BOSS/EVERCONTROL OS /Middleware
Target ultra high Dependability

Sergio Montenegro, Felix Holzky
FhG-FIRST
Kekulestr 7, D-12489 Berlin, Germany
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

BOSS targets ultra high dependability using a principle which the world forgot a long time ago: *Simplicity.* The opposite, complexity, is the root of most development errors – if you eliminate complexity, you eliminate most development errors and you can understand better the system.

Simplicity does not mean, however, lack of functionality. Resource management, synchronization, communication, I/O and interrupt handling and all the functions one can expect from a microkernel are implemented – just as simple as possible. An important BOSS design target is the irreducible complexity; this is the minimal possible complexity for a determined function. When it is not possible to implement it simpler without destroying the functionality.

BOSS is based on very few and simple basic functions, which can be proofed very faithfully and partially using formal methods. These basic functions are used for almost every operation of the kernel.

BOSS was designed to support fault tolerance, this implies replication of services, redundancy and redundancy management. The basic replication unit is a computer node, this implies a distributed or a parallel system. To administrate the resource distribution BOSS provides a middleware (running on the top of the BOSS-kernel), which hides the location and number or replicas.

2. An Example: BIRD Satellite

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.

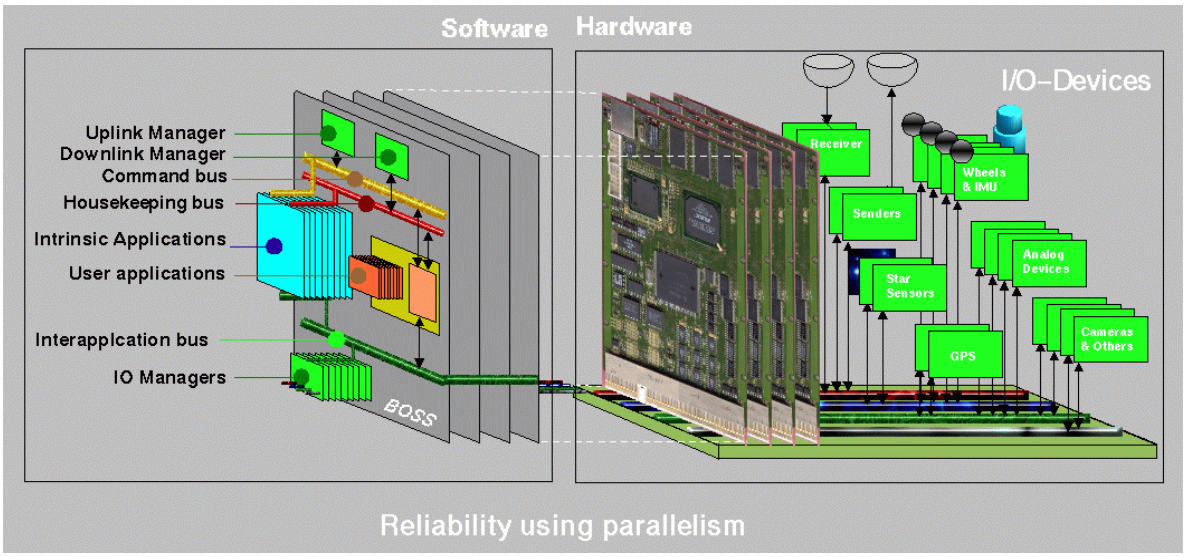


Figure 2: BIRD's Control System: software and hardware

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 AVR (TinyBoss) 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. BOSS Middleware and Fault Tolerance Support

On the top of the kernel runs the BOSS middleware which administrates the distribution of services. BOSS middleware was designed, besides normal message distribution, to support fault tolerance. The services in the system are provided by a network of tasks distributed in the computer nodes. In the simplest configuration there is one task for each service, but for fault tolerance reasons there can be for each service several identical tasks with normal or degraded versions of each of them, running on different nodes.

All tasks running on top of the BOSS Middleware can exchange messages asynchronously using a publisher/subscriber protocol: Tasks which provide a service make it public (publishes it) under a given name. Any task can subscribe to one or more services (message types by name). When a task publishes a service it sends messages of a given type (name), each subscriber to this name receives a copy of this messages. 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/services or hardware and software functionality. The system can be configured or reconfigured dynamically simply by plugging software modules or hardware devices into/out of the middleware. There are no fixed communication paths. Each data transfer is resolved just in time using the registered names.

The simplest example of building a fault tolerant application is a controller sending commands (messages) to a device (figure 3a).
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 given type. 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 3b).

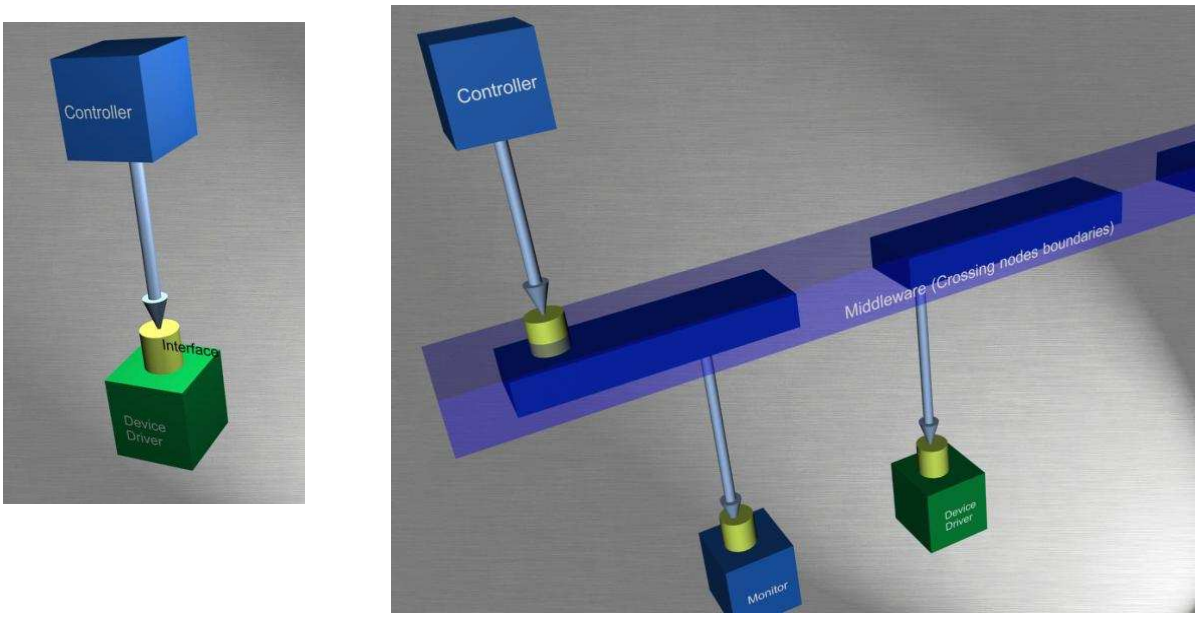


Figure 3: Middleware insertion

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).

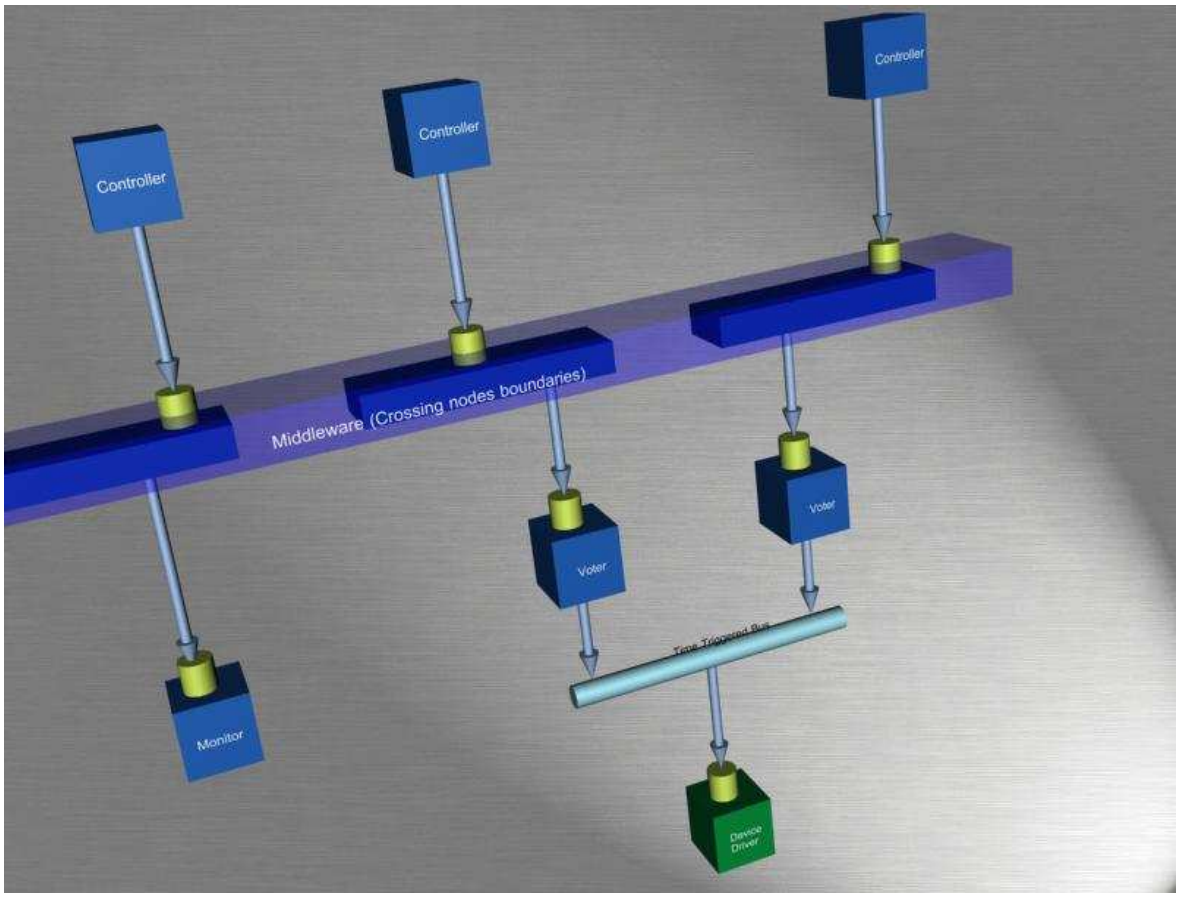
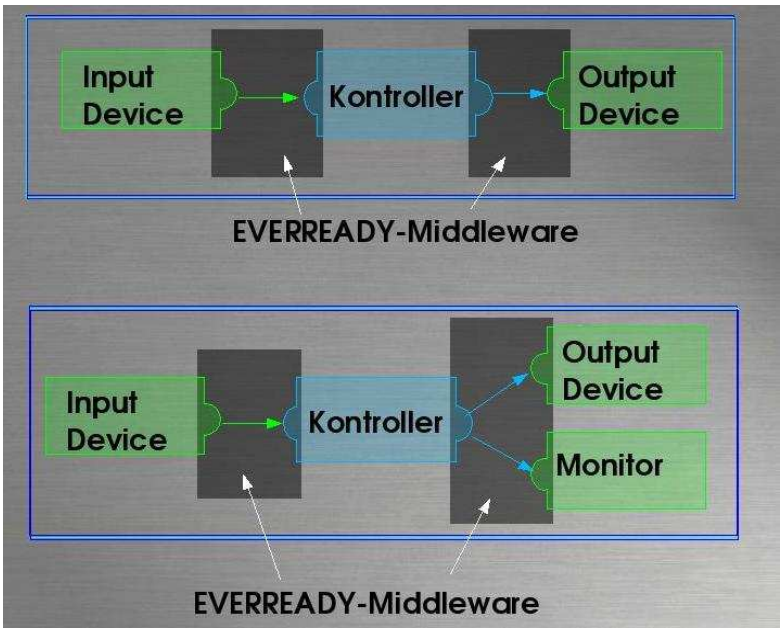


Figure 4: Middleware and voters

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.



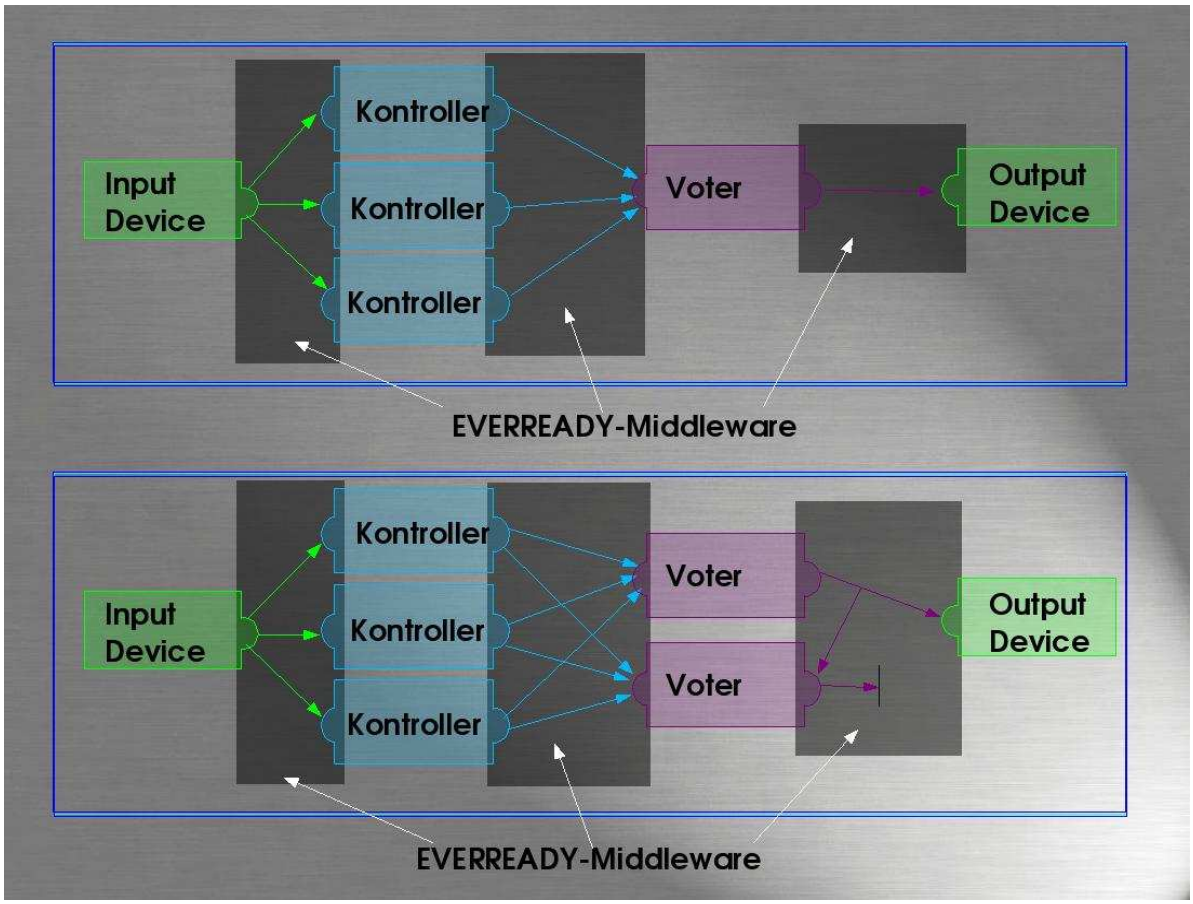


Figure 5: Examples of message routings

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