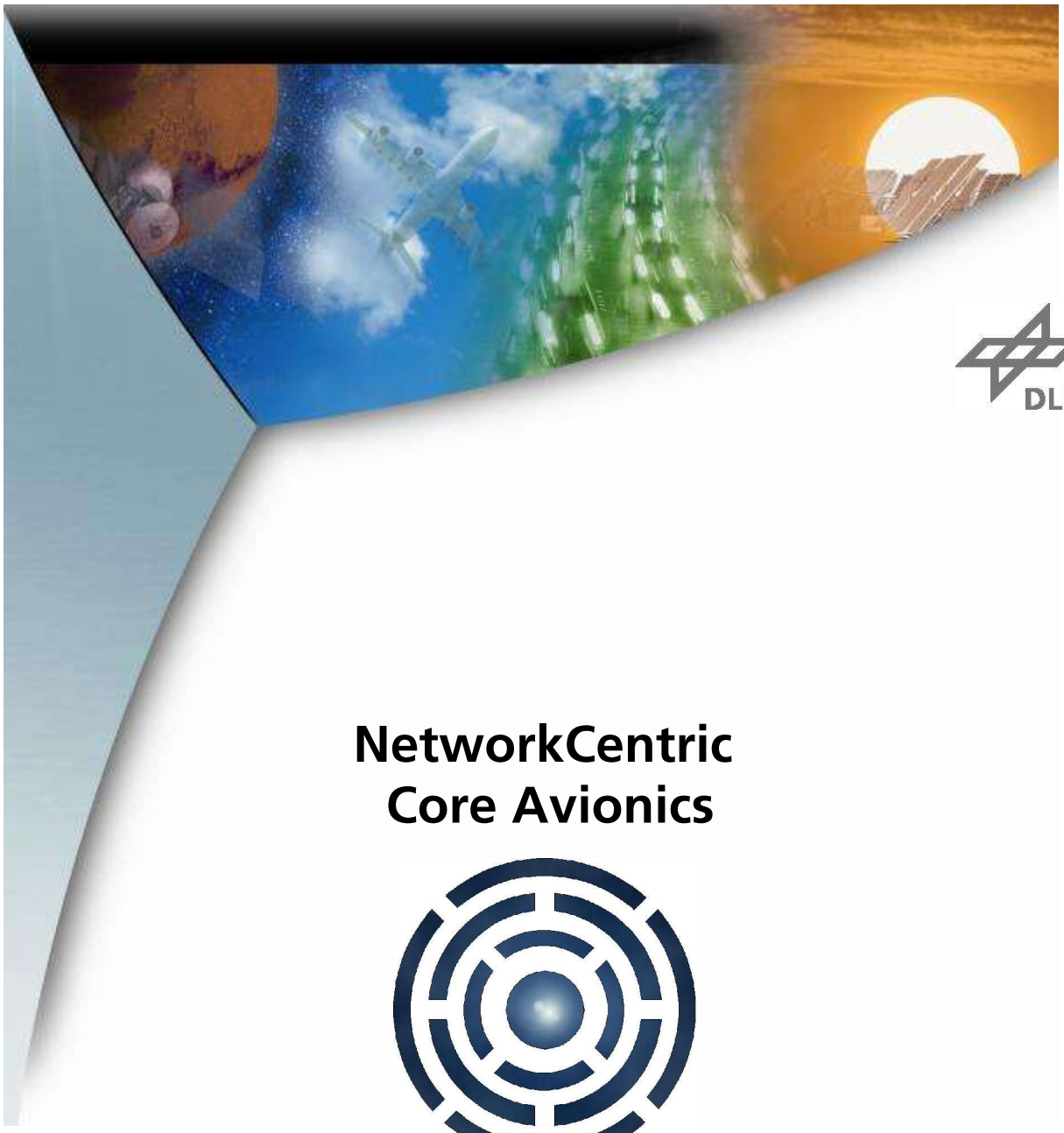


NetworkCentric Core Avionics

NetworkCentric Computing

Version: 01
Date: 02.02.2009

NetworkCentric Computing

Sergio Montenegro
DLR-RY (Bremen)
sergio.montenegro@dlr.de

The Core Avionics department aim to develop high dependable bord computing system using not reliable components, eg COTS elements.

Dependability is a main issue for space applications and after more than 30 years of research, how to achieve dependable computing, the general solution has not been found. There are many proposals how to achieve fault tolerance, or robustness or fault prevention etc, but not a single global accepted solution.

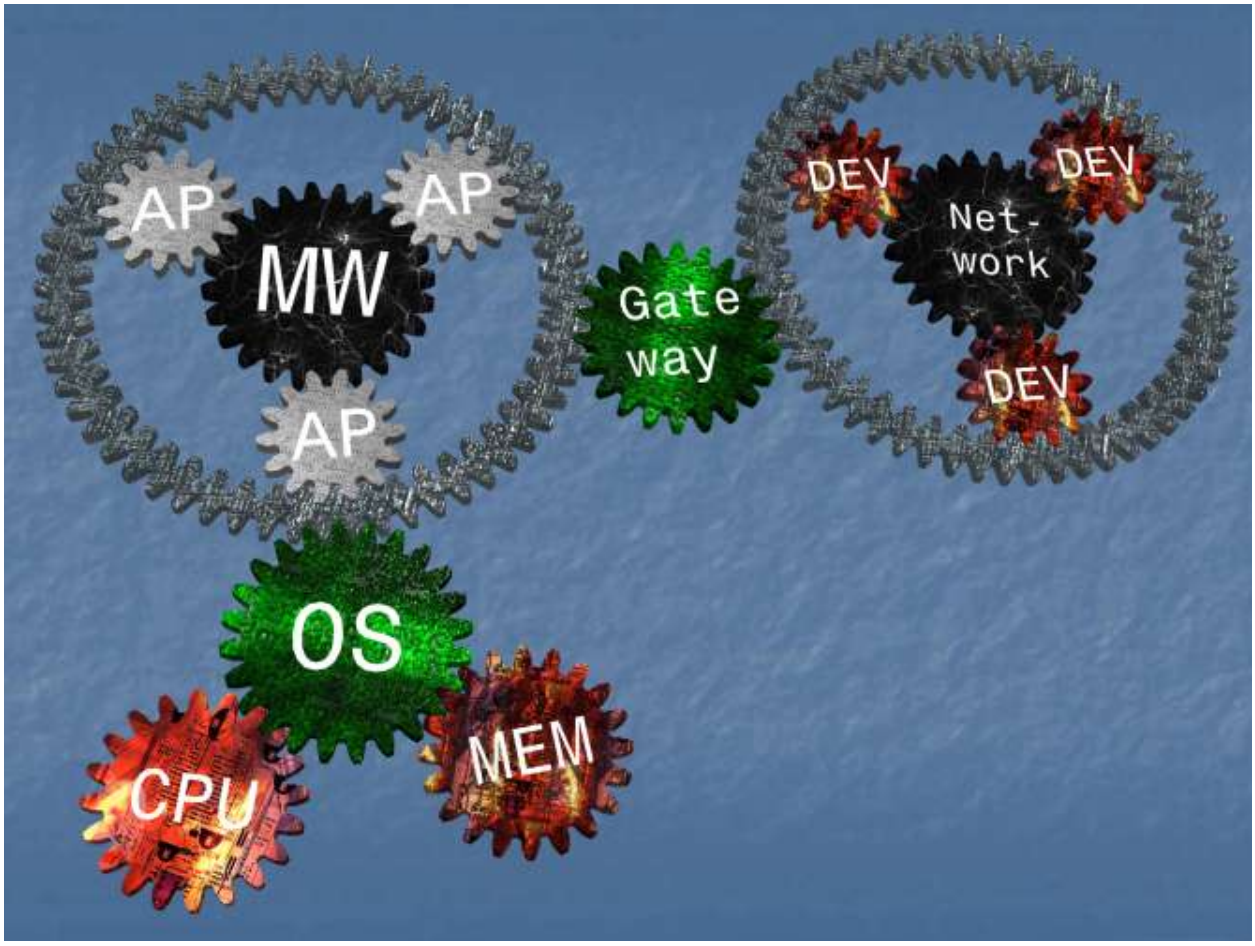
The main risk factors in a typical core avionics development are the complexity, software-hardware interfaces and the difficulties to handle many different interfaces in a single system. These topics shall be addressed in order to get high dependability.

Typical data systems for space applications are computer centric. The central component is a computer to which all (many) devices are attached. The computer has to handle devices, communication, computing, and storage of data.

We aim to create a new concept of core avionics systems which targets fault tolerance as a natural part of the concept. In our approach the central component shall not be the computer but a distributed fault tolerant network system. We call it NetworkCentric core avionics. We provide dependability to the network and to this network a set of undependable redundant components can be attached like for example devices, simple computing units, mass memory units, etc. Any of these devices may fail and the network manager will deactivate the failed device and activate a redundant one producing the same services like the failed.

The NetworkCentric core avionics machine consists of several harmonised components which work together to implement dependable computing in a simple way.

The NetworkCentric core avionics machine consists of several harmonised components which work together to implement dependable computing in a simple way.



The NetworkCentric Machine

Computing units (CPU +MEM) are managed by the local real-time kernel operating system (OS) RODOS. On top of the kernel runs the software middleware (MW) of RODOS and around this middleware the user can implement its applications (AP). To communicate with external units, including devices and other computing units, each node provides a gateway to the network and around the network several devices (IO Devs and computing nodes) may be attached to the system.

The network is built using middleware switches which implement different protocols to different units and converts them to the internal NetworkCentric protocol.

The most effective and safe way to implement a complex parallel system is to compose it as a network of simple sequential tasks. These tasks may be executed by software like for example steering control or by hardware components like for example providing temperature measurements. We aim to unify software and hardware so there shall no be difference if services are provided by software or by hardware. All service providers communicate using the same communication protocol and unified messages. All services use the same interface. For the user of a service there shall be no difference in how it was implemented (software, hardware, both) and where is being executed (in which computing node or device).

The core avionics system becomes a distributed computer system. No single node is required to be dependable. The nodes are connected by a dependable network, which is the heart of the system. Software services can be distributed on all computer nodes and may migrate from one node to another for example in case of node-failures, overloading or for power management purposes. In this way it is possible to compose a reliable system out of unreliable parts. The network is based on a publisher/subscriber protocol which is implemented in a software middleware for the software tasks and in a FPGA as a middleware switch for hardware devices and to interconnect computing nodes and mass memories.

1. The First step toward dependable computing

Our first step designing dependability for space computers was first used in the (DLR-) BIRD satellite. In this architecture there are two or four 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 old worker node is enforced to execute a recovery function and if there is no permanent error detected, it becomes the supervisor node.

2. Integrated software and hardware structures

The next step to improve this structure was a software-only step. 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 among applications or between application and I/O-drivers, there is only one interface for all communications in the system. The middleware provides a message-interface which can be used to interchange data among all 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-managers.

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 (see following figure)

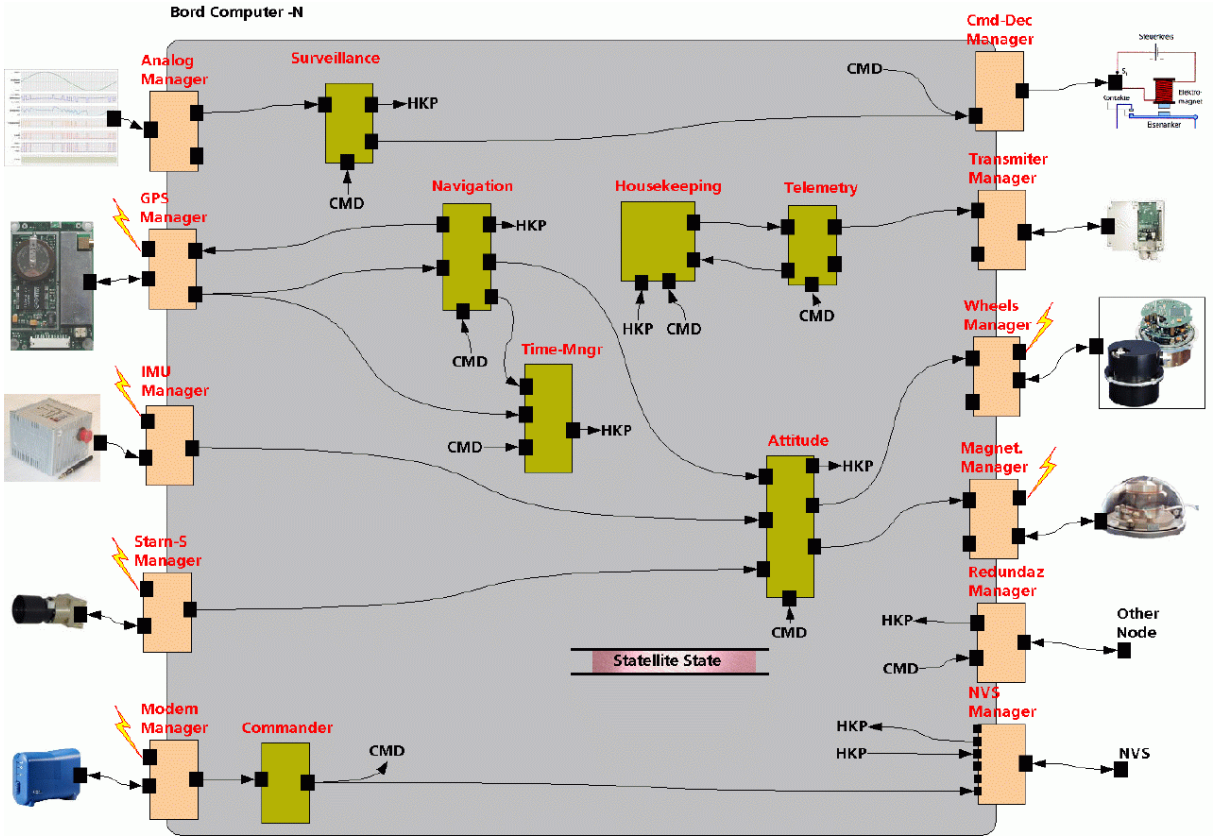


Figure 1: communicating applications

3. The Middleware Switch

The next step is to unify software and hardware in an integrated architecture. Figure 2 shows a typical data/control flow to access I/O-devices.

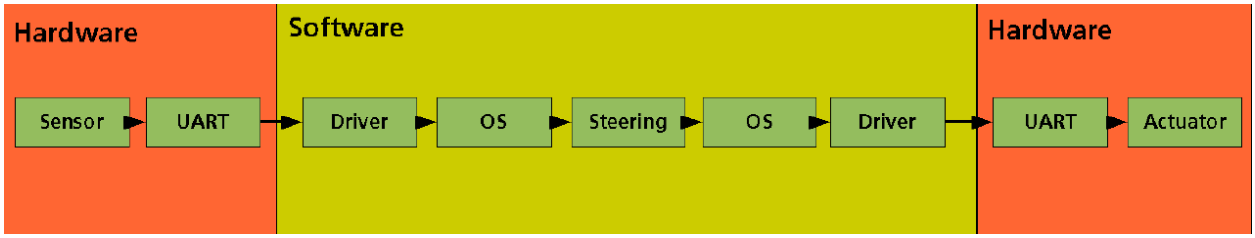


Figure 2: typical data/control flow from devices to applications

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 3. Our intension is to implement our middleware in form of an Application Specific Integrated Circuit (ASIC).

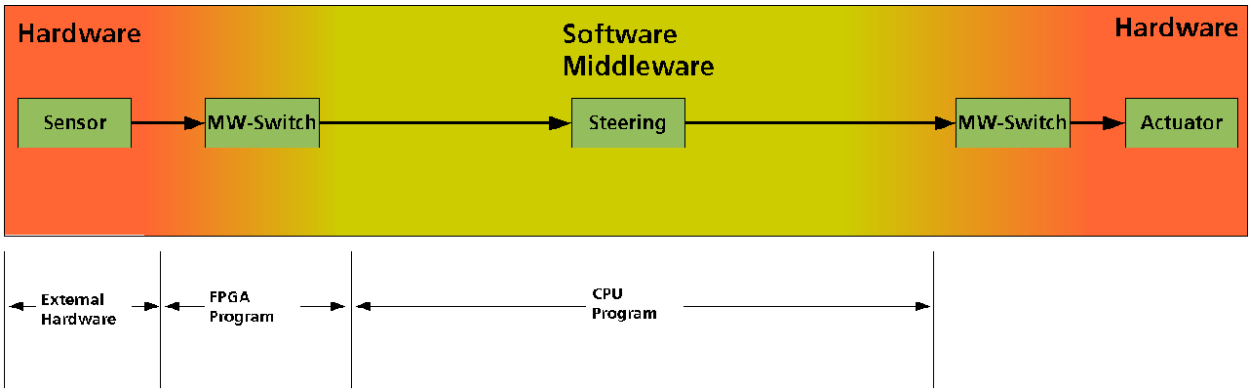


Figure 3: merging software and hardware in the Middleware

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 2 can then be extended to the structure in figure 4.

An embedded controller in the middleware switch recognizes communication requests from the I/O ports and connects/disconnects the ports accordingly. For cost-sensitive applications we will also investigate how it could be done to manage I/O links automatically by hardware without need of software controlled embedded processing resources.

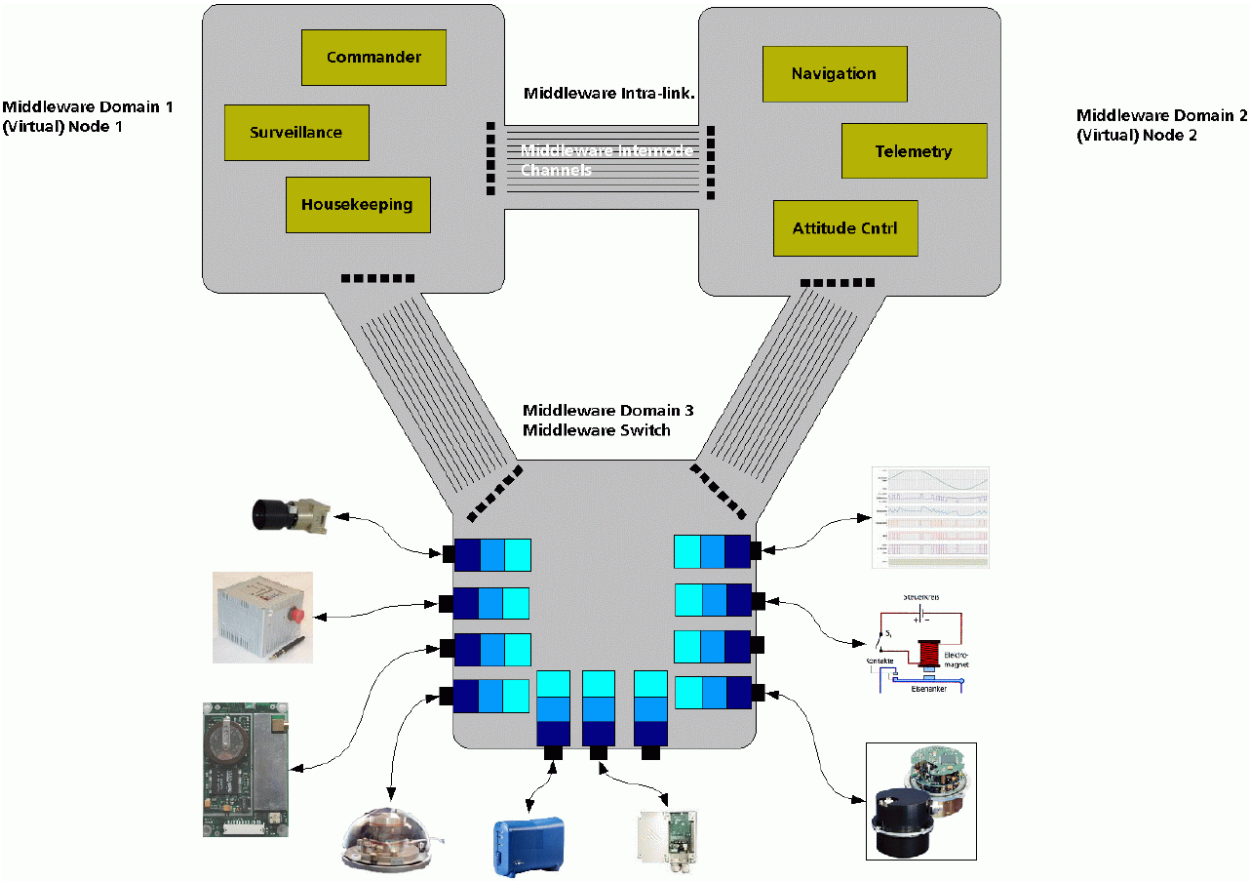


Figure 4: I/O-interfaces integrated in the Middleware