On Board Data Analysis and Realtime Information System

1st Schwenk Kurt
German Space Operations Center (GSOC)
German Aerospace Center (DLR)
82234, Wessling
kurt.schwenk@dlr.de

2st Daniel Herschmann
German Space Operations Center (GSOC)
German Aerospace Center (DLR)
82234, Wessling
daniel.herschmann@dlr.de

***Abstract*—For many information applications like alarming services or surveillance systems, reactivity is a very important criterion. This is also true for many applications based on satellite data, for example in the field of remote sensing or space system operation. These applications are currently bottlenecked by classical satellite operation, where contact to the satellite is only established within a few communication windows, when the satellite is in reach of an assigned ground station. The amount of time between collection of the data on the space system and having it available for ground-users can take from several hours up to days.**

At our institution, we are working on a cost-efficient method to decrease the latency until information, derived from satellite data, becomes available for usage on ground. The basic approach is trying to utilizing satellite based low-latency real-time telecommunication networks, designed for ground-based satellite phone usage in combination with on board information extraction. Information extraction on the satellite will be necessary as the telecommunication networks only offer a very limited data-transfer bandwidth in the range of a few hundred bytes and the transmission of raw sensor data is not possible.

***Index Terms*—on board data processing, on board data handling, system reconfiguration, commercial of the shelf components, artificial intelligence, system architecture, software development, quality of service, safety**

I. Introduction

Currently many satellite missions follow a time-window based, static management philosophy. When a satellite has contact to a ground station, telecommands containing a time-based static command list are sent to the satellite and telemetry and payload data are received. Payload data is usually processed at a data center on ground, after the data was downloaded from the satellite. Many satellites, particularly smaller low earth orbit satellite missions, have a contact window every 6-24 hours, to reduce the operational costs. For data collection applications, these ramifications might be quite sufficient. For time-critical applications the long duration in-between flight windows, where no information transfer between the ground and space segment can be performed and no information from the sensor data can be utilized, is quite a limiting factor for the quality of service of the mission. Two typical application scenarios, which are currently bottlenecked by current satellite operations, are alarming services, for example fire detection, and space system health services.

Having the capability to detect a critical event on board and getting immediately informed about it, would significantly improve the quality of service of satellite operations, due to the opportunity of a timely reaction. At the moment, implementing a real-time information system is challenging, especially for smaller satellite mission designs, as real-time links are not commonly available on these platforms and critical event detection on board requires, compared to traditional mission designs, advanced on board data management and processing capabilities.

Therefore, we are currently working on the ”On-board Data Analysis And Real-Time Information System” (ODARIS), with the goal to be able to implement real-time services with an information latency around a few minutes on board of especially small- to medium-sized satellite systems.

Our approach for providing an always-on real-time connection link, especially for small satellite missions, is to re-utilize satellite telecommunications networks like the Iridium network [1]. These networks are designed for ground usage but have also been successfully used in a space environment for low earth orbit satellites [2]. These communication networks may be able to provide a low latency 24/7 communication link, with the downside that the data bandwidth is very limited. The service we are currently using enables us to send and receive text messages from the ground with a size of around 200 bytes and a total volume of up to 1 MB / day, assuming very good connectivity with the communication network [3]. To circumvent these limitations, we are currently working on an on board data processing and management software system which collect and extract useful information from sensor data on board as well as sending alarming messages to the ground via the real-time communication link. Furthermore, the system shall also enable us to access information at any time from the satellite platform.

II. Related work

The EO-ALERT project founded by the European Commission H2020 program, has similar goals as ODARIS. The aim of the project is to propose, design and validate a next generation data processing chain for earth observation satellites through ground testing. The goal of the EO-ALERT architecture for providing SAR and optical EO products to the end user with a

latency of less than five minutes, and targeting a latency of one minute for alarming services. The prototype application chosen to show the concept are ship detection based on SAR data and extreme weather monitoring. Data is processed on board, using a standard LINUX ARM-based embedded computing platform, based on the AMD ZYNQ computing series. For real-time communication the INMARSAT geostationary satellite service, providing a maximum data rate of about 250kbit/s, was chosen [4].

The ESA OPS-SAT mission is one of the many recent examples, using a standard ARM-based Linux operating system embedded computing system for for the payload processing unit. This choice enables the (re-)usage of modern software components, like Tensorflow Lite Library [5] , in a space environment, which were not specifically developed for space usage [6]. It was demonstrated that even a 3U Cubesat could provide enough computing resources to run state of the art AI-based data analysis applications [7].

III. History

The first phase of ODARIS development started 2015 with a requirements analysis and the creation of a concept for a real-time information system. 2018 a first prototype of the information system was demonstrated on an aircraft platform performing on board ship detection. The goal was to detect ships from high resolution visible optical camera data and identify ships which are not sending mandatory Automatic identification system (AIS) signals [8]. With begin of 2019, we started to prepare a couple of satellite missions, with the goal to be able to demonstrate the ODARIS concept within a space environment [9], [10]. These involve tasks, like finding a flight opportunity, identifying system requirements for space operation, identify strategies for implementation and set up a development and test environment.

IV. Objectives of this paper

The object of this article is to give an overview of the background, motivation, goals, features, system architecture, boundaries and planned verification missions of the ODARIS project.

V. System architecture

In the following section a brief overview over the system architecture is presented. The main design targets of the ODARIS system are to enable satellite-based real time alarming services, increase the efficiency of satellite operation, a broad support for flight platforms and providing a host platform for space service applications. Based on these design targets, the design of ODARIS has been derived.

A. Hardware architecture

In figure 1 a schematic of the hardware components of a typical ODARIS implementation is shown. The system consists of

- an on board payload processing unit, responsible for sensor data processing and system management
- an real-time communication link for enabling real time communication
- various sensors, typically cameras or system monitoring sensors
- an connection to the satellite bus

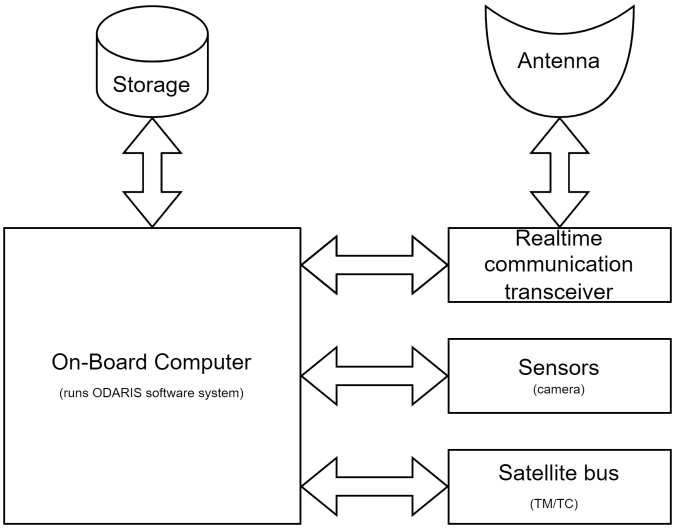

Fig. 1. Schematic of the hardware architecture of a typical ODARIS implementation

At the moment a "RaspberryPi-1" [11] like computing system running a Linux operating system (LINUX) is sufficient to execute the ODARIS software system. For developing and testing we are currently using a Commercial Off-The-Shelf (COTS) AMD Zynq-7020 SoC, which contains a dual core ARM Cortex-A9 processor [12]. For the real-time communication link the EyeStar-S4 Radio from Near Space Launch (NSL) [13] is planned for usage. Derived from the specification, it should enable us to send and receive text messages with a size of around 200 Bytes and a latency of 5-30 seconds. The maximal data rate is stated with 13.5 bytes/sec which adds up to around 1 MB per day [3]. Additionally, a sensor, like a camera or system monitoring sensors, will be attached to the ODARIS system as a data source. Finally, the ODARIS system will be attached to the satellite bus for power and TM/TC communication. To show the size of the ODARIS hardware components, the currently used development hardware is shown in figure 2.

B. Software architecture

Based on the design targets, the following requirements for the software system have been derived. The software systems should be:

- responsive, to allow immediate reaction on user requests and data input
- sufficient reliable and efficient to be executed in a space environment
- extensible to be able of adding third-party data processing applications and attach a broad range of sensors

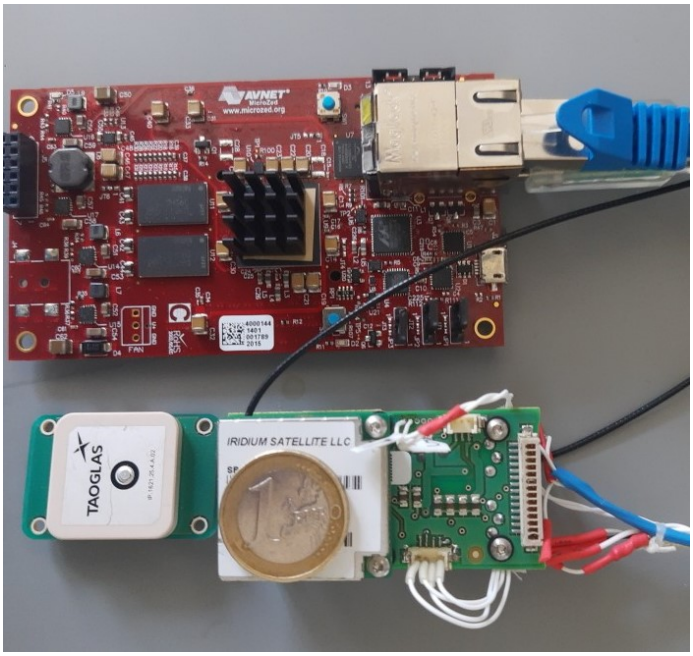

Fig. 2. Image of the development hardware components currently used. Top: Zynq-7020 MicroZed Board [14], Botton-right: EyeStar-S4 radio, Botton-left Iridium patch antenna

- flexible for adequate usage in highly dynamic mission scenarios, and at last provide sufficient computing resources to be able to perform modern data processing tasks as AI-based object detection

To suite these requirements, we decided to use a service driven, database centric, message-based software architecture The core components of the ODARIS software system are on one hand stateless software services for performing data processing and system management task. And on the other hand, a database system for information storage and intra-service communication. Currently the ODARIS core services are implemented as C++/Linux applications, but in principle every application which is able to be connected to a database can be attached. As database system we are at the moment using the SQLite [15] file-based database engine.

A schematic of the service architecture is shown in figure 3. Currently the ODARIS system is developed and tested on an embedded LINUX system [16] executed on the ArmV7 architecture and additionally to identify architecture specific issues also on an LINUX based x86-64 system. For an exemplary AI-based object detection application the Tensorflow Lite Library [5] (TFLite) is utilized.

.

C. Service architecture

The services itself can be executed as standalone applications on a LINUX system. Or they can be integrated in a host application via (library) function calls. If started as standalone processes a dedicated service manager is required for starting and stopping the ODARIS services. Currently we use

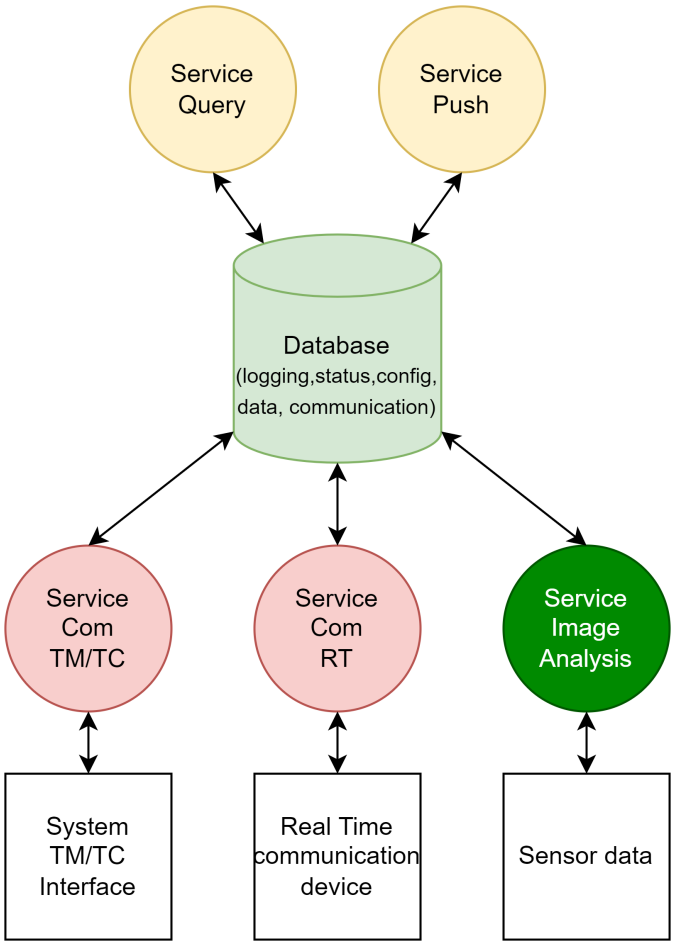

Fig. 3. Schematic of the ODARIS system service architecture

Bourne-again shell (Bash) shell script for starting, observing and stopping the services for standalone applications. If the services are integrated via function calls, the host application is responsible for the service management. Initial configuration of the services is performed at startup time using configuration files. The services are only loosely coupled, in the sense, that they can be started and stopped independently of other services execution states. Services are also not directly communicating with each other. There is no point to point communication channels necessary between the services. The services exclusively exchange information via the system databases. There is also no service communication protocol used. The services write data they want to publish in the system databases. Other service are able to consume the information, by querying the database. The (consuming) services are designed not to require any input from another service to increase flexibility and safety. External system resources are accessed by exactly one (interfacing) service. As an example, an TM/TC service will query a TM database table for TM data, inserted by other services, and propagate it to one of the available TM channels. As another example, an image analysis service will directly access image data from a camera, analyze it and also

write the result in a database, which could be consumed by other services. More details about this process is provided in the next section. Different application modes can be defined simply via a mode specific list of active services and mode specific configuration files. The services itself are currently implemented as stateless application which have periodically triggered activity windows. State data is read and written from the databases.

D. Communication architecture

As explained, intra-service communication is performed over system databases. Communication with external devices is performed via dedicated services, which are exclusively communicating over the specific device interfaces. The ODARIS services can be categorized in three service classes: data processing services, communication services and information management services. In figure 4 a typical data flow is presented, visualizing the role of the service classes. Data processing services fetch data from a data source, like a camera, compute it, and write mission relevant data to a system database. Communication services are responsible for sending and receiving data over one of the available communication channels. For example, the ServiceComRT is responsible for sending and receiving data over the real-time (RT) communication link and the ServiceTM/TC is responsible for sending and receiving data over the TM/TC channel. If a service wants to send or receive data from one of the communication channels, it uses the internal ODARIS message system, by fetching and storing messages from internal post-in or the post-out boxes. Post-in and post-out boxes are implemented as databases tables with a specific format. Internal ODARIS message functions are used by the services to ensure that the messages are inserted in the database table in the expected format. The ODARIS message system is exclusively used by the communication and information management services. The information management services are responsible for generating messages which are sent to the user on the ground. Currently the only representatives of this class are the ServiceQuery and the ServicePush. The ServiceQuery fetches user query messages from the post-in box, generates a response message by querying the system databases based on the query message content and puts the response message back to a post-out box. The ServicePush periodically accesses the entries of a push event database table and checks regularly if the event has occurred, by also querying system databases. If an event occurs, an event response message will be generated an put in one of the post-out boxes. The push event database table can be modified by the user during operation via one of the communication links. The design choice to use different services for different tasks, has the advantage that data generation is completely decoupled from system communication. Data processing services do not send or receive any data from the message system directly. This has the advantage that all data processing services can share the same communication infrastructure and application developers don't have to decide when and what data has to be communicated to the ground segment. They just have to ensure that their application puts the product data in an consumable format in one of the system databases. An operator can define alarming events and query data required directly from the system, at mission operation time. For formulating query requests and push events the Structured Query Language (SQL) is used.

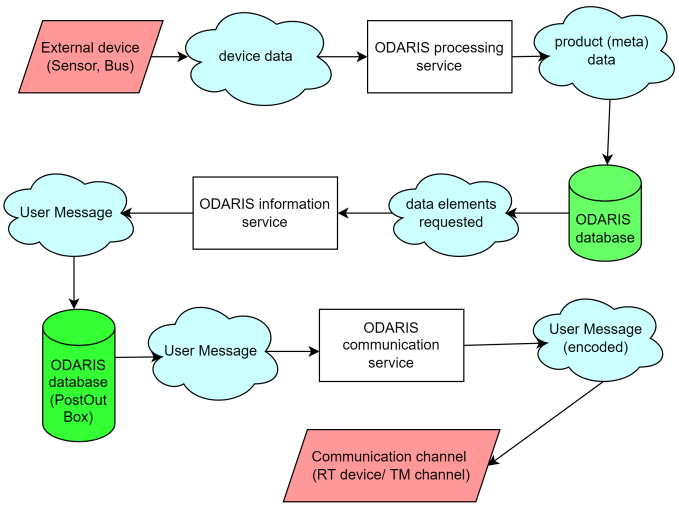

Fig. 4. Diagram of a typical dataflow within the ODARIS context

E. Data management

Currently all data which should be accessible via the communication services is stored in system databases. Large data artifacts like sensor data and data analysis results may be stored as files. These files could be transmitted to the ground via data downlink channels. Currently the downlink of large data sets to the ground via the real-time channel is not supported, as the maximal data rate mentioned by the specification is 13.5 bytes/sec which is around 1 MB per day [3]. Nevertheless, small sized data as for example an image section, containing an detected object, can be sent down via the RT link using multiple messages. Meta data, for example the image acquisition time, is also inserted in the system database. The database is also used for storage of system logs and status information. A logging library is used to ensure coherent system logging of the ODARIS services. In figure 5 an excerpt of of a ODARIS time based system log and in figure 6 the status information of the ServiceQuery is shown. Note, that these information can be accessed by the RT link via a user query or via an push event. During operation, we also plan to periodically send down a snapshot of the ODARIS databases to the ground via data downlink channels.

F. Safety concept

At the moment the ODARIS system is designed with the requirement to be executed in an safe enclave, which enforces that ODARIS cannot affect critical system components. On our currently planned missions this is realized by executing ODARIS on a dedicated payload processing unit where no mission critical tasks are performed. Furthermore ODARIS does not access

	id	Timestamp	Severity	Module	Message
	Fi…	Filter	Filter	Filter	Filter
29	29	2023-07-05 15:28:59	info	serviceQuery	Initialized service_query
30	30	2023-07-05 15:29:09	info	serviceQuery	Received SQL-Query: system.db SELECT *
31	31	2023-07-05 15:29:09	warning	serviceQuery	Error SQL-Query: SELECT *, (likely wrong sql …
32	32	2023-07-05 15:29:09	error	Exception_DbSQLite_c	SqlQuery:no tables specified [/home/kurt/…
33	33	2023-07-05 15:29:26	info	serviceQuery	Received SQL-Query: system.db SELECT *
34	34	2023-07-05 15:29:26	warning	serviceQuery	Error SQL-Query: SELECT *, (likely wrong sql …

Fig. 5. Excerpt of the ODARIS system log, displayed via SQLiteBrowser [17]

	id	Module	Name	Value	LogTime
	Fi…	Filter	Filter	Filter	Filter
1	1	serviceQuery	QueryErrors	3	2023-07-05 17:05:18
2	2	serviceQuery	ReceivedQuerys	28	2023-07-05 17:06:58
3	3	serviceQuery	Initialized	yes	2023-07-05 17:05:18
4	4	serviceQuery	ExecutionCycles	203	2023-07-05 17:08:41

Fig. 6. Status log of the ServiceQuery, displayed via SQLiteBrowser [17]

any system components which are critical to the system. The design choice not to be classified as critical system component, enables us to focus on functionality and reducing development cost. The main advantage is that we can use state of the art software components as for example the TFLite for machine learning in a space environment. Nevertheless, high quality of service is a very important aspect of a space applications, as space system integration and the deployment of the system in the space environment is very expensive compared to ground-based applications. Furthermore, the capability to service the application at operation time is also very restricted due to the non-existent option of having physically access. Updating the system software may also be not feasible or very limited due to the available up-link capacity. To ensure high quality of service, especially for the software components, we have established a modern state of the art software developing infrastructure, focusing on:

- automatically enforcing strong coding rules
- using modern software developments tools and guidelines
- a strict coding review process
- automatically checking for code safety violation
- extensive automatic testing of the system components on the target platforms and
- regularly preforming manual system tests

In detail the ODARIS software components are written in C++, compliant to the C++-17/20 standards [18]. We focused on writing robust and safe code, by trying to avoid unsafe and error prone programming constructs and by utilizing modern safe language features. For test and integration of the software components an automatic GitLab [19] CI-pipeline is set up, building the software for several target specific architectures. Automatic software tests are executed directly on the development hardware used for mission specific system development. Only code which has at least reviewed by one person is allowed to be merged in the code base. We enforce usage of coding guidelines by very strict compiler settings (-Werror) and the usage of the static-code analysis tool clang-tidy [20], with a large testing set activated. For detection of dynamic memory access violations, we use address sanitizers [21], which are built-in in modern compilers like the GNU compiler collection [22] (GCC) or the LLVM-CLANG Compiler [23] (CLANG). Also, we have the rule, that any operation performed in space has to be tested on ground before usage. The hardware is also tested extensively in a laboratory environment to ensure functionality and safety. At the moment we have not built-in sophisticated countermeasures addressing hardware failures. We expect that the hardware is capable to be executed in a space environment. Nevertheless, as ODARIS is a non-critical system component, we can accept sporadic system failures. As ODARIS services are stateless there should not be an issue to restart some or all of them, if a critical system error occurs. The only hardening measure we are currently planning to implement, is to ensure, that the ODARIS system is capable of handling corrupt configuration and database files, and that abortions of services are detected and an automatically restart of the service is performed. For being able to inspect the health status of the ODARIS system, the services provide status information and logging information via databases. Status and logging information can therefore be accessed via the communication system when required.

VI. Concept Verification

We are currently in preparation of two satellite missions with the goal to demonstrate the ODARIS concept. The overarching goal of the experiments is to verify, that the proposed methodology is in general a feasible approach for implementing satellite based real-time information systems on ”low earth orbit” space missions. The experimental results shall provide sufficient information to evaluate the potential of using the proposed real-time information concept within future regular satellite mission operations. To achieve these scientific goals, first, it shall be shown, that the proposed methodology can be implemented for a real-world space mission. Second, to evaluate the feasibility for operational usage of the method, performance criteria of different aspects of the system shall be derived in a real-world experiment scenario. Third, weaknesses of the system and development potential of the methodology shall be identified. There are three major aspects which shall be evaluated within the ODARIS experiments, the performance of the real-time communication system, the usability of the information system in a real space-mission environment and the feasibility to perform data analysis on board a satellite. The start of both missions is planned around 2025 and they are both based on different on board computing architectures. For one mission we are part of the Scalable On-board Computing for Space Avionics (ScOSA) project team, which has the goal to demonstrate the concept of a next generation high performance distributed heterogeneous computing system, utilizing a combination of high performance COTS processing nodes for computationally intensive tasks and reliable nodes based on traditional certified space hardware for system management and supervision tasks [24]. For the mission the ODARIS system has to be integrated in a software system supporting the operation of the distributed heterogeneous computing system

[24]. The second experiment in preparation is planned to be conducted within the Seamless Radio Access Networks for Internet of Space (SERANIS) satellite mission from the University of the Bundeswehr Munich [25]. It is planned to integrate ODARIS as a standalone application within a Docker container on a rather powerful GPU accelerated Linux based payload processing unit.

VII. Conclusion and next steps

With this article we gave a comprehensive overview of the background, the goals, the system architecture and the concept verification approach of the ODARIS on board data analysis and real time information system. The next development steps are as mentioned in the last section preparing and performing two concept demonstration satellite experiments. If the feasibility of the concept can be demonstrated, the next step will be the deployment of the system on a variety of space missions to increase the quality of service and simplify the operation. Additionally, the information system shall serve as a host platform for all kinds of applications, enabling developers to bring their application in a space environment without deep understanding of space system software development.

References

[1] Iridium Communication Inc. Iridium-llc home page, 2023. https://www.iridium.com.

[2] Hank D Voss, Jeff F Dailey, Matthew B Orvis, and Matthew C Voss. Thin cubesats and compact sensors for constellations in vleo to deep space. In *Small Sat Conference*, 2022.

[3] NearSpace Launch. Eyestar-s4 product page, 2023. https://www.nearspacelaunch.com/eye-star.

[4] Murray Kerr, Stefania Tonetti, Stefania Cornara, Juan-Ignacio Bravo, Robert Hinz, Antonio Latorre, Francisco Membibre, Alexis Ramos, Alvaro Moron, Chiara Solimini, Stefan Wiehle, Helko Breit, Dominik Günzel, Srikantha Mandapati, Björn Tings, Ulrich Balss, Otto Koudelka, Franz Teschl, Enrico Magli, Tiziano Bianchi, Andrea Migliorati, Paolo Motto Ros, Michele Caon, Maurizio Martina, Riccardo Freddi, Fabio Milani, Guido Curci, Silvia Fraile, and Cecilia Marcos. A novel architecture for the next generation of earth observation satellites supporting rapid civil alert. In *The 35th Annual Small Satellite Conference - SSC*, pages 1–7, August 2021. to cite the paper from the USU archive: Kerr, M. et al. 2021. "A Novel Architecture for the Next Generation of Earth Observation Satellites Supporting Rapid Civil Alerts," Proceedings of the Conference on Small Satellites, Technical Session 8: Advanced Technologies II, SSC21-VIII-07.

[5] Google Inc. Tensorflow project homepage, 2023. https://www.tensorflow.org.

[6] Georges Labrèche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, and Vladimir Zelenevskiy. *Agile Development and Rapid Prototyping in a Flying Mission with Open Source Software Reuse On-Board the OPS-SAT Spacecraft*. DOI: 10.2514/6.2022-0648.

[7] Georges Labrèche, David Evans, Dominik Marszk, Tom Mladenov, Vasundhara Shiradhonkar, Tanguy Soto, and Vladimir Zelenevskiy. Ops-sat spacecraft autonomy with tensorflow lite, unsupervised learning, and online machine learning. In *2022 IEEE Aerospace Conference (AERO)*, pages 1–17, 2022. DOI: 10.1109/AERO53065.2022.9843402.

[8] Katharina Willburger, Kurt Schwenk, and Jörg Brauchle. Amaro - an on-board ship detection and real-time information system. *Sensors*, 20(5):1324, Februar 2020.

[9] Kurt Schwenk and Daniel Herschmann. On-board data analysis and realtime information system - status & outlook. In *Deutscher Luft- und Raumfahrtkongress 2022 (DLRK 2022)*, September 2022.

[10] Kurt Schwenk and Daniel Herschmann. On-board data analysis and real-time information system. In *Deutscher Luft- und Raumfahrtkongress 2020*, Oktober 2020.

[11] Raspberry Pi Foundation. Raspberry pi 1 model b+, 2023. https:/www.raspberrypi.com/products/raspberry-pi-1-model-b-plus.

[12] Advanced Micro Devices, Inc. Zynq-7000 soc product description, March 2023. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[13] NearSpace Launch. Nearspace launch company website, 2023. https://www.nearspacelaunch.com.

[14] Avnet Inc. Product page avnet microzed board, 2023. https://www.avnet.com/wps/portal/us/products/avnet-boards/avnet-board-families/microzed.

[15] SQLite Project. Sqlite project homepage, 2023. https://sqlite.org/.

[16] Yocto Project. Yocto project website, 2023. https://www.yoctoproject.org/.

[17] SQLiteBrowser Community. The official home of the db browser for sqlite, 2023. https://sqlitebrowser.org.

[18] Standard C++ Foundation. Standard c++ foundation, 2023. https://isocpp.org.

[19] GitLab B.V. Git lab documentation, 2023. https://docs.gitlab.com.

[20] Clang Project. Clang-tidy documentation, 2023. https:clang.llvm.org/extra/clang-tidy.

[21] Konstantin Serebryany, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov. Addresssanitizer: A fast address sanity checker. In *USENIX ATC 2012*, 2012.

[22] GCC Team. Gcc project homepage, 2023. https://gcc.gnu.org.

[23] Clang Project. Clang project homepage, 2023. https://clang.llvm.org.

[24] Andreas Lund, Zain Alabedin Haj Hammadeh, Patrick Kenny, Vishav Vishav, Andrii Kovalov, Hannes Watolla, Andreas Gerndt, and Daniel Lüdtke. Scosa system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture. *CEAS Space Journal*, May 2021.

[25] Emir Gadzo, Francesco Porcelli, Roger Förstner, and Kristin Paetzold. Requirements engineering process for seranis, a small satellite mission. 09 2022. DOI: 10.25967/570212.