Deutscher Luft- und Raumfahrtkongress 2019
DocumentID: 490112

Space application development at "The On-Orbit Servicing and Autonomy" group at the DLR

Kurt Schwenk, Katharina Willburger, Eicke-Alexander Risse

Abstract

The On-Orbit Servicing and Autonomy group of the Space Flight Technology department conducts applied research for future missions in the fields On-Orbit Servicing and On-board Data Analysis and Real-time Information Systems.

In On-Orbit Servicing, new Guidance, Navigation and Control concepts based on optical navigation are developed for the approach to non-cooperative targets like space debris or satellites at the end of their life-cycle. In the field of On-board Data Analysis and Real-Time Information Systems, the goal is the development of real-time on-board data processing systems that allow accessing information obtained from sensor data within minutes after recording of the data. Both of these applications require computationally expensive data processing that has to be executed directly on board the satellite. This brings two major challenges:

The first challenge is to develop on-board computing systems powerful enough for these demanding applications. The second challenge is to lower the software system requirements to a point where a future modern on-board computer can handle the tasks. The current generation of on-board computer systems were meant for controlling a satellite and are far from being powerful enough to support data intensive applications. Therefore the development of these applications is done hand in hand with the development of a high performance on-board computing system within a DLR cooperation.

In this article the difficulties, approaches and progress of space application development of our group in the last year are presented.

1 INTRODUCTION

1.1 BACKGROUND

The On-Orbit Servicing and Autonomy group of the Space Flight Technology department at the German Aerospace Center (DLR) conducts applied research for on-board intelligence and autonomy in the fields of On Orbit Servicing (OOS) and Onboard Data Analysis and Real-Time Information Systems (ODARIS) [1].

In the field of OOS, Guidance, Navigation and Control (GNC)-concepts based on optical navigation[1] are developed. The objective is to gain know-how in approaching non-cooperative targets like satellites at the end of their life-cycle, or space debris [3]. With the European Proximity Operations Simulator 2.0 (EPOS) [1], the Rendezvous and Docking (RvD) process as well as the developed GNC-methods are verified by Hardware in the Loop (HIL) simulations.

Within the ODARIS section, the goal is the development of concepts and prototypes for accessing real-time information from a satellite. The main question tackled here is, how relevant information can be extracted from sensors or other sources on a satellite and delivered to the user within minutes after retrieval of the data. Important applications in this regard are alarming and real-time query services. In the last three years the prototype system Autonomous real-time detection of moving maritime objects (AMARO) was developed and successfully tested on an airplane campaign in 2018 [4]. One main feature of the AMARO system is a real-time query service. An example query is getting the list of all ships (position, size etc.) that have been detected in the last three minutes. The other feature is the alarming service. For example, the system can be configured on runtime to inform the user when a ship was detected in a predetermined region.

The OOS as well as the AMARO system software are currently developed for x86-64 Linux workstations. However, on a future space mis-

[1]sensors currently used: optical cameras, Photonic Mixer Device (PMD) [2], LIDAR sensors

sion these applications have to run on space-qualified hardware on-board the satellite. Therefore our group is also heavily involved in the development of a future high-performance on-board space computer, primarily in application development [5].

1.2 CHALLENGES

Making the OOS and ODARIS applications compatible for running on-board of a satellite comes with three major challenges:

1. Computing performance
2. Reliability
3. Platform compatibility

First, the **computing performance** available on the current generation of DLR small-medium sized on-board computers that we target as carrier platform of our applications, is far from being sufficient. For example, a typical system may be built on top of an LEON3 processor [6], running with less than 100 MHz and having 64 MiB RAM available. This kind of systems are meant for control and light computing tasks. They are not sufficient for data processing tasks, like image or video data analysis, which are part of our applications.
Second, the applications have to be far more **reliable** than the laboratory versions, due to the very restricted supportability of the system. Applications also have to be prepared for on-board computer failures due to single event upsets and alike. It may even be necessary to keep control of the spacecraft while handling those events in critical situations.
Third, the applications have to be **platform compatible** to the on-board computer (OBC) system, both from the hardware and the software perspective. From the hardware perspective the application has to support the OBC target computing architecture, which under normal circumstances differs essentially from the platform used for developing the laboratory version of the application. Also hardware interfaces to system components used for the laboratory version of the application may not be supported on the on-board system. From the software perspective the application has to be compatible with the on-board systems software and development environment. Significant differences between the laboratory development environment and the on-board computing environment contest the re-usability of already developed code. Differences can be the Operating System (OS) used, supported computing language, supported build tools, availability of software libraries, interfacing to external hardware and system resources.

1.3 OBJECTIVES

Hereafter, an overview of the on-board application development activities of DLR-OOS is presented. First, a short description of the applications currently in progress is given. Then it is shown how the challenges described in section 1.2 are approached. In the end an outlook of research topics planned in the future is presented.

2 APPLICATIONS

As stated in section 1, two applications are currently developed at DLR-OOS: the AMARO system and the GNC system. This section explains both applications in detail.

2.1 REAL-TIME ON-BOARD INFORMATION SYSTEMS

In the last years a prototype of a real-time on-board information system was developed [4]. Its main features are analysing sensor data on-board and allowing a user to query the data or get an alarm when predefined events happen. It was developed within the AMARO project at DLR. An idea sketch can be seen in figure 1.

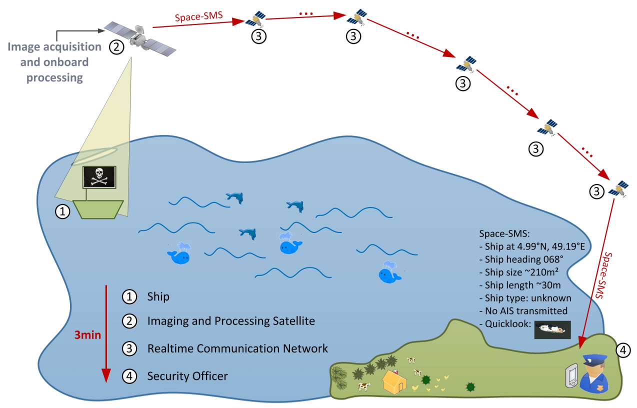

Figure 1: AMARO idea

Within a flight campaign 2018 over the North Sea it was shown, that ships could be detected on board, that a user could query information via direct communication with the satellite and get the information within three minutes after sending the request [4]. Furthermore, it was shown that the user could be informed automatically when a predefined event happened.

The AMARO system consists of a camera, an Automatic Identification System (AIS) receiver [7], an Iridium satellite Internet of things (IOT) communication device [8] and a x86-64 based on-board computer. On the latter, a standard openSUSE Linux distribution [9] was installed, on top of which the AMARO system software was executed. The AMARO system software is mainly programmed in C++ [10], and its main tasks were the execution of data analysis and the communication with the ground [2]. The system software itself consists of several independent working services communicating over a database. An overview is shown in figure 2.

[2]answering user requests and sending alarm messages

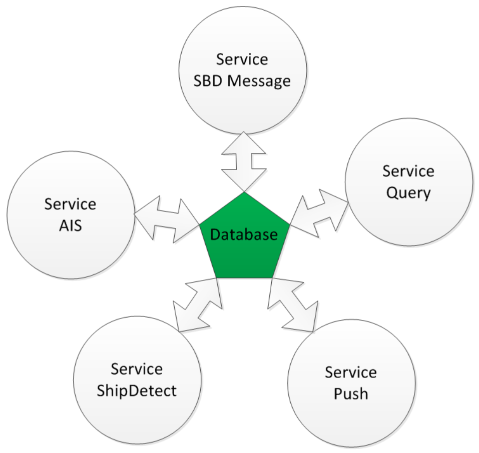

Figure 2: AMARO service architecture

The system handles intensive data processing while being responsive and reliable. At the current state a user can query data from the system's internal database by directly sending Structured Query Language (SQL) [11] requests via email over the Iridium IOT service. Also for the alarm service SQL requests are used, which are executed periodically. In case of the result of the query not being empty, the user is informed.

2.2 GNC-APPLICATION FOR DEBRIS-REMOVAL & ON-ORBIT SERVICING

The rendezvous GNC application [12] controls a chaser satellite's approach towards a possibly uncooperative target object of an OOS mission by processing the information from optical 2D and 3D sensors. Mainly dictated by the capabilities of the associated hardware-in-the-loop environment EPOS [1], the GNC application focuses on the final approach beginning at roughly 20 meters up to a mating point close to the target. In its current state, the rendezvous GNC system is equipped with a Charge-Coupled Device (CCD) mono-camera and a PMD 3D camera [13], with the former serving as the primary navigation sensor. An edge tracking algorithm calculates the relative pose of the target with respect to the chaser satellite from the CCD image data. An Extended Kalman filter computes a smoothed solution from the combined estimates of CCD - and PMD - camera-based trackers. The guidance function provides a smooth and continuous trajectory assembled from a succession of automated sub-trajectories. Finally, the position controller computes control forces from filter estimates and guidance trajectory.

In the future, GNC application development will lead to more sensors, like a scanning LiDAR, and more sophisticated pose initialization and tracking algorithms to increase robustness, reliability and flexibility.

The GNC application is designed as decentralized modular software in C++ to support parallel and asynchronous as well as limited-performance computer architectures. For example, the Kalman filter processes input data with time stamps, such that a considerable and varying delay, a typical property of non-deterministic pose estimation algorithms in general, can be handled robustly and elegantly. All its modules provide separate telemetry and telecommand (TM/TC) capability via ESA Packet Utilisation Standard (PUS) [14] conform packets to support control of the approach in a real multi-mission control room at German Space Operations Center (GSOC).

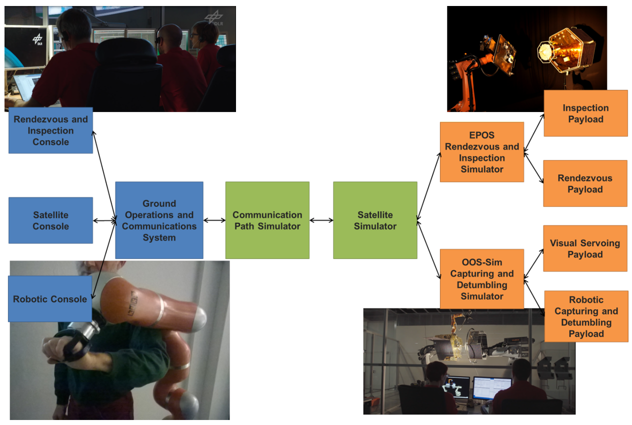
Rendezvous and Inspection Console
Satellite Console
Robotic Console
Ground Operations and Communications System
Communication Path Simulator
Satellite Simulator
EPOS Rendezvous and Inspection Simulator
OOS-Sim Capturing and Detumbling Simulator
Inspection Payload
Rendezvous Payload
Visual Servoing Payload
Robotic Capturing and Detumbling Payload

Figure 3: Architecture of End-to-End Simulation (E2E): blue: ground segment, green: software simulation, orange: hardware-in-the-loop simulation

This close-range approach is routinely performed as part of the distributed End-to-End-Simulation [3] shown in figure 3, which is developed by a cooperation of four DLR Institutes at Oberpfaffenhofen and Berlin, Germany. This setup allows to simulate and test the most critical phases of an OOS mission in a unique simulation environment. E2E includes a full ground segment and simulations of the communication path and the space segment. The space segment comprises a satellite simulator, two hardware-in-the-loop test facilities and the inspection, rendezvous and robotic payloads. Future OOS and space debris removal missions can use E2E, for example for servicing of DLR compact satellites. Single subsystems can be easily replaced due to its decentralized and modular architecture. This allows to integrate for example new sensors or algorithms developed by partner institutes, industry or universities.

3 ON-BOARD COMPUTING PLATFORM

Migrating the applications described in section 2 to a space computing platform poses major challenges.

First, the space systems we target are small to medium sized (experimental) satellites like Bi-spectral InfraRed Optical System (BIROS) [15] or Eu:CROPIS [16]. On these platforms, the available computing systems were designed to execute control tasks, and are not capable of handling computationally intensive applications. Neither do they offer enough computing resources nor do they support a system software stack that is adequate for developing larger software applications.

To address these issues, DLR launched the project Scalable On-board Computing for Space Avionics (ScOSA) [17]. In this section, an overview of the hardware- and software architecture of this on-board computer is presented.

3.1 HARDWARE PLATFORM

ScOSA builds upon a distributed computing architecture [18], where the on-board computer consists of several independent computing nodes, which are connected via spacewire or ethernet. Managing the nodes is done on a software level [19]. A schema of the system can be seen in figure 4.

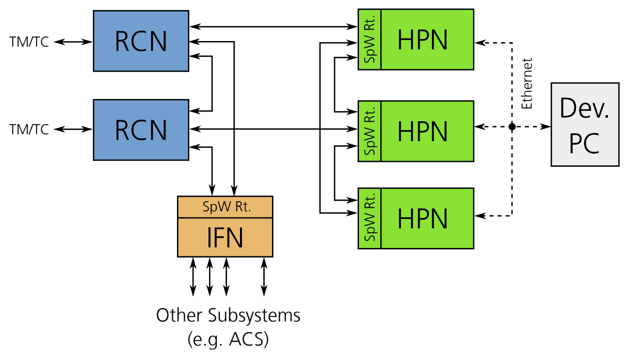

Figure 4: block diagram of the ScOSA system base design and test-bed [17]

Three types of nodes are currently used, each specialized for a different aspect: Reliable Computing Node (RCN), High Performance Node (HPN) and Interface Node (IFN). Reliable nodes are very similar to classical on-board computers, using certificated space hardened hardware components. The focus is reliability and endurance. The current version of the RCN uses a radiation hardened LEON3 System on a chip (SOC) accompanied by a flash-based Field Programmable Gate Array (FPGA). The RCN offers a 50 MHz dual LEON3 processor [6], a 64 MiB error correcting RAM and non-volatile memories for boot-images and arbitrary uses [17]. The RCNs are primarily used for critical system management and surveillance tasks, which are less computationally demanding.
For executing computationally intensive tasks, the HPNs are used. To achieve high computing performance and keep system costs down, Commercial off-the-shelf (COTS) hardware is used. At the moment a Xilinx-Zynq Z7020 SOC [20] is used accompanied by 1 GiB of Random-access memory (RAM) and 4 GiB flash-based non-volatile memory for the operating system and the application software. The dual core ARM Cortex-A9@866 MHz CPU [21] of the Xilinx-Zynq SOC offers significantly more performance than the LEON3 used for the RCNs. Apart from this, an FPGA is part of the Xilinx-Zync SOC, which is intended to be used for implementing highly computationally intensive tasks. Due to the nature of a distributed system, it is possible to use other kinds of high-performance nodes, like GPU accelerators or in the future replace them with modern, more performant hardware, without the need of replacing the whole system.
For designing the applications we are currently using the development board shown in figure 5, which consists of three HPNs. For software development RCNs are not necessary, because the HPNs can execute the system management tasks, too. For connecting external components and for controlling the system, standard Ethernet is used.

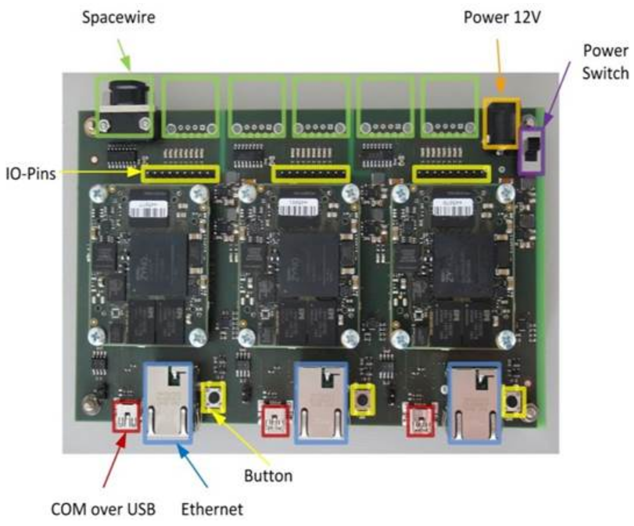

Figure 5: development board consisting of 3 HPNs

For connecting with external components like cameras IFNs are used. IFNs are specially built for connecting with external components. A bread-board version of an IFN is shown in figure 6.

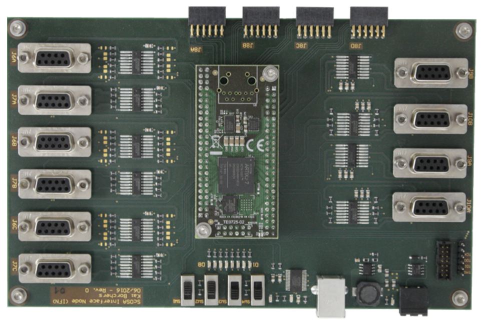

Figure 6: Interface Node with 12 RS-422 connectors, 4 space wire ports and an artix7 control FPGA

3.2 SOFTWARE PLATFORM

In the next chapter an overview of the software stack of an HPN is given, as this is the relevant part for application developers. A graphical illustration of the software stack is shown in figure 7.

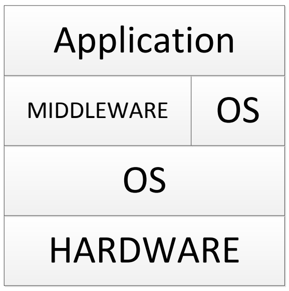

Figure 7: middleware based software stack of the ScOSA system.

The software stack of the HPN supports Yocto Linux and the RTEMS Real Time Operating System [22]. In this article we focus only on the Yocto Linux software stack, as this is used for the applications. Yocto Linux is a customizable Linux distribution provided by the Yocto Project [23]. Yocto Linux is specialized on embedded platforms and supports a wide range of open source libraries and tool. Since our applications are Linux based, we can reuse a substantial portion of our software for the on-board computing versions. The ScOSA middleware is managing the different nodes of the on-board computer and is responsible for deploying the tasks on the different nodes, for inter-node communication, for failure detection and recovery, for reconfiguration and many other system aspects [19]. For programming applications, the middleware offers a data-flow driven Application Programming Interface (API) which allows application developers to take advantage of the distributed architecture of the ScOSA system. A schema of a typical data processing application can be found in figure 8.

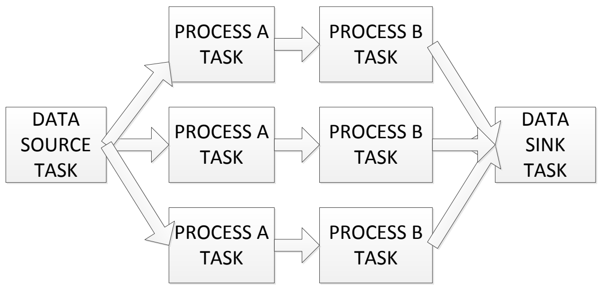

Figure 8: data-flow schema of a typical application, with the data source distributing the data to three different execution paths, to speed up processing.

The tasks can be distributed to different computing nodes, while the ScOSA middleware handles the communication. One feature of the ScOSA middleware is, that it can detect if a node is failing, move the tasks on an other functional node and restart them without stopping the application. Furthermore, the application developer can use the check-point feature of the ScOSA framework, such that the interrupted tasks are restarted by the framework with their last valid state. More about application development for the tasking framework can be found in [5] and more details about architecture and features of the ScOSA middleware in [19].

4 DEVELOPMENT METHODOLOGY

In this section our development methodology for migrating the laboratory versions of our applications to the ScOSA on-board computing platform is presented. Our migration process involves several steps:

1. Porting the software from the laboratory computing platform to the on-board computing platform.
2. Integrating the applications in the on-board computing software framework.
3. Optimizing the application performance.
4. Adding Fault Detection, Isolation and Recovery (FDIR) functionality to the application.

The first step is **porting the applications** from our laboratory computing platform to the on-board computing platform. As stated in the section 1.2, this may be a big issue, if the laboratory platform is too distinct from on-board platform, even leading to complete redevelopment of the application. The applications presented in section 2 are developed for x86-64 Linux, written mostly in the C++ programming language [10] using the CMake build system [24] and using open source libraries as Boost [25], OpenCV [26] and SQLite [27]. A major feature of the ScOSA system is, that it supports the Poky Linux Distribution as an OS. One important advantage of using the Poky Linux Distribution is, that it offers quite a complete collection of open source libraries, programs and tools. All important libraries and programs used for the laboratory version of the applications are available for the on-board platform. Without changing a lot of code, we were able to build the applications for the ScOSA on-board system and to successfully run them on one ScOSA HPN.

The second migration step is **integrating the application** in the ScOSA software framework. This is a necessary step for getting support for starting and executing the applications on satellite operation, getting access to external interfaces and using the performance and FDIR features of the ScOSA system. For integrating applications in the ScOSA framework, it offers a C++ API. Nevertheless porting a free-standing Linux application in a framework requires some refactoring. We split the applications in a system-independent library part and an system-dependent application part. The library part contains all program logic and is independent of the system API. The system-dependent application part contains code parts for the access of system resources and the execution of the application. With this approach we were able to build up a shared code base, which we use for laboratory as for the on-board setup.

The third point is **optimizing application performance**. The distributed computing architecture of the ScOSA system is quite different from the shared memory architecture of our applications. To be able to use more than one node of the ScOSA system, the applications have to be divided in smaller execution units, called tasks. Tasks are only able to share data, by sending and receiving messages from other tasks. In the ScOSA framework the tasks are connected like parts on an electric circuit, as illustrated in figure 8. Each task can be configured to run on one of the HPNs of the ScOSA system. Communication and execution management is handled by the ScOSA system software.

With both our applications based on a shared memory architecture, quite a lot of work has to be done to make them compatible to a data-flow architecture. The OOS application is a parallel application, where different threads communicate over a shared memory pool. The ODARIS application is working similar, with the difference that the different services are communicating by using (shared) SQLite database files.

The last migration step is **adding FDIR support** to the application. The ScOSA framework can detect the failing of a computing node and it can migrate the tasks executed on the failing node to functioning nodes. For configuring the initial state of a migrated node, the application can use a checkpoint service. With the checkpoint service, important status information of a task can be continuously registered, which can then be used for the initialization procedure of a migrated task.

More details about developing applications for the ScOSA platform can be found in [5].

5 RESULTS

In this section some results of our approach and insights, regarding the space application development of the group OOS is presented.

As noted in section 1.2 there are three major challenges, that we have to handle, to being able to migrate our applications, presented in section 2 to a on-board computing platform: Computing performance, reliability and platform compatibility.

Let's start with the **computing performance**. The computing performance of the current generation of small-medium sized DLR satellites, we target as carrier platform for our applications, is not sufficient to handle our data processing applications. But with DLR's progress in the development of the next generation high performance on-board computing platform ScOSA, we are at a point, where we can start migrating our applications to an on-board platform. We were able to migrate and execute the OOS and the ODARIS applications to one ScOSA HPN. Since the computing performance of one ScOSA HPN is about 10-100 times less compared to our workstation laboratory PCs, we can execute the applications, but the application performance is not sufficient for operation yet. We could already demonstrate a successful rendezvous for one depicted test case, but the system is not yet able to handle more difficult cases. For the OOS application we are currently in the process of optimizing the performance of the application. With the ScOSA computing platform there are two starting points, where we suppose that a significant increase of the performance of our applications is possible:

1. multi-node support (distributed computing)
2. hardware acceleration (with FPGA)

Unfortunately these two options require a huge amount of development effort. For the OOS application we are currently in the process of adding multi-node support. As the distributed computing architecture of the ScOSA platform is quite different from the shared memory architecture of the laboratory version of the application we have to heavily refactor the way how data is distributed within the application. Currently we think that it will also be necessary to accelerate some intensive parts of the OOS application, by using the FPGA available on the ScOSA HPNs. For the ODARIS application computing performance is not as critical, but if we want to be able to analyse huge amount of sensor data (for example multi-spectral image data) within a moderate time span, we also have to face computing performance. But currently we are still in the process of migrating the application to the on-board platform.

The second challenge is making the applications **reliable** enough for space operation. There are two levels we have to address concerning reliability:

1. application reliability
2. system reliability

With application reliability we mean that the application should be able to handle each situation that can occur during a mission when the computing system is operating regular. This means that the application does not have any software bug that effects the operability of the application and that it is able to handle each scenario which can occur. For both of our applications we are currently working on enhancing the software quality and the test coverage to a point, where the occurrence of software bugs concerning the operability of the mission is quite unlikely. The harder point is to ensure that the application can handle each scenario which can occur during a mission, specially for the OOS application, as it is system critical. The most critical point here is the pose estimation of the target object. This is also related to computing performance, as with higher computing performance we are able to use more reliably but also more computationally intensive algorithms. Furthermore, we can increase the sensor sample time and we can even use additional sensors for target identification and tracking observation. With system reliability we mean that the system should even be reliable if the computing system is performing irregular. This happens for example if a computing node malfunctions because of an radiation impact. For this kind of problem the ScOSA system offers FDIR features. Until now, we have not started to implement this features.

The last challenge is **platform compatibility** of our applications. Our applications have to be compatible with the on-board computing system. With the current generation of DLR satellites this would have been a big issue as they support only real-time operation systems or specialized systems, which have a significantly different API as the Linux architecture. This would have required to write specialized versions of the applications for the on-board system, which would have required the redevelopment of a huge amount of the codebase. Specially, some software libraries we are using, may not be supported by the on-board platform. And last but not least, the on-board version and the laboratory version do not share code anymore, which would force us to support and develop two systems in parallel. But as stated in section 4 with the ScOSA on-board computing system, supporting Yocto Linux, we where able to migrate the laboratory Linux applications to the ScOSA computing system, nearly one to one. As stated in section 4 still large changes have to be done, to integrate the applications in the ScOSA middleware framework, due to the software architecture differences. But with splitting the applications up in an API-independent library part and an API-dependent application part, we are able to share most of the codebase with the laboratory versions. This means that we only have to support one version of the library part, which inhibits most of the codebase. With that, for example new features and bug fixes integrated in the laboratory version, are immediately available for the on-board version, too.

Altogether, with the modern system architecture and the higher computing performance of the ScOSA system, we were able to get a big step closer towards a future on-board implementation of our applications.

6 SUMMARY

This article gives a short overview over the space application development we are conducting at the DLR OOS group. The three main challenges we face for on-board computing application development are computing performance, reliability, and platform compatibility. We are currently working on two applications, an application for "Guidance, Navigation and Control for On Orbit Servicing" and an "Onboard Data Analysis and Real-Time Information Systems" application. These applications require much more computing resources as available with the current generation of on-board computers used by the DLR. To address this issue, DLR has started to develop a reliable high performance on-board computer platform, called ScOSA. For getting high performance and reliability, the ScOSA system is using a distributed computing architecture in combination with FPGA hardware acceleration. An important feature of the ScOSA system for application development, is the offering of a rather complete Linux open source development stack. With support of the Linux API and a rich set of open-source libraries and tools, developing applications and deploying software is much more simplified, relative to specialized on-board computing operation systems. Linux laboratory application versions can be migrated to the on-board computing system bit by bit, performing following steps: Porting the software to the on-board computing platform, integrating the application in the on-board computing software framework, optimizing the application performance and adding FDIR functionality to the application. One important result is, that we were able to migrate the applications from the laboratory version to the ScOSA platform, with sharing a large portion of the source code with the laboratory version of the applications. One other result is, despite the ScOSA system being a performant on-board computing system out of the box, without optimizing our applications for the ScOSA system, the application performance is barely sufficient for real-world scenarios yet. But with optimizing the application for the distributed computing architecture of the ScOSA system and with making use of the hardware acceleration via the built-in FPGA, we are confident that we can reach an acceptable application performance.
As an overall conclusion, we think that we are on the right track to bring our applications to space, and that we have the tools and knowledge available to achieve this.

7 ACKNOWLEDGMENTS

A thanks to my colleagues Matthias Burri, Florian Rems, Heike Benninghoff and Michael Beer whose comments and suggestions significantly improved both clarity and precision of the paper. Also thanks to the ScOSA project colleagues for providing information and illustration material.

References

[1] Benninghoff, H., Rems, F., Risse, E.-A., and Mietner, C., “European proximity operations simulator 2.0 (epos) - a robotic-based rendezvous and docking simulator,” *Journal of Large-Scale Research Facilities JLSRF* (April 2017).

[2] Klionovska, K., Benninghoff, H., and Strobl, K. H., “Pmd camera- and hand-eye-calibration for on-orbit servicing test scenarios on the ground,” in [*14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA)*], (Juni 2017).

[3] Benninghoff, H., Rems, F., Risse, E., Brunner, B., Stelzer, M., Krenn, R., Reiner, M., Stangl, C., and Gnat, M., “End-to-end simulation and verification of gnc and robotic systems considering both space segment and ground segment,” *CEAS Space Journal* **10**, 535–553 (Dec 2018).

[4] Schwenk, K., Willburger, K., and Pless, S., “Amaro-autonomous real-time detection of moving maritime objects: introducing a flight experiment for an on-board ship detection system,” (2017).

[5] Schwenk, K., Ulmer, M., and Peng, T., "Scosa: application development for a high-performance space qualified onboard computing platform," in [*Proc. SPIE 10792, High-Performance Computing in Geoscience and Remote Sensing VIII*], *Proceedings of SPIE* **VIII**, SPIE Remote Sensing (September 2018).

[6] Cobham Gaisler AB, "LEON3 product description." https://www.gaisler.com/index.php/products/processors/leon3 (2019).

[7] Wikipedia, "Automatic identification system." https://en.wikipedia.org/wiki/Automatic_identification_system (2019).

[8] Iridium Communications Inc., "Iot Products." https://www.iridium.com/iot-products (2019).

[9] SUSE LLC., "Open Suse." https://www.opensuse.org (2019).

[10] Standard C++ Foundation, "The standard C++ foundation." https://isocpp.org (2019).

[11] Wikipedia, "Structured Query Language." https://en.wikipedia.org/wiki/SQL (2019).

[12] Rems, F., Risse, E.-A., and Benninghoff, H., "Rendezvous GNC-system for autonomous orbital servicing of uncooperative targets," in [*Proceedings of the 10th International ESA Conference on Guidance, Navigation & Control Systems*], (2017).

[13] Klionovska, K., Ventura, J., Benninghoff, H., and Huber, F., "Close range tracking of an uncooperative target in a sequence of photonic mixer device (PMD) images," *Robotics* **7**(1) (2018).

[14] European Cooperation for Space Standardization, "Ecss-e-st-70-41c – telemetry and telecommand packet utilization," (Apr. 2016).

[15] DLR, "Mission Firebird." https://www.dlr.de/dlr/desktopdefault.aspx/tabid-10891/1596_read-17992/#/gallery/23115 (2019).

[16] Schulze, D., Philpot, C., Morfill, G., Klein, B., and Beck, T., "Food production in space – operating a greenhouse in low earth orbit," in [*SpaceOps 2016*], (Juni 2016).

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

[18] Lüdtke, D., Westerdorff, K., Stohlmann, K., Börner, A., Maibaum, O., Peng, T., Weps, B., Fey, G., and Gerndt, A., "Obc-ng: Towards a reconfigurable on-board computing architecture for spacecraft," in [*2014 IEEE Aerospace Conference*], 1–13 (March 2014).

[19] Peng, T., Höflinger, K., Weps, B., Maibaum, O., Schwenk, K., Lüdtke, D., and Gerndt, A., "A component-based middleware for a reliable distributed and reconfigurable spacecraft onboard computer," in [*35th Symposium on Reliable Distributed Systems (SRDS)*], *Proceedings of the IEEE Symposium on Reliable Distributed Systems* , 337–342, IEEE (September 2016).

[20] Xilinx, "Zynq product description." https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html (2019).

[21] ARM Ltd., "Cortex a9 product description." https://developer.arm.com/ip-products/processors/cortex-a/cortex-a9 (2019).

[22] OAR, "Rtems offical website." http://www.rtems.com (2019).

[23] Yocto Project, "The Yocto Project." https://www.yoctoproject.org (2019).

[24] Kitware, "CMake." http://www.cmake.org (2019).

[25] boost.org, "Boost C++ Libraries." https://boost.org (2019).

[26] OpenCV team, "OpenCV." https://opencv.org (2019).

[27] SQLite Project, "SQLite." https://sqlite.org (2019).