greater than that of sequential case because the commands which are being processed by the initial modules still have to wait to be uplinked when a command being transmitted experiences an uplink_failure and is being retransmitted. Although this difference is not much (see figure 4), for a particular command the pipelined case might take a lot longer to uplink the command due to failures in system modules and uplink failures of the previous commands as well as itself. Also when realtime commands are read from a preplanned script then the rate of input commands would be greater and this would make the commands wait longer than usual in the pipeline case. This also adds to the risk involved for the pipelined execution. Since under real time conditions a command might miss the deadline for uplinking which may result in hazardous conditions. This needs to be taken care of in the system design.

5 Conclusions and Lessons Learned

This paper presented an example for generating formal specification models based on Colored Petri Nets. The models are generated from specifications based on Structured Analysis and Real Time (SART). One of the important characteristic of the methodology used is scalability. It can be applied to large scale distributed systems. The example presented is based on a component of the Flight Operation Segment of NASA's Earth Observing System. Results of performability analysis of this model are discussed. There were several lessons learned from the above application. Some of these are described as follows.

1. When software is developed to implement complex systems for real time applications, allowing concurrency in execution of different subsystems is an important design decision. There is a need for tools supporting the design and verification of such software. Colored Petri Nets provide an effective way of modeling such parallelism. As discussed in this paper the analyst can make use of the Colored Petri Net notation to model the concurrency in execution of different subsystems.

2. For the EOS Commanding model, the concurrent execution of the system components in the pipelined specification improves the performability of the system but it also adds to the complexity and risk factor of the system specifications [10].

3. In our methodology of verification of fault tolerance requirements, code segments were used for simulating the failure and repair activities of system functions as in the case of BSRC and Verify_Command functions of the EOS Commanding model. Code segments can also be used to inject faults into the module behavior by changing the module output to correspond to a fault based on a certain probability. Different algorithms for a system function can be implemented using code segments, so that the analyst has a handy prototype for testing fault tolerant specifications.

4. Control Specification in the Structured Analysis domain can have ambiguous or undefined attributes. The analyst should collaborate with the developer to correct these attributes during the formal model development process.

6 References

[1] Hooker S., Lockyer M.A., Fencott P.C., "CASE Support for Methods Integration: Implementation of translation from a structured to a formal notation", Proceedings of the Methods Integration Workshop, Leeds, 25-26 March 1996, Springer-Verlog.

[2] Amoroso E. G., "Creating formal specifications from informal requirement documents", ACM SIGSOFT, Software Engineering Notes, Vol 20 no 1, pp 67-70, Jan 1995.

[3] Fraser, M.D. & Kumar, K. "Informal to formal requirement specification languages: Bridging the gap", IEEE transactions on Software Engineering, pp 454-, Vol. 17, No. 5, May 1991.

[4] Pezze M., Elmstrom R., Lintulampi R., "Giving Semantics to SA/RT by means of High-Level timed petri nets", The international journal of time critical computing systems, Vol. 5, no 2/3, May 1993.

[5] "Automatic translation of SA/RT to high level time Petri nets" Espirit report, URL:http://www.ifad.dk/projects/iptes-ifad.html, IPTES-PDM-17-V2.3. 1994

[6] Jensen K., "Coloured Petri nets: basic concepts, analysis methods and practical use", Springer-Verlag, Berlin; New York April 1992.

[7] Functional and Performance Requirements Specification for the Earth Observing System Data and Information System (EOSDIS) Core System. Revision A and CH-01.

[8] Flight Operations Segment (FOS) Requirements Specification for the ECS Project, Volume 1: General Requirements, November 1994. By Hughes Applied Information Systems.

[9] K. Lateef, H.H. Ammar, V. Mogulothu, T. Nikzadeh, "A Methodology for Verification and Analysis of Parallel and Distributed Systems Requirement Specifications", in Proceedings of the 2nd IFIP International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE-97), IEEE Computer Society, May 1997.

[10] H. Ammar, T. Nikzadeh, and J. B. Dugan," A Methodology for Risk Assessment of Functional Specifications of Software Systems Using Colored Petri Nets," in Proceedings of the 4th International Software Metrics Symposium (Metrics'97), IEEE Computer Society, Nov. 1997.

Thus the average response time for the pipelined case is almost equal to that of the sequential case, but if a sequence of preplanned realtime commands is read from a script of commands then the average response time of a command will be much larger than the sequential case. This is because the input rate would be higher and there will be a build up at the communication channel. The commands have to wait for longer times at the channel for being uplinked. This is an added risk in the pipelined design since some of the commands will have a large waiting time. These commands may violate the deadlines of uplinking and execution because of this delay.

Performability analysis of the pipelined design:

In this case, the commanding model was simulated

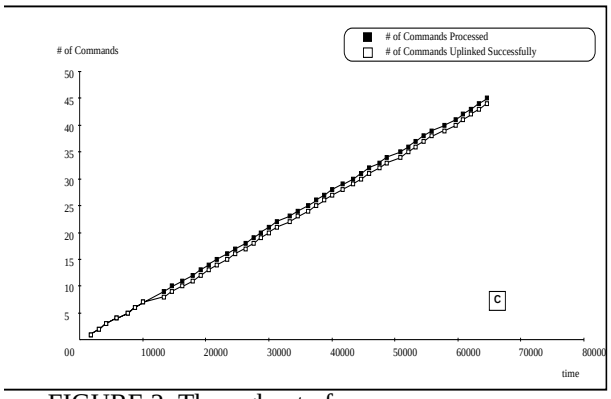


FIGURE 2: Throughput of
Scenario 5 : Sequential Execution under Failures in Communication and Fault Injection

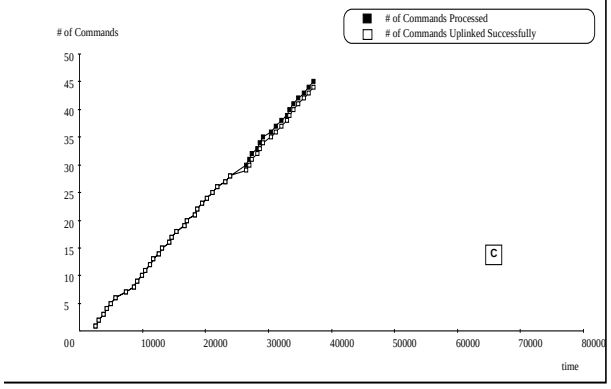
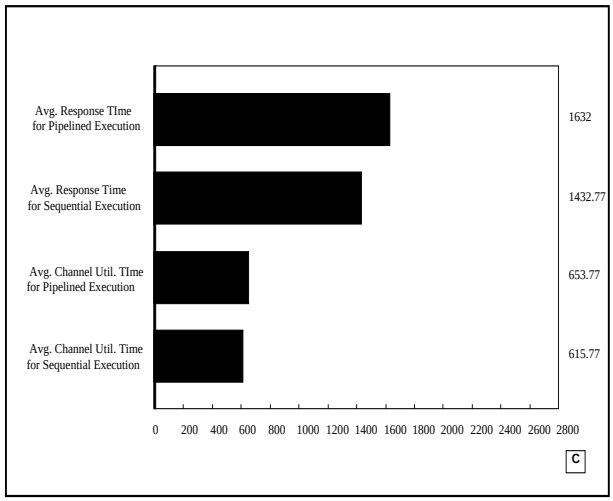


FIGURE 3: Throughput of
Pipelined Execution under Failures in Communication and Fault Injection

with faulty behavior in the system. This was accomplished by simulating the effects of failure and recovery in the system functions such as BSRC and Verify_Command. The failure and recovery activities of these modules were simulated by adding CPN ML code in the code segments associated with the respective transitions to simulate the activity of causing a failure with which an estimated recovery time is attached. The degraded performance of the system under failures and repairs is observed in the simulation. This scenario is simulated for a sequential design of the system and also for a pipelined design and the relative performance is evaluated.

The throughput of commands for the pipelined execution under failures of the system modules is again larger when compared to the throughput for sequential execution under the same conditions (see figures 2 & 3). This indicates that the pipelined design has improved the system performance under faulty conditions also. However, the throughput of the pipelined model should reduce when commands are flushed from the pipeline due to an uplink or execution failure. The response time per command in both pipeline and sequential execution is greater than (almost 2.5 times) the channel utilization time per command. When there are failures in the system modules then the commands experience a delay in processing and there will be a decrease in the build up at the communication channel. When the faulty modules like BSRC or Verify_command fail and the system takes some time to recover to normal execution, the communication channel can uplink the commands which were already processed and verified.



Sequential Execution under Failures in Communication and Fault Injection
Pipelined Execution under Failures in Communication and Fault Injection

FIGURE 4: Response time & Channel utilization time per command for Sequential & Pipelined execution under failures

It is also observed that the processing time for each command is not the same throughout and that some of the commands are lost without being uplinked. Whenever there is a loss of command we can observe this by the widening of the gap between the lines representing the total commands processed and total commands uplinked (see figures 2 & 3). The average response time (the response time per command) for pipelined execution is

The Controller enables the Transmission_Command function to uplink the next command. When the uplinking of a command is done then the Next_Command token is set to true so that the next command can be uplinked (Transition T13 puts a token in the Next_Command place). When a command is being uplinked the Controller enables the Count_Transmission_Number function (transition New_Com fires).The transmission command function can be fired for retransmitting a command when an uplink failure occurs. The retransmission is done only when the Retransmission_Command token is set to Enable and when the Transmission_Limit_Reached is not false.

When the Spacecraft_Command_Status token received from Evaluate_Spacecraft_Command_Status function is set to Uplink Failure then a retransmission of the command is attempted. The operator is also notified of the Uplink Failure. If there was no failure then the Controller goes back to the Wait_Command state to process the next command (transition T16 fires).

The code segments of the transitions update the number of Commands processed and the number of Commands uplinked and collect the statistical data for the execution.

The CPN model for Commanding was validated using the formalism of Petrinets and through the simulation of this model several inconsistencies were found in the Teamwork model of Commanding.

4 Performance Analysis of the Commanding Component

This section discusses the performance analysis carried out on the Commanding component of the EOS system and presents the results and conclusions of the analysis.

The following scenarios were simulated to assess the performance and/or performability of different execution profiles of the Commanding model.

Performance under normal sequential execution:

The Commanding model was simulated to analyze the performance under favorable conditions. The timing behavior of each module was specified and it was assumed that all the modules function normally. The simulation of this scenario produces measures on the throughput and total execution time of the operator commands. The average response time for a command and the average channel utilization time are also calculated. The average channel utilization time was almost equal to the average response time. This indicates that the bottleneck in the system is the communication channel. This is because the time taken to build, validate & verify a command is comparably less than the time taken for uplinking of the command and downlinking of the command_status. The total execution time for a sequence of 45 operator input commands was more than the sum total of the response times for these commands because of the time taken for the input of the commands. The response time per command was constant, since each command encounters the same conditions.

Performance under pipelined normal execution:

In this case instead of processing one command at a time, a sequence of operator commands are pipelined through the system. Several functions are concurrently active to process the command sequence.

The flexibility of Colored Petri Net notation to express the control flow of a system greatly eases the anlayst's efforts to design alternate specifications and explore the system behavior under such specifications. The analyst may need to make minor modifications to come up with a different specification of the system. The State Transition Diagram specification of TeamWork is limited in the sense that it does not allow the specification of a parallel design without the introduction of many more states and transitions making the system too complex to visualize and analyze. The specification of the pipeline design in the Design/CPN model is almost identical to the sequential design. The minor modification that was done in the EOC controller is an addition of an arc from the Transition T-10 to the State Wait_Command (see figure 6). In the sequential scenario the when Verify_Command processes the output of Build_Spacecraft_Realtime_Command, all other processes are inactive and the next Operator_Command_Input will be processed only after the current command is transmitted. In the pipeline design the firing of transition T-10 deposits a token in the Wait_Command state as well as Verify_Command state. This enables the Build_Spacecraft_Realtime_Command to process the next Operator_Command_Input even while the previous command is still being processed. This is propagated to all other processes down the line. Thus a pipelined execution of the system is effected.

The performance improvement under this pipelined design is observed. The throughput of commands for the pipelined execution is double the throughput for sequential execution under the same conditions. The average response time is only slightly greater than the sequential case. The response time per command was constant throughout the execution of the simulation. This is because the input rate of commands is low considering the time taken for the operator to input the command.

performed by the Commanding module are listed below according to the NASA requirement specifications [7],[8].

- Generate and verify real-time commands. This is accomplished by the functions "Build_Spacecraft_Realtime_Command" and "Verify_Command" [7].
- Merge and uplink the pre-planned and real-time commands to EDOS. The functions "Merge_Command" and "Transmission_Command" are responsible for this job [7].
- Receive and evaluate the command status. This is done by functions "Evaluate_ Spacecraft_ Command_Status" and "Receive_ Command_ Status_Data" [7].
- The automatic retransmission is also provided when an unsuccessful transmission occurs. This is managed with the help of the function "Count_ Transmission_ number" [8].

The Build_Spacecraft_Realtime_Command function generates the spacecraft realtime command based on the operator command input and the pre-planned command script. Verify_Command checks the authorization level of a command and determines whether a specific command is critical based on its definition [8]. Merge_Command merges a Valid_Realtime_Command and a Valid_Preplanned_Command. Transmission_Command receives Uplink_Data_ Stream from the Merge_Command sub-function and sends it to the space crafts as a Spacecraft_Uplink_Data. Count_Transmission_Number controls the retransmission efforts needed when the data received from the space craft indicate that the command has been rejected. Receive_Command_Status_Data is used to monitor the status of data received by the space craft. The "Evaluate_Spacecraft_Command_Status" function verifies the successful receipt and execution of all commands by the spacecraft [7].

3 CPN model of the Commanding subsystem

The model of the Commanding subsystem as described in the previous section was built using the CASE tool TeamWork. The Teamwork model of Commanding was translated to the Design/CPN environment for Dynamic Analysis. The translation was preformed by mapping the semantics from Teamwork to Design/CPN as described in [9]. The Hierarchy page of the CPN model is shown in figure 1. The CPN page corresponding to Commanding DFD is shown in figure 5 and the CPN page corresponding to the Controller for Commanding is shown in figure 6. The CPN model mapped from the Teamwork model needs to be completed by adding the missing semantics needed for dynamic analysis. CPN Meta Language code is written to map the outputs from the inputs. The information needed to implement a particular scenario is also added to the model. Thus the model is customized for each simulation and the behavior of the model is analyzed under different scenarios.

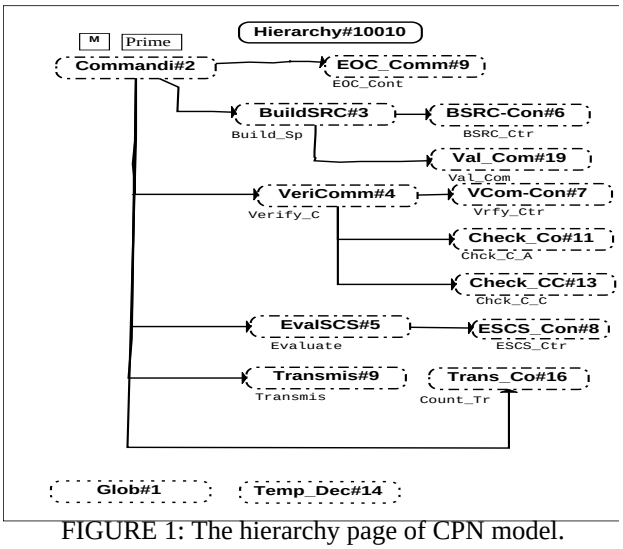


FIGURE 1: The hierarchy page of CPN model.

The CPN page corresponding to the Controller for Commanding is shown in figure 10. It contains the control specifications for the Commanding module as a whole. It produces the tokens necessary for the invocation of the functions of the Commanding module.

The transition T1 is enabled by the presence of the token START. When the transition T1 fires the Controller goes to the state Wait_Command and waits for the input of a command. The function init() is invoked which initializes the statistical variables needed for the dynamic analysis.When an Operator_Command_Request token is received by the Controller it goes to the state Build_Spacecraft_Realtime_Command and enables the BSRC module (transition T9 is fired). When the BSRC module is done processing the request the Controller enables the Verify_Command function (transition T10 is fired). If the Command_Authority_Violation (CAV) is false and the Command is Non_Critical then the Controller enables Merge_Command (transition T4 fires). If the Command is Critical then the operator is informed (transition T5 fires) and upon receipt of a positive response from the operator goes to the Merge_Command state (transition T7 fires).

When Merge_Command function is done and the Uplink_Required token is set to true then the Controller goes to the Uplink_Command state (transition T11 fires).

Performability Analysis of the Commanding Component of NASA's Earth Observing System**

V. Mogulothu, H. H. Ammar, K. Lateef, T. Nikzadeh, and Z. Miao
Department of Computer Science and Electrical Engineering
West Virginia University
P.O.Box 6104, Morgantown, WV 26506-6104

Abstract

This paper deals with the problem of performability analysis of Parallel and Distributed Systems (PDS) specifications originally developed using a CASE tool. The specifications are mapped from a Structured Analysis Realtime (SART) environment to a rigorous notation based on Coloured Petri Nets (CPNs). CPNs are used to model and analyze concurrency in the specification and design phases. Dynamic simulations of CPN models are conducted for performance/performability analysis. A model of a large industrial scale PDS is presented to illustrate the usefulness of this approach. The model is based on a component of NASA's Earth Observing System (EOS).

1 Introduction

The objective of this work is to develop methods and techniques for generating verification and analysis models from notations used for Parallel and Distributed Systems specifications. The resulting verification models can be subjected to extensive and exhaustive verification of the requirement specifications.

This paper presents an application of the methodology developed by us to integrate a CASE environment based on SART (Structured Analysis with Real Time) notation and CPN (Coloured Petri Nets) based verification environment. The methodology of integration of these tools is explained in detail in a separate paper [9]. Semantics mapping rules are used to map SART objects to corresponding CPN objects. A model of a large scale parallel and distributed system based on requirement specifications of NASA's EOS (Earth Observing System) was developed to illustrate the scalability of our methodology. This paper focuses on the application of our methodology for verification & analysis of this model and presents the results of the dynamic analysis of this PDS model.

** This work is supported in part by a grant from NASA Goddard to West Virginia University under Contract No. NAG 5-2129, and by the DoD grant No. DAAH04-96-1-0419, monitored by the Army Research Office.

Background

Informal specification languages use combination of graphics and semiformal textual grammars to describe and specify software system requirements [2], [3]. These languages tend to be imprecise and ambiguous. Hence there is clearly a need to use formal specification languages for the requirements analysts domain. Colored Petri Nets (CPNs) can be used for software requirements and design specification. It is especially useful in rigorous analysis of the dynamic behavioral properties such as concurrency analysis, performance analysis, safety, and reliability analysis. This work shows the applicability of CPN based analysis to large scale models.

This paper is organized as follows. In Section 2, we briefly describe the SART model of the Commanding subsystem of EOS. In Section 3, the corresponding CPN model is described. In Section 4, the performability analysis results of the CPN model for a set of scenarios are.presented. Section 5 presents the conclusions and lessons learned.

2 Description of the Commanding subsystem of EOS

The Earth Observing System (EOS) being developed by NASA is a large scale Parallel and Distributed System. Based on the requirement specifications, and scheduling scenario, it was observed that the commanding module plays an important role. A model of the Commanding Subsystem was built based on the requirement specifications of NASA. The major tasks