements to benefit from hierarchy in the specification.

3 A STEP TOWARDS AN INTEGRATED METHODOLOGY

In this section, we propose report what features an integrated methodology should include, and some aspects, that make this task a grand challenge.

Figure 4 summarizes the approach used in the case study. As before mentioned, the informal specification was manually implemented in OCCAM. From the OCCAM code a CSP specification has been derived partly automated, for deadlock analysis. All this information together has been used to build the GSPN model for performance analysis, together with manually derived timing information.

This approach has some obvious disadvantages.

Three costly models have been derived mostly manually from the original specification. This may lead to inconsistencies and is a waste of man power. Furthermore performance analysis may yield some insights that necessitate modifications in the system layout. But at this point of the development phase, the back-tracking of changes from the performance model to the implementation is very costly, if not impossible. Especially because modifications usually cannot be made automatically. Non-formal backward changes from the performance model further rise the danger of inconsistencies.

In [13] an integrated software development approach with Petri nets is presented. There an automatic generation of a Petri net model from the data flow diagram is described for a special kind of data flow diagram, namely IDEF0 [8]. The applied Petri net concept are hierarchical colored Petri nets. The automatically generated Petri net is used in order to get insight into the system and deduce qualitative properties of the system. Moreover the Petri net serves as base for the implementation, by refining the Petri net, until the Petri net simulator fulfills the functionality of the software to be implemented.

Why not use Petri net also as first specification language instead of IDEF0? (The question raised by [13] is: why using IDEF0 instead of Petri nets). The integration of performance evaluation is not considered in this concept, it is only described that at the end of the development process the performance of the software is measured at the implemented system.

We imagine an approach, that enhances the methodology presented in [13] by methods for integrated development and analysis of a performance model.

3.1 Necessary Features

Figure 5 describes a hint towards a methodology for integrated software engineering, that supports specification, implementation, qualitative and performance analysis as well. In comparison with the procedure used in the case study (fig. 4) it is much more straightforward and efficient. The development of the formal model and specification of performance requirements of the software are integrated into the analysis and design phase (A&D) of the software development process.

Already in the informal specification of the system timing estimations and performance requirements are defined. At this point, timing information about existing components can be exploited. This might be software, that communicates with the system to be developed, as well as existing software that is to be reused.

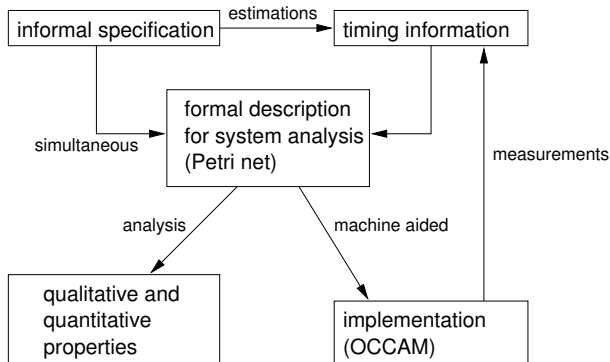


Figure 5:

The development of the performance model is integrated in the formal specification of the system, changes in either concept result in automatic modification of the other model. Hierarchy as used in common informal specification techniques (such as DFD) should be supported.

The resulting integrated formal specification and performance model can then be used for both qualitative and performance validation. The derivation of the implemented code from the formal specification should be automated as far as possible.

3.1.1 Requirements

In this section we list in detail, what requirements a modeling and specification concepts must fulfill in order to be suitable for our vision.

The concept

1. must support software modelling/development.

Elements used in software engineering (such as data-types, method invocation, etc.) should be supported.

An important aspect is the possibility of deriving implemented code out of the specification. This should be automated as far as possible. Like in integrated software development environments, at least the head and input/output data of methods should be generated, if the automatic building of the body is not possible.

The generated code then serves as base for further refinement/completion of the implementation. Perhaps use UML as base concept, because it is the most wide spread formalism used in software engineering, and also steps toward integration of performance evaluation have been undertaken. ([10, 12, 11]) Otherwise, there exist Petri net Tools, that can produce Java-Code out of a Petri net (Renew, J-THOR-Nets), or C++ Code (THOR-Nets).

2. must allow hierarchical modelling

The definition of the software with data flow dia-

grams is made by defining the main processes of the system and the data flow between these processes in the top level diagram. The processes are refined in further data flow diagrams which can also be refined.

This top-down approach with stepwise refinement has proven to be useful for system engineers and should also be supported by the in our concept. During the hierarchical development of the informal specification a formal model should be developed in parallel to maintain consistency between formal and informal specification.

3. must have timing concept (deterministic/stochastic)

As crucial requirement for performance analysis of the software system, the used modeling concept must support timing specification (such as duration of procedures), definition of timing requirements during the specification phase (e.g. maximal duration of procedures) and of course offer methods for performance analysis.

The timing concept must also be hierarchical, in order to be able to make rough performance estimation at an early phase.

4. must include data types/individual tokens

Individual tokens allow the representation of data-structures and timing that depends on the object to be handled. Individual tokens furthermore simplify the automatic code-generation, if data-types of the destination implementation language can be used.

3.1.2 Possible Solutions

We consider a graphical execution of the model as crucial feature, as help for the developer of the model as well as communication aid between developer and system engineer. Therefore we examined a number of Petri nets dialects for their usability for our purpose.

We examined a number of established Petri net concepts and their properties with respect to our requirements (Amongst them GSPN, PNiq [2], CPN [6], THORN [5]). The Petri net class fitting closest to our requirements are THOR-nets [5]. THOR-nets integrate a hierarchy resembling to method invocation in programming languages. The enabling of a hierarchical transition invokes the dynamic generation of a subnet, that is executed and deleted afterwards. Token can be of C++ type and if it is desired, a C++ code can be generated, executing the behaviour specified by the THOR-net. The before-mentioned properties predistinct THOR-nets for modeling of software.

But qualitative analysis methods for UML or high-level Petri nets are not very well developed, related to the complexity introduced by object-oriented data-structures. Related to the proof of correct-

ness of software, finding a solution for less complex questions still seems very hard.

The integrated system development environment has to offer some crucial features. Methods for maintaining consistency between different formalisms must be provided. For that purpose, connections between parts of different formalisms have to be established.

4 CONCLUSION

In this paper, we described an important problem, inspired by a case study. At the moment, software engineering and software analysis are done separately (if done at all). But the deployment of software in more and more places in our everyday-life necessitates the software to be engineered quickly in order to contribute to rapidly changing requirements, furthermore software must be safe and convenient to use (no software crashes, no rebooting anymore, quick response time) otherwise it will not be accepted by the users.

Software is usually specified using hierarchically approaches, which can generate code from the specification e.g. in UML. The hierarchy can also be used to deduce other models (for performance analysis, or deadlock analysis) from a higher level of abstraction, because analysis of the implemented software is impossible. Thus abstract models for performance analysis and qualitative analysis are necessary, their integrity and consistency with the original (changing!) specification must be guaranteed. Another factor that is to be considered, are hardware specific preliminaries, which have to be included into the performance models somehow.

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] Becker, M. und H. Szczerbicka (1998a, July). Combined Modeling with Generalized Stochastic Petri Nets including Queueing Nets. In *Proceedings of the 14th Performance Engineering Workshop*, Edinburgh, UK, pp. 48-62.
- [3] B. Buth, M. Kouvaras, J. Peleska, H. 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.
- [4] DASA Daimler Benz Aerospace AG, *DMS-R FTC SW DDD vol. 1,2*, Doc. DMS-R-RIBRE-DDD-0001, 1996.
- [5] H. Fleischhack, U. Lichtblau, Ulrike, M. Sonnenschein, R. Wieting, Ralf: *Generische Definition {hierarchischer} {zeitbeschrifteter} {höherer} Petrinetze*. Technical Report Nr. AIS-13, University of Oldenburg, 1993
- [6] K. Jensen: *Coloured Petri Nets, Volume 1 Basic Concepts*, Springer Verlag, Berlin 1992
- [7] C. Lindemann: *Performance Modelling with Deterministic and Stochastic Petri Nets*, Chichester, John Wiley and Sons, 1998.
- [8] D. A. Marca, C. L. McGowan: *SADT*, McGraw-Hill, New York, 1988
- [9] E.-R. Olderog: *Operational Petri Net Semantics for CCSP*, In Grzegorz Rozenberg (Ed.): *Advances in Petri Nets*, Berlin, Springer LNCS 266, 1987, pp. 196-223.
- [10] R. Pooley and P. King, *The Unified Modeling Language and Performance Engineering*, in: *IEE Proceedings-Software*, 1999, pp.2-10.
- [11] R. Pooley, *Using UML to Derive Stochastic Process Algebra Models* in: *UKPEW '99*, Proceedings of the Fifteenth UK Performance Engineering Workshop, 1999.
- [12] P. King and R. Pooley, *Using UML to Derive Stochastic Petri Net Models*, *UKPEW '99*, Proceedings of the Fifteenth UK Performance Engineering Workshop, 1999.
- [13] V. O. Pinci, R. M. Shapiro: *An Integrated Software Development Methodology Based on Hierarchical Colored Petri Nets* In: *Lecture Notes in Computer Science Vol. 524*, Springer Verlag, Berlin, 1991, pp. 227-252
- [14] H. Schlingloff, L. Twele: *DASA Project FTC - Stochastic Load Analysis of AVI*, University Bremen, TZI-BISS, Internal Report, 1998.
- [15] L. Twele, H. Schlingloff, H.Szczerbicka: *Performability Analysis of an Avionics-Interface*. In: *Proceedings of the 1998 IEEE Conference on Systems, Man and Cybernetics*, 1998.
- [16] G. Urban, H.-J. Kolinowitz, J. Peleska: *A Survivable Avionics System for Space Applications*. In: *Proceedings of the FTCS-28, 28th Annual Symposium on Fault-Tolerant Computing*, Munich, June 23-25, 1998, pp. 372-381.