An example from space: new development for high dependability for space board computing.

Sergio Montenegro[(1)], Lutz Dittrich[(2)]

[(1)]DLR, German Aerospace Center, Am Fallturm 1, D-28359 Bremen, Germany, Email:Sergio.Montenegro@dlr.de

[(2)]DLR, German Aerospace Center, Am Fallturm 1, D-28359 Bremen, Germany, Email:Lutz.Dittrich@dlr.de

1. Introduction

The Standard Satellite Bus's (**SSB**) core avionics system is a further step in the development line of the software and hardware architecture which was first used in the bispectral infrared detector mission (BIRD). The next step improves dependability, flexibility and simplicity of the whole core avionics system. Important aspects of this concept were already implemented, simulated and tested in other ESA and industrial projects. Therefore we can say the basic concept is proven. This paper deals with different aspects of core avionics development and proposes an extension to the existing core avionics system of BIRD to meet current and future requirements regarding flexibility, availability, and reliability of small satellites and the continuously increasing demand of mass memory and computational power.

The term "core avionics system" as used by the authors of this paper is nearly synonym to the term "command and data handling system" (C&DH). In particular we mean a special system tailored to the needs of a small satellite with a computer as an integral part of this system. This on-board computer is also the execution platform for attitude control algorithms as well as for navigation an on-board intelligence.

The main risk factors in a typical core avionics development are the complexity, software-hardware interfaces and the difficulties to handle many different interfaces in a single system. The new core avionics concept targets these problems and aims to provide a very simple integrated solution of software and hardware. The border between both shall vanish. This concept can handle both; bus control and payload control in one system.

The emerging and fast growing FPGA technology allows us to implement the biggest part of the core avionics in software, including classical CPU software and FPGA software. The use of fixed hardware is kept to a minimal limit.

In this concept the core avionics functionality is provided by a network of services. Some of them are implemented in classical CPU-software, some in FPGA-software and some in hardware devices, for example sensors and actuators. To access any service there shall be no difference in how it was implemented (CPU, FPGA, hardware) and where it runs. The core avionics system is a distributed computer system. No single node is required to be dependable. The computers are connected by a dependable hardware network which is the heart of the system. Software services can be distributed on all computers and may migrate from one to another for example in case of failures, overloading or for power management purposes.

In the same way like the hardware network, there is a software network, which interconnects all services, including software tasks running on the same computer, on different computers, FPGA programs and even hardware devices. This global software interconnection network is called the Middleware. The middleware is implemented in both: CPU-software and FPGA-software. Both implementations use the same communication protocol. This allows us to have only one interface type in the whole system: The Middleware Interface.

The most effective and save way to implement a complex parallel system is to compose it as a network of simple sequential co-operating tasks which communicate by offering and consuming services. For the implementation there is no difference where the tasks will be deployed/execute and whether they run on a CPU or in a FPGA or which services are provided and/or consumed by hardware devices. The location of tasks can even change at run time, without requiring any explicit reaction of the other involved tasks. Tasks communicate with each other by using the middleware. There is only one interface to the middleware, which is encapsulated in messages. The residual is transparent to the user.

2. Requirements for Space Applications

The data management system on board is similar to many other embedded systems on ground; but is space we have very strict constraints and difficulties:

Special requirements for space applications:

Reiability: This property is extremely important for the data management system and is the main cause for its high expenses and costs in comparison to terrestrial embedded systems. Three factors belong to reliability: Robustness, Self-healing (self-regeneration) and fault tolerance.

Self-healing self-regeneration): The spacecraft must be able to handle and to treat failures and anomalies by reconfiguration using redundant resources (reserves and spares). This is especially importantly for missions with long lifetime, e.g., 15 years. In these cases, just Self-healing using cheap components in not enough. Here we need rather very robust components. This originates a Trade-off: How much to invest in **robustness** o avoid failures, and how much to invest in **redundancy** o compensate for failures.

Fault tolerance: Even if components are not permanently damaged, malfunctions are to be expected at any time. The system must be able to recognize such anomalies, to compensate and to correct them, best before they have wider consequences. On Earth we find similar requirements (and even more strict) in the case of **safety-critical ystems**, as for example railway, airplane or nuclear reactor control. Nevertheless, n the ground we are protected from approx. 50 km of air very well against the cosmic radiation, the spacecraft, however, not. This radiation causes a huge amount of **data corruption** (**bit flips**) and leads to quicker**ageing** of the electronic components. Exactly as with terrestrial safety-critical systems the data corruption must be treated, however, their frequency is higher in space by around the factor of 10 up to 1000; 1 to 10 **bit-flippers** per tag can be expected to occur in the board computer memory (e.g. 16 Mbytes).

Very limited space, energy (power) and dimension resources: As with hand-held-Devices (cellular phone, portable navigation systems, mp3-players, Organizer, cameras) the

system has to go very economically with these resources. Solutions for such earth-products can be interesting also for space missions… if they are able to survive space conditions.

Cold / Heat: Although the spacecraft is exposed to extreme temperatures, e.g., from-170° to +120° Celsius, the temperature range of the data management segment is relatively mild. The data management segment is mostly well protected in the middle of spacecraft. The temperature mostly stays between +10° and +40° Celsius, which produces no difficulties for the electronics. With some terrestrial applications the requirements are more extreme, e.g., the automotive electronics must be able to work from -40 ° (winter nights) to +80 ° (direct sun in summer).

Vacuum: For electronics on ground, we can use air (even ventilators) to conduct head away from the heat-source. In space, in vacuum it is not so simple to conduct the heat away. This can lead to a heat traffic jam and lead to a very irregular temperature distribution on the electronic components. Metal conductors must be used to transport heat from the boards to the main spacecraft structure.

No gravity: This should be no problem for the electronics and the software, but, a loose small metal part (e.g., a cuted wire) could freely move and cause short circuits at different places. We have similar problems in terrestrial applications by strong vibrations, e.g., in vehicles.

Vibrations: In free fly in orbit vibrations are hardly to be expected, but during the launch on the rocket for about 10 to 15 minutes, we can expect extreme vibrations up to 7-fold earth acceleration (g). At the separation we can expect a shock for few microseconds with up to 1000g. This requires especially robust hardware.

Software complexity: As in all other IT areas, the software complexity increases too fast. It has reached dimensions which are hardly controllable. With today's complexity one must use proven software engineering methodology and a strict quality assurance program. These expenses for the software quality assurance are often underestimated; it is between 50% and 80% of the complete software development const.

Software-Uploads: Almost always the software efforts are underestimated. Hence, some missions start, before the software is ready. This is not a good concept and rescues many dangers in itself. Nevertheless, it is important to create the possibility to be able to reload new software or software updates. This functionality is also usual with mobile consumer electronics and navigation systems. Software-Uploads are necessary because it is not possible to foresee all situations for the spacecraft. In operation-time improvements and errors can be recognized and new software must installed on the spacecraft which is already in space. Another reason is that in the course of the mission, components are permanently damaged or change their working properties. It would be conceivable to have software which foresees all these situations. Indeed, the complexity of such a system would be too big. It is usual; to wait until a failure or unexpected situation is discovered to update the software.

Remote diagnosis: It is not always possible that the software can recognize and identify all possible anomalies. Hence, a remote diagnosis will be required, so that the developers can diagnose the system from the ground. The software should be able to collect state information of all subsystems and to transmit them to the ground. This functionality is also used for big machines (newspaper-printers, excavators, mining machinery, packing machines etc.). These producer companies seat worldwide their machines, but their maintenance engineers can not be everywhere. They have to diagnose errors from their remote offices.

High computing performance: usually the computing performance in spacecrafts is around the factor from 10 to 100 slower than comparable devices on the ground. The demanded high reliability and limited resources excludes high computing performance. Today space applications are becoming more and more hungry for computing performance. New ways must be searched to combine high computing performance with high reliability and with limited resources (not only for space applications).

These requirements are not new and they are also not so different to the ones found on terrestrial applications. However, the problem with the data processing in spacecrafts is that one must fulfill all these requirements at the same time.

3. Tasks of the core avionics system

The whole control of the satellite is performed by the core avionics system. Typically this big job is divided into the following subtasks:

Command handling

The command handling subsystem receives, validates, decodes, and distributes the received commands from the ground station [5]. It is responsible for generating acknowledgements, counting received and executed messages. It shall provide means for executing time-tagged commands and pre-defined command lists.

Data handling

The data handling subsystem's job is it to collect telemetry data and status information from different devices on a regular basis as well as on demand. It has to store this gathered data and to convert this data in a format compatible with the communication system, e.g. CCSDS and PUS [6][7].

Time management

For time-tagged command execution and time-tagging measurements a precise time source is necessary. High precision oscillators are expensive. An alternative is to use a lower precision oscillator and to synchronise with external time sources. One possible solution is to use a GPS receiver to derive the time from the transmitted GPS signals. Another way is to use the communication system to synchronise with the ground station.

Health monitoring

Health monitoring is the surveillance of all systems and subsystems. This is typically done by consistency checks and watchdog timers. Another important aspect of health monitoring is memory scrubbing to prevent the accumulation of bit-flips induced by single-event effects.

Attitude control

The attitude control system determines the angular position of the spacecraft. To do this it controls reaction wheels, magnetic torquers or thrusters. Reaction wheels can be accelerated or slowed down in order to rotate the spacecraft. The execution platform for these algorithms has to be provided by the core avionics system.

Onboard navigation

If the onboard navigation system computes the position and orbit of the spacecraft, it needs software and an execution platform. This platform has to be provided by the core avionics system.

Power management

The power management system of the spacecraft also needs control algorithms for charge control of the batteries and to shunt the solar cells to prevent overloading.

Thermal management

A thermal management system containing active elements like heaters or coolers needs control. So, if this control is performed by software to increase flexibility, decrease mass, volume and costs, it also needs an execution environment which has to be provided by the core avionics system.

4. State of the art

Figure 1 shows as example the board computer structure used in BIRD.

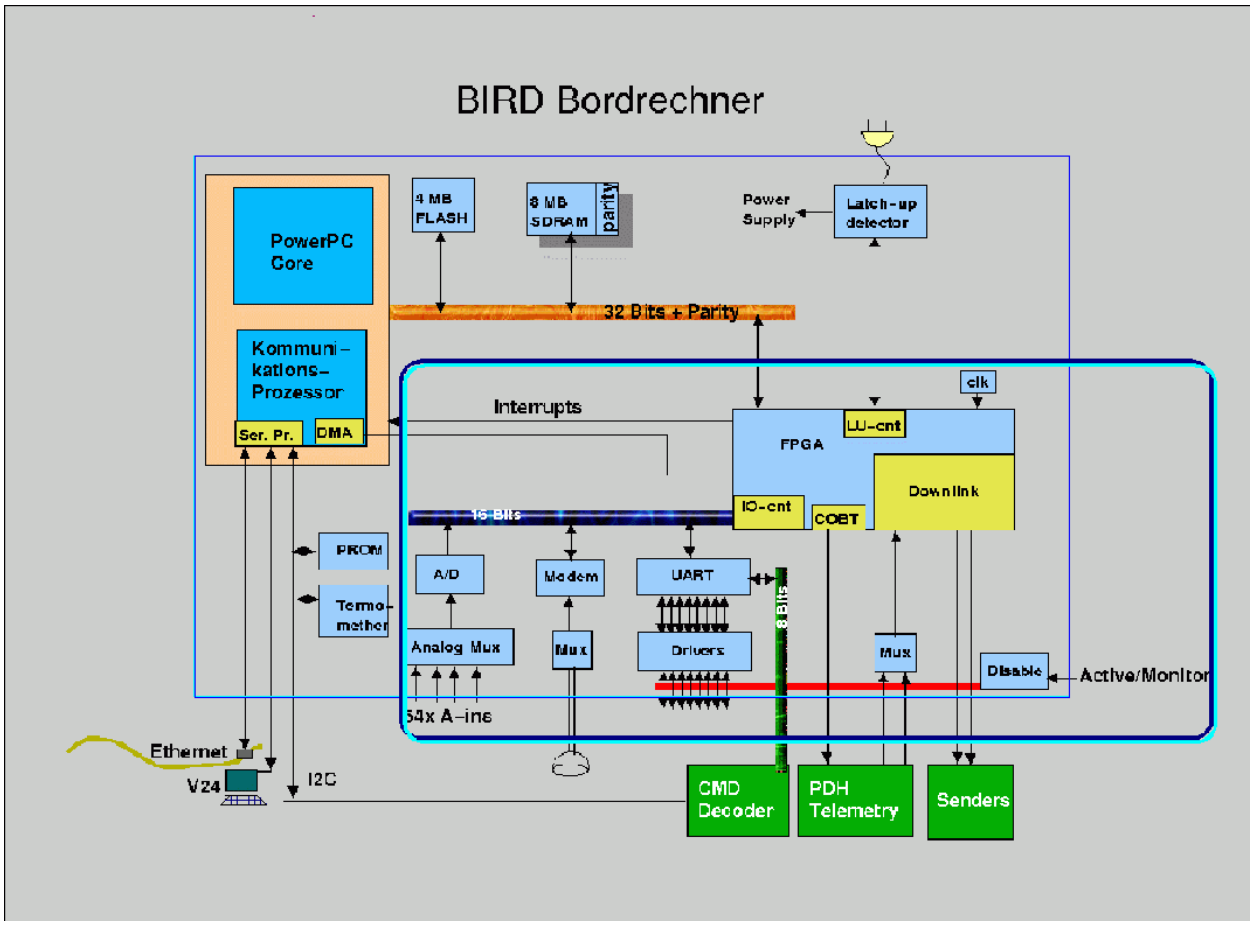

Figure 1: BIRD board computer

This structure can be represented in a more generic way as shown in figures 2 and 3. Usually the protocol conversion from the CPU-interface to the different device interfaces is performed by an UART (as IP).

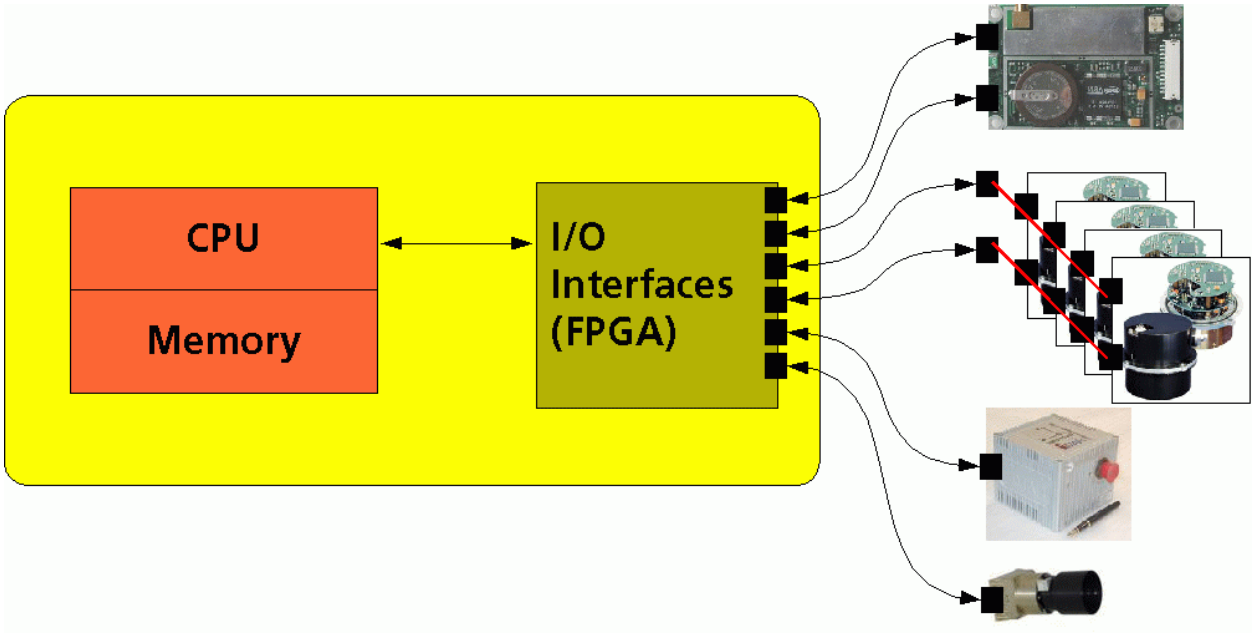

Figure 2: typical board computer

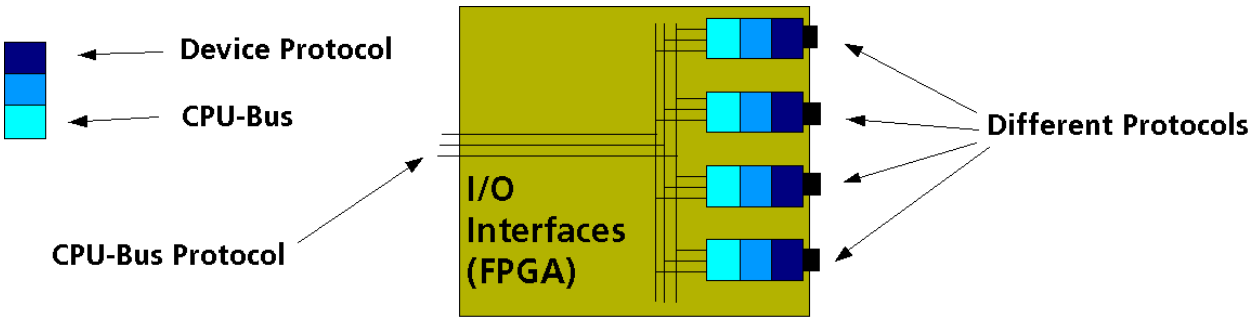

Figure 3: typical I/O interface

Figure 4 shows the typical software structure (e.g. from BIRD).

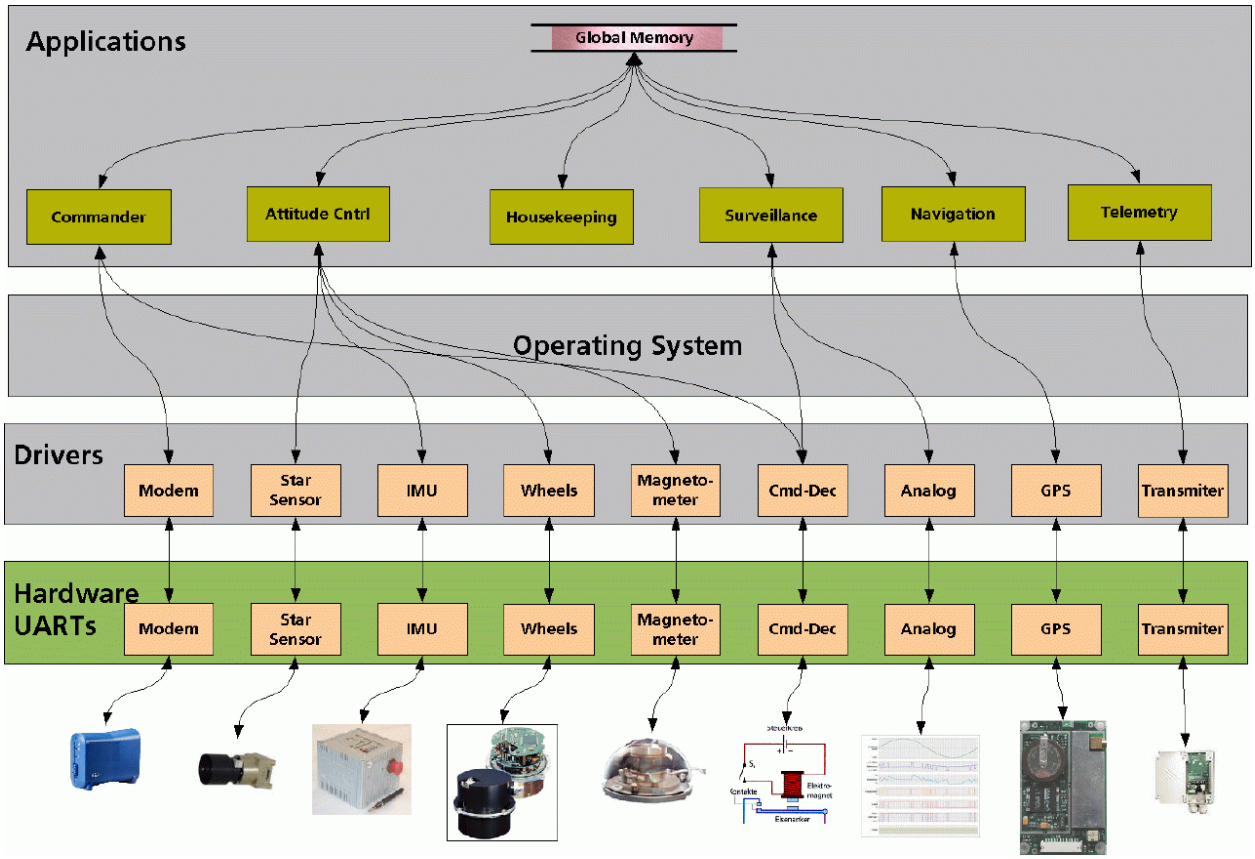

Figure 4: typical software structure

To improve dependability two board computers can be connected in parallel like it was done in BIRD (See figure 5).

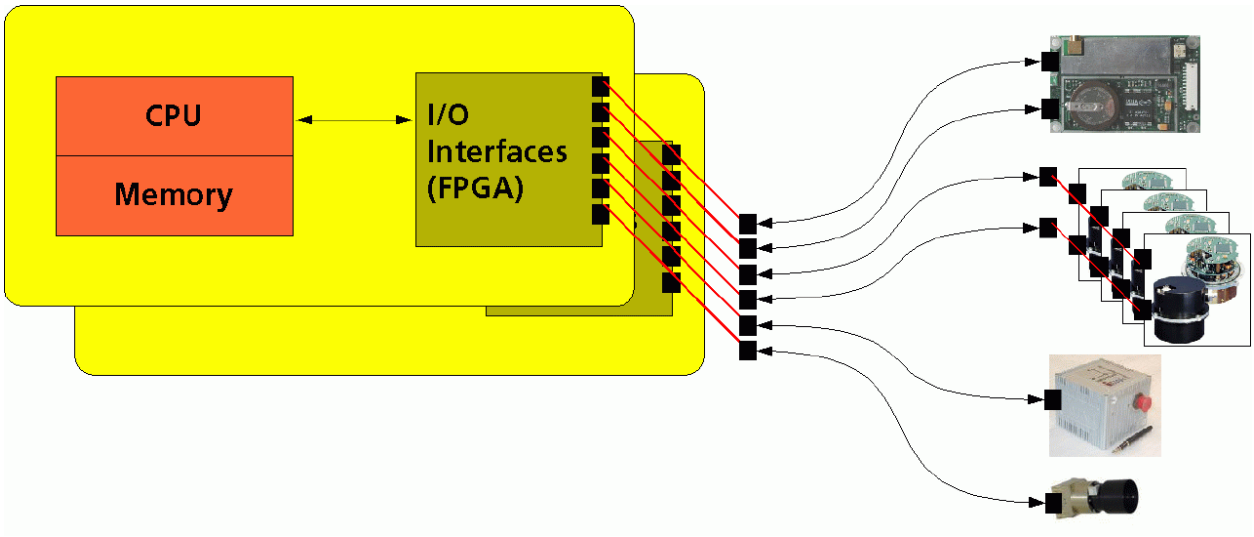

Figure 5: two computers in parallel

In this architecture of redundant control computers each of the nodes is able to execute all control tasks. One node (the worker) is controlling the satellite while a second node (supervisor) is supervising the correct operation of the worker node. If an anomaly of the worker node is detected by the supervisor node, the supervisor takes over the control of the satellite and becomes the new worker node. The faulty worker node is enforced to execute a recovery procedure and if there is no permanent error detected, it becomes the supervisor node.

5. One step ahead

The next step we did to improve this structure was a software-only step (e.g. TET). While the hardware structure stayed the same, the software structure was improved by adding a middleware (for communication). Instead of having many different interfaces, for example between applications and between application and I/O-drivers, there is only one interface for all communication in the system. The middleware provides a message-interface which can be used to interchange data between any entities in the system. Therefore there is no extra I/O- driver interface. I/O-devices are controlled by applications which are called I/O-manager and the I/O-managers communicate with the rest of the system using the middleware messages.

Another improvement is the inter-node communication. The functionality of the system is implemented as a network of applications which can be distributed among many computers in the system. The applications interchange middleware messages (services) without having to know in which node the communication partner is running. Figure 6 shows an example (Hipercar) of communicating applications, which are distributed over two nodes (computers) and figure 7 shows a typical example of application.

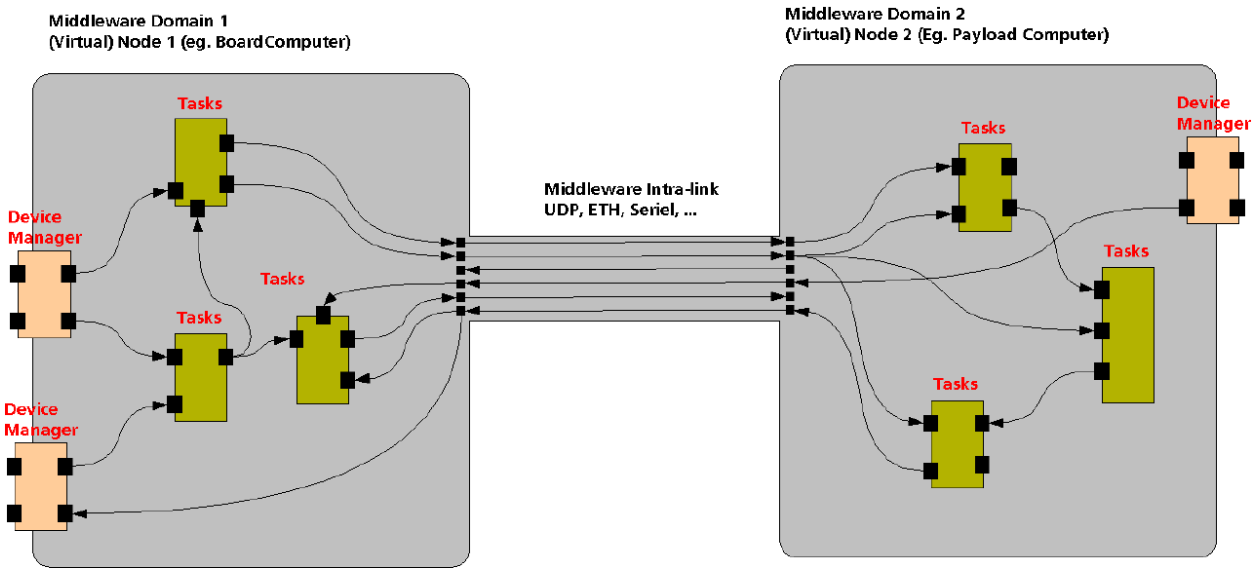

Figure 6: communicating applications in two nodes

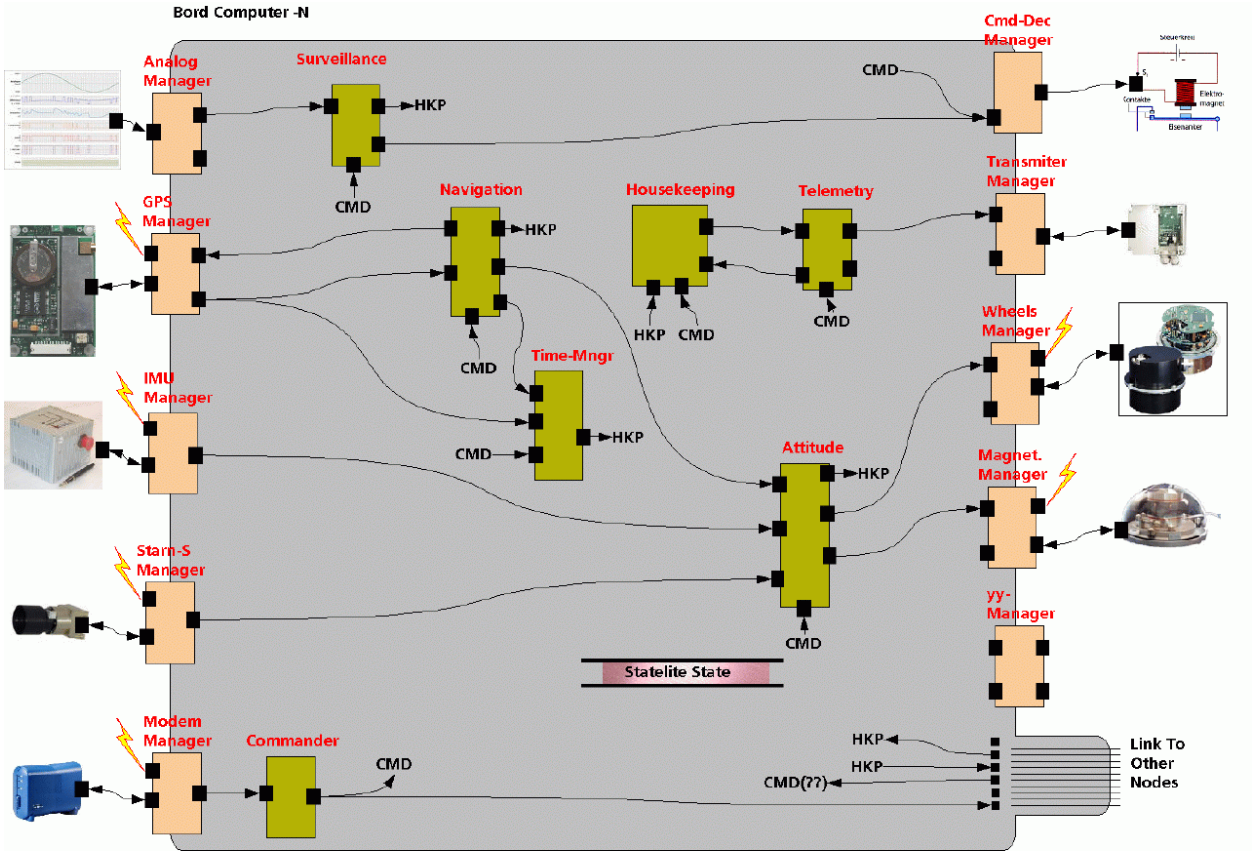

Figure 7: communicating applications in TET

6. The next step: The Middleware Switch

The next step is to unify software and hardware in an integrated architecture. Figure 8 shows a typical data/control flow to access I/O-devices.

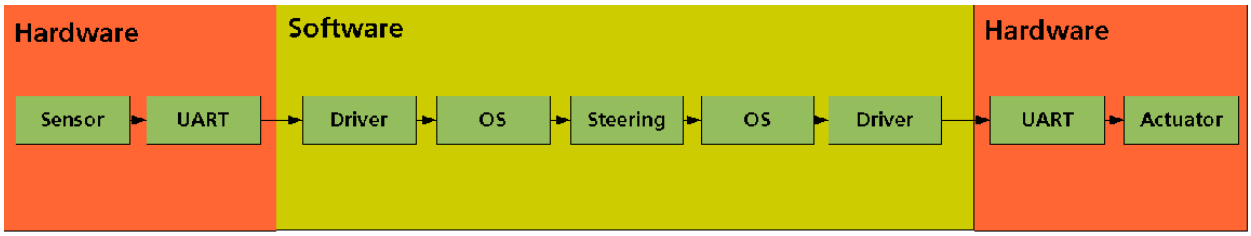

Figure 8: typical data/control flow from devices to applications

The capabilities of the FPGA (programmable hardware) emerging technology allows us to implement middleware functionality directly in the hardware I/O-interface to reach a structure like in figure 9.

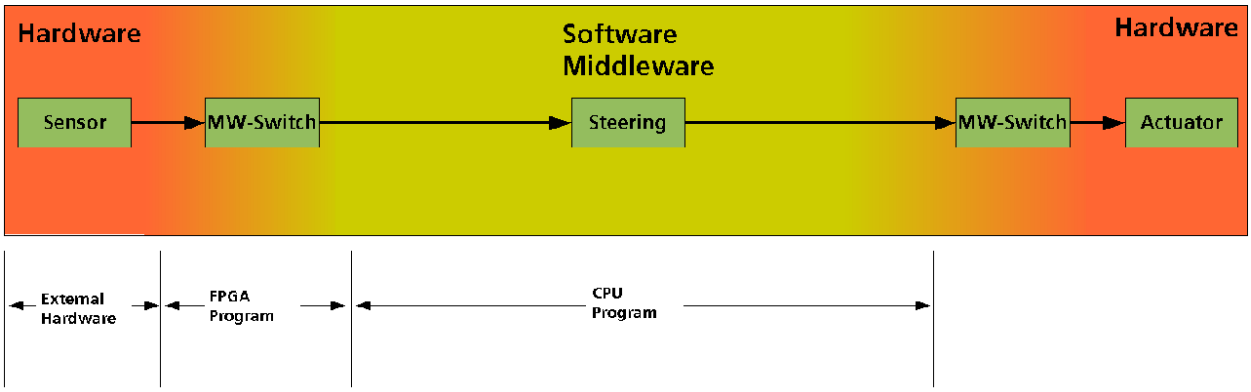

Figure 9: merging software and hardware in the Middleware

The I/O interface (traditionally an UART) will then have on one side the required device interface and on the other side it will be directly integrated to the middleware protocol. The structure from figure 6 can then be extended to the structure in figure 10.

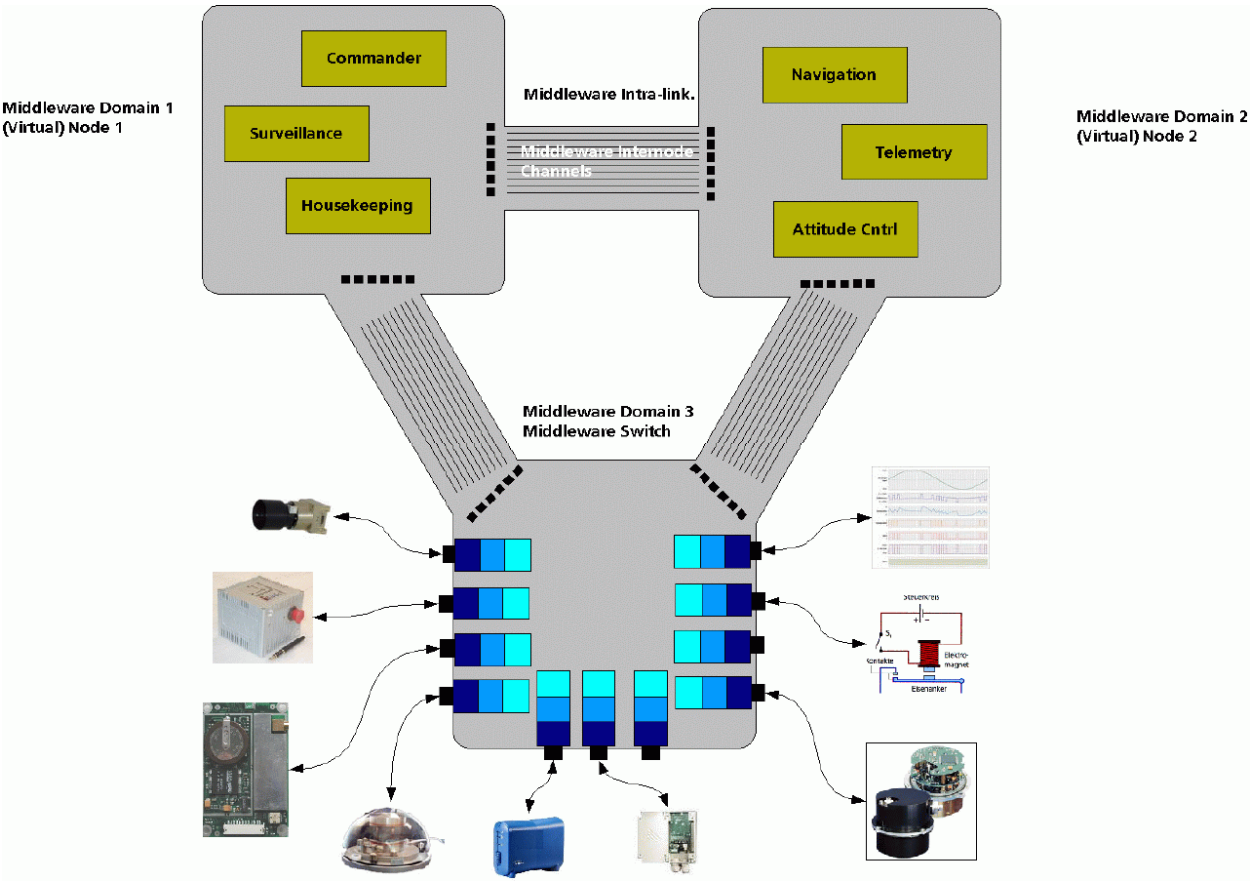

Figure 10: I/O-interfaces integrated in the Middleware

Figures 11 and 12 show the hardware view of such a system.

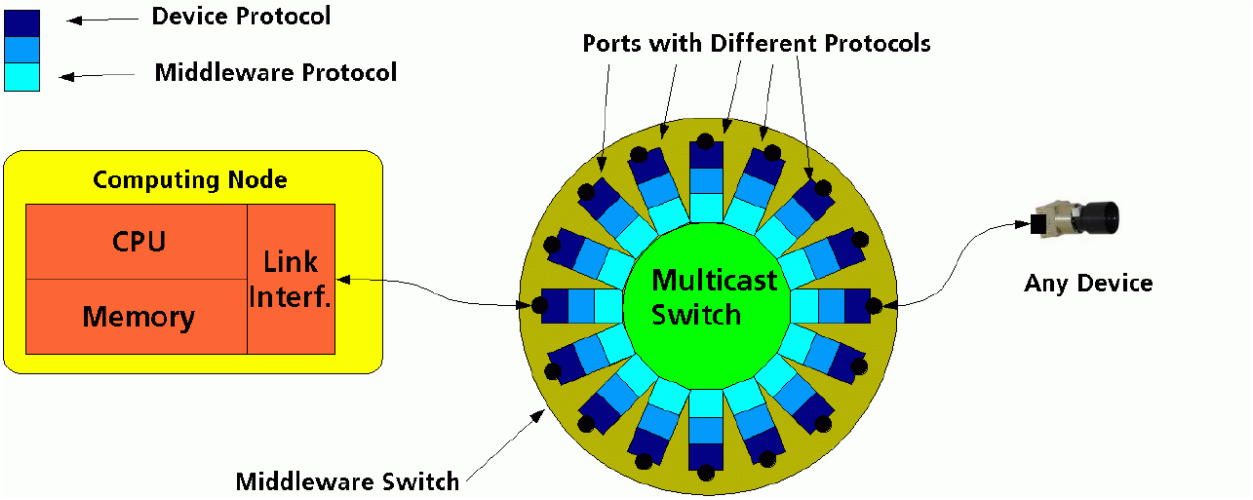

Figure 11: Middleware Switch and node Interface

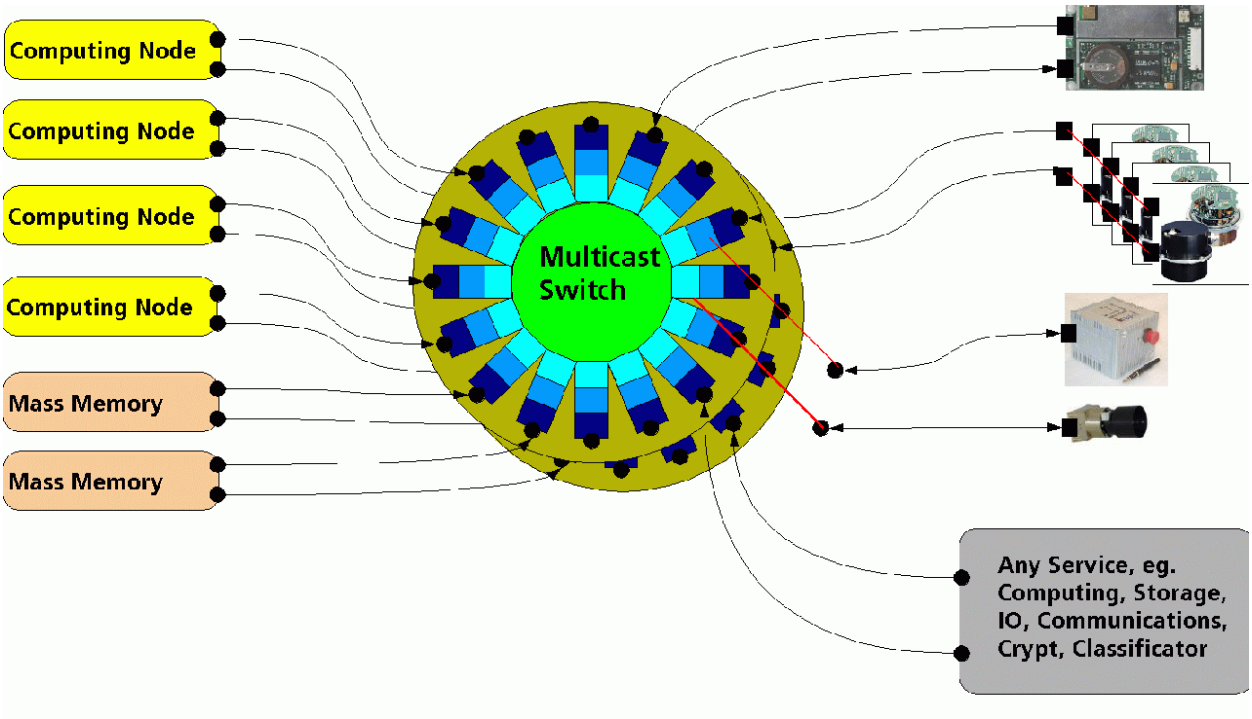

Figure 12: a network of services

The use of resources can be different from mission to mission, or from mission-phase to mission-phase. Resources can be used as redundancy to increase dependability or can be used to increase computing power. See figure 13.

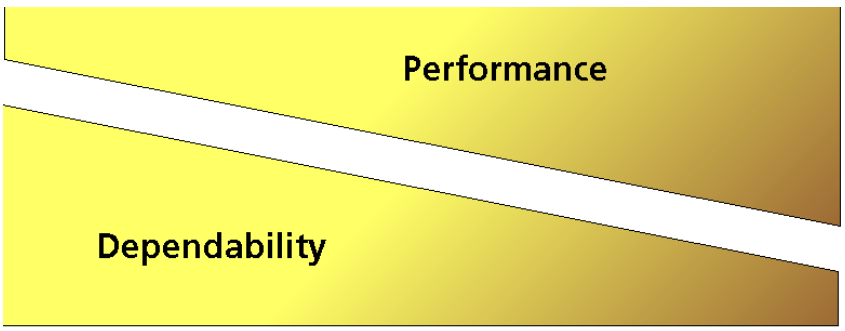

Figure 13: Use of resources

7. References

1. Bärwald W. and Montenegro S., *BIRD-Spacecraft bus controller*, Small Satellites for Earth Observation, Vol. 3, 371-373, 2001

2. Ley W. and Wittmann K. and Hallmann W. (editors), *Handbuch der Raumfahrttechnik*, Hanser Fachbuch, München, 2008

3. Brieß K et al. , *Technology Demonstration by the BIRD-Mission*, 4th IAA Symposium on Small Satellites for Earth Observation, Berlin, 2003

4. Montenegro S. and Bärwald W., *BIRD – Spacecraft bus controller*, 3rd IAA Symposium on Small Satellites for Earth Observation, Berlin 2001

5. Wertz J.R. and Larson W.J. (editors), *Space Mission Analysis and Design*, Springer, New York, 2007

6. CCSDS 102.0-B-5, *Packet Telemetry*, CCSDS secretariat, 2000

7. ECSS-E-70-41A, *Space engineering; Ground systems and operations – Telemetry and telecommand packet utilization*, ESA publications division, 2003

8. Kazeminejad B., *AsteroidFinder/SSB - Space Radiation Environment Specification*, DLR internal study,not published, 2008

9. Velazco R. and Fouillat P. and Reis R. (editors), *Radiation Effects on Embedded Systems*, Springer, 2007

10. Xilinx Inc., *Radiation Effects & Mitigation Overview*; http://www.xilinx.com/esp/mil_aero/collateral/presentations/radiation_effects.pdf