RODOS operating system for Network Centric Core Avionics

Dr. Sergio Montenegro & John Richardson
Core Avionics Group
DLR – German Aerospace Agency
Bremen, Germany
sergio.montenegro@dlr.de

Abstract—*Network centric core avionics attempts to solve the question of simplicity and dependability in computing by means of a fault-tolerant and robust architecture called middleware. By this means software services can be distributed via nodes which are non-dependable and which can migrate in the case of node failure, thus creating a reliable network of unreliable parts.*

Proposals*-With the network centric approach the aim is to try a new paradigm of dependable computing for space applications. Instead of having a powerful all-embracing computer to the centre of the design, there is instead a powerful network which connects many undependable and simple units including computers, mass memory and I/O devices.*

Fault tolerance shall be supported by turning devices and computers on and off and turning equivalent functionality on. Only the network shall remain dependable.

Results*-The corresponding operating system and middleware (RODOS) and software simulations of the network, computers and some devices have been implemented. Using this simulator the fault tolerance can be shown by turning simulated computers and devices off and the system will stay operable in real time.*

Currently a physical demonstrator is being implemented which will be called "Levitator". The Levitator will have the ability to make a piece of metal float in a 3-dimensional space in real time. Even if there is a complete failure of the computer or peripheral devices the piece of metal shall continue to float and move smoothly.

Keywords-component; middleware, RODOS, NetworkCentric

I. INTRODUCTION

Dependability is a main issue for space applications but after more than 30 years of research into how to achieve dependable computing, the general solution has still not been found. There are many proposals as to how to achieve fault tolerance, robustness, or fault prevention etc., but not a single solution is globally accepted.

The main risk factors in a typical core avionics development are the complexity, software-hardware interfaces and the difficulties in handling many different interfaces in a single system. These topics shall be addressed in order to achieve high dependability.

Simple devices have one fixed topic to publish messages and one fixed topic to subscribe messages. More complex devices, however, may subscribe to a set of fixed topics which will be written in the topic list of the corresponding output port. Computing nodes may have a variable list of topics, because the subscribed topics depend on the software which is currently running. Such synapses may receive from the corresponding input port new lists of topics to update dynamically the corresponding subscriber list of the output port, at any time. The current middleware switch implementations are FPGA-based without CPUs. Using (internal) CPUs more intelligent middleware switches may be implemented which provide higher functionality, network routing, automatic link interruptions detection and, for example, may collect periodically all subscriber lists to inform computing nodes which topics are expected by any attached nodes.

The NetworkCentric protocol relies on the exchange of messages asynchronously using a publisher / subscriber protocol. This provides a very high flexibility and users do not have to differentiate between local/remote communication and between any combination of software/hardware device communication.

Publishers make messages public under a given topic. Zero, one or more subscribers to a given topic receive all of the messages which are published under this topic. To establish a transfer path, both the publisher and subscriber have to share the same topic. Both components - software middleware and network switch (called the middleware-switch) interpret the same publisher/subscriber protocol in the same way.

Communication relationships can be very dynamic. Units may disappear or appear (turned off/on), and software tasks may be migrated, activated or deactivated at any time. The position of publishers and subscribers can even change (migration) at runtime, without requiring any explicit reaction of the other involved communication partner. There are no fixed communication paths, and each data transfer is resolved just in time using the registered communication topics. Topics are global communication channels in the whole system, covering software applications and hardware devices, without making any difference of node / hardware boundaries.

Typical data systems for space applications are computer centric. The central component is a computer to which all or most devices are attached. This computer is responsible for handling devices, communication, computing, and storage of data.

The aim is to create a new concept of core avionics systems which targets fault tolerance as a natural part of the concept. In this approach the central component is not the computer but a distributed fault tolerant network system. This provides dependability to the network to which a set of independent redundant components can be attached such as devices, simple computing units, and mass memory units, etc. Any of these devices have the possibility to fail in which case the network manager will deactivate the failed device and replace it with an activated, redundant one providing the same services.

The most effective and safe way to implement a complex parallel system is to compose it as a network of simple sequential tasks. These tasks may be executed by software, such as steering control, or by hardware components such as those providing temperature measurements. The aim is to unify software and hardware so there shall be no difference between services provided either by software or by hardware. All service providers communicate using the same communication protocol and unified messages and use the same interface. When using a service there shall be no difference between how it was implemented (software, hardware, or both) and where it is being executed (in which computing node or device).

different units and converts them to the internal NetworkCentric protocol.

Different nodes, including computing nodes, mass memory nodes, and devices may be attached to the network using at least one link. The network switching points are implemented by the Middleware Switch. The connection links to the network may be redundant and in this case they will be treated by the network as different independent connections. Links may be of different types such as RS422, MIL-1553, CAN, SpaceWire, SPI, I2C , etc. Each interface from the network to a link is called a Synapse and each synapse has to be adapted to the corresponding link in order to support the corresponding protocol. The Synapses translate the external link protocols to the NetworkCentric publisher/subscriber protocol which is the “common universal language” for all units in the system, including software tasks and I/O devices. The Software middleware in RODOS (the NetworkCentric Operating System) directly supports this protocol. For all other units, translators have to be provided, and this is the function of the synapses. All synapses have the same structure but a different implementation according to the attached inner network centric protocol.

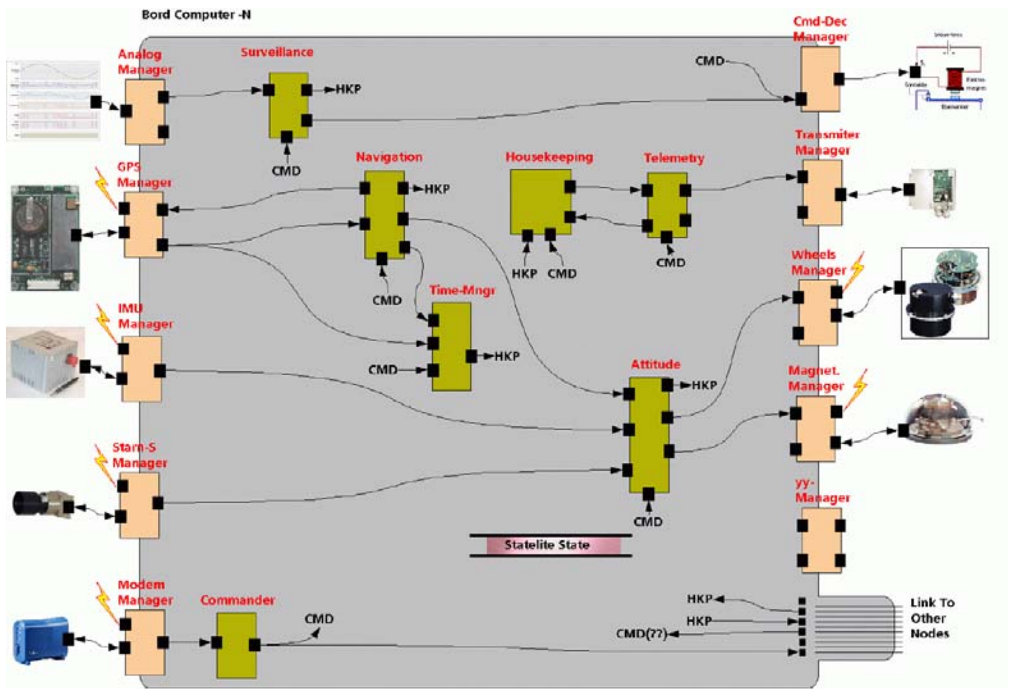

Figure 1: Communicating Applications

The NetworkCentric core avionics machine consists of several harmonised components which work together to implement dependable computing in a simple way. The network is built using basic building blocks called middleware switches which implement different protocols to

II. The First step toward dependable computing

The first step to designing dependability for space computers was first used in the (DLR-) BIRD satellite. In this architecture there are two or four redundant control computers, each of the nodes capable of executing all of the control tasks. One node (the worker) controls the satellite while a second node (supervisor) supervises the correct operation of the worker node. If an anomaly of the worker node is detected by the supervisor node, the supervisor takes over the control of the satellite and becomes the new worker node. The old worker node is enforced to execute a recovery function and if there is no permanent error detected, it then becomes the supervisor node.

The core avionics system becomes a distributed computer system through which no single node is required to be dependable. The nodes are instead connected through a dependable network which is the heart of the system. Software services can then be distributed on all computer nodes and may migrate from one node to another in the 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 a software middleware for the software tasks, and in a FPGA as a middleware switch for hardware devices and to interconnect computing nodes and mass memories.

III.INTEGRATED SOFTWARE AND HARDWARE STRUCTURES

The next step to improving this structure was a software-only step. While the hardware structure stayed the same, the software structure was improved by adding a middleware for communication purposes. Instead of having many different interfaces among applications or between application and I/O-drivers, there is only one interface for all communications in the system. The middleware provides a message-interface which can be used to interchange data among all entities in the system. Therefore there is no extra I/O- driver interface. I/O-devices are controlled by applications called I/O-managers.

Another improvement is the inter-node communication. The functionality of the system is implemented as a network of applications which can be distributed among many computers in the system (Figure 1).

Computing units (CPU +MEM) are managed by the local real-time kernel operating system (OS) RODOS. On top of the kernel runs the software middleware (MW) of RODOS and around this middleware the user can implement its applications (AP). To communicate with external units, including devices and other computing units, each node provides a gateway to the network and around the network several devices (IO Devs and computing nodes) may be attached to the system.

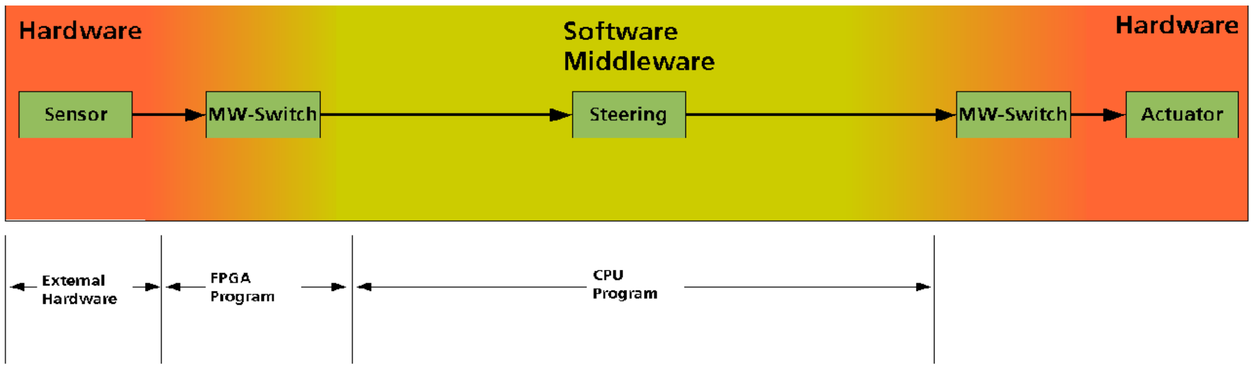

Figure 3: Merging software and hardware in the Middleware

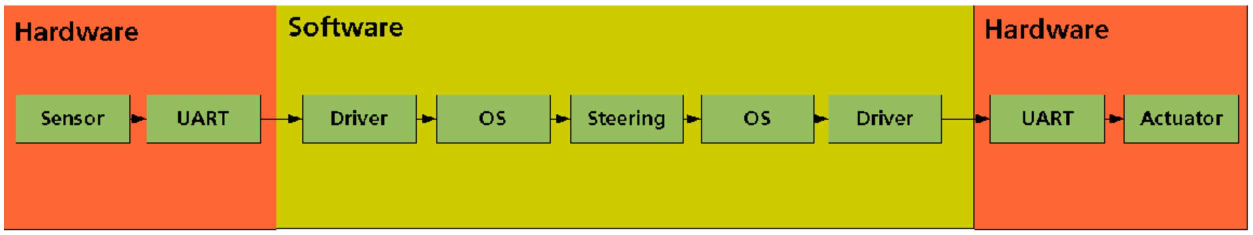

Figure 2: Typical data/control flow from devices to applications

IV. The Middleware Switch

A. *Middleware architecture*

The next step is to unify software and hardware into an integrated architecture. Figure 2 shows a typical data/control flow to access I/O-devices.

The capabilities of the FPGA (programmable hardware) emerging technology allows the implementation of middleware functionality directly into the hardware I/O-interface of a structure such as that depicted in Figure 3. The intention is to implement the middleware in the form of an Application Specific Integrated Circuit (ASIC).

The I/O interface (typically a UART) will have on one side the required device interface while on the other side it will be directly integrated into the middleware protocol. In this way the structure in figure 2 can be extended to simulate the structure depicted in figure 4.

An embedded controller in the middleware switch recognizes communication requests from the I/O ports and connects/disconnects the ports accordingly. For cost-sensitive applications an investigation will be carried out into how the hardware management of I/O links can be done automatically without the need of software-controlled embedded processing resources.

V. RODOS

The RODOS real-time kernel and middleware provide an integrated Object-oriented (OO) framework interface to multitasking resources management and to the NetworkCentric communication infrastructure. The RODOS framework seeks to offer the simplest and smallest possible interface to user applications, while still providing all the required functionality and flexibility. It includes time management, CPU and memory management.

Simplicity does not mean, however, lack of functionality. Real time scheduling, resource management, synchronization, middleware and simple communication and all the functions expected from a microkernel are implemented – just as simply as possible. An important RODOS design target is the irreducible complexity; this is the minimal possible complexity for a determined function - when it becomes no longer possible to implement it simpler without destroying the core functionality.

RODOS is based on utilising very few and simple basic functions. Applications running on top of RODOS are implemented using object-oriented technology, resulting in highly modular application software. Applications running on the top of the RODOS middleware are built using the schema of software building blocks. Several (simple)

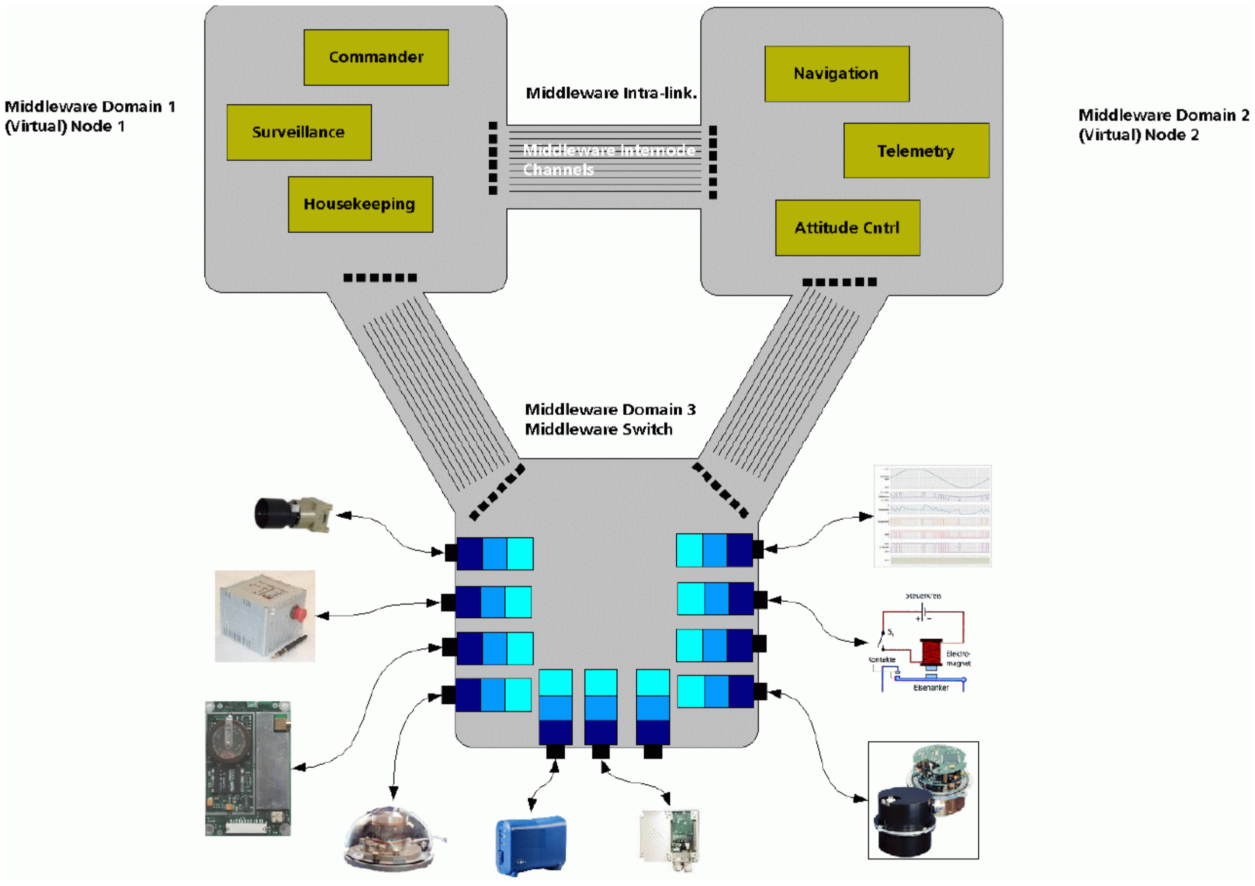

Figure 4: I/O-interfaces integrated in the Middleware

building blocks (called applications) can be distributed and interconnected in a computer+devices network using the NetworkCentric core avionics protocols thus building more complex functionality. Building blocks can be implemented and tested independently of each other and can be interchanged without having to modify other blocks or interfaces.

RODOS can be executed on embedded target hardware and on top of Linux. Applications can be moved from one host to another without modifications. 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.

Applications may be implemented by creating active and passive objects. Active objects may get CPU-time from the underlying core as reaction to time, to events, to message distribution and to requests from the object itself. To create an active object the user just needs to inherit and instantiate from the interface classes Thread, Event and/or instantiate Subscriber (Middleware Interface) objects.

The central element in the core is the time. The time begins at 0 (boot time) and increments continuously in nanosecond steps until “End-Of-Time” is reached which is about 150 years in the future. This time control controls almost all activities in the core.

Many threads may run (apparently) simultaneously. Each thread may run until it is suspended for a time period by itself (typically) or by another thread (not usual). To suspend a thread one may call explicitly the suspend() method, or access a synchronised object which may suspend its caller. Examples are entering a semaphore, reading from synchronised fifos, waiting for Messages, etc. Such synchronised objects just call the suspend() method of the caller if it has to be blocked.

To build a complex functionality and even many different functionalities in the same system, which is typical in satellite systems, it is advisable to encapsulate simple and clear-cut related functions in building blocks – here called applications – and to plug such blocks together thereby building a network of applications/building blocks (not to be confused with the hardware network). Adding devices to the network we get a network of services.

Applications (or building blocks) may encapsulate threads, messages, event handlers, passive objects etc. From outside it shall not be visible to the innermost being of the application. The only interfaces to an application are messages which can be distributed/subscribed. Each application shall provide *one* specific service to the system.

A services network may be extended (or modified) to provide more functionality without having to modify the individual building blocks. For example we may add instrumentation for debugging and a data logger by just adding new building blocks to the network.

To build fault tolerance, both producers and consumers of services may be replicated. For the programmer there is no difference if one or more replicas are working in the same network. To merge outputs from several sources we need voters which may intercept messages, select the most probable correct one and forward it to the device. Even voters may be replicated.

CONCLUSIONS

This paper describes a minimalitic operating system and middleware which has been developed to support the network centric protocols which are the most simple implementation of the publisher / subscriber protocol imaginable. This protocols are similar to the DDS protocols but much simpler.

The fault tolerant capability of the system in real-time simulations , as well as the interface to programmers, have been shown to be very simple and easy to use. The levitator demonstrator is currently in development.

RELATED WORK

This work was inspired by the DDS protocol which implements a similar functionality in the most simple possible way to be used in very small embedded systems and to be ported to different systems. Another inspiration is the AFDX, a Real-Time protocol for Ethernet.

REFERENCES

[1] Sergio Montenegro and Lutz Dittrich, Network Centric Core Avionics,7th Symposium on Small Satellites for Earth Observation, May 04 - 08, 2009, Berlin, Germany

[2] Sergio Montenegro and Frank Dannemann, The software architecture for TET and AsteroidFinder satellites, 7th Symposium on Small Satellites for Earth Observation, May 04 - 08, 2009, Berlin, Germany

[3] Francisco Afonso, Carlos Silva, Adriano Tavares, and Sergio Montenegro, Application-Level Fault Tolerance in Real-Time Embedded Systems, Third International Symposium on Industrial Embedded Systems, 11 - 13 June 2008, Montpellier France

[4] Francisco Afonso, Carlos Silva, Nuno Brito, Sergio Montenegro, and Adriano Tavares, Aspect-Oriented Fault Tolerance for Real-Time Embedded Systems, 7th Workshop on Aspects, Components, and Patterns for Infrastructure Software (ACP4IS '08), 31st March 2008, Brussels, Belgium

[5] Sergio Montenegro and Lutz Dittrich, The Core Avionics System for the DLR Compact-Satellite Series, Small Satellites Systems and Services The 4S Symposium, 26 - 30 May 2008, Rhodes, Greece

[6] Sergio Montenegro and Raffaele Vitulli, An unusual Approach to Dependability for Space SWApplications, DASIA 2007 DAta Systems In A erospace EUROSPACE, 29th May - 1st June 2007, Naples, Italy

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.