# DEPENDABLE AND FLEXIBLE BOARD COMPUTER SOFTWARE FOR PICO SATELLITES

Sergio Montenegro*, Klaus Briess**, Hakan Kayal**

*FhG FIRST
Kekulestr 7
12489 Berlin
www.first.fhg.de/~sergio
sergio@first.fhg.de

**Tecnical University Berlin
ILR institute for air and space
Marchstr 12
10587 Berlin
Klaus.briess@ilr.tu-berlin.de
Hakan.Kayal@TU-Berlin.de

## 1. ABSTRACT

An increasing number of pico satellites, are currently being developed and tested in space. Based on the development of California Polytechnic State University and Space Stanford University, these so called CUBESAT's with max 1 kg mass and 10 x 10 x 10 cm has the potential to become a very useful tool for space applications, if some of the key technologies such as a powerful and flexible board computer and attitude control devices can be adopted to this the very small form factor.

The Technical University of Berlin is also working on a CUBESAT family (See figure 1). Some of them could fly in different constellations, missions and with different payloads. Thus, such projects require a very high flexibility without lowering dependability. Based on this requirements we propose a board computer with following characteristics: Modular, high performance, configurable hardware, different IO and interconnection network capabilities.

The control computer of the first generation is based on the PowerPC CPU Family. The first Version is a compact single board Controller running at 60 Mhz. On the same board are a CAN controller, UARTS, Ethernet controller and several digital I/O signals. The next Version will implement a very low power CPU and configurable hardware like FPGAs. to be able to reload hardware to match different applications and payloads. In this way the TUB-Cube-Sat Bus can be configured very easily for different missions, devices and payloads.

*Figure 1: TU-Cube-Sat Model*

To be able to experiment with different configurations, architectures and devices, the system software has to be modular and constructed using independent components. For

this purpose we use the real time Operating System BOSS, which was developed originally for the BIRD satellite (already 3 years in orbit). BOSS was designed for dependability and simplicity, because complexity is the root of most development errors – if you eliminate complexity, you eliminate most development errors.

Our aim is to obtain the greatest possible dependability of embedded systems by reducing development errors (through simplicity) and handling runtime anomalies (by fault tolerance support). The principles underlying the creation of BOSS and its middleware were: find and build the irreducible complexity, use modern framework technology for the underlying operating system (BOSS) and component technology for the middleware and its applications.

## 2. CONTROL SYSTEM MICROWHEELS, PRECURSOR OF TU-CUBE-SAT

The first step toward the TU-CUBE-Sat was the Microwheel project, where the basic control mechanisms were developed and tested. An important design decision was to use a CAN-BUS to interconnect all devices in the satellite (See figure 2).

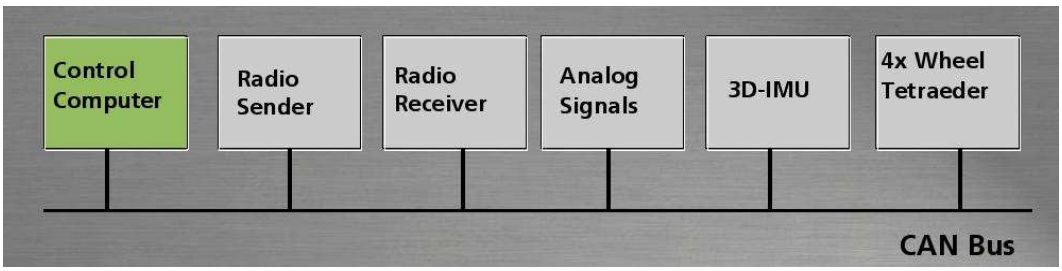


Figure 2: Devices and CAN bus in the Microwheel prototype

In this way, the control computer has direct access to all devices, in the same way like any other device. In normal case the control computer has the total control over the satellite, but in an emergency case it is possible to send commands from the radio receiver (telecomanding) directly to any device in the satellite using the same (CAN) protocol like the board computer.

## 3. THE OPERATING SYSTEM BOSS AND ITS MIDDLEWARE

BOSS (from FhG FIRST) is a real-time embedded operating system and middleware, which were designed for safety and simplicity and to allow their own mathematically formal verification. Complexity is the root of most development errors – if you eliminate complexity, you eliminate most development errors. This was one target of BOSS. The BOSS-middleware simplicity allows the system to be easily understood, used and ported to other platforms even to FPGA (Field Programmable Gate Array ) logic!

## 3.1. The BOSS-Middleware principle

The BOSS-Middleware provides very simple communication mechanisms for applications running on the top of it, regardless if they are implemented in software or in (FPGA) hardware (now in development). It was designed to support fault tolerance. All processes running on top of the BOSS-Middleware can exchange messages asynchronously using a subscriber protocol: a process or a hardware device can subscribe to one or more message types by name. When a process or a hardware device sends a message of a given type (name), each subscriber to this name receives a copy of the message. For communication purposes, the node and even the software/hardware barriers/boundaries are transparent. The messages are distributed across these barriers. Using this approach, we obtain very high flexibility and users do not have to differentiate between local/remote functions or hardware and software functionality. The system can be configured or reconfigured simply by plugging software modules or hardware devices into/out of the middleware.

The BOSS-Middleware provides transparent support for fault tolerance. The simplest example of this is a controller sending commands (messages) to a device. As a first step, we insert the middleware between the device and the controller by implementing the same interface on both sides of it. Neither the controller nor the device notices this intervention. The middleware forwards the messages across node boundaries, which means that controller and device no longer need to be located in the same node. Furthermore, messages can be replicated if there is more than one subscriber to a message type (name). Now we can add a monitor to hear messages of the same type, like the device. The monitor can create a log file and/or execute an online diagnosis of the system. Again, no one will notice this intervention (see Figure 3).

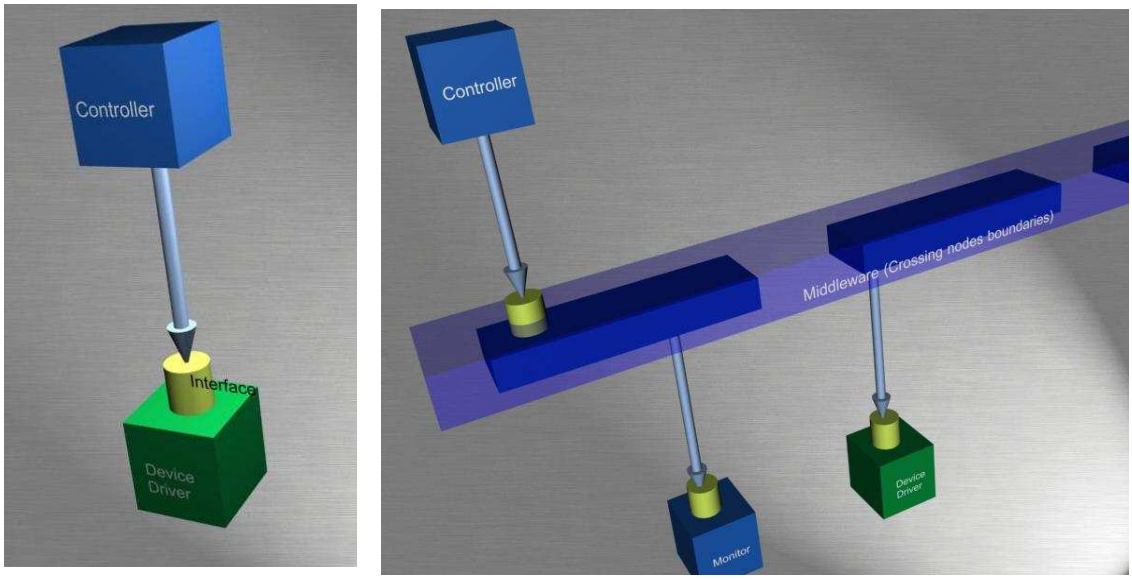


Figure 3: Middleware insertion

The next step is to replicate the controller, simply by creating several instances of it, if possible running on different nodes. They need not know about the existence of the other replicas. What are needed now are voters that intercept all messages to the device, compare

them and send only those that are most likely to be right (a democratic decision, e.g. two of three) to the device. If required, it is possible to replicate the voter, too. One voter – the worker (as in BIRD) – is in charge and the other one – the supervisor (as in BIRD) – is a hot redundancy. The supervisor is ready to take control if the voter in charge fails to respond (see Figure 4).

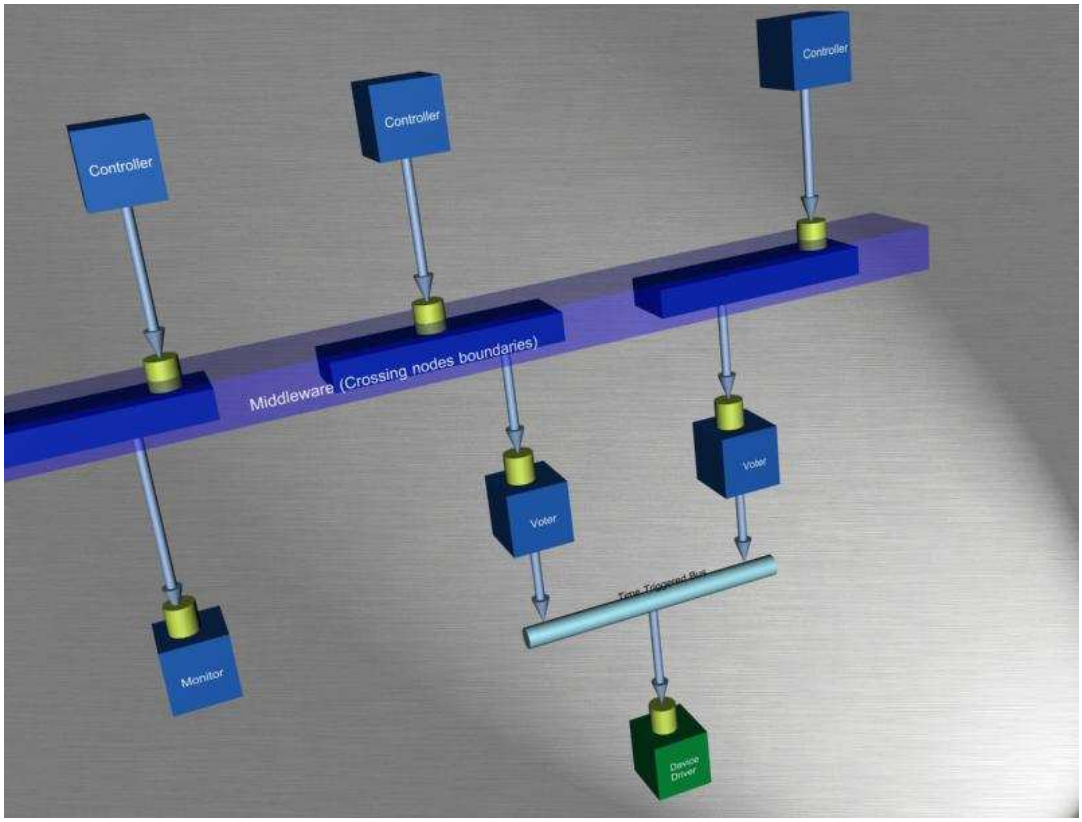


Figure 4: Middleware and voters

The routing of messages depends only on the types/names of the messages and on who is subscribed to each name.

## 4. REFERENCES

1999: Sergio Montenegro. Buch Entwicklung sicherheitsrelevanter Systeme, Hanser Verlag, ISBN: 3-446-21235-3

2003: Briess, K., Baerwald, W., Gill, E., Halle, W., Kayal, H., Montenbruck, O., Montenegro, S. Skrbek, W., Studemund, H., Terzibaschian, T., Venus, H.
Technology demonstration by the bird-mission
4th IAA Symposium on Small Satellites for earth observation, April 7 -11, 2003
ISBN 3-89685-569-7

2002: Briess, K., Bärwald, W., Hartmann, M., Kayal, F., Krug, H.3, Lorenz, E., Lura, F., Maibaum, O., Montenegro, S., Oertel, D., Röser, H.P., Schlotzhauer, G., Schwarz, J., Studemund, H., Turner, P., Zhukov, B.
Orbit experience and first results of the bird-mission
53rd International Astronautical Congress The World Space Congress -2002, 10-19 October 2002 / Houston, Texas

2002: K. Briess, S. Montenegro, W. Bärwald, W. Halle, H. Kayal, E. Lorenz, W. Skrbek, H. Studemund, T. Terzibaschian, I. Walter
Demonstration of Small Satellite Technologies by the BIRD Mission
16th Annual AIAAA/USU COnference on small satellites, Logan,Utah, USA 2002

2002: Sergio Montenegro, Volkert Barr

BOSS/Ada: An Open Source Ada 95 Safety Kit

Ada Deutschland Tagung 2002 6. 8. März 2002, Jena