Integration of Software and Hardware Building Blocks for space applications

Sergio Montenegro
University Würzburg Am Hubland, 97074 Würzburg

ABSTRACT: Two of the biggest problems by building complex embedded systems like spacecraft are the integration of software/hardware and the management of the increasing complexity. To handle the complexity we build complex systems as a network of (more) simple independent components. In our approach we aim to manage both, software and hardware components, in the same way and to build a network of services implemented in Software and Hardware.
KEYWORDS: Building blocks, Middleware, complexity, RODOS, Real time Kernel.

Introduction / Motivation

Two of the biggest problems by building complex embedded systems like spacecraft are the integration of software/hardware and the management of the increasing complexity. The best known method today to handle the complexity is to build complex systems as a network of (more) simple independent components. Normally this method is used independently for software and for hardware components. In our approach we aim to manage both, software and hardware components, in the same way and to build a network of services. Some services will be produced in software and others in hardware. The services are distributed in the (Software/Hardware) network from producers (publishers) to consumers (subscribers). This does not depend on if the services are produced by software components or by hardware components. The same applies to the consumer of services.

But to handle the complexity is not enough. Firstly we aim to reduce the complexity to the minimum possible. The best method to manage complexity is to eliminate it, and then to handle only the rest, what we can not eliminate without losing functionality. This requires some willingness to compromises, to accept some looses of performance and to dispense with not absolutely required functionality. The engineering paradigm is already changing. A few years ago the most important aspect was to get higher and higher performance, the increase of complexity was not a theme. Today the most important aspect is to handle the complexity and better to reduce it even if some performance will be lost.

In our approach we handle both main problems – hardware/software integration and complexity management – with one single solution: an unified execution environment for software and hardware building blocks. For the development of our execution environment for building blocks (RODOS) our first goal was to find and implement (only) the most simple possible solution (and we think we did it).

1. Reducing complexity

The complexity of the avionic/electronic has already reached a critical value. The automotive industry is already fighting the complexity trying to reduce it. The space industry is now beginning to realize that the complexity has to be reduced. To manage it, is not enough any more, we do have to reduce it.

The main obstacle to reduce complexity (and produce simplicity) is the missing willingness and determinedness of the developers. One have to make the commitment to do it, because it is not so easy like it seems from the first glance.

Complex systems are easy to thing but very difficult to build and to keep running. Simple solutions on the other side, are very difficult to find but then they are simple to build and to keep running. The Problem is the first step: To find the “simplest” solution. It may take very long time a one have to think a lot, discard other (nice) solutions and begin from zero again.

Most of our system components were implemented three times or even more, searching for more simple solutions until we thing “simpler is not possible”. This is a difficult and expensive commitment but we can show why it is necessary to do so.

The first problem is: we thing we can control the complexity, “we have created it and therefore we can control it”. The true is shown in figure 1, we did create it and we feed it and it grew and grew and is still growing, now we have lost control.

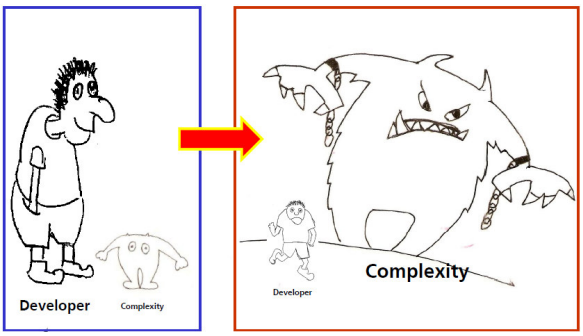

Figure 1: the Growth of complexity

Second: we aim to create dependable systems, but complexity is the source of most (development) errors and failures. Complex systems can fail in many ways, complexity hides (bad) surprises (difficult to see, too complex to have an overview) and simple systems can cope better with uncertainty.

Let's analyze fragility: Fragility grows proportional to complexity. Complex systems have many components and ways to fail, this increases the fragility of complex systems. Fragility grows proportional to uncertainty too. Systems behave robust in total deterministic environment which we know and have tested. If the circumstances are always the same and there are no surprises, we can assume we will get (hopefully) no surprises from our system. In a unknown and indeterministic environment we shall expect more (bad) surprises from our system. As shown in Figure 2 lets say:

Fragility = complexity * uncertainty (- adaptability + other factors)

Fragility = Complexity * Uncertainty

Figure 2: Fragility = complexity * uncertainty

If this is true, then to increase robustness (reduce fragility) we have to reduce complexity or uncertainty. We can not reduce uncertainty if we what to reach some thing new like for example in space exploration, but we can reduce complexity … if we really want to. (figure 3)

Figure 3: A compromises to do, to reduce fragility

Thirdly: Complexity has to limits, the lower limit is the irreducible complexity. For every functionality there is a limit where we can not do it simpler without losing the required functionality. The upper limit: if the fragility grows with the complexity, then there is an upper limit where the fragility reaches its Maximum, where the system can not be controlled any more, it may not work properly any more and it collapses (See figure 4). This point is the critical complexity.

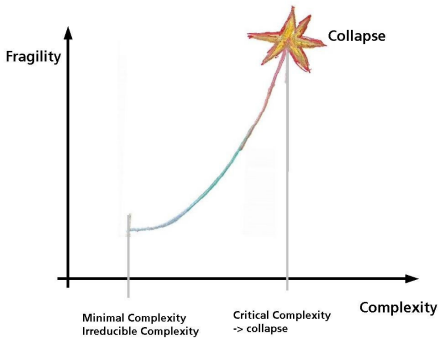

Figure 4: Two limits for complexity

Any functioning system is in between these two limits. The distance to the critical complexity is the margin we have before our system collapses. The distances to the irreducible complexity are self made problems. Any (normal) development begins in between these limits and keeps growing towards the critical complexity until a redesign is performed. It is true, new tools and methods allows us to shift the positions of the critical complexity, but this has a limit too and going closer to this limit is not a good idea. We aim to go in the other direction. We invested a lot of effort (and money) to reach the irreducible complexity for our building blocks execution platform RODOS.

It is worth to invest in simplicity. We did.

2. Building blocks execution platform for Software and Hardware

The main risk factors in a typical core avionics development are the complexity (already handled), software-hardware interfaces and the difficulties to handle many different interfaces in a single system. Our new core avionics concept targets these problems and aims to provide a very simple integrated solution of software and hardware. The border between both shall vanish. There is only one interface (middleware interface) which is used by all building blocks including hardware components.

The most effective and safe way to implement a complex parallel system is to compose it as a network of simple tasks. These tasks may be executed by software like for example steering control; or by hardware components like for example providing temperature measurements; or by FPGA programs for example FFT or access to simple devices. 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).

In our approach, 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 RODOS as a software middleware for the software tasks and in a FPGA as a middleware switch for hardware devices and to interconnect computing nodes (an ASIC implementation of the network is in work).

2.1. Software Control on RODOS

RODOS (Real Time On board Dependable Operating System) is an open source building block execution platform/environment designed for space applications and for applications demanding high dependability. Simplicity is our main strategy for achieving dependability, as complexity is the cause of most development faults. The system was developed in C++, using an object-oriented

framework simple enough to be understood and applied in several application domains. Although targeting minimal complexity, no fundamental functionality is missing, as its micro-kernel provides support for resource management, thread synchronization and communication, input/output and interrupts management. The system is fully preemptive and uses priority-based scheduling and round robin for same priority threads. On the top of this kernel the RODOS middleware distributes messages locally and using gateways globally.

The RODOS execution platform provides a (software) interconnection network between applications / building blocks (the middleware). A building block requires some services (incoming messages) in order to be able to provide other services (outgoing messages). The execution platform distributes such services (messages) from producer to consumers. (see Figure 5).

RODOS may be executed on the top of other operating systems or TSP (Time Space partitioning systems) or directly on the hardware in case no other operating system is running on the target hardware. In all cases the interfaces to the building blocks (or applications) remains the same, and a network of applications may be executed on different platforms and operating systems without modifications.

RODOS provides a middleware which carries out transparent communications between applications and computing nodes. All communications in the system are based on the publisher/subscriber protocol.

Publishers make messages public under a given topic. Subscribers (zero, one or more) to a given topic get all messages which are published under this topic. To establish a transfer path, both the publisher and the subscriber must share the same topic. A topic is represented by a topic ID and a data type.

On the software side the middleware implements an array of topics which may be compared to hardware buses. Each time a message is published under a given topic, the middleware checks for all subscribers that wish to receive it. Each one will receive a copy of its content. To go beyond the limits of the computing node we use gateways which may read all topics and forward them to the network and vice versa (Figure 6)

Using this approach, no fixed communication paths are established and the system can be reconfigured easily at run-time. For instance, several replicas of the same software can run in different nodes and publish the result using the same topic, without knowing each other. A voter may subscribe to that topic and vote on the correct result. The core of the middleware distributes messages only locally, but using the integrated gateways to the "NetworkCentric" network, messages can reach any node and application in the network. The communication in the whole system includes software applications, computing nodes and IO devices. (see Figure 6).

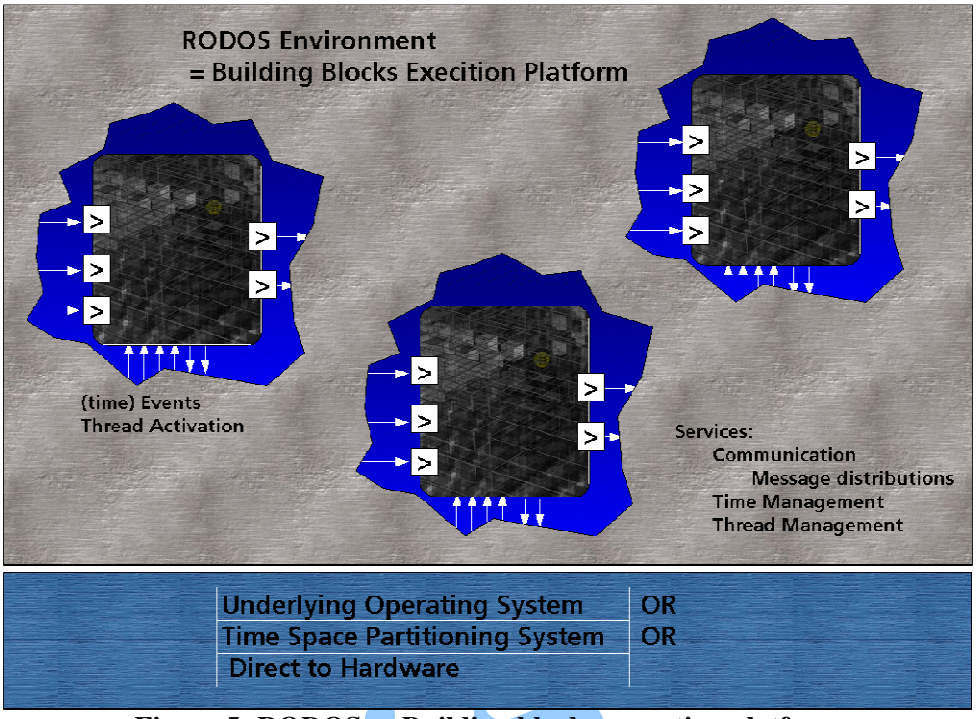

Figure 5: RODOS as Building blocks execution platform

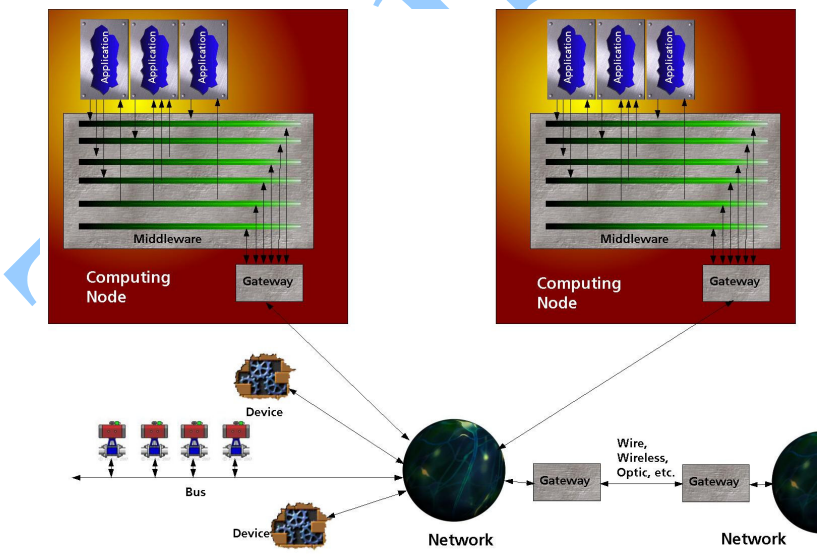

Figure 6: Network of software tasks and hardware devices

This method is used to interconnect service providers and consumers including hardware and software. All our devices and software components

provide this interface. In order to be able to attach COTS (Commercial Off The Shelf) devices (with their own protocols) to the network, the network provides the required interfaces and protocol converters. The COTS devices receive and send their own messages, and then the protocol converters translate them into our internal "universal language". The network will perform all required transformations in order to make the message transport transparent.

2.2. Software Protocol and Hardware Network

Our interconnection link defines a very simple message distribution protocol. Each message in the network receives a topic tag which identifies its content and purpose. Examples of topics could be: position, temperature, and attitude. Services will be published as topics regardless of whether they are produced by software tasks or by hardware devices. Any application or device may subscribe to the topics which it needs to help to produce its services.

The current FPGA implementation of the network supports different own and COTS protocols. The next generation (being building now) will be based on an ASIC switch, which can support only two different physical links: our own high speed "Simple Synchronous Serial Protocol" (S3P) and simple UARTS. To access other low level protocols (e.g. CAN, SPI, I2C, SpaceWire) we aim to develop bridges and remote terminal controllers (RTU).

3. Experience from former Projects

Our concepts are based on our experience from former satellites we have build and from new spacecrafts which are being developed now.

BIRD is a German small satellite. It was built to observe terrestrial fires via infrared spectroscopy and was launched in 2001. TET is similar to BIRD in design and its mission is to validate new space technologies. The launch of the TET satellite is scheduled to take place in 2010. In both cases the central element in their architecture is the board computer which has to provide computing power, storage (mass memory), input/output interfaces for many (> 20) data buses and devices. The central element has to be dependable and robust. It has to manage the redundancy and, additionally, it has to execute its real-time job. Therefore such a computer system becomes very complex, expensive and susceptible and is not reusable in different missions with different input/output devices. Even TET's similarity to BIRD required a new development of the board computer, because a few of the devices needed to be substituted.

The BIRD and TET main computers were implemented using four computers. BIRD is totally based in COTS technology, TET has both Rad-Hard and COTS. One of the 4 nodes (the so called worker node) controls the satellite, while a second node (monitor node) monitors the correct operation of the worker node. In case the worker fails, the monitor takes over the job of the worker while the crashed node performs a recovery cycle.

Two other node computers are spare components and are disconnected. In case of a permanent failure of one running computers, the spare computer pair will be used instead of the failed pair.

Both BIRD and TET are only a small step beyond state of the art. They have more redundancy than is normal because they are based on COTS (not space-qualified) components. The redundancy management is performed in software. A lot of functionality which normally is implemented in hardware, in BIRD and TET is implemented in software and takes advantage from the higher redundancy and redundancy management.

We are currently developing, for a series of satellites, a high performance generic board computer (called SSMMU) based on FPGAs. These board computers may be used to control the satellite and at the same time to process and store the payload data, but the central element will not be the computer any more, but the network, which can interconnect many buses and devices, including computers and mass memory devices. In this architecture the computer is not a central element, therefore it may be very simple, cheaper and have lower dependability requirements.

The entire functionality of the avionic system will be obtained by plugging hardware and software components together. Each of these components shall be replaceable without having to modify the rest of the system. To guarantee dependability, the only component which shall be dependable is the network. Our technology research is concentrated on the network which will include intrinsic redundancy and will be built using European radiation resistant components. The redundancy management will again be executed through software.

The Network Centric approach provides much higher dependability, flexibility, scalability and performance. The current implementations of the network relies on commercial high-capacity reprogrammable gate arrays (FPGAs). We are currently developing, together with a microelectronic partner IHP (Innovations for High Performance Microelectronics, http://www.ihp-microelectronics.com), a 32-Ports middleware Switch ASIC using their submicron radiation tolerant technology (0.25 micron). 16 of such ports are simple UARTS and 16 implement our high speed simple serial link S3P.

References

[BM01] **W. Bärwald, S. Montenegro –** *BIRD - Spacecraft bus controller*, Small Satellites for Earth Observation, Vol. 3, 371-373, 2001

[Mon09] **S. Montenegro** – *Network Centric Core Avionics for Dependable Systems*, CEAS 2009: Air and Space Conference, Europe, October 26-29, 2009 - Manchester, UK

[MD09] **S. Montenegro, F. Danemann -** *RODOS: Real Time Kernel Design for Dependability*; DASIA 2009 DAta Systems In Aerospace 26 to 29 May 2009, Istanbul

[MH09] **S. Montenegro, E. Haririan –** *Fault-Tolerant Middleware Switch for Space Applications*, IEEE Computer Society Washington, DC, USA 2009, SMC-IT, Proceedings of the Third IEEE International Conference on Space Mission Challenges for Information Technology - ISBN:978-0-7695-3637-8

[M+08a] **S. Montenegro, J. Grundmann, B. Kazeminejad, P. Spietz –** *The new DLR Standard Satellite Bus series (SSB)*, Small Satellites Systems and Services - The 4S Symposium, German Aerospace Center, 2008

[M+08b] **S. Mottola, A. Börner, J. Grundmann, G. Hahn, G., B. Kazeminejad, E. Kührt, H. Michaelis, S. Montenegro, N. Schmitz, P. Spietz –** *AsteroidFinder: Unveiling the Population of Inner Earth Objects*, 59th International aeronautical congress, 29th September to 3rd October 2008, Glasgow, Scotland.

[BIRD] **BIRD** (Bispectral InfraRed-Detector http://www.dlr.de/rb/en/desktopdefault.aspx/tabid-2731/6724_read-6311

[COM] **Complexity measurement and management** http://www.ontonix.com/

[TET] **Technology Experiments Carrier** http://www.dlr.de/rd/en/desktopdefault.aspx/tabid-2274/3396_read-5085