BOSS/EVERCONTROL
Dependable RT Operating System and Middleware

Sergio Montenegro, Felix Holzky
FhG-FIRST
Kekulestr 7, 12489 Berlin
sergio@first.fhg.de
www.first.fhg.de/~sergio
Tel +49 30 63921878

Abstract

BOSS/EVERCONTROL are a real-time embedded operating system and middleware, which were designed for safety and simplicity and to allow their own mathematically formal verification (on which work is currently in progress). Nowadays, formal verification is not possible for complex systems; what is needed is a simple and well-structured description of the system (like BOSS) to be verified,. Further advantages of simplicity are obvious: the system can be easily understood, used and ported to other platforms. Besides, complexity is the root of most development errors – if you eliminate complexity, you eliminate most development errors.

1. Introduction

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 EVERCONTROL 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 (EVERCONTROL). The results are very promising. BOSS has been in continuous use in space (BIRD satellite) and in medical devices for a number of years now. Even complex functionality can be implemented very easily using BOSS/EVERCONTROL.

2. An Example: BIRD Satellite

Figure 1: BIRD in space

BIRD is a microsatellite (German Aerospace Center, DLR) designed for the early detection of fires around the globe. It is able to detect any fire larger than 12 m^2, to compute temperatures to an accuracy of half a degree, and compute energy radiation from fires, cities, industry, etc. Microsatellites have to meet a major challenge: fulfilling high performance requirements using small-scale equipment and in particular on small budgets. Cost is one of the most important factors in microsatellite missions. To keep costs within the low budget limits, demonstrating new and non-space-enabled technologies for the spacecraft is a key factor in meeting high performance mission requirements. To achieve high dependability and safety and a long useful life, the on-board computer consists of four identical computers. As shown in the block diagram in Figure 2, the redundant nodes and satellite devices are interconnected by several bus systems.

The architecture of the redundant control computer allows each of the nodes to execute all/any control tasks. One node (the worker) controls the satellite, while a second node (supervisor) supervises the correct operation of the worker node. The other two node computers are spare components and are disconnected. If an anomaly in the worker node is detected, the supervisor becomes the worker and takes over control of the satellite. The former worker node is forced to execute a recovery function and, if there is no permanent error, it becomes the new supervisor node. If the recovery procedure fails or a permanent hardware error is detected, the faulty node computer is switched off and replaced by one of the spare nodes. Using this strategy, up to three permanent node failures can be tolerated, with the on-board computer remaining operable.

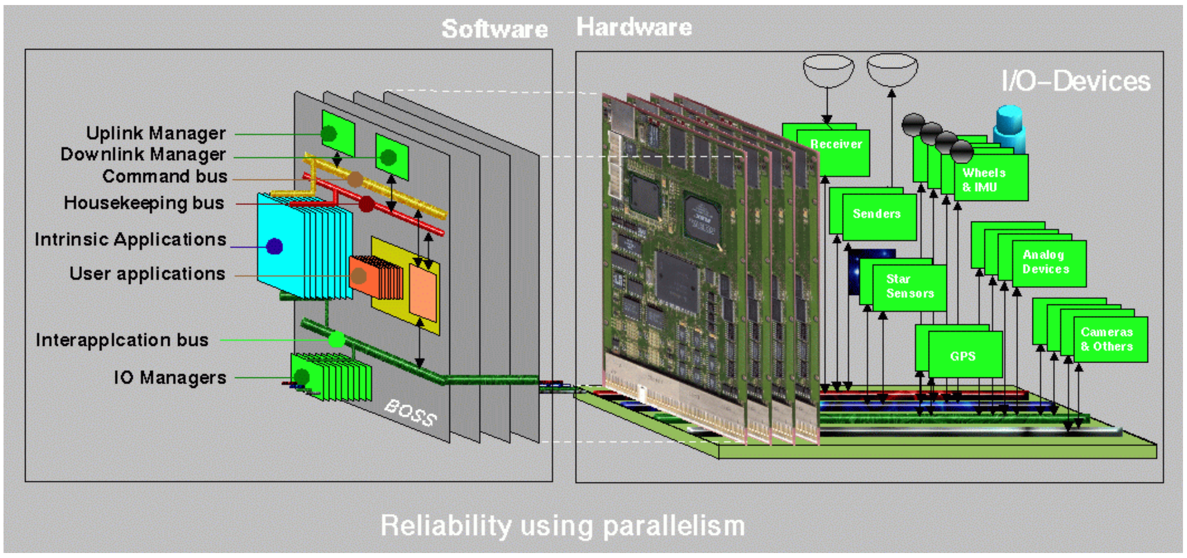


Figure 2: BIRD's Control System: software and hardware

The whole system is controlled by BOSS. The highly modular operating-system software was implemented by using the latest software technology and the critical parts were formally verified. The applications running on top of BOSS are implemented using object-oriented technology, resulting in highly modular application software. To achieve a well-structured application system, we defined a software backplane (the precursor of the middleware EVERCONTROL) which consists of two software buses. Each application implements an interface to each of them. One software bus is used to distribute commands to the applications, and the second collects status information, which has to be sent down to the control stations on Earth. The principle of a software backplane allows easy configuration of the system by simply plugging the software components in and out of the SW backplane. This principle was further developed to produce the EVERCONTROL middleware.

3. BOSS Description

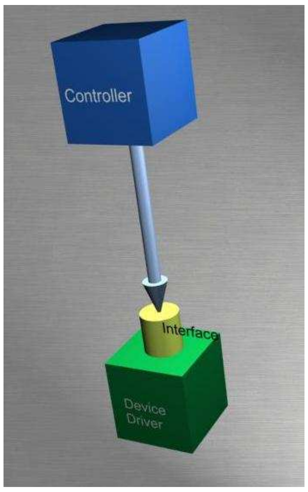
BOSS was designed as a framework to provide a dependable real-time embedded operating system that can be easily certified because it is very simple. Simplicity does not, however, mean lack of functionality. Resource management, synchronization, communication, I/O and interrupt handling and all the functions we can expect from a microkernel are there – only in as simple as possible. BOSS offers the following features: multithreading; priority-managed pre-emption; real-time and fault tolerance support; communication support; object-oriented design and implementation; C++ interface; time resolution: 1 microsecond; thread switch time: 3 microseconds (PPC at 48 Mhz); reaction time: under 3 microseconds (PPC at 48 Mhz); boot time from flash memory: under 300 milliseconds.

There are several implementations of BOSS on different platforms, e.g. PowerPC, x86, Atmel and an on-top-of-LINUX implementation. Applications written on BOSS can run without changes on any of these platforms. The on-top-of-LINUX implementation helps developers to work locally on their workstation without having to use the target system. To move to the target, they have only to recompile the code. The behaviour is the same, except for timing requirements and time resolution, which on LINUX cannot be as exact as in the target systems.

4. EVERCONTROL and Fault Tolerance Support

Running on top of BOSS is the EVERCONTROL middleware, which was designed to support fault tolerance. All processes running on top of EVERCONTROL 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 EVERCONTROL 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).



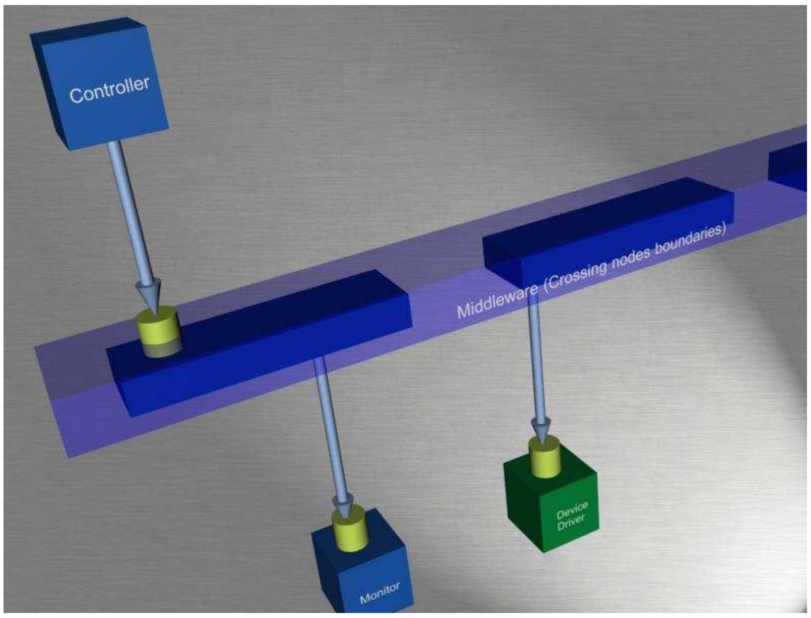


Figure 3: Middleware insertion

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).

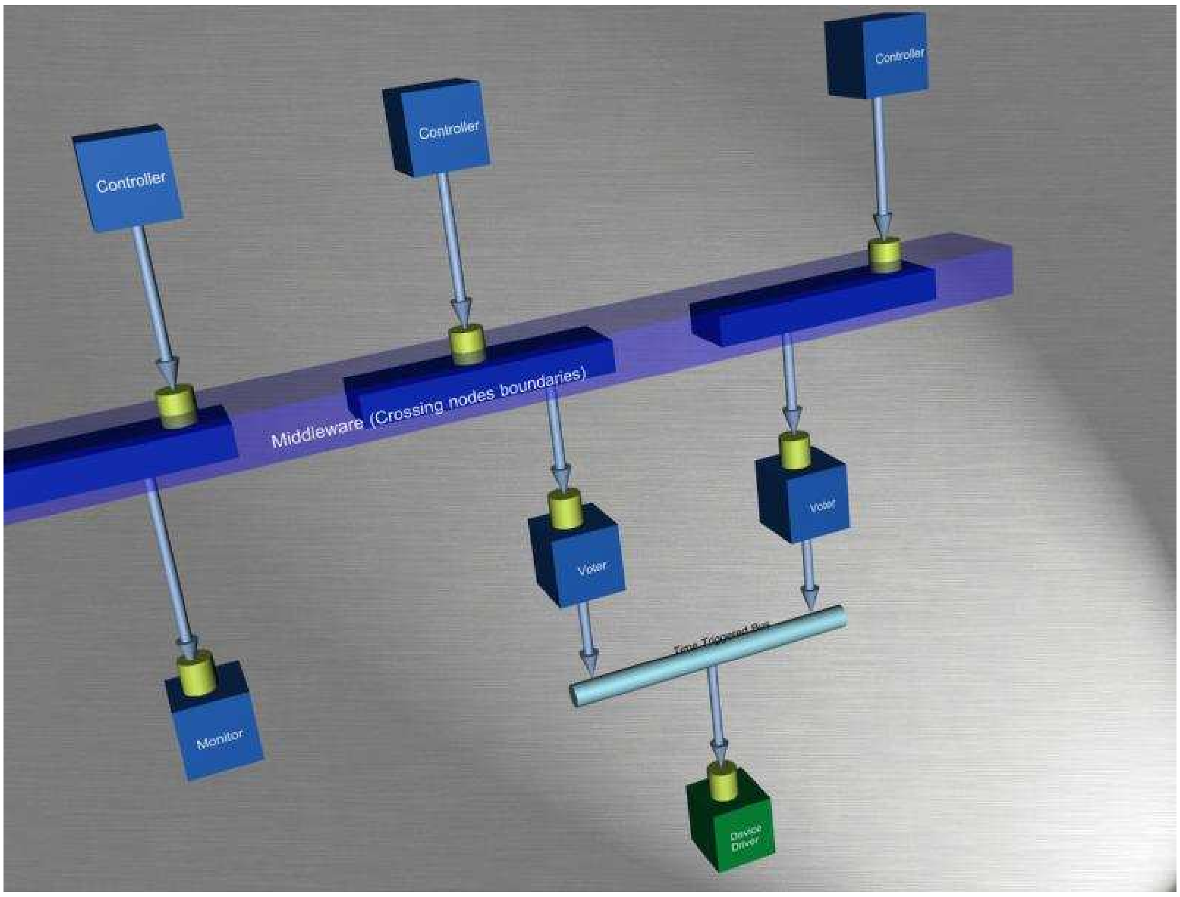
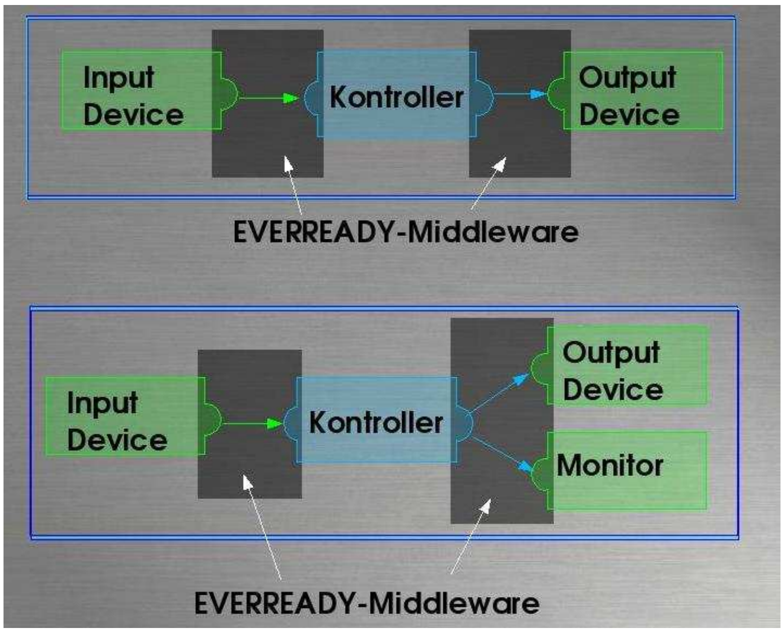


Figure 4: Middleware and voters

The routing of messages depends only on the types/names of the messages and on who is subscribed to each name. Figure 5 shows some simple examples of useful potential configurations.



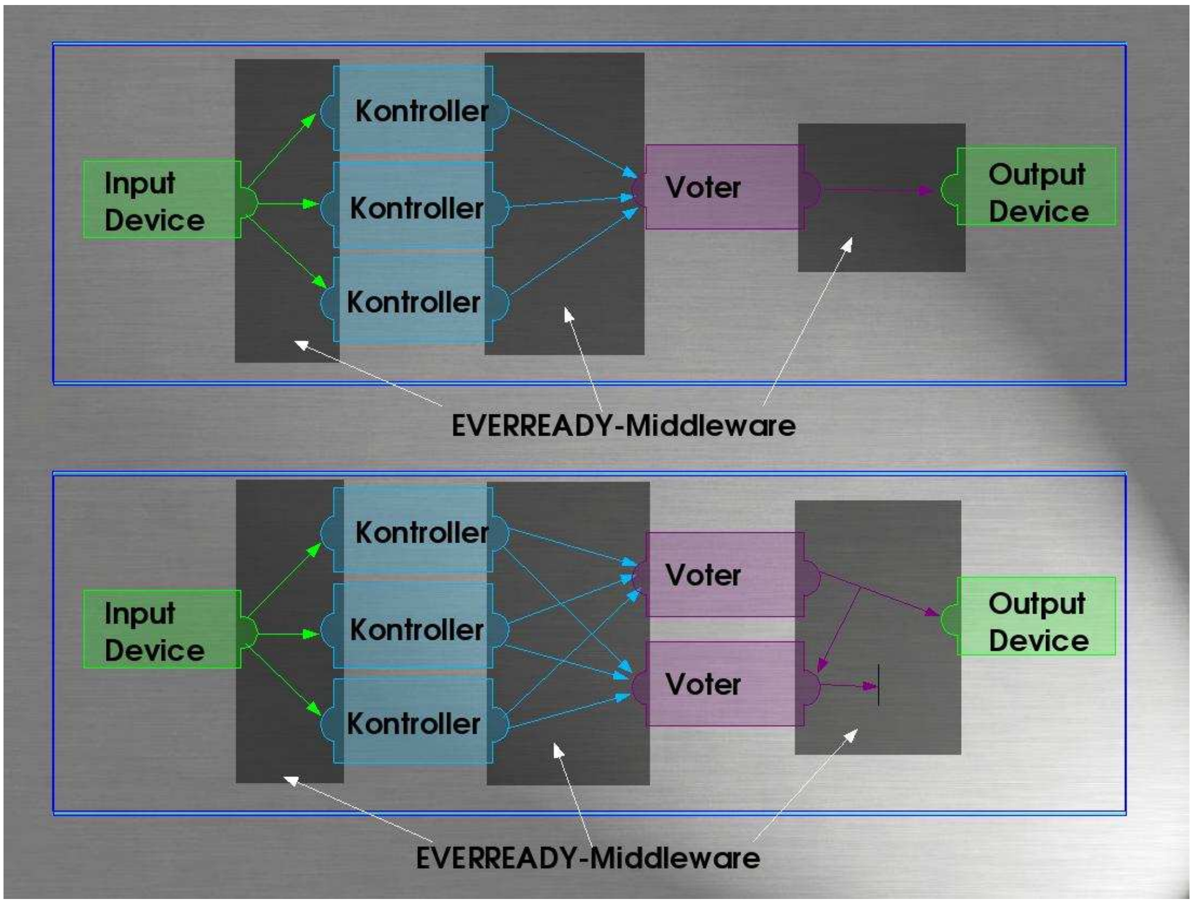


Figure 5: Examples of message routings

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