From Graphs to the Science Computer of a Space Telescope

The power of Petri Nets in Systems Engineering *

Rafal Graczyk[1][0000−0003−4570−3431], Waldemar Bujwan[2], Marcin Darmetko[2], Marcin Dziezyc[3], Damien Galano[4], Konrad Grochowski[3], Michal Kurowski[3], Grzegorz Juchnikowski[2], Marek Morawski[2], Michal Mosdorf[3], Piotr Orleanski[2], Cedric Thizy[5], and Marcus Völp[1][0000−0002−8020−4446]

[1] Interdisciplinary Centre for Security, Reliability and Trust, University of Luxembourg, Esch-sur-Alzette, Luxembourg
rafal.graczyk@uni.lu
[2] Centrum Badań Kosmicznych Polskiej Akademii Nauk, Warsaw, Poland
piotr.orleanski@cbk.waw.pl
[3] N7 Space, Warsaw, Poland
mmosdorf@n7space.com
[4] European Space Agency, Noordwijk, The Netherlands
damien.galano@esa.int
[5] Center Spatial de Liege, Liege, Belgium
cedric.thizy@uliege.be

Abstract. Space system engineering has to follow a rigorous design process to manage performance/risk trade-offs at each development stage and possibly across several functional and organizational domains. The process is further complicated by the co-development of multiple solutions, each contributing differently to the goal and with different trade-offs. Moreover, the design process is iterative, involving both changing requirements and specifications along the different ways that lead to the set goal of the mission. The above requires rigorous modeling that, in addition, must be easily extendible and maintainable across organizational units. On the example of the PROBA-3 science computer (instrument control unit, CCB DPU), we show how Petri Nets can serve as such a simple-to-maintain, holistic model, combining finite-state characterizations with dynamic system behavior caused by hardware-software interactions, to express the component-state dependent end-to-end performance characteristics of the system. The paper elaborates on how the proposed Petri-Net-modeling scheme allows for system architecture optimization that result in safely reduced technical margins and in turn substantial savings in components costs. We show that performance metrics, obtained from simulation, correlate well with the real performance characteristics of the flight model of PROBA-3's science computer.

* The Proba-3 ASPIICS project is developed under the auspices of the ESA's General Support Technology Programme (GSTP) and the ESA's Prodex Programme thanks to the sponsorships of seven member states: Belgium, Poland, Romania, Italy, Ireland, Greece, and the Czech Republic. This work is also supported by the Fond Nationale de Recherche, Luxembourg, through grant CS20/IS/14689454 - HERA.

Keywords: Petri Net · Systems Engineering · Performance Modeling · On-Board Computer · Scientific Payload

1 Introduction

While space missions may have a clearly specified goal, there are many ways of achieving it. Not all solution pathways are equal: some entail higher implementation costs, others are limited in the performance they achieve, introduce unacceptable risks, or utilize spacecraft resources too extensively. Designing spacecrafts and their subsystems is a gradual, iterative, incremental process and has to happen concurrently across functional and organizational boundaries. Gradual advances and small increments lead the project from initial requirements and specification up to construction, test and deployment. At all times, subsequent steps in the development process have to be evaluated from the perspective of risk mitigation, reducing unknowns and removing technical obstacles.

Classically, space projects in the ecosystem of the European Space Agency (ESA) progress through the following seven phases:

Phase 0 - *Early Conceptualization*
Phase A - *Mission Definition*
Phase B - *Preliminary Technical Design*
Phase C - *Critical Technical Design*
Phase D - *Flight Equipment Manufacturing, Assembly, Integration and Test*
Phase E - *Deployment in Flight*
Phase F - *Disposal*

Subsequent phases thereby build upon design decisions, trade-offs and experiments from earlier phases and the high formalization of the process has the goal of reducing programmatic risk. Examples of the latter include overruns on costs or of the schedule, which are often caused by wrong requirements or assumptions, that were discovered too late for quick and easy fixes. This calls for the deployment and active use of all possible means for simulating, testing, and verifying assumptions, solutions and decisions, as early as possible and sensible.

Space projects often experience a clash between opposing forces: on the one hand, there is a demand for large technical margins on designed subsystems to ensure with a very large likelihood that evolving requirements will be met; on the other hand, there is the need to optimize the design, reduce the typically very large costs incurred by an extensive use of high reliability components and resources (mass, energy, processing power, memory size) utlization. This dilemma drives the strong need to analyze, evaluate and experiment with the designed system to judge technical decisions and to ensure that, as the project matures, risks are indeed reduced while striving for optimal utilization of available resources.

In this paper, we would like to share our experience of using Petri Nets in practice for modeling the processing performance of the Instrument Control Unit for Coronagraph, a sun observing telescope, which has been the primary payload of ESA's Proba-3 mission. Application of Petri Nets has been essential in the early stages of the project (Phases B and C) where architectural decisions and

optimization attempts had been confronted with the need to prove the system would be capable of meeting required performance.

Aside from our experience report, this paper contributes:

- the design of Coronagraph's instrument control unit architecture,
- an initial Petri Net model and simulation exploring the system's design space,
- an advanced Petri Net model correlated with flight equipment,
- a concept for the hardware-software co-design of embedded systems beyond the space domain, and
- the tangible system optimization results we achieved applying the above methods.

Parts of this work (Sections 3.1, 3.2 and 4) have been previously published as a PhD thesis [14].

2 Background

Modeling and simulation of space systems, be it avionics, command and data handling systems, or mission payloads, is crucial for their correct high-level design, for their technical specification and, finally, for an initial verification of their key requirements. Early modeling and simulation helps making tested functionalities traceable for future user needs and for ensuring a common understanding on what is actually being built and why. Modeling and simulation of the system design allows for idea feasibility checks, architecture trade-offs, initiates early prototyping, brief requirements verification, and de-risking any critical aspects of the system [19, 4, 6, 26, 15, 7]. Two modeling approaches are frequently used for spacecrafts, which we describe in the following.

2.1 Analytical and mathematical models

Every system operation can be modeled by means of affecting the energy, mass or information flow (in sense of enabling or disabling the flow, accumulation or transformation of resource). Modeling along those lines starts with capturing the basic building blocks of the system, defining transfer functions and their transient response, which leads to a description of components in form of a mix of algebraic and differential equations. From these equations, one further derives system balancing equations to describe the system dynamics in simplified form, by linking storage, flow or transformation of mass, energy or information. Balance equations are a valid method for modeling fluid systems, attitude determination and control, and some electrical systems (e.g., power systems) are conveniently modeled by means of flow and storage. In this approach, the ultimate goal is to describe the state space of a system by formulating state equations, which define the internal state of the system, and output equations, which define the system response as a function of its current system state and received inputs.

2.2 Dynamic and functional models

While flow or variation of physical attributes are conveniently expressed in terms of differential equations, the modeling of computation or data flow needs to take into account the stateful and event-driven nature of the data processing and control systems. State charts or graphs and finite-state machines (FSMs) are mathematical models used to represent such computation in classic logic devices or, when extended like in UML [16], SDL [2] or AADL [9], to represent software execution paths and software component relations in an embedded system [18]. FSMs can be in only one state at a time. External events induce state changes. This makes FSMs a convenient way of representing a single activity over time and showing the dependency of modeled systems on transition triggering conditions. The operation of one or more FSMs over time, and their interaction can be extended into flow networks. A flow network is a particular example of a directed graph where each edge has its maximum capacity and has some temporary flow value. Flow values cannot exceed the maximum capacity of an edge. Flow has to follow the preservation rules, meaning that effective network-node inflow must be equal to outflow (with the exception of source and sink nodes). This simple methodology allows for brief analysis of the dynamic behavior of systems. Flow networks are especially useful for modeling system aspects related to transportation like electric current, liquid or heat flow or data transfer. Flow networks are useful not only for the analysis of system evolutions over time, but also for finding the maximum flow capability of the whole network. Quite often, computing and control systems have to be analyzed as an evolution over time (in continuous domain), while system state transitions occur at discrete events, when associated conditions trigger desired reactions. Such events exhibit a competition against other triggers and each one of them, typically, has its own stochastic mechanisms that govern determining a new system state. For each state transition, new events may be scheduled and previously scheduled events may be canceled. Petri Nets provide a versatile analysis framework for this kind of modeling, especially if the modeled system exhibits randomness, state-transitions, concurrency and scheduling [22, 23, 29, 13]. Some works on Petri Nets applied to space systems engineering can be found, including software modeling [21], satellite constellation modeling [8] and instrument simulation [20].

2.3 Hybrid models

Hybrid models, combining discrete and dynamic aspects, such as PDL [10, 5] and hybrid-systems state machines [25] have been investigated in the research community, but are not yet widely deployed in space system design. We therefore leave the investigation of such models as future work.

2.4 State-of-the-art modeling approaches

A representative example of a state-of-the-art modeling approach for embedded cyber-physical systems, used in ESA, is TASTE [1, 3] and its toolchain. TASTE

focuses on the software aspect of system operations and supports the creation of systems using formal methods and automated code generation. The system under evaluation shall be defined in AADL, but a large number of other tools and languages are supported as well, including SDL, VHDL, ASN.1, SCADE, and Simulink. Cheddar and MAST are used for model verification. Verified models can be converted into Ada, C, or C++ and deployed on the target hardware. The TASTE framework targets the development of safety and mission-critical communication, control and data processing systems and associated real-time applications. The goal for using TASTE is to facilitate the understanding of a specification or design to get an early executable representation of the system and allow independent testing and verification at different levels of abstractions [24].

3 The Proba-3 mission, ASPIICS Coronagraph and Instrument Control Unit

Proba-3 includes, among others, the ASPIICS (Association of Spacecraft for Polarimetric and Imaging Investigation of the Corona of the Sun) primary payload. The mission is devoted to demonstrating in-orbit the precision formation flying of two satellites. The first one produces a nearly perfect eclipse allowing the second one — the PROBA-3 Coronagraph ASPIICS — to observe the sun corona closer to the rim than ever before. To achieve that, both satellites need to keep, their distance and alignment, precisely and accurately, during the observations. The coronagraph will cover the range of radial distances between ∼1.1 and 3 solar radii, thus providing continuous observational conditions very close to those during a total solar eclipse, but without the effects of Earth's atmosphere. ASPIICS will provide novel solar observations capabilities to achieve two major science objectives in the area of solar physics [27, 11]:

1. understanding the physical processes that govern the quiescent solar corona; and
2. understanding the physical processes that lead to coronal mass ejections (CMEs) and that thus determine space weather.

The Coronagraph Control Box (CCB) is the electronic controller of the ASPIICS Coronagraph Instrument (CI). The CCB consists of a compact housing that contains:

- a Data Processing Unit (DPU), i.e., an embedded payload computer module that is capable of processing and buffering data and of executing management and control algorithms. The DPU is responsible for interacting with the Proba-3 Coronagraph Satellite on-board computer, called Advanced Data and Power Management System (ADPMS);
- a Power Conditioning Unit (PCU), i.e., a power supply module that provides all the voltages required by the CI, along with voltage / current measurement capabilities for telemetry data generation and protection circuits. The PCU is switched on at the moment when the ADPMS provides power, and supplies all units instantaneously;

– an Ancillary Electronics Unit (AEU) contains switches for ASPIICS power on and off control, as well as advanced actuation control for Filter Wheel Assembly (FWA) and Front Door Assembly (FDA) stepper motors, the Coronagraph Optical Box (COB) heater system and the ADCs to gather telemetry data. AEU functionality is controlled by the DPU;

The Data Processing Unit (DPU) of the Coronagraph Instrument is responsible for all control and scientific processing algorithms. It is built on two main components. One, is a processor (CPU, GR712RC), executing the control software (Boot Software and, target, Application Software). The other is a configurable logic device (FPGA, RTAX2000S), that acts as the processor's coprocessor, implementing all features that are not available in the processor. The processor interfaces to Flash (for boot-image and application software storage) and to SDRAM as operations memory and as storage for scientific data. The FPGA is equipped with an external SRAM acting as a cache for scientific data packet formation. The processor's main functions are:

– scientific data acquisition, processing and transfer to On-Board Computer
– flight software and operations schedule execution
– motor and temperature control, power management
– fault detection, isolation and recovery

Main FPGA functions are:

– extending the CPU features through Space Wire
– providing missing communication interfaces (Packet Wire)
– Shadow Position Sensor data acquisition and preprocessing

See Figure 1 for an overview over the DPU functional blocks and their relation to the overall system architecture.

3.1 Classic approach to performance modeling

The system architecture, as described at the beginning of Section 3, shall be evaluated during its development against the system requirements. System characteristics estimation is needed to, first, understand the feasibility of the proposed solution, and, second, to identify the system's technical margins, which are needed to understand the project's technical risks.

The classical approach for modeling complex embedded system like the CCB DPU, would be to create a simple pipeline model. Such a model assumes tiles (chunks of images taken by Coronagraph) are transferred from one buffering place to another, immediately, with maximum rate. Typically, such a model will not include any functional dependencies or blocking operations and is therefore limited to giving theoretical maximum performance estimates of a given hardware configuration of the system. It can help detect of performance bottlenecks, which are typically located at the communication interface with the lowest bandwidth.

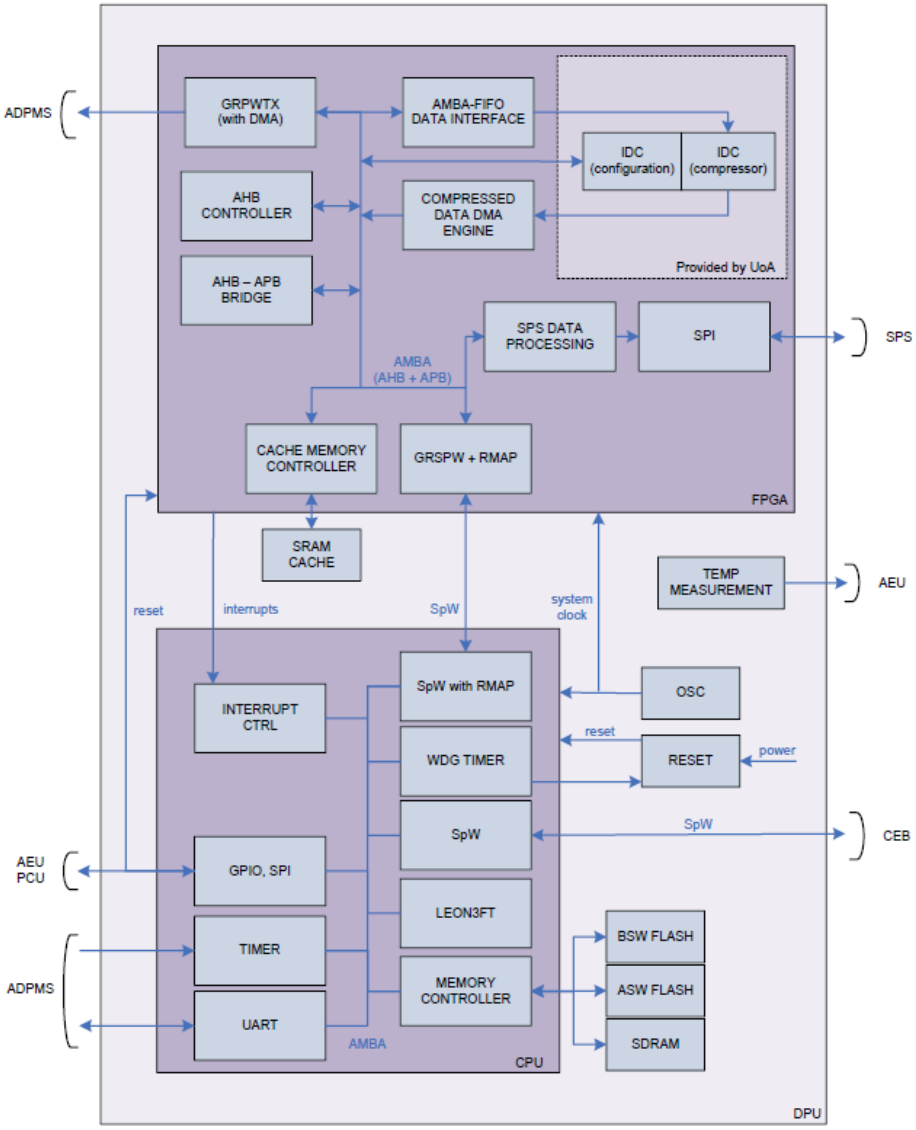


Fig. 1. Data Processing Unit (DPU) architecture

To be more specific, let us outline a simple DPU pipeline model, where the input stream of scientific data is fed through the "pipeline" over a data transfer channel. A data transfer channel consists of all sub-channels (communication interfaces) through which data is transferred. For convenience, we could express the bandwidth (BW) of each of part of the data transfer channel in terms of tiles per second (BW_{Coff} for compression off and BW_{Con} for compression on, since compressed tile is about 3 times smaller than raw). The simple DPU pipeline model is shown in Table 1. Knowing that planned system operations require the CCB DPU to be capable handling 192 tiles/s, in worst case scenario, we could immediately conclude, that, there should be no performance bottleneck present. Then, we could draw a positive conclusion regarding the implementation feasibility of PROBA-3 CCB Data Processing Unit.

Table 1. DPU pipeline dataflow model

Communication Path	BW [Mbps]	BW_{Coff} [tiles/s]	BW_{Con} [tiles/s]
Space Wire (from CEB)	50	800	800
AMBA bus (in CPU)	1600	25600	25600
Space Wire (between CPU & FPGA)	50	800	800
AMBA bus (in FPGA)	200	3200	9600
Packet Wire (to ADPMS)	25	400	3200

3.2 Tile-flow peculiarities that need to be taken into account

The pipeline model presented in Section 3.1 is simple but dangerously coarse. It can be noticed, that within the processor and within the FPGA, all activity — i.e., control and data transfer — is conducted over the AMBA bus, which interconnects all peripheral and IP-core blocks. On the processor side, AMBA is involved whenever a tile is transferred, whenever the processor cache has to be filled, during regular housekeeping, scheduling, control and health monitoring activities of the on-board software, and whenever interactions with other parts of the Coronagraph Instrument or the satellite bus take place.

The FPGA is designed and built around the same philosophy of operation, also utilizing the AMBA bus and the set of IP-cores to provide the required functionalities. However, the FPGA does not contain any processor itself. Its operation therefore crucially depends on remote configuration by the main processor (CPU). Therefore, although some bus transactions inside the FPGA (like DMA transfers) happen automatically, they have to be configured beforehand by the CPU. In consequence, the CPU remains in full control of scientific data flows, both in the GR712RC processor and in the RTAX 2000 FPGA.

The FPGA further reports back to the CPU any detected events related to the communication and Shadow Position Sensor (SPS) operation. Reporting is

via 3 interrupt lines originating from the Packet Wire block denoting successful transmission of a tile to the spacecraft mass memory, from the IDC data compression engine, denoting end-of-tile processing, and from the SPS Readout Engine block. This generates additional asynchronous events, on top of those received by the CPU directly (e.g., in relation to the communication with the satellite's On-Board Computer and the ADPMS).

Let us take a brief glimpse how the AMBA bus works to understand a major issue we have to address. AMBA (also called here AHB) is a multiplexed bus (see Figure 2). A multiplexer, connecting the master (i.e. source of data) to a slave (i.e. data sink), blocks any connection from happening until the bus is freed again. Only one type of operation can be executed at a time and by only one master at a time. That is either a tile can be transferred from Space Wire to SDRAM, or from SDRAM to the FPGA for compression, provided the FPGA is configured or interacts with the CPU. The same holds for the FPGA side AMBA bus: either the IDC compression engine (or the CPU) feeds tiles directly into the SRAM cache or tiles are sent through Packet Wire to the On-Board Computer. On a given AMBA bus (of the CPU or FPGA), each AMBA operation blocks all other operations from being executed. Introducing multiple separate buses allows increasing the degree of parallelism, but the dependence on the CPU to manage data flows continues to significantly affect system performance. This aspect is ignored in classic modeling, causing it to be potentially flawed.

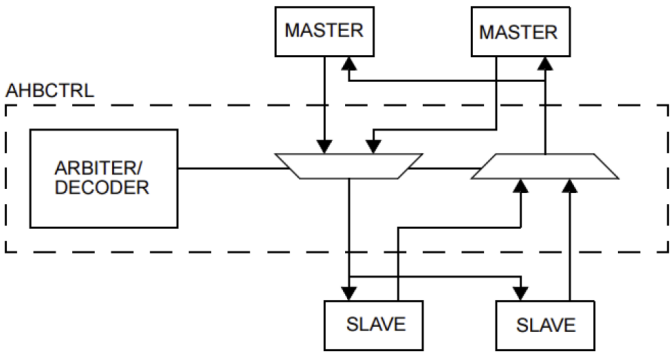


Fig. 2. AMBA AHB controller functional block diagram

4 Processing performance modeling

In order to build a Petri Net model capable of correctly characterizing the CCB DPU processing performance, we need to map DPU functional components and other aspects of the system to Petri Net (PN) primitives.

We used a Extended Deterministic and Stochastic Petri Net (eDSPN) to model the CCB DPU. Table 2 summarizes the PN primitives and the physical

reality they represent. All the models presented in the paper were built and simulated with the TimeNet software [12, 17, 28].

Table 2. Physical system element to Petri Net primitive mapping.

Physical system	Petri Net	Comment
tile, control signal	token	tokens are all the same, token role is interpreted depending on topology of places and transactions
memory buffer, cache, processing block, process mode change	place	place function is subject to abstract interpretation
buffer or communication interface capacity	inhibitor arc	arc ended with a circle, connecting place and inhibited transaction, activated when number of tokens in starting place exceeds defined threshold
communication interface	deterministic transaction	black rectangle, firing rate inversely proportional to tile throughput, enabled in presence of control token, absence of inhibiting signals
communication bus	exponential transaction	white rectangle, firing rate expected value , enabled in presence of control token, absence of inhibiting signals
software process preemption	immediate transition	black bar, controlled by guard signals

The basic idea behind the proposed DPU performance model is to convert the classic pipeline model to Petri Nets and to augment the latter with the missing dynamic aspects of the system that affect the scientific tile-flow and that characterize AMBA bus blocking.

4.1 Initial processing performance model

The proposed CCB DPU Petri Net model is shown on Figure 3. It characterizes the CCB DPU configuration where compression is enabled. In the model, the places P_CEB and P_ADPMS represent the scientific data stream source and the sink, respectively. To obtain a more flexible simulation, we introduced a parameter N (typically set to 10000) for the number of tokens (or tiles) that are to be transferred. Simulation ends once N tokens arrive at P_ADPMS. The time to transfer all tokens thereby serves as a performance indicator of the system throughput, as shown in Equation 1.

$$System\_Throughput = \frac{N}{time\_to\_transfer\_N\_tokens} \tag{1}$$

Tokens need to traverse along the whole DPU science data path to be stored in P_ADPMS. Some places along this path represent internal buffers of the communication interfaces (P_SPW[0..2]), two places represent the key buffers of the DPU system - the SDRAM memory attached to CPU (P_SDRAM) and SRAM memory attached to FPGA (P_SRAM). The IDC compression engine, assumed to process 1 tile at a time, is labeled as P_IDC. The maximum buffer size (in number of tiles it can hold) is enforced by the use of transition feedback, an inhibitor arc.

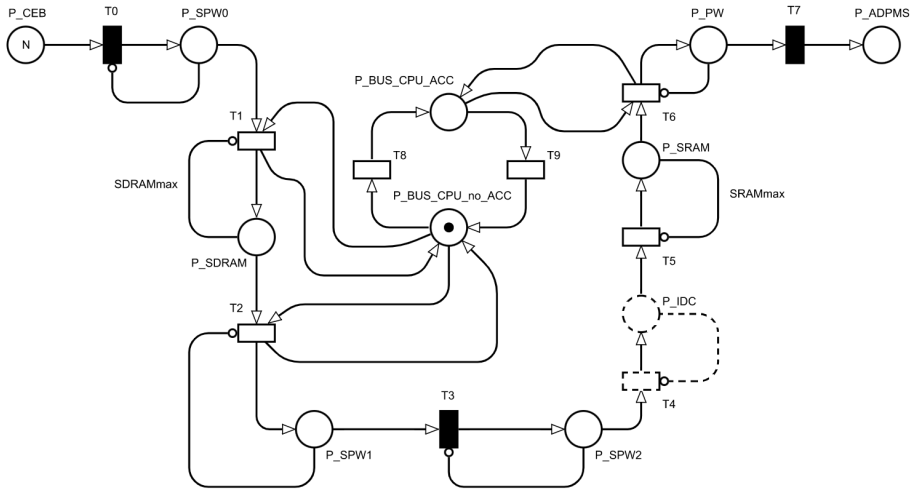


Fig. 3. Data Processing Unit (DPU) performance model. Compression engine, disabled in raw mode, marked with dashed line.

Petri Net places are connected using transitions, representing the throughput of the interfaces and buses that are involved in transferring tiles between the respective buffers and processing nodes in physical system. The transitions T0 and T3 are Space Wire interfaces from the CEB and between the CPU and the FPGA. They operate at 50 Mbps. Transition T7 is a Packet Wire interface to ADPMS and operates at 25 Mbps. Transition T1 and T2 represent the AHB transfers to and from the SRAM buffer. Similarly, T4, T5, T6 represent AHB transitions in the FPGA. The AHB bus in the processor operates at 1600 Mbps and the AHB in the FPGA at 200 Mbps. Equation 2 denotes the firing rate (i.e., expected time to fire).

$$Time\_to\_Fire = \frac{Tile\_Size}{Throughput} \tag{2}$$

There are three modes of scientific data flow processing: compression, bit-stuffing and raw. Compression and bit-stuffing make use of the IDC engine to

perform their activities. This part of the system's Petri Net model can remain the same. Only the transition firing rate needs to change in order to reflect changes in the tile throughput in the FPGA. This is because tile sizes are reduced after compression, utilizing only a fraction of the bandwidth. For raw mode the location P_IDC and transmission T4 are removed, T5 then connects directly to P_SPW2, and firing rates are modified to reflect the transition of full size tiles.

In order to model the dynamic behavior of the system, at the stage of development, where perhaps only hardware prototypes are available, but no software, simplifying assumptions have to be made. Plain throughput, as shown in the pipeline model, is not sufficient to validate architecture design in the presence of AMBA bus blocking behavior, which is introduced by competing functional chains (science vs SPS control vs maintenance vs TC/TM control). Moreover, even within the scientific data transfer functional chain, a competition for AMBA bus access occurs. This is because only one of the following two situations happen: tiles are circulated within the processor; or the processor configures and monitors the semi-automatic operation of the Direct Memory Access engines in the FPGA. To model this behavior we introduce the subnet P_BUS_CPU{∅,_no}_ACC, T8 and T9, where the presence of a token (interpreted as flow control) in place P_BUS_CPU_ACC denotes the CPU having access over AHB to perform its activities (including FPGA control) and a token in P_BUS_CPU{∅,_no} denotes that the CPU is either idle or performs activities using only it's cache memories, while the AHB is free for science data transfer. The T1 transfer (from CEB data source to SDRAM) will fire only if a token is present at P_BUS_CPU_no_ACC, which in turn blocks the T2 transfer (from SDRAM buffer to FPGA via Space Wire), which mimics the same limitations as present in the processor. Firing of T1 or T2, besides moving tokens that represent tiles of scientific data moving along the scientific datapath, will also put the flow control token back to P_BUS_CPU_no_ACC. A similar concept of operation is implemented for enabling T6, with small but important difference, that this transition can fire only when the CPU takes control over the GR712RC AMBA bus and the FPGA Space Wire link with RMAP enabled to configure the Packet Wire DMA engine.

4.2 Simulation of the initial processing performance model

Since the initial model is designed to validate the architectural considerations before actual hardware is built and software deployed on it, there is not much gain in simulating performance to evaluate overall absolute values. Obtained measures would be questioned during technical reviews as being too affected by model assumptions. Much more appealing at this early stage is leveraging the model for exploring numerous design variants and configurations, to see which of them are promising to fulfill system and, eventually, mission requirements.

To evaluate the CCB DPU architecture, one needs to show the system performance goal can be met and under what conditions. Therefore, on the one hand, simulation needs to show that the most demanding (data-intensive) observation

scenario can be served within the expected payload operation window (here 25-30% of the orbital period) and that the influence of AMBA bus blocking on the scientific data flow is tolerable. The first can be obtained by simulating the time required to transfer and process the selected amount of tokens from CEB to ADPMS, extrapolating the result to estimate the time required for transferring all tiles generated during the CME-Watch scenario ($\sim$ 2.9M tiles). The second measure can be obtained by defining $AMBA\_blocking\_ratio$ (Eq. 3) as the fraction of time during which the token is in place P_BUS_CPU_ACC in contrast to being in either in P_BUS_CPU_ACC or P_BUS_CPU_no_ACC.

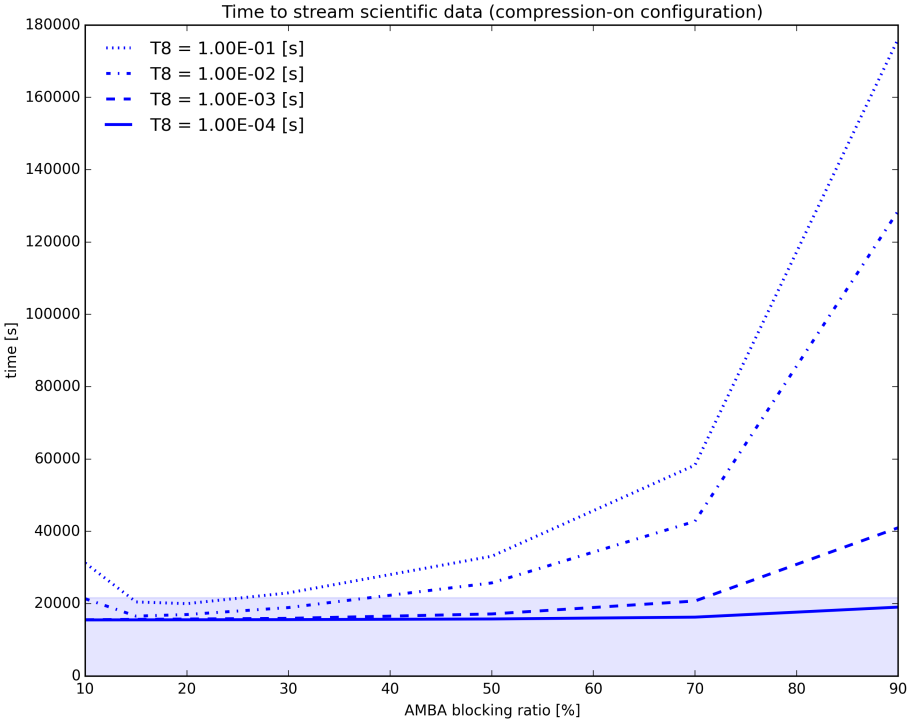
$$AMBA\_blocking\_ratio = \frac{T9}{(T8 + T9)} * 100\% \tag{3}$$

Figure 4 shows DPU's performance obtained from simulating the initial model outlined above, configured with (a) and without (b) compression. The blue-shaded area at the bottom of the figures denotes the time limit available for DPU operations: 21600 s. Scientific data streaming to ADPMS is considered successful when it finishes before this time limit. The plots show the time to stream data to ADPMS that was generated in the worst-case CME-watch scenario for different CPU AMBA bus blocking ratios (i.e., for different fractions of total time that the AMBA bus is not used for transferring scientific data tiles, refer to Eq. 3). Simulations are performed for few cases, showing fours orders of magnitude of AMBA blocking ratio granularity. The more the AMBA bus is blocked for other activities than transferring tiles from the CEB and further into the DPU and ADPMS (horizontal axes), the more time it takes to complete the whole data dump to the on-board computer. This actually confirms initial suspicions, since when tile-reception is blocked at the GR712RC, then tile inflow to the DPU is limited as well and no advantage can be drawn from the fast internal DPU interfaces.

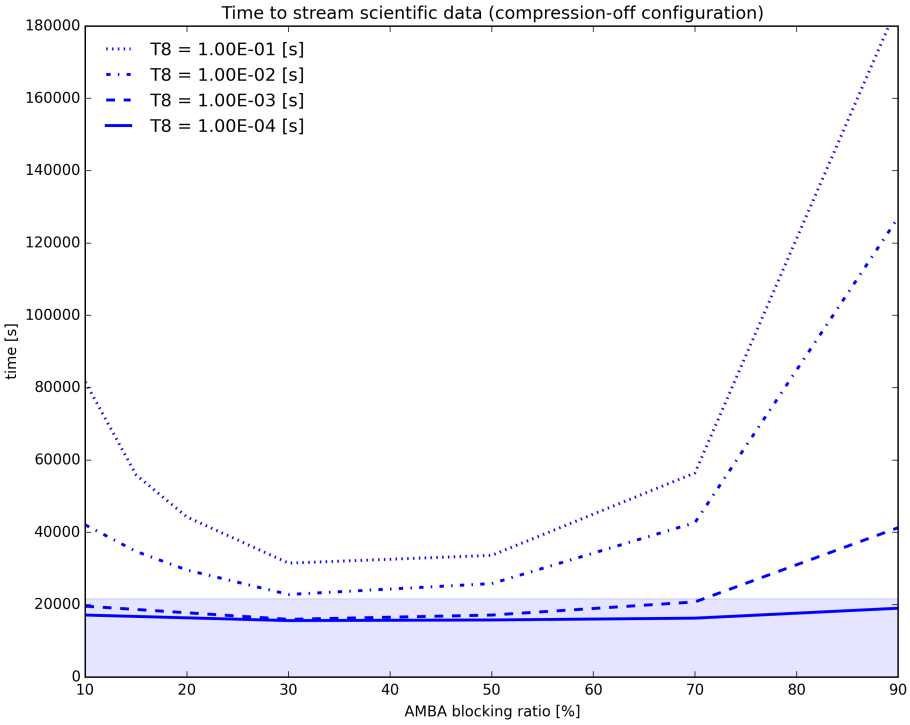
Another insight into CCB DPU operations can be obtained in relation to the impact of the CPU AMBA bus mode switching time granularity. The larger this granularity, the more time is spent in a given mode before switching and the lower will be DPU's capability to stream the data. Keeping mode switching granularity high and ensuring buffers are emptied quickly, prevents tile pile-ups and appears to be a key to achieving the required performance and drove further hardware and software codesign.

5 Implementation

Validating the architecture with the help of Petri Nets, as described in the previous section, confirms the initial assumptions and indicate that it will be feasible to implement the CCB DPU. They provided insight into the system's internal operation and allowed to better understand the involved hardware-software interactions, allowing the project to advance into subsequent phases, where the focus is on building and deploying the actual equipment.



(a) with compression



(b) without compression

Fig. 4. Data Processing Unit (DPU) performance simulation

Subsequent CCB DPU models are developed in an iterative approach, where each following model tests the assumptions, solutions and design decisions that have been made with the previous model and that are most critical. The outcome of analyzing such a refinement may require discarding certain design decisions or they may remain uncertain in the sense that still multiple options may be viable. The best viable state in the final phase is the desired system. A summary of the models built within the CCB DPU project is provided in Table 3. For clarity, only the models that are most relevant for the performance evaluation are shown.

Table 3. CCB DPU models

Model	Phase	Applicability	Purpose
Software Mock-Up	B & C	Prototyping	Used for verifying schedulability of software components
Development Model	B & C	Prototyping	Used for testing the hardware and software, verification of initial assumptions, getting acquainted with technology stack
Engineering Model	C & D	Advanced Prototyping	Built be as close to Flight Model as possible, used for testing critical functionality and interfacing with the rest of the system. Intermediate software revisions deployed.
Proto-Flight Model	D & E	To conduct the mission	Full flight configuration, materials, quality and processes, tested to acceptance levels. Flight software revision deployed.

The DPU Development Model allowed for early prototyping and was built for reducing the risks associated with technology and components selection. It further served to create first mock-ups of the critical hardware and software parts and for testing initial assumptions about operations, including first performance measurements. For example, at this stage, measurements already indicated a tile throughput of 252 tiles/s on average, for which we had to implement only part of the final software functionality.

Following the above, we conducted (as part of our Verification and Validation campaign) a performance verification of the DPU processing capabilities on a physical implementation of the models (the Engineering Model and Proto-Flight Model, shown in Figure 5). The campaign attempted to answer whether the requirements defining the system have been met, but also, what were the actual limits of the hardware and of the software scientific data processing power. The data flow test involved verifying that the rate of transfer of compressed science data from the CCB to the ADPMS memory module met the requirements set out in the system requirements specification. The scenario measured the average number of tiles processed by the CCB and sent to the ADPMS each second based

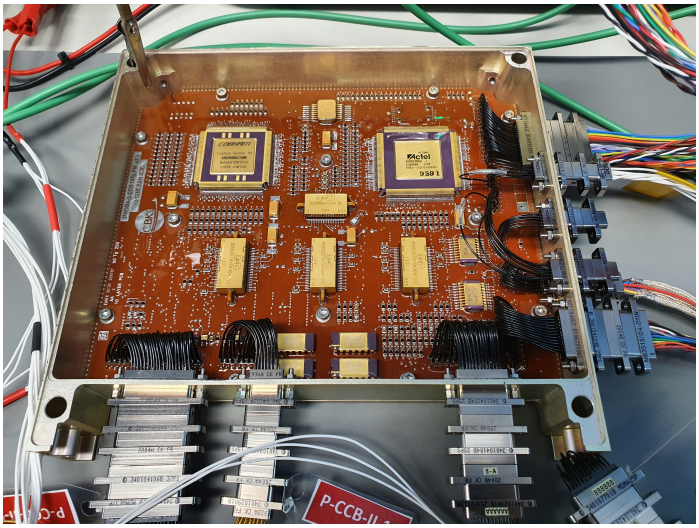


Fig. 5. Flight model of Coronograph Control Box (CCB) Data Processing Unit (DPU)

on the events logged by the software for each tile. The numbers obtained from validation test logs, collected from runs performed on the engineering model of the CCB, connected to the electrical ground support equipment, oscillated around 205 tiles per second. It therefore met the requirement of at least 192 tiles per second, as defined in the requirements specification.

However the DPU engineering model, at time of test, did not operate under the final version of flight software. Similar tests conducted with the Proto-Flight Model in its final software configuration revealed a scientific tile throughput of 233 tiles per second (all values for compression mode enabled), still well within the specification requirements.

6 Correlating the Petri Net performance model with the physical model

It is worth noticing, that the performance model and actual verification measurements were organized around different methodologies. The Petri Net performance model was targeting simulation of what is happening inside the DPU, while it is fed with a worst case expected data stream to figure out what are required buffer sizes and what control-performance bottlenecks might be encountered in the system. Modeling was performed for the purpose of reducing uncertainty (and associated programmatic risks) and for optimizing the technical margins of the system. The verification measurements, however, were targeting the discovery of actual performance limits in the system. Therefore it was fed with the maximum data stream allowed by the input interface bandwidth, measuring the resulting processing performance.

To correlate these performance results, both have to follow the same experimentation philosophy. However, since at the time of writing, the CCB underwent the integration process with the satellite platform, it was easier to adjust the Petri Net model rather than repeating the performance measurements with the payload, although the latter will have to happen anyway as part of the validation campaign of the integrated satellite.

A quick look at the Petri Net models in Figure 3 allows noticing that to replicate the performance experiments, which had been performed on the implemented equipment, it suffices to modify transaction T1, defining the data stream input rate, from 192 tiles/s to 800 tiles/s which is about the maximum that can be handled by the SpaceWire interface at a configured bit rate of 50 Mbps.

The other aspect that has to be adjusted in our models is the T8 / T9 transactions to resemble the behavior of the CCB Application Software. In the initial modeling of the CCB DPU, the expected value of the T8 and T9 transactions firing time was of main interest, but not their absolute values, which at that time would have to be guessed. The analysis had therefore focused on the relation between T8 and T9 and on the order of magnitude of the firing time (granularity of translation activation) as described in Section 4.2. Now, after the Flight Software (ASW) is available and deployed in target architecture, it is possible to estimate these values more accurately.

T8 indicates for how long the CPU manages tile-inflow to the system, receiving in the first part of the data pipeline the tiles from imager electronics (CEB) and within the CPU itself (storing in SDRAM, passing to FPGA, configuring FPGA DMAs and IDC operation). T8 is defined by execution times of ASW components summed up in Table 4.

T9 defines how long the CPU manages tile-outflow of the system in the second part of the data pipeline, pushing out the tiles stored in the FPGA SRAM to On-Board Computer(ADPMS) through PacketWire. T9 is defined by execution time of ASW components (summed up as well in Table 4).

Table 4. Relevant ASW components average execution times, modeled by respective transactions.

Transaction	Component	Average execution time [ms]
T8	*tile_related_rmap_transaction*	0.012
	tile_spw_transmission	1.250
	fpga_idc_compression	0.660
	Total:	1.922
T9	*fpga_pw_transmission*	2.73
	Total:	2.73

As for T8's and T9's expected firing times, the key architecture quality factor derived from initial modeling, AMBA blocking factor (Eq. 3) is 59% and it has

a very fine granularity (order of milliseconds) which are the exact circumstances that have been discovered as offering sufficient CCB DPU performance margins and flexibility to meet the system requirements (refer to Sec 4.2).

However, setting realistic values of T8 and T9 is necessary, but not sufficient, to ensure adequate similarity between Proto-Flight Model and Petri Nets. At this stage, we also needed to accommodate the fact that the Application Flight Software is much more complex than it was considered (and feasible for taking into account) at initial modeling time. Application Flight Software is responsible for real-time SPS sensor measurements and their delivery further down the Guidance, Navigation and Control loop. This is a 2 Hz period process and some associated processing load. Among other loads there is a telecommand decoding and telemetry packet generation, as well as, all the housekeeping and control activities and, last but not least, fault detection, isolation and recovery functions (internal monitoring).

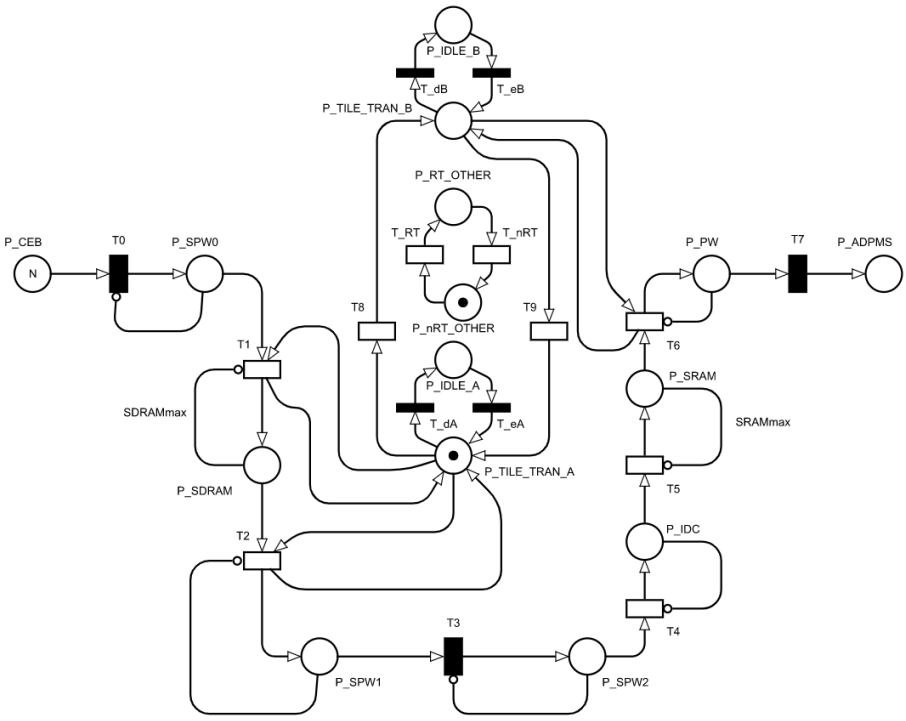


Fig. 6. Data Processing Unit (DPU) performance model, with enabled compression, correlated with Proto-Flight Model

All in all, those real-time (deterministic scheduling) and asynchronous (interrupt based) loads block, when executed, the AMBA bus for tile transfer and this

effect can be modeled by controlling the presence of tile transmission enabling token rotation in P_TILE_TRAN_A & T8 & P_TILE_TRAN_B & T9 path. This is done by adding the P_IDLE_A and P_IDLE_B places which siphons in (by immediate transitions T_dA and T_dB) or siphons out (by immediate transitions T_eA and T_eB) the token so none of the transition responsible for modeling actual tile transfer can fire. Disabling the tile transmission in the system, effectively is performed by activating T_dA or T_dB. Both these transition are controlled by an enabling function, asserted when the newly introduced place P_RT_OTHERS contains a token. By symmetry, the tile transmission within the system recovers when T_eA or T_eB are activated, by asserting the enabling function when place P_nRT_OTHER contains the token. It shall now be obvious that a token in the place P_RT_OTHER models the times when the CPU executes the scheduled and asynchronous loads, while a token in P_nRT_OTHER models the situation when the CPU performs best-effort operations and tile-transfers. The transition T_RT ensures that periodic and other loads are executed in simulation every 500 ms and the T_nRT sets the expected load execution time to about 200 ms, which estimate is based on execution loads of major components of ASW. The model is presented in Figure 6. The enabling connections from P_RT_OTHERS to T_dA and T_dB, as well as from P_nRT_OTHER to T_eA and T_eB are omitted for maintaining clarity of figure.

Simulation of the updated models in TimeNet software yields that, in order to transfer 10000 tiles the DPU would need 41 and 59 seconds, respectively, for models with compression enabled and disabled. This corresponds to about 244 and 169 tiles/s peak performance capability of DPU for compression-on and -off mode. For the mode of operation involving the bit-stuffing (each pixel is coded with 16 bits but contains only 14 bits of information) time to transfer while simulated tile batch is about 51 seconds, which corresponds to 196 tiles/s on average.

Simulation results correlate well with experimental results, as presented in table 5, in the whole spectrum of DPU modes of operation. Following models are taken into account: Development Model (DM), Engineering Model (EM), Proto-Flight Model (PFM) and Petri Net model (P/N-model).

Table 5. The processing performance (in tiles per second) of CCB DPU models.

Mode of operation	DM	EM	PFM	P/N-model
raw tiles	—	—	180	169
bit-stuffed tiles	—	—	202	196
compressed tiles	252	205	233	244

7 Gained Insights and Conclusions

Petri Nets are not new to industry. However, they are not used very widely. They can serve addressing the increasing importance, in logistics, network and computer architectures design and analysis and workflow systems.

The work presented in this paper documents the way Petri Nets were involved in systems engineering of complex control and data processing systems. This method of modeling has been proven as a useful tool for obtaining insights in architecture trade-offs, but more importantly, the model could be easily updated to track and reflect the changes in the system, as development gains maturity and up to the stage of quite accurate correlation of simulation results with the measurements of processing performance of DPU flight hardware and software. Our main achievements of the CCB DPU modeling process using Petri Nets can be summed up in following way:

- DPU Petri Net performance model revealed that DPU has to be treated as streaming, not buffering, device, as in worst-case operation scenario of "CME-Watch" there will be no time to send all scientific data tiles after the end of observations;
- to the last point, DPU amount of on-board SDRAM memory has been reduced significantly, which in terms of Flight Units delivery is equivalent of buying 9 SDRAM memory modules less (savings of roughly 50 thousand Euro in components costs);
- DPU performance model analysis provided insights in how the Application Software controlling the DPU has to manage the scientific data flow and circulation of tiles from CEB, to compression engine and to ADPMS Mass Memory Modules in order to meet the processing performance demands;
- the Petri Net model validity has been proven by, first, building the flight equipment meeting performance requirements, second, accurately correlating the updated Petri Net model with the measurements performed on the DPU Flight Model.

All in all, we found that time and effort spent on developing modeling tools, such as Petri Nets, are likely to pay back in the future. Modeling, if it allows early concept prototyping in order to fail and pivot or to consolidate and move forward, without any doubt, is an invaluable support, that is directly traceable to savings in time and money spent on the project. The described creation of Proba-3 Coronagraph Control Box and its Digital Processing Unit in particular, provides solid evidence for this claim.

The presented work contributes a small step towards providing new systems engineering tools for use in the aerospace or space industry. It is a perfect moment for such discussions, as space business undergoes deep changes, starting from the New Space revolution up to evolution of Model-based Systems Engineering into a Digital Twin Spacecraft concept. Petri Nets, thanks to their high level of abstraction and versatility, tackle a large class of system engineering issues, especially in the early design phases.

References

1. TASTE. Available at https://taste.tools/ (2022/01/26)
2. Blommestijn, R., Fuchs, J.: Specification and description language (SDL) (Jul 1999), z.100
3. Blommestijn, R., Fuchs, J.: Technical Dossier on System Modelling and Simulation Tools (Jul 2012), iss 2, rev 2A
4. Bluff, R.: Avionic system modelling. In: Simulation '98. International Conference on (Conf. Publ. No. 457). pp. 11–18 (Sep 1998). https://doi.org/10.1049/cp:19980610
5. Bollig, B., Fortin, M., Gastin, P.: Communicating finite-state machines, first-order logic, and star-free propositional dynamic logic. Journal of Computer and System Sciences **115**, 22–53 (2021). https://doi.org/10.1016/j.jcss.2020.06.006
6. Eickhoff, J.: Simulating Spacecraft Systems. Springer Science & Business Media (Sep 2009)
7. Eickhoff, J.: Onboard Computers, Onboard Software and Satellite Operations: An Introduction. Springer, Berlin ; New York, 2012 edition edn. (Dec 2011)
8. Ereau, J.F., Saleman, M.: Modeling and simulation of a satellite constellation based on Petri nets. pp. 66–72 (Jan 1996). https://doi.org/10.1109/RAMS.1996.500644
9. Feiler, P.: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley Professional, 1st edition. edn. (2012)
10. Fischer, M.J., Ladner, R.E.: Propositional dynamic logic of regular programs. Journal of Computer and System Sciences **18**(2), 194–211 (1979). https://doi.org/10.1016/0022-0000(79)90046-1
11. Galano, D., Bemporad, A., Buckley, S., Cernica, I., Dániel, V., Denis, F., de Vos, L., Fineschi, S., Galy, C., Graczyk, R., Horodyska, P., Jacob, J., Jansen, R., Kranitis, N., Kurowski, M., Ladno, M., Ledent, P., Loreggia, D., Melich, R., Mollet, D., Mosdorf, M., Paschalis, A., Peresty, R., Purica, M., Radzik, B., Rataj, M., Rougeot, R., Salvador, L., Thizy, C., Versluys, J., Walczak, T., Zarzycka, A., Zender, J., Zhukov, A.: Development of ASPIICS: a coronagraph based on Proba-3 formation flying mission. In: Lystrup, M., MacEwen, H.A., Fazio, G.G., Batalha, N., Siegler, N., Tong, E.C. (eds.) Space Telescopes and Instrumentation 2018: Optical, Infrared, and Millimeter Wave. vol. 10698, pp. 906 – 918. International Society for Optics and Photonics, SPIE (2018). https://doi.org/10.1117/12.2312493
12. German, R., Kelling, C., Zimmermann, A., Hommel, G.: TimeNET-a toolkit for evaluating non-Markovian stochastic Petri nets. In: Proceedings of the Sixth International Workshop on Petri Nets and Performance Models, 1995. pp. 210–211 (Oct 1995). https://doi.org/10.1109/PNPM.1995.524333
13. Girault, C., Valk, R.: Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer Science & Business Media (2003)
14. Graczyk, R.: Reliability and performance modeling of configurable electronic systems for unmanned spacecraft. Editorial Series on Accelerator Science, Warsaw University of Technology Publishing House, Warsaw (2016)
15. Hall, A.D.: A Methodology for Systems Engineering. Van Nostrand (1962)
16. Holt, J.: UML for systems engineering (2004)
17. Kelling, C., German, R., Zimmermann, A., Hommel, G.: TimeNET: evaluation tool for non-Markovian stochastic Petri nets. In: Computer Performance and Dependability Symposium, 1996., Proceedings of IEEE International (Sep 1996). https://doi.org/10.1109/IPDS.1996.540206

18. Kordon, F., Canals, A., Dohet, A.: Embedded systems analysis and modeling with SysML, UML and AADL. Electonics engineering series, ISTE, London (2013)
19. Kossiakoff, A., Sweet, W.N.: Systems Engineering Principles and Practice. Wiley-Interscience, New York, 1 edition edn. (Dec 2002)
20. Lloret, J.C., Roux, J.L., Algayres, B., Chamontin, M.: Modelling and evaluation of a satellite system using EVAL, a Petri Net based industrial tool. In: Jensen, K. (ed.) Application and Theory of Petri Nets, pp. 379–383. No. 616 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (Jan 1992)
21. Malott, L., Palangpour, P.: Small Spacecraft Software Modeling: A Petri Net-Based Approach. AIAA/USU Conference on Small Satellites (Aug 2013)
22. Molloy: Performance analysis using stochastic petri nets. IEEE Transactions on Computers **C-31**(9), 913–917 (1982). https://doi.org/10.1109/TC.1982.1676110
23. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE **77**(4), 541–580 (Apr 1989). https://doi.org/10.1109/5.24143
24. Perrotin, M., Grochowski, K., Verhoef, M., Galano, D., Mosdorf, M., Kurowski, M., Denis, F., Graas, E.: TASTE in action. In: 8th European Congress on Embedded Real Time Software and Systems (ERTS 2016). TOULOUSE, France (Jan 2016), https://hal.archives-ouvertes.fr/hal-01289678
25. Platzer, A.: Logical Analysis of Hybrid Systems Proving Theorems for Complex Dynamics. Springer (2010)
26. Rechtin, E.: Systems Architecting: Creating and Building Complex Systems. Prentice Hall (1991)
27. Renotte, E., Buckley, S., Cernica, I., Denis, F., Desselle, R., Vos, L.D., Fineschi, S., Fleury-Frenette, K., Galano, D., Galy, C., Gillis, J.M., Graas, E., Graczyk, R., Horodyska, P., Kranitis, N., Kurowski, M., Ladno, M., Liebecq, S., Loreggia, D., Mechmech, I., Melich, R., Mollet, D., Mosdorf, M., Mroczkowski, M., O'Neill, K., Patočka, K., Paschalis, A., Peresty, R., Radzik, B., Rataj, M., Salvador, L., Servaye, J.S., Stockman, Y., Thizy, C., Walczak, T., Zarzycka, A., Zhukov, A.: Recent achievements on ASPIICS, an externally occulted coronagraph for PROBA-3. In: MacEwen, H.A., Fazio, G.G., Lystrup, M., Batalha, N., Siegler, N., Tong, E.C. (eds.) Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave. vol. 9904, pp. 1112 – 1126. International Society for Optics and Photonics, SPIE (2016). https://doi.org/10.1117/12.2232695
28. Zimmermann, A.: Modeling and evaluation of stochastic Petri nets with TimeNET 4.1. In: 2012 6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS). pp. 54–63 (Oct 2012)
29. Zurawski, R., Zhou, M.: Petri nets and industrial applications: A tutorial. IEEE Transactions on Industrial Electronics **41**(6), 567–583 (Dec 1994). https://doi.org/10.1109/41.334574