Software Evolution from TET-1 to Eu:CROPIS

Olaf Maibaum (1), Ansgar Heidecker (2)

(1) German Aerospace Center (DLR), Simulation and Software Technology, Lilienthalplatz 7, 38108 Braunschweig, Germany

(2) German Aerospace Center (DLR), Institute of Space Systems, Robert Hooke Str. 7, 28359 Bremen, Germany

ABSTRACT

The base of the Eu:CROPIS (Euglena Combined Regenerative Organic food Production In Space) Attitude and Orbit Control System (AOCS) is the three layer AOCS software architecture of the TET-1 satellite (Technology demonstrator). Because of different AOCS requirements between TET-1 and Eu:CROPIS, a software reuse is only possible for software components in the interface layer. In the other two architecture layers, the software components have to be replaced by new implementations to fulfil the changed requirements of the Eu:CROPIS mission. In contrast to the former software evolution from BIRD (Bispectral Infra-Red Detection) to the TET-1 AOCS, the software evolution is forced in Eu:CROPIS by the reuse of software design principals applied in TET-1. This enables the change of the underlying scheduling mechanisms from a fixed-time approach to a more reactive software behavior presented in this paper.

1. INTRODUCTION

The reactive scheduling mechanism used in the Eu:CROPIS AOCS is a result from experiences made in the BIRD mission [7]. This mechanism, named “Tasking Framework”, resolves one weakness in the BIRD and TET-1 AOCS software [8]: the scant timing for the control torque computation. The Tasking Framework is the core element in the runtime system development of DLR’s OBC-NG (Onboard Computer – Next Generation) project [5], which will provide a distributed onboard computer platform for reconfigurable and high redundant systems. Currently the Tasking Framework is used for several developments like DLR’s ATON (Autonomous Terrain-based Optical Navigation) project, MAIUS (Atom-optical experiments on sounding rockets) mission, and Eu:CROPIS.

The next chapter outlines the Eu:CROPIS mission, the used satellite bus, and the AOCS. Chapter 3 presents the Tasking Framework and implementation details for the AOCS. The focus is set on the usage of the reactive behavior for the implementation of the diagnostic report service defined by the packet utilization standard (PUS) [6], which is one of the main benefits of the Tasking Framework. The paper closes with a conclusion.

2. EU:CROPIS

The mission Eu:CROPIS is the demonstration of the feasibility of restartable and sustainable life support systems on a pure biological basis. Such systems enable the production of food and atmosphere, and the utilization of waste like urine and phosphate [1]. Furthermore, the system should be reliable enough for long duration missions. The biological experiments require different levels of gravity. This is achieved by a spin stabilized satellite which can change its rotation speed and thereby the centrifugal force (gravity) in the biological experiment compartments during mission time. The target gravities of Eu:CROPIS are 0.16 g (Moon) and 0.38 g (Mars) respectively in the biological experiment compartments. They are placed at a reference radius of 0.35 m measured from the designed spin axis. The launch is planned in 2017 with a Falcon 9 as a piggy back start.

The used satellite bus is based on the DLR compact satellite bus program, which is a research and development platform in a component-oriented design. The bus for the Eu:CROPIS mission is a spin stabilized platform with a cylindrical body with a diameter of 1000 mm and a height of 1100 mm [3]. The bus has two sections: one with the separated payload compartments at the upper deck, and one with the bus section. Figure 1 shows both sections. The mass of the satellite is around 230 kg. The orbit is sun synchronous with at least 500 km altitude.

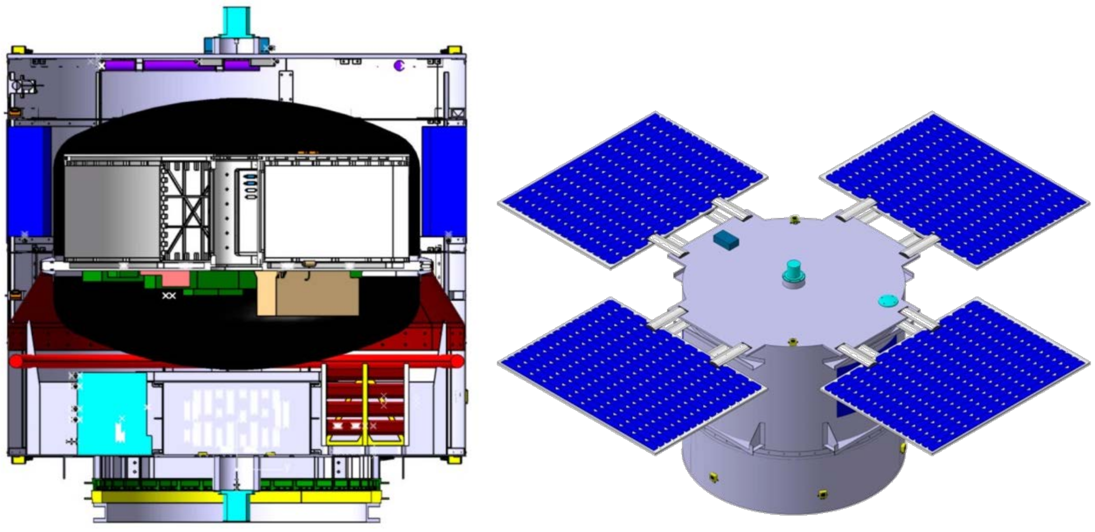

Figure 1 Satellite Bus Structure

2.1 AOCS

The main requirement to be fulfilled by the AOCS is the generation of gravity in the biological compartments. Beside this, the AOCS is responsible to orient the z-axis into sun direction to ensure power generation by the solar panels. Thereby, the satellite bus is spinning around the z-axis between 5 to 31 rpm. To satisfy the power generation with the solar panels the spin axis has to be reoriented by ~1 deg/day to keep a sun pointing attitude.

As actuators, the AOCS uses three magnetic torquers to control the rotation and spin axis. It uses two magnetometers, 10 sun sensors, and liquids inside the experiment compartments as damping device. Table 1 show the key figures of the AOCS hardware components.

The attitude controller uses five control state modes. The detumbling mode damps the spin rates of the satellite body sufficiently. It is the first mode when the AOCS boots up. The second mode is the spin up/down mode to change the spin rate around the z-axis. The spin mode holds a spin rate and orients the solar panels into the sun. For the solar panel deployment, the AOCS provides a deployment mode to handle the moments of inertia changes. To indicate a problem in the AOCS, the fifth mode is the AOCS safe mode that keeps the solar panels into sun direction.

The attitude controller uses an Unscented Kalman Filter (UKF) approach as core of the attitude determination which is designed for spin stabilized satellite. For a detailed description of the used UKF and the AOCS see [4].

Amount	Hardware	Key figure
3	Magnetic Torquer	30 Am^2
2	Magnetometer	noise < 1000 nT (3σ)
10	Sun Sensor	noise < 0.3 deg (3σ)
4	Angular Rate Gyroscope	RW 0.08 deg/$\sqrt{h}$, Bias 9 deg/h (3σ)
2	GPS Receiver	DLR-Phoenix

Table 1 Attitude control system hardware devices

3. TASKING FRAMEWORK

The Eu:CROPIS AOCS uses a reactive scheduling mechanism to control the order and timing of computation tasks, named as Tasking Framework. Starting point for the implementation is the way how estimator and predictor modules were organized in the BIRD AOCS. These modules are sensor data filtering and fusion algorithms to transform the raw sensor measurements to an accurate attitude state vector.

In BIRD and TET-1 the estimator and predictor modules are computed in a fixed-order at a fixed-time in the control cycle. The timing is a combination of the sensor latency and an additional gap time to satisfy the availability of data for the computation. For TET-1, this static approach with overestimated safe gap times has been led to a scant timing between controller computation and the commanding of the control torque to the actuators. During the launch and early orbit phase (LEOP), a timing violation in another bus application occurred which result to an unexpected AOCS state caused by another ordering of independent computations. Figure 2 A) depicts an example of such a timing.

In the new Tasking Framework, the timing behavior has been changed. Instead starting the computation at a predefined time in the computation cycle, it starts now whenever the information is available. All information values are stored in messages distributed by channels. The channels initiate a computation when all defined conditions for the computation are met. The timing with the Tasking Framework is depicted in Figure 2 B). In addition, the computed values of the estimator and predictor modules are not stored in one AOCS state vector anymore but handled as messages on channels inside the AOCS software. These channels provide the synchronization mechanism for the data. Channels can be implemented as single or double buffers, or as more complex data structures like FIFO queues. The scheduler satisfies that a computation task always sees synchronized data and that all computations are performed in the right order.

In the Tasking Framework, all computations are performed by tasks instead of threads. A task can subscribe to channels needed to provide the information for the computation. Each task is started when a specified amount of information is available on all subscribed channels. It can also start immediately when a corresponding channel is subscribed as final input. These specifications are configured by task inputs, which establish the subscription between channel and task. In contrast, a thread in TET-1 is only started once during the run time of the onboard software and should be implemented as an endless computation loop for the specified time frame.

For computations requiring predefined timings in the computation cycle, the Tasking Framework provides a task event. A task event triggers on three different ways, periodically, relative to a trigger from a task, or relative to the computation cycle of a group of tasks. A task can subscribed to a task event instead of a channel in order to operate with a defined timing. A time-out is reached when a task event is subscribed as optional and final input.

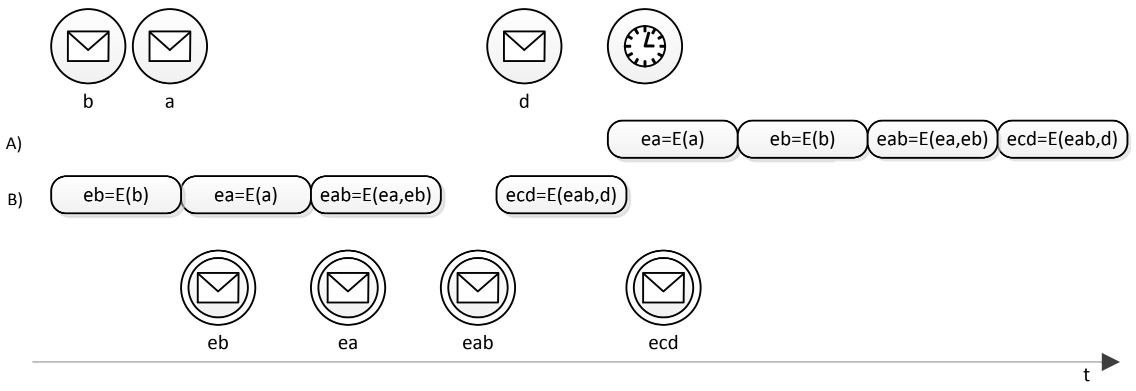

Figure 2 Timing: A) Procedural B) Tasking

One computation can split into several tasks but managed as a group. This allows the concurrent execution of a computation on multi-core processor platforms. The behavior of tasks inside such a task group is slightly different to the behavior of an isolated task. By default, the scheduler performs a reset operation on a task and all its inputs when it finishes the computation. The reset operation on a task input set the number of arrived data items on the associated task channel back to zero. Each data item on the associated task channel increase the number of arrivals at the associated inputs and becomes activated when the specified number of arrivals is reached. The input associated to a task is started again when all its inputs are activated or when the activated input is marked as final. For a task group, this reset operation is only executed when all tasks in the task group have been executed. Thus, a task inside a task group can only start again when all other tasks inside the group have been executed at least once in the previous execution. An example for such a group is the magnetic torquer interface introduced in section 4.1.

4. IMPLEMENTATION AND USAGE

The Tasking Framework is split in two parts, the API with all system independent parts of the framework and the bare metal model with the run time environment specific parts. The API consists of a set of C++ classes specify the interfaces of the elements in the Tasking Framework. The functionalities of application software like the AOCS shall be implemented for this interface specified by the API. The principal to implement the behavior is similar to the BOSS operating system used in BIRD and TET-1.

Currently several implementations exist as bare metal model for the Tasking Framework. The first implementation is a bare metal model for Linux run time systems. This implementation is used in the DLR's ATON project and for the MAIUS mission. For OBC-NG a bare metal model for the RODOS operating system and the Linux porting is used for experiments to address inhomogeneous distributed run time environments. These versions have also support for multi core computing platforms. For Eu:CROPIS a bare metal model for the run time platform of the DLR compact satellite series [2] is used, called libCOBC. Figure 3 shows the layered software architecture used for the Eu:CROPIS AOCS.

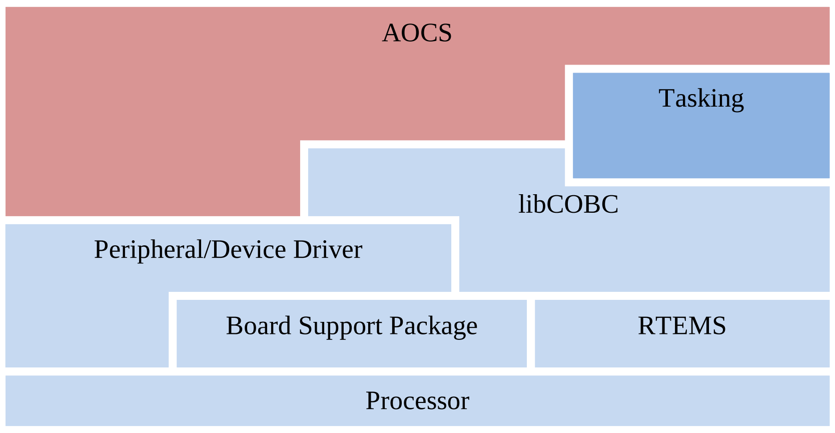

Figure 3 The layered software architecture of the AOCS

4.1 Tasks in Eu:CROPIS

The tasks of the Eu:CROPIS AOCS are divided in three groups. The first group encloses tasks with a strict timing. All of these tasks are triggered by a periodic clock and perform input and output operations. The second group of tasks receives data from the sensor and actuator interfaces and performs first checks on the incoming data and converts it from the message format into the representation of physical dimensions. The last group consists of computation tasks like controller modules or the state machine.

For example, tasks in the first and second group are necessary for the interface to the magnetic torquer. This interface uses a master/slave protocol for the communication with the magnetic torque electronics. To implement the protocol three tasks are necessary to drive the communication with the magnetic torque electronics. The tasks are part of a periodic group. Figure 4 depicts the software architecture of the magnetic torque interface with all channels, input configurations and tasks.

The first step in the processing of the magnetic torque communication protocol is the commanding of the control torque to the magnetic torquer electronics. This step is done by the command task which consumes the desired torque from the control torque channel and the interface state detected in the last cycle from the MTC (Magnetic Torque Control) state channel. The processing is initiated by the periodic MTC timer event and ends with pushing the command state to the command state channel. The MTC timer event is subscribed as final input to satisfy the execution even no data is send on the control torque channel.

The second step is to receive a command acknowledge and request telemetry data from the electronics. This step is performed by the request task and initiated by an incoming answer on the communication line from the magnetic torquer electronics. If no answer is received the time out event will start the task. This event can be a periodic timer or is triggered with a time offset by the command task. The request task pushes the current communication state to the request state channel which is consumed by the next processing step.

The last step is to analyze the health state of the magnetic torque electronics with the received telemetry data. This step is started by the TM data message on the MTC message in channel or by a time out if no message is received. Because the MTC Message in channel is also used as input to the request task, the input is configured with two incoming messages, the command acknowledge message and the requested TM message. The current state of the interface is pushed to the MTC state channel to distribute this information to all tasks which need the knowledge of the state. All three steps run as a group. By thus the termination of the telemetry tasks resets all inputs to zero arrivals on the corresponding channels and the tasks can be activated again.

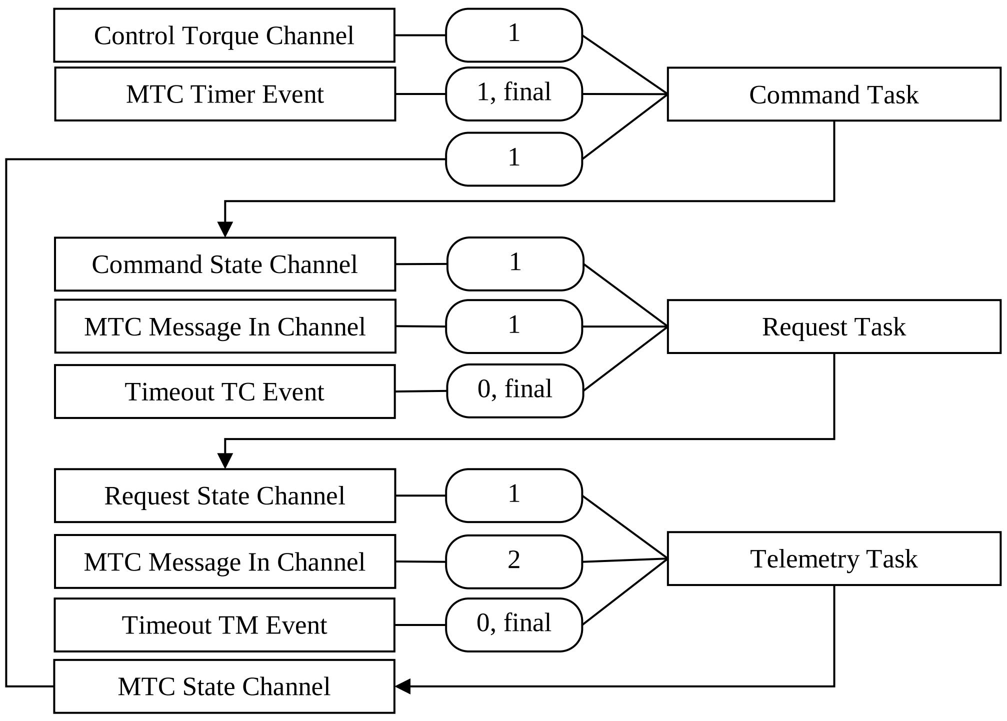

Figure 4 Channels, input configurations, and tasks of magnetic torquer interface

4.2 Data Synchronization

Data synchronization is only done by the different implementations of the task channels. The API of the Tasking Framework provides three virtual methods as entry points for the synchronization. The scheduler calls them before a task is executed, after a task is executed and when a task and the associated inputs are reset. As an example, in a double buffer a data switch during the reset is enough. This will satisfy that all tasks in a group access to the same data in the double buffer. When the double buffer is used as input by several ungrouped tasks a switch of the double buffer is only useful when all of them are executed. So the synchronization performs after the last of the receiving tasks is executed. In contrary a FIFO with several readers needs an adjustment of the reader to the current FIFO state just before a reading task is started.

4.3 Diagnostic Reporting Service

The diagnostic reporting service defined by the Packet Utilization Standard (PUS) allows the request of extended telemetry data from the subsystems of a spacecraft. For the Eu:CROPIS AOCS all task channels are addressable in a diagnostic report. This allows a deep inspection of the current software state during the operational phase. Beside single or periodic reports PUS allows the definition of filters for the reporting. For example it is possible with the current implementation of the diagnostic service of the Eu:CROPIS AOCS to define a filter on the current state of the magnetic torque software interface. In case of a switch of the health state of the magnetic torques the diagnostic reporting service generate a diagnostic report. This report encloses all task channels in the software interface except the timers for a deeper inspection of the switch in the health state.

From an implementation point of view the diagnostic reporting service is defined as periodic task. For single or periodic requests this task observes all enabled diagnostic reports. A diagnostic report is send when the conditions for the report generation fit. For filterable task channels a scalar filter value exists which is calculated from the last message data on the channel. These filter values could be e.g. a rotation speed, a binary value, or the angle. If a filter is defined for a diagnostic report, the periodical task applies the selected filter check on the filter values.

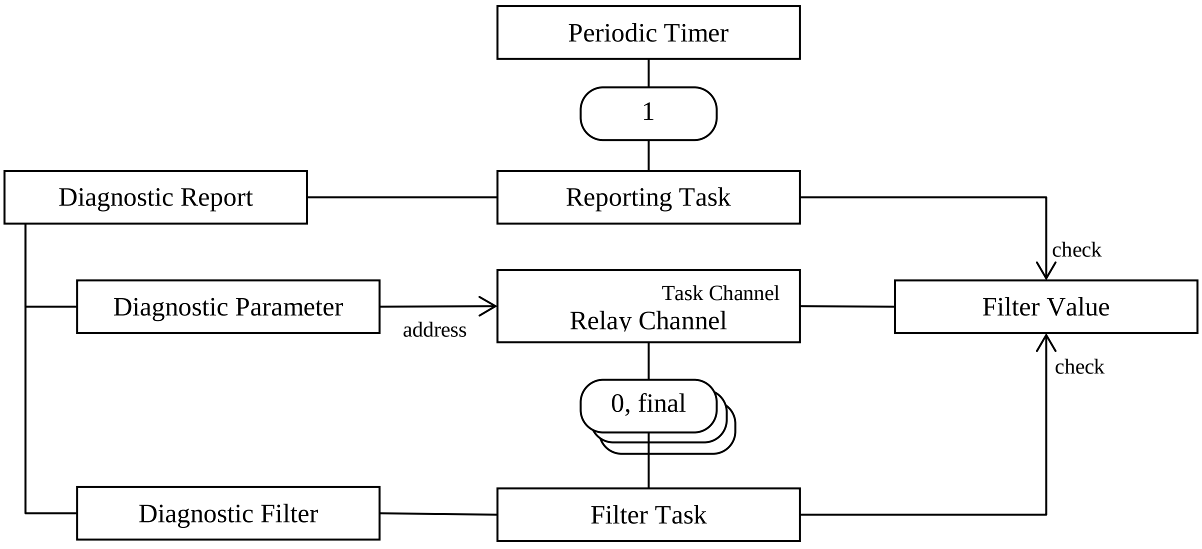

Figure 5 Software architecture diagnostic reporting service

PUS defines as filters only absolute or percentage difference checks. Beside these, the implementation of the Eu:CROPIS AOCS extends the diagnostic reporting service by upper, lower and epsilon environment checks. These three checks can executed immediately when data is distributed on a task channel. For this purpose an additional filter task exists in the diagnostic reporting service which is associated with the task channel to filter. New data on one of the observed channels starts the filter task immediately to check the filter condition on this channel and sends a report when the filter matches the condition. This enables the observation of all data exchanged between AOCS tasks as extended telemetry. Figure 5 shows the software architecture of the diagnostic reporting service.

5. CONCLUSIONS AND OUTLOOK

This paper presents the reactive scheduling mechanism used in the Eu:CROPIS AOCS. This kind of scheduling removes the overestimated time gaps between processing steps in the BIRD and TET-1 software. Data inconsistencies by the violation of the timing are no longer possible. The ordering of computations is defined by the presence of data and computations are performed as soon as possible. With usage of the Tasking Framework only the lack of computing power or implementation failures in the control algorithms can violate the correct functionality of software.

The described filter concept used by the implementation of the diagnostic reporting service shows how observations can be integrated into a computation process. Besides the normal operation of the AOCS, such injections can also be used for FDIR (Failure detection and recovery) and surveillance checks or more sophisticated checks during the test phase of the software. Furthermore, the split of processing chains in several computation tasks allows the parallelization of onboard computing which yield more computation power. This is addressed in DLR's OBC-NG project. The Tasking Framework used for the Eu:CROPIS AOCS demonstrates already today the potential power of distributed and reconfigurable onboard systems expected to become available in the next years also for the space domain.

6. KEYWORDS

AOCS, software, reactive system, scheduling

7. REFERENCES

[1] G. Bornemann, K. Waßer, R. Hemmersbach, R. Gerzer, R. Anken, J. Hauslage. C.R.O.P.-Combinded Regenerative Organic Food Production: waste and wastewater processing by trickling filters. In: EHBLSS 2013 Proceeding & Abstracts Book. Beijing, China (2013).

[2] F. Dannemann, F. Greif, Software Platform of the DLR Compact Satellite Series. In: Proceedings of the 4S Symposium, Mallorca, Spain (2014).

[3] O. Mierheim, T. Glaser, C. Hühne, H. Müller, E. Kheiri, Modal Frequency Adjustment of the Eu:CROPIS Satellite Structure. In: Proceedings of the 13th European Conference on Space Structure, Material & Environmental Testing. Braunschweig, Germany (2014).

[4] A. Heidecker, T. Kato, O. Maibaum, M. Hölzel, Attitude Control System of the Eu:CROPIS Mission. IAC 2014, Toronto, Canada (2014).

[5] D. Lüdtke, K. Westerdorff, et. al. In: Proceedings IEEE Aerospace Conference 2014: OBC-NG: Towards a Reconfigurable On-board Computing Architecture for Spacecraft. Big Sky, USA (2014).

[6] ECSS-E-70-41A: Packet Utilization Standard. (2003)

[7] K. Brieß, W. Bärwald, T. Gerlich, H. Jahn,F. Lura, H. Studemund. The DLR Small Satellite Mission BIRD. In: Digest of the 2nd International Symposium of the International Academy of Astronautics. pp 45-48. Berlin, Germany (1999)

[8] O. Maibaum, T. Terzibaschian, C. Raschke, and A. Gerndt. Software Reuse of the BIRD ACS for the TET Satellite Bus. In: Digest of the 8th International Symposium of the International Academy of Astronautics. pp. 409-412. Wissenschaft und Technik Verlag, Berlin (2011)