Enabling Rapid Development of On-board Applications: Securing a Spacecraft Middleware by Separation and Isolation

Andreas Lund*, Carlos Gonzalez Cortes†, Zain Haj Hammadeh†, Fiona Brömer*, Glen te Hofsté*, Daniel Lüdtke†

*Institute for Software Technology
German Aerospace Center (DLR)
Weßling, Germany

†Institute for Software Technology
German Aerospace Center (DLR)
Braunschweig, Germany

Today's space missions require increasingly powerful hardware to achieve their mission objectives, such as high-resolution Earth observation or autonomous decision-making in deep space. At the same time, system availability and reliability requirements remain high due to the harsh environment in which the system operates. This leads to an engineering trade-off between the use of reliable and high performance hardware. To overcome this trade-off, the German Aerospace Center (DLR) is developing a special computer architecture that combines both reliable computing hardware with high-performance commercial-off-the-shelf (COTS) hardware. This computer architecture is called *Scalable On-Board Computing for Space Avionics (ScOSA)* and is currently being prepared for demonstration on a CubeSat, also known as the *ScOSA Flight Experiment* [1].

The ScOSA software consists of a middleware to execute distributed applications, perform critical on-board software functionalities, and do fault detection and recovery tasks. The software is based on the *Distributed Tasking Framework* which is a derivate of the open-source, data-flow oriented *Tasking Framework* [2], for this reason, developers organize their applications as a set of *tasks* and *channels*. The middleware handles the task distribution among the nodes [3]. ScOSA will detect failing compute nodes and reallocate tasks to maintain the availability of the entire system. The middleware can also change the set of allocated tasks to support different mission phases. Thus, ScOSA allows software to be reloaded and executed after startup. By this the software can be tested quickly and safely on the system. Combined with an upload strategy, ScOSA can be used for *in-situ* testing of on-board applications.

Since ScOSA will also perform mission-critical tasks, such as an *Attitude and Orbit Control System* or a *Command and Data Handling System*, the opening of the platform leads to the problem of mixed criticality [4]. This problem is already present in the *ScOSA Flight Experiment*, since the demonstration will include typical satellite applications developed by different teams in the *DLR*. Thus, not only the teams implement different quality standards for their software, but also the applications themselves have different *Technical Readiness Levels (TRLs)*.

The challenge of mixed criticality is often met by completely separating and isolating the different software components, e.g. by using a hypervisor or a separation kernel [5], [6]. Due to the distributed nature of the ScOSA system and its execution platform a separation using hypervisor technique is not easily achievable.

For this reason, we discuss in this work how we separate the critical services and communication components into their own *Linux* process to guarantee that best-effort applications are not inflicting the critical components of the middleware. We also consider and discuss in this work how to implement further mechanisms of the *Linux* kernel in order to strengthen the separation, i.e. the *cgroups* and the *kernel namespaces*. However, a complete isolation between software components is undesirable, due to the necessary interaction between them. Given that the applications themselves can be spread over several nodes, the application tasks need to communicate and this can be only done if the critical software components relays messages from other nodes to the separated application processes. For this reason the middleware provides a relay service which takes care of the intra-node-inter-process-communication. Using a relaying mechanism simplifies development and does not require a complete rewrite of the existing middleware network stack.

The proposed techniques were applied in a case study to integrate applications of unknown quality standards into the ScOSA software system in an agile way. We discuss how the presented measures ensure that the resultant software is sufficiently tested and meets the required quality level.

Finally, we discuss possible improvements to our existing separation and isolation solution for *ScOSA* and outline how these techniques can be used in other platforms such as the *RTEMS* operating system.

Index Terms

Mixed-Criticality, On-board software, Fault-Tolerance, Computer Architecture

References

[1] C. J. Treudler, H. Benninghoff, K. Borchers, B. Brunner, J. Cremer, M. Dumke, T. Gärtner, K. J. Höflinger, D. Lüdtke, T. Peng, E.-A. Risse, K. Schwenk, M. Stelzer, M. Ulmer, S. Vellas, and K. Westerdorff, "ScOSA - scalable on-board computing for space avionics," in *International Astronautical Congress (IAC), Bremen, Germany, Oct. 1-5*, 2018.

[2] Z. A. H. Hammadeh, T. Franz, O. Maibaum, A. Gerndt, and D. Lüdtke, "Event-driven multithreading execution platform for real-time on-board software systems," in *15th Workshop on Operating Systems Platforms for Embedded Real-Time applications (OSPERT), Stuttgart, Germany, July 9, 2019*, A. Lackorzynski and D. Lohmann, Eds., 2019, pp. 29–34. [Online]. Available: https://elib.dlr.de/128249/

[3] A. Lund, Z. A. H. Hammadeh, P. Kenny, V. Bensal, A. Kovalov, H. Watolla, A. Gerndt, and D. Lüdtke, "ScOSA system software: The reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture," *CEAS Space Journal*, Mai 2021.

[4] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2022.

[5] E. Salazar, A. Alonso, and J. Garrido, "Mixed-criticality design of a satellite software system," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 12 278–12 283, 2014, 19th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016435688

[6] D. Sanán, A. Butterfield, and M. Hinchey, "Separation kernel verification: The xtratum case study," in *Verified Software: Theories, Tools and Experiments: 6th International Conference, VSTTE 2014, Vienna, Austria, July 17-18, 2014, Revised Selected Papers 6*. Springer, 2014, pp. 133–149.