New Software Technology in Space: BOSS – a Dependable Open Source Embedded Operating System

Sergio Montenegro
FhG FIRST
Kekulestr 7, 12489 Berlin
www.first.fhg.de/~sergio
sergio@first.fhg.de

Abstract: BOSS targets a principle which the world forgot a long time ago: Simplicity. BOSS is an embedded real time operating system for safety critical applications. Our experience shows that the first enemy of safety is the complexity. If you need safety, use only what you can understand. This was the philosophy creating BOSS. First: build every thing as simple as possible. Second: use modern framework technology to reduce the complexity of the applications. Third: use component technology to be able to handle the remaining complexity. The result is very promising. BOSS is already working for years without interruptions, for example in space (Satellite BIRD [BIRD0]), in medical devices, protocol tester and other Applications [ROD03]. Furthermore BOSS is open-source, so you can look inside, adapt it and move to other platforms/hardware.

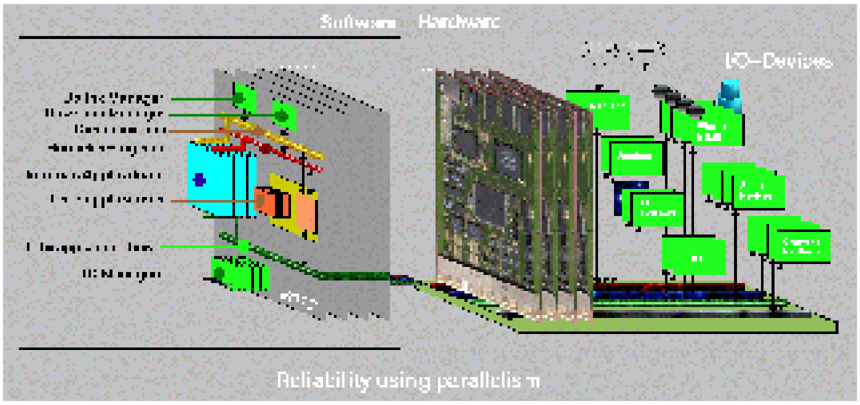
1 Introduction

BOSS was designed as a framework to be a dependable real time embedded operating system which can be easily certified and verified. Due to the fact, that complexity is the first enemy of safety [MONT99], BOSS is intended to be as simple as possible, so, it is easier to understand, to review, to verify, to use etc. The kernel can be printed in a few pages.

Some parts of BOSS are being verified mathematically and formally using model checker and theorem prover. The basic components -- list management and communication buffers -- were verified using the VML and SPIN model checkers [SPIN], [VERIFUN]. Now we are working in the verification of more complex components using the theorem prover ISABELLE [ISABELLE]. With the current state of the art on formal verification, complex systems cannot be verified formally, but BOSS can be. BOSS is based on very few and simple basic functions, which can be proofed very faithfully, and these functions are used for almost every operation of the kernel.

2 An example: Satellite BIRD

Small satellites have to meet a big challenge: to answer high performance requirements by means of small equipment and especially of small budgets. Out of all aspects the cost aspect is one of the most important drives for small satellite missions. To keep the costs within the low-budget frame the demonstration of new and not space-qualified technologies for the spacecraft is one key point in fulfilling high performance mission requirements [BIRD1]. Taking this into account the DLR micro-satellite mission BIRD (Bi-spectral Infra-Red Detection), which mission is to find fire all around the world, [BIRD0] has to demonstrate a high performance capability of spacecraft bus by using and testing new technologies including modern software technology. The control system of BIRD relays on BOSS. The spacecraft bus is controlled by the redundant dependable board computer. To achieve a high dependability, safety and lifetime, the board computer is formed of four identical computers (see picture 1) [MONT00].



Picture 1: Bird control system.

Each of the nodes is able to execute all control tasks. One node (the worker) is controlling the satellite while a second node, the monitor, is supervising the correct operation of the worker. The two other node computers are spare components and are disconnected. If an anomaly of the worker node is detected, the supervisor becomes the worker and takes over the control of the satellite. The old worker node is enforced to execute a recovery function and, if there is no permanent error, it becomes the role of the monitor. If the recovery procedure fails or if a permanent hardware error is detected, the faulty node computer will be switched off and replaced by one of the spare nodes. By this strategy up to 3 permanent node failures can be tolerated while the board computer stays operable.

The applications running on top of BOSS are implemented by using object oriented technology, resulting in a highly modular application software. We defined a software backplane which consists of two software buses. Each application implements an interface to each software bus. One software bus is used to distribute commands. The second collects status information of the applications. The principle of a software backplane allows us to easily configure the system by simply plugging the software components in or out of the back plane.

3 BOSS Description

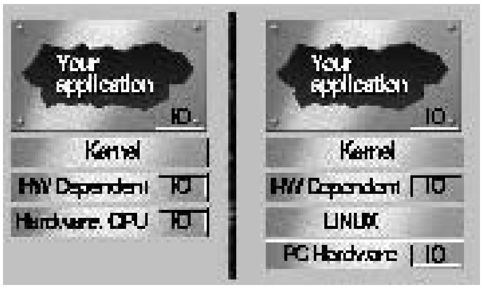
BOSS characteristics are: multithreading, pre-emptive priority managed scheduling; real time support; fault tolerance support; communication support ; OO-design and implementation; C++ interface; Time resolution 1 microsecond; Thread switch time 3 microseconds PPC at 48 Mhz.; Reaction time: under 3 microseconds PPC at 48 Mhz; Boot time from Flash: under 1/2 Second.

Currently there are 3 implementations of BOSS on different platforms: powerPC, x86 and an on-top-of-LINUX. Applications written on BOSS can run without changes on any of these platforms. The on-top-of-LINUX implementation helps the developer to work locally on his workstation without having to use the target system. To move to the target he/she needs only to recompile the code. The behavior will be the same except for timing and time resolution which on LINUX can not be as exact as in the target systems.

4 Development environment

The application development can be done on a LINUX workstation using the on-top-of-LINUX BOSS implementation. Devices can be simulated on LINUX or on BOSS. For the execution on the target system we provide a debugger interface and serial connections to load and debug the system. Using LINUX as front-end to the target system, the implementation/debug can be done remotely using internet. It is possible to capture log files of the activities in BOSS in order to visualize and to verify timings automatically (still in work).

5 Macro structure



BOSS is structured in layers from hardware up to the final application, each providing a virtual view of the lower layer. This virtual view is always the same even if the lowest layers are substituted. The bottom layer is the target hardware CPU, IO devices and other Hardware units or even a LINUX-platform. The bottom layer is for the kernel and applications transparent and can be changed very easily. The second layer is the only hardware dependent layer. This layer implements the functionality which is different on each platform, e.g. load/store of CPU registers, low level Hardware-Drivers and basic interrupt management.

To move from one platform to another, only this layer has to be rewritten. The third layer is the kernel, it implements the interface to the applications and manages threads, time and resources.

6 BOSS Kernel

The Kernel is so simple that it can be explained in one page. The basic class is Thread. Threads are executable objects with context, stack and own data. They can be executed (run), suspended, reactivated (resumed) and they react to time, and to internal and external events.

The basic operation is lists-management. All resources are managed (sorted) in chained lists. No element can be in two lists at the same time. Before inserting an element in any list, it will be removed from any other list. The kernel lists are:
Ready list: list of threads which are ready to use the CPU. It is sorted by priorities. The first thread in the list has the CPU - is running now. All others are waiting. If the list is empty the idle-thread gets the control and consumes the free CPU time.
Timer list, IOLists: list of threads which are waiting for a time point, a time event or an I/O event/interrupt/signal. It is sorted by time or by priority. The next Thread to be resumed is at the front of the list.
Semaphores: list of threads sorted by priority. Semaphores are used to implement monitors, to protect exclusive sections and to implement synchronisation.
Messages and communication lists: messages from one thread to other are inserted in a list of messages sorted by priorities or time. A thread attempting to read from an empty Messagebox will be suspended.

7 Literatur

[BIRD0] www.dlr.de/bird
[BIRD1] www.first.fhg.de/~sergio/public/IAA2003_BIRD_Technol.html
[MONT00] http://www.first.gmd.de/~sergio/public/bird-iaaa.html
[MONT03] http://www.sciencedirect.com/, Sergio Montenegro, Wolfgang Baerwald, PowerBird - Modern Spacecracrt Bus Controller, Acta Astronoautica 52 (2003) 957-963
[MONT99] S. Montenegro, Sichere und Fehlertolerante Steuerungen , Hanser Verlag, Sept. 1999, ISBN: 3-446-21235-3
[ROD03] Rodionow, Montenegro, Behr, BOSS for Hyperspectral Analyse (IAA 2003), http://www.first.fhg.de/~sergio/public/iaa2003.pdf
[SPIN] Spin model checker netlib.bell-labs.com/netlib/spin/whatispin.html
[VERIFUN] www.inferenzsysteme.informatik.tu-darmstadt.de/verifun
[ISABELLE] Isabelle Theorem Prover: http://isabelle.in.tum.de/