PLATO N-DPU ON-BOARD SOFTWARE: AN IDEAL CANDIDATE FOR MULTICORE SCHEDULING ANALYSIS

Philippe Plasson [(1)], Gabriel Brusq [(2)], Frank Singhoff[(3)], Hai Nam Tran[(3)], Stéphane Rubini[(3)], Pierre Dissaux [(4)]

[(1)] LESIA, Observatoire de Paris, Université PSL, CNRS, Sorbonne Université, Université de Paris, 5 place Jules Janssen, 92195 Meudon, France, philippe.plasson@obspm.fr
[(2)] CNES, 18 avenue Edouard Belin 31400 Toulouse France, gabriel.brusq@cnes.fr
[(3)] Lab-STICC, CNRS UMR 6285, Univ. of Brest, 20, av Victor le Gorgeu, 29200 Brest, France, surname@univ-brest.fr
[(4)] Ellidiss Technologies, 24 quai de la douane, 29200 Brest, France, pierre.dissaux@ellidiss.com

INTRODUCTION

Each day, space missions require ever more computing power. PLATO mission, parts of ESA's Cosmic Vision program, is no exception. To answer the demand, multiprocessors chips are now used in space industry. In several ways it impacts space engineering habits. Architects have to rethink their software designs to benefit these extra processing resources, while considering shared accesses constraints. Their real-time applications are based on complex dynamic architectures and require a schedulability proof for flight. A model based analysis approach has been selected for PLATO N-DPU ASW. An AADL model representative of the application has been designed, and analysed through AADLInspector and Cheddar tools. In this paper, we present the PLATO N-DPU ASW architecture, and the LESIA GERICOS framework it relies on. We also discuss the model and the tools used to perform schedulability analysis.

1. PLATO MISSION PRESENTATION

1.1. PLATO mission goal

PLATO ("PLAnetary Transits and Oscillations of stars") is an M-class mission of the European Space Agency foreseen to be launched in 2026 [3]. PLATO aims to characterize exoplanetary systems by detecting planetary transits and conducting asteroseismology of their parent stars.
The PLATO payload concept is based on a multi-camera approach, involving a set of 24 normal instruments monitoring stars fainter than mV=8, plus a smaller set of two fast instruments observing extremely bright stars with magnitudes brighter than mV=8.

1.2. PLATO On-Board Data Processing System

The PLATO Data Processing System, called DPS, is the sub-system of PLATO payload module in charge of the on-board data processing (data acquisition, data reduction, data compression, monitoring, etc.) [4]. The DPS is a set of several on-board computer boards connected via a SpaceWire network. The DPS architecture is composed of:

- 12 Normal Data Processing Units (N-DPU) embedding a GR712RC dual-core LEON3FT SPARC V8 processor.
- two Fast Data processing Units (F-DPU)
- two Instrument Control Units (ICU) working in cold redundancy

The N-DPU and F-DPU are connected via SpaceWire interfaces to the front-end electronics of the normal (N-FEE) and fast (F-FEE) cameras at one end and to the ICU at the other.
The N-DPU and F-DPU are in charge of processing respectively the data of two normal cameras for each N-DPU and the data of one fast camera for each F-DPU.
The active ICU is responsible for managing the payload, communicating with the spacecraft service module, and compressing the scientific data before transmission as telemetry packets.

1.3. PLATO N-DPU Application Software Overview

The N-DPU Application Software (N-DPU ASW) is the embedded software deployed in each of the 12 N-DPU boards. Each N-DPU ASW manages two cameras (four 21-million pixel detectors per camera) and collects science and housekeeping data from their N-FEE. During the Observation mode, each N-DPU unit is receiving window segments of observed stars from both N-FEE as inputs. The window segments are transferred every 6.25 seconds through a single SpaceWire link per N-FEE. The window segments are then reconstructed back using a lookup table into windows and are processed by the N-DPU ASW.
The detector exposure time is 25 seconds, but the four detectors are read sequentially every 6.25 seconds with a read-out time of four seconds.
In the observation mode, the N-DPU ASW role is to produce 6x6-pixel square-shaped windows (i.e. raw star windows not transformed on-board) for 21% of the incoming stars (out of 132350 stars per camera) and photometry products using binary mask algorithms for 79% of them.
The photometry products are made up of star flux and centroids of stars. The N-DPU ASW computes also the offset, background, smearing and performs outlier detection.

The N-DPU ASW calculates light fluxes and centres of brightness (COB) every 25 seconds. These photometry products are then averaged over 50 and 600 seconds (samples of two and 24 measurements respectively). The averaged photometry is then sent to the ICU.
The other 21% stars (27500) of each camera are directly transmitted to the ICU.
In addition to the science services, the N-DPU ASW offers services dedicated to calibration activities: full-image acquisition service and window acquisition service.
Last but not least, the N-DPU ASW implements services for checking the telecommand packets, managing the production of housekeeping reports and event reports, handling the time, accessing to the N-DPU and N-FEE memories, managing and monitoring the on-board parameters, etc.

2. SOFTWARE FRAMEWORK AND DYNAMIC ARCHITECTURE

2.1. GERICOS framework

The architecture of the N-DPU ASW is built according to an Asymmetric Multi-Processing approach (AMP) using the GERICOS framework running on top of the RTEMS 4.8 real-time kernel (Edisoft version).
The GERICOS platform (GEneRIC Onboard Software), which is a generic platform for the development of space payload software, is made up of two parts:

- GERICOS C++ framework: a set of layered, reusable and customizable software components,
- GERICOS::TOOLS: a set of tools for automatizing the development process of embedded software (building chain, code generation, UML profile, UML diagram transformation, AMP support, …).

The GERICOS::CORE layer, which is part of C++ framework and which acts as a middleware, offers an extremely lightweight (small memory footprint), optimized (low CPU resources) and space qualified implementation of the active object paradigm on top of a real-time kernel.
With the GERICOS::CORE layer, a real-time application is built as a set of active objects (called "tasks"), each active object having its own message queue and its own computational thread. Each Thread processes incoming messages one by one by executing the function associated with the message [5].
The support of the AMP architecture in the GERICOS framework relies on the intrinsic features of the active object paradigm of the GERICOS::CORE layer. With this paradigm, two objects communicate via marshalled messages. Each object is split in an implementation object (which implements the services offered by the object) and in a stub object which is responsible for the marshalling aspects (message serialization, message unserialization). The marshalling process implemented in the stub has been extended so that the objects are able to communicate from a CPU core to another one using simple communication mechanisms based on shared memory for passing the messages and spin locks based on the LEON3 atomic compare-and-swap operation (CASA instruction) to make safe the inter-core concurrent access to the memory.
The GERICOS::CORE task components include functionalities for recording and reporting either response time statistics or execution time statistics of the various threaded functions. These features are used to assess by measurements the worst-case execution times (WCET) needed by the scheduling analysis model.

2.2. Dynamic architecture

The dual-core AMP architecture of the N-DPU ASW means that two applications coexist.
The first application, which acts as the master, is deployed on the LEON3-FT CPU core #0 and is in charge of managing the ICU interfaces (telecommand and telemetry packets), processing the data of camera #A, managing modes and technical services. This first application is made up of 21 tasks supporting 57 threaded functions. Among these threaded functions, only 23 have been identified as playing a key role in the computational model used for performing the scheduling analysis of the observation mode, which is the most demanding and critical mode of the software.
The second application is deployed on the LEON3-FT CPU core #1 and is in charge of processing the data of camera #B and scrubbing the memory. This second application is made up of 9 tasks supporting 35 threaded functions. Among these threaded functions, 13 have been identified as playing a key role in the computational model.
In terms of real-time sequencing, the fundamental period to consider is 6.25 seconds: the software receives every 6.25s a new set of star windows to process. The transmission of these star windows is spread over the 4.1 seconds corresponding to the readout of the detector. Up to 25 packets of 32 kilobytes are transmitted per second by the camera over the readout period. At the end of the packet reception, the software has to extract the window segments from the packets and reconstruct the windows in less than 2.15 seconds, that is to say before the end of the cycle of 6.25 seconds. Once the reconstruction phase is complete, the software can start the photometric processing. These photometric treatments are carried out during the transmission of the star windows of the next detector and must imperatively be finished 4.1 seconds after the starting of the following cycle. Figure 1 illustrates the sequence of the N-DPU ASW tasks with the main periods and deadlines.
This real-time sequencing and the related constraints are the same for both applications. The main difference is related to the periodic technical functions (data pool management, housekeeping packet production, monitoring of the parameters, transmission of telemetry

packets, etc.) which are not the same for both applications. These threaded functions execute periodically in the cycle of 6.25 seconds with a period that depends on the function (between 50 ms and one second).

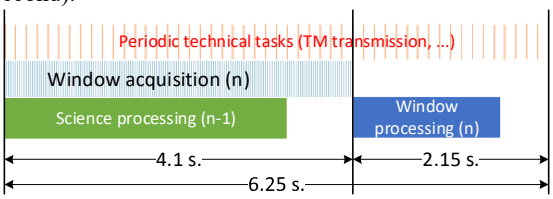

Figure 1: sequence of the N-DPU ASW tasks with the main periods and deadlines

During the observation mode, both applications communicate through the circular buffers containing the telemetry packets. The second application builds the telemetry packets corresponding to the data from camera B and stores them in a shared queue. The first application retrieves these packets from the shared queue and transmits them to the ICU. There can also be inter-core message exchanges when, for example, an event is detected by the second application and a report must be transmitted by the first application.

2.3. Real-time constraints

The N-DPU ASW is a real-time software where failure to conform to timing constraints can result in a loss of science data or of the instrument. Particular attention shall be paid to the window and science processing tasks which are considered critical for the scientific mission. A schedulability proof must be provided to validate that the dynamic architecture design prevents any deadline miss for those critical activities.

2.4. Dynamic model

The real-time application presented above is based on a complex dynamic architecture. A schedulability proof and CPU margins are required for flight in order to comply with ECSS standards. Such results are tricky to obtain because of the multi-core architecture. In particular, every access conflicts and locks must be taken into account. A model of the PLATO N-DPU ASW has been specifically designed to describe all its dynamic properties, and in particular, shared resources access, priority inheritance protocols, synchronization mechanisms, tasks precedence's constraints, WCET, …
Dynamic model design is a trade-off between model representativeness and model complexity. The goal is to propose a model using simplification hypothesis as little pessimistic as possible, to end up with sufficient representativeness to guarantee the schedulability analysis validity. In particular, some modelling hypothesis have been proposed for caches and DMA burst transfers. The model is populated thanks to real-time measurements performed on the on-board computer, via a dedicated GERICOS feature.
The next section details how the model of the application can be analyzed to get a schedulability proof and a worst-case CPU occupation ratio. These results are then confronted with the CPU margin requirements.

3. SCHEDULABILITY ANALYSIS

3.1. Cheddar Scheduling Analysis Tool

Cheddar is an open-source real-time scheduling analysis tool developed and maintained by the University of Brest and Ellidiss Technologies [1]. Recent improvements have been implemented to support scheduling analysis of software running on multi-core architectures [2]. Although Cheddar can be used in a standalone mode, i.e., using its own internal modelling language and editor, it is also semantically compliant with the SAE-AS5506 international standard (AADL: Architecture Analysis and Design Language) and can thus be easily integrated within interoperable systems and software development toolchains.

3.2. AADL Language

The AADL standard [7] has been defined to describe software intensive real-time system architectures and to embed a sufficient level of semantics to enable early analysis at model level. It is used worldwide for mission critical programs in the aerospace, ground transportation and medical device industries.
AADL is supported by a variety of modelling and analysis academic and industrial tools, including Osate [11], Ocarina [12], Masiw [13], Ramses [14], SCADE Architect, FASTAR, TASTE [15], Stood, and AADL Inspector. An alternate approach would have been to use the UML MARTE profile associated with the MAST scheduling analysis tool [8,9]. However, the strength of the AADL ecosystem [6] militates in its favour.

3.3. AADL Inspector Model Processing Framework

AADL Inspector (see Figure 2) is a model processing framework. It can load hand-written or automatically generated AADL models from modelling tools or inward model transformations. It provides dedicated outward model transformations to feed real-time, safety and security analysis tools. For this project, AADL Inspector has been used to load the hand-written AADL representation of the PLATO N-PDU ASW into Cheddar to perform the required multi-core scheduling analysis. A possible enhancement would consist in adding a custom model transformation to directly load the GERICOS UML model into AADL Inspector.

4. FEEDBACK FROM PLATO N-DPU ASW ANALYSIS

AADL offers a large panel of entities and properties to

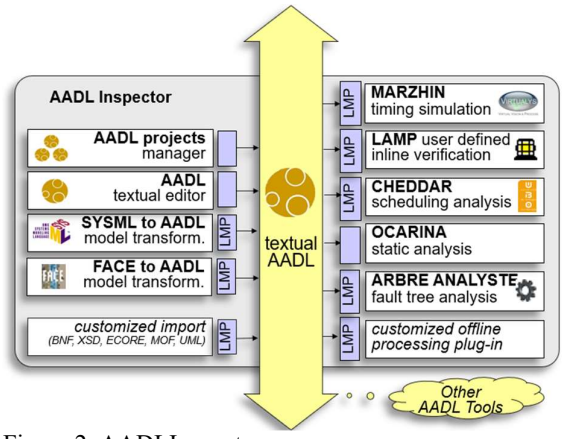

Figure 2. AADLInspector

model dynamic systems, that AADLInspector and Cheddar kernel tools can analyze. There are often several ways to model a dynamic behavior combining different elements of the language. For example, PLATO N-DPU ASW tasks synchronization is performed through messages exchanges. This behavior can be modeled in AADL by events sent through an event port, that can trigger the dispatch of a task. Another example is the modeling of task precedency by the dispatch offset property: to model that a task B starts after a task A, we can set a dispatch offset property for task B with a duration equal to task A period. The challenge is to find a valid model which is a trade-off between model design complexity and its pessimism. Assumptions can make the model more pessimistic, but also easier to design, and sometimes easier to analyze. Another solution is to tune the dynamic architecture model to make it compliant with schedulability analysis. For instance, the task set periods have been reworked to decrease as much as possible the Least Common Multiple (LCM) in order to decrease the time required to perform the scheduling analysis.

Some features that were missing to analyze the complete PLATO N-DPU ASW, such as the management of resources shared between cores, have been quickly integrated in AADLInspector and Cheddar tools. Then, the access to those resources can be accurately represented in AADL combining a resource protected by a spinlock (blocking statement) for core access, and a semaphore protected against deadlock by a priority inheritance protocol for task accesses (See Figure 3).

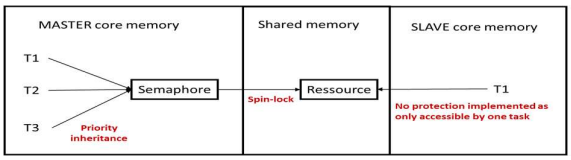

Figure 3. Modeling of core shared resources

For PLATO ASW, the memory accesses slowdown caused by DMA transfer has been computed based on SDRAM burst accesses configuration. A DMA burst access consumes 12 cycles every 153 cycles. It represents 7,84% of the SDRAM access cycles. In the worst case, 3 DMA transfers can occur at the same time, leading to a 24% slowdown ratio. This ratio is added to each task for which the WCET has been measured out of any DMA perturbation. Conversely, some WCET are obtained from measures on worst case scenarios, and already take into account the DMA impacts in this case the slowdown ratio is not applied. Regarding caches, no theoretical impact has been computed. Instead, the impact is also taken into account when measuring WCET on worst case scenarios.

The schedulability proof can be obtained with Cheddar by static analysis or by simulation performed on the feasibility interval, which is the interval of time that captures all possible events of the analyzed model. It can be computed with the following rule [10]:
Feasibility interval = [0 .. 2 * LCM + max (first dispatch time)] = 54 seconds for PLATO N-DPU ASW.
Both methods allow to check that the WCRT are always below the deadline, which is the expected schedulability proof.

CONCLUSION

In this paper, we presented how an AADL model of the PLATO N-DPU ASW system has been designed for schedulability analysis. The model has proved than all the tasks of the system, and in particular the more critical ones, theoretically cannot miss their deadlines. Execute window processing has a deadline of 2150 ms, and a computed WCRT of 1552ms on core 0 and 1142 ms on core 1. Execute science processing has a deadline of 4100 ms, and a computed WCRT of 3616 ms on core 0 and 2699 ms on core 1. Moreover, we were able to compute CPU margins from the model. Since the model is significantly pessimistic, due to hypothesis we took, we are confident that real CPU margins are above the one computed from the model.
The dynamic model helped to improve the dynamic architecture of the on-board software, in particular since the design of the model requires a flawless understanding of the real-time architecture and constraints. Outputs of analysis tools were precious to find real time optimizations. For instance, improvements were made by relaxing unjustified too hard timing constraints, or on priority and period assignments. Finally, WCET measurement mechanisms have been improved to be as less pessimistic as possible. Once an optimal dynamic architecture was reached, the model was used to get the analytical proof of the schedulability thanks to dedicated tools. Moreover, the model is helpful to validate any dynamic architecture evolution before its software implementation, and to continuously monitor the schedulability until the software design is frozen.

REFERENCES

[1]. Singhoff, F., Legrand, J., Nana, L., & Marcé, L. Cheddar: a flexible real time scheduling framework. In Proceedings of ACM SIGAda international conference. 2004, November, pp. 1-8

[2]. Rubini, S., Fotsing, C., Singhoff, F., Tran, H. N., & Dissaux, P. Scheduling analysis from architectural models of embedded multi-processor systems. ACM SIGBED Review, 11(1), 68-73. 2014

[3]. Ragazzoni, R., Magrin, D., Rauer, H., Pagano, I., Nascimbeni, V., Piotto, G., Piazza, D., Levacher, P., Schweitzer, M., Basso, S., Bandy, T., Benz, W., Bergomi, M., Biondi, F., Boerner, A., Borsa, F., Brandeker, A., Br andli, M., Bruno, G., Cabrera, J., Chinellato, S., Roche, T. D., Dima, M., Erikson, A., Farinato, J., Munari, M., Ghigo, M., Greggio, D., Gullieuszik, M., Klebor, M., Marafatto, L., Mogulsky, V., Peter, G., Rieder, M., Sicilia, D., Spiga, D., Viotto, V., Wieser, M., Heras, A. M., Gondoin, P., Bodin, P., and Catala, C., PLATO: a multiple telescope spacecraft for exo-planets hunting, in [Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave], MacEwen, H. A., Fazio, G. G., Lystrup, M., Batalha, N., Siegler, N., and Tong, E. C., eds., 9904, 731-737, International Society for Optics and Photonics, SPIE (2016).

[4]. Ziemke, C., Witteck U., Peter G., Plasson P., Galli E., Ulmer B., Ottensamer R., Ottacher H., Windsor J. PLATO DPS: State of the art on-board data processing for Europe's next planet-hunter. OBDP2021 - 2nd European Workshop on On-Board Data Processing. 2021, June.

[5]. Plasson, P., Cuomo, C., Gabriel, G., Gauthier, N., Gueguen, L., Malac-Allain, L. GERICOS: A Generic Framework for the Development of On-Board Software. In proceedings of DASIA Conference. 2016, May. ISBN: 978-92-9221-301-5. ESA-SP Vol. 736, 2016, id.39

[6]. Boydston A., Feiler P., Vestal S., Lewis B., Architecture Centric Virtual Integration Process (ACVIP): A Key Component of the DoD Digital Engineering Strategy. https://resources.sei.cmu.-edu/library/asset-view.cfm?assetid=634965

[7]. Feiler, Peter H., Bruce A. Lewis, and Steve Vestal. The SAE Architecture Analysis & Design Language (AADL) a standard for engineering performance critical systems. 2006 IEEE Conference on Computer Aided Control System Design.

[8]. Harbour, M. González, et al. Mast: Modeling and analysis suite for real time applications. Proceedings 13th Euromicro Conference on Real-Time Systems. IEEE, 2001.

[9]. Faugere, Madeleine, et al. Marte: Also an uml profile for modeling aadl applications. 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007). IEEE, 2007.

[10]. Goossens, Joël, Emmanuel Grolleau, and Liliana Cucu-Grosjean. Periodicity of real-time schedules for dependent periodic tasks on identical multiprocessor platforms. Real-time systems 52.6 (2016): 808-832.

[11]. Osate, https://osate.org/

[12]. Hugues, Jerome, et al. From the prototype to the final embedded system using the Ocarina AADL tool suite. ACM Transactions on Embedded Computing Systems (TECS) 7.4 (2008): 1-25.

[13]. Khoroshilov, Alexey, et al. AADL-based toolset for IMA system design and integration. SAE International Journal of Aerospace 5.2 (2012): 294.

[14]. Blouin, Dominique, and Etienne Borde. AADL: A Language to Specify the Architecture of Cyber-Physical Systems. Foundations of Multi-Paradigm Modelling for Cyber-Physical Systems. Springer, Cham, 2020. 209-258.

[15]. Perrotin, Maxime, et al. TASTE: An open-source tool-chain for embedded system and software development. Embedded Real Time Software and Systems (ERTS2012). 2012.