Visionary data management system for nano satellites (VIDANA)

Sergio Montenegro, Thomas Walter
sergio.montenegro@uni-wuerzburg.de | thomas.walter@uni-wuerzburg.de
Tel 0931/31 83715
Universität Würzburg
Am Hubland
97074 Würzburg
Germany

1. Introduction

Nano satellites and nano spacecraft are becoming more and more important for orbital and interplanetary missions. The expectations to such vehicles increases very fast. They shall be autonomous, self adaptable to different situations, provide self healing and many self-x's. Such vehicles shall be able to operate in swarm, cooperate for docking and auto assembly, and many other expectations. All this relays primary on the capabilities of the data management system (DMS). VIDANA aims to provide high dependable and high performance computing systems using very limited physical resources. We aim to exceed current data management capabilities using very simple (COTS) components.

Another challenge is the variety of different target missions. Developing a DMS which is applicable only for one missions would not be the best solution. A DMS which is good for every one ("one size fits all") would be too large in many cases and under-sized in other (almost never the right size). Our DMS should therefore be adapted with minimal effort to the specific mission (tailoring).

At this point, let's differentiate between static adaptability which is performed by the developer for each mission (tailoring) and dynamic adaptability which is performed autonomously by the spacecraft when circumstances change. Both are targets of VIDANA.

2. A VIDANA Data Management System

A VIDANA data management system is a network of software and hardware components. This implies a software network, a hardware network and a smooth connection between both of them. Our strategy is based on our innovative middleware. A reliable interconnection network (SW & HW) can interconnect many unreliable redundant components such as sensors, actuators, communication devices, computers, and storage elements. Component failures are detected, the affected device is disabled and its function is taken over by a redundant component. Our middleware doesn't connect only software, but also devices and software together. Software and hardware communicate with each other without having to distinguish which functions are in software and which are implemented in hardware. Components may be turned on and off at any time, and the whole system will autonomously adapt to its new configuration in order to continue fulfilling its task.

Our targets are:

1. Dynamic adaptability, dynamic scalability for dependability and for speed: At any point in time we have a list of required tasks and redundancy, and a list of available resources (computers, power, IO). The current resources will be dynamically allocated to tasks and redundancy to best fulfil the current mission requirements. The same resources can be used to achieve speed or to achieve dependability therefore when one increases the other decreases. eg: low speed high dependability: for critical operations like docking; High speed with lower dependability for example for video analysis; low power (-> low speed, low dependability) for example for the cruise phases. The same mechanisms can be used to migrate software tasks among hardware components. This properties will be used to implement fault tolerance (self healing) and load balancing.

2. Static adaptability (tailoring) for different missions: highly (static) scalability without structural limits,

down scalable to pico satellite boundaries. We take exemplar solution from nature for adaptability, communication strategies, distribution of services/capabilities and fault tolerance. VIDANA provides a building blocks architecture for both software an hardware and means to interconnect systems of different sizes. The systems is based on a service oriented architecture (for space applications)

3. Unified communications protocols for software and hardware. In this way services may be produces and consumed by hardware or software. For the producers (publishers) and consumer (subscribers) there is no difference if the communication partner is implemented in software or in hardware. This gives us a very flexible building blocks box which makes the tailoring tasks very simple.

4. The same communication protocols will be used inside (intra) and outside (inter) the space craft. This allow us to distribute services among different communicating spacecraft with no extra software effort. This is a basic support for cluster, swarms and assembly tasks. The same applies for the space segment <-> ground segment communication. This allows us to distribute tasks among space and ground and simplifies in an incredible way the commanding and telemetry tasks.

All these properties are focused on the requirements of nano and mini satellites, but can be used down to pico satellites and up to very big systems. Our software and hardware are absolutely ITAR-free.

3. Implementation

3.1. Software

Our strategy to implement complex software (control) systems is to decompose the system in simple communicating building blocks (BB). The whole application is then executed by a network of simple BBs. Software BBs are similar to Hardware chips: Only the interfaces (pins) and their behaviour

has to be known to interconnect them. They can be seen as ports (like in hardware-pins), on which the BB expect or publish services in terms of messages (see Figure 1). Note: the same concept is used in hardware since years and only by that modern complex hardware-systems could be realized.

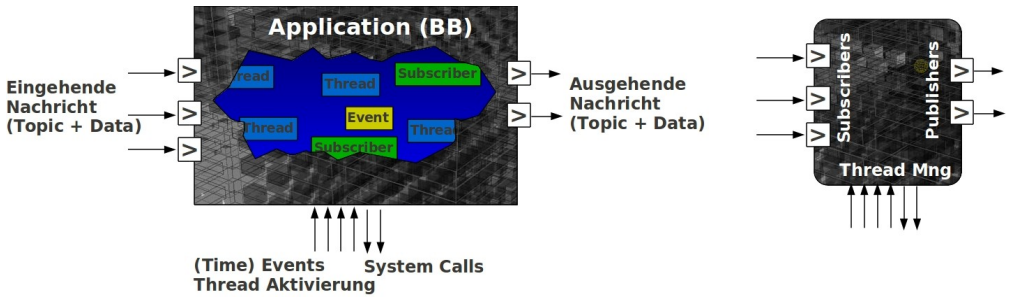


Figure 1: Software Building Blocks / Components

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 synchronisation and communication, input/output and interrupts management. The system is fully pre emptive and uses priority-based scheduling and round robin for same priority threads. On the top of this kernel the RODOS middleware distributes messages globally

using gateways and locally.

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 2).

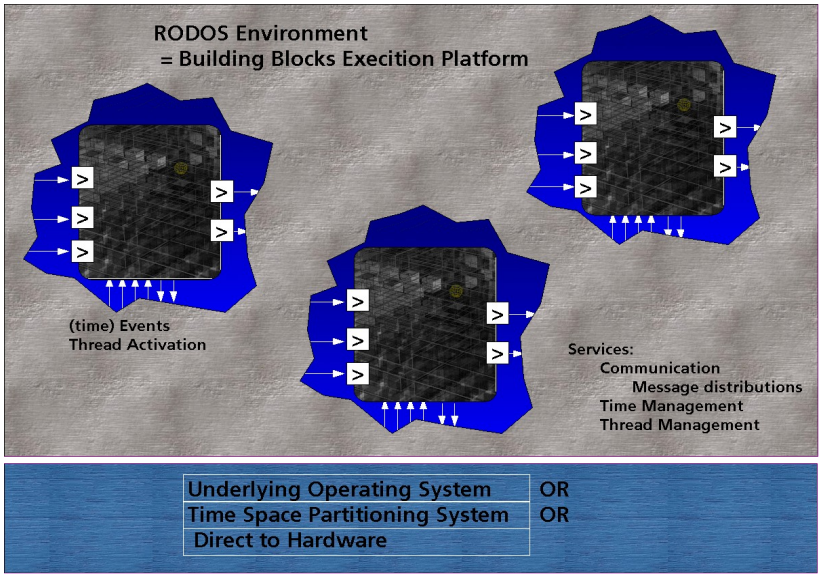


Figure 2: RODOS as Building blocks execution platform

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 busses. 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).

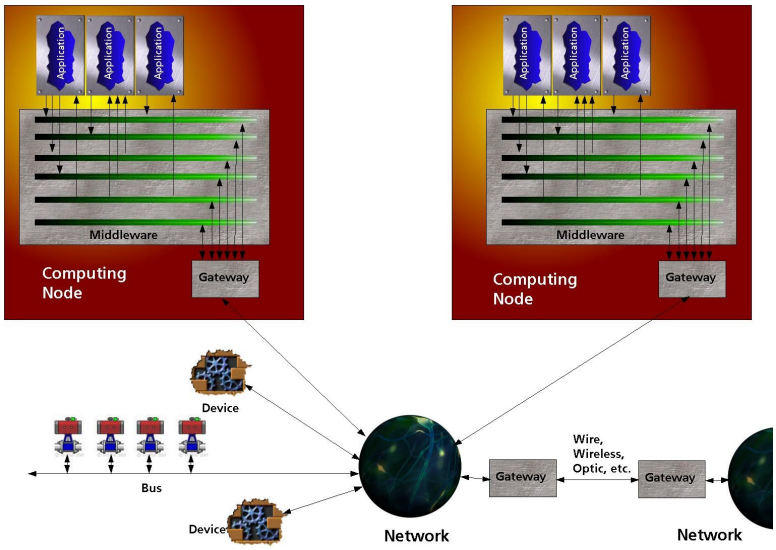


Figure 3: Network of software tasks and hardware devices

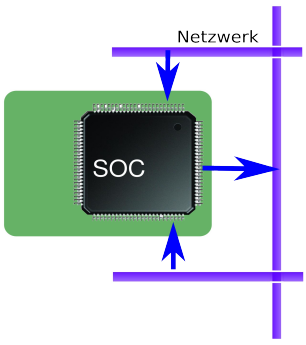
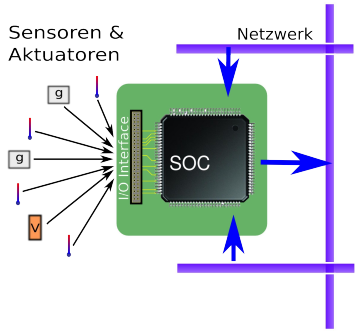
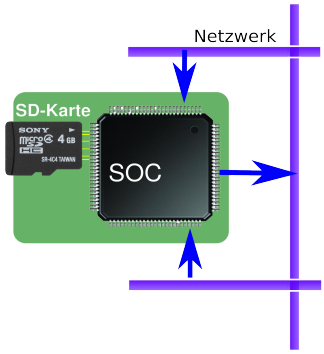
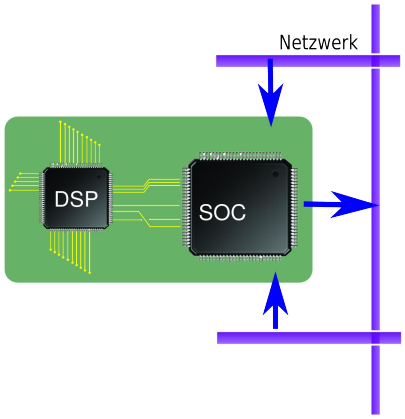
This method is used to interconnect service providers and consumers including hardware and software. All our devices and software components provide this interface. In order to be able to attach COTS (Commercial Off The Shelf) devices (with their own protocols) to the network, the network provides the 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.

3.2. Hardware

A VIDANA computer is a network of (very) small computer nodes. We concentrate our efforts on the communication system which includes software and hardware components. Each node has an intelligent interface to the network, which is able to understand and execute the software protocols in the system. Attached to the communication interface we have a node core which is a micro controller with different IO interfaces, which will be differently used in different nodes or missions. Attached to the node core we have some specialisations like flash mass memory, number crunchers accelerators (DPS) and special IO devices to implement "intelligent" sensors and actuators.

Nodes attached to the network can be turned off and on at any time to be able to adapt to the current mission situation. The required tasks for the current mission phase will be then distributed among the active nodes. For fault tolerance purposes we shall have at least two active nodes at any time and important tasks shall be executed at least twice at any time point. Tasks are able to flow from any node to any other. We call this "flowing tasks"

For different missions we can have different configuration of nodes. we consider "One size fits all" is not a feasible solution for all missions. The only important aspect is the interoperability among different nodes so it shall be possible to attach any kind of node to the network and use it in a "plug and play" way without having to modify the software.
We may have following kind of nodes:

The **basic node** is implemented in a single SOC (System-on-a-chip) chip. This SOC implements all required interface to the network and implements (in software) the required communication and routing protocols. With the rest of the computing power (a lot) it can be used as simple computing node. As extension of the basis node we may have different kinds of specialized nodes (see following nodes).	
The **intelligent Sensors and Actuators** (similar to a remote terminal controller) transform the signals and protocols of (not intelligent) devices to the VIDANA network protocol. So it is possible to attach any intelligent device to the network and use it in a plug and play fusion. Beyond signals and protocol conversions the intelligent devices provide functions like calibration data transformation (eg from speed to acceleration, from rotations to attitude) and sensor fusion.	
The **Mass-Memory**-Device is another extension of the basic node. It just adds SD-Flash-Memory cards (eg. 4 GB) and implements a file system.	
Number Crunchers are required to process a "big" amount of data for example Image processing/recognition, optical navigation, etc. For these cases we can attach a Digital Signalprocessor (DSP) or a vector processor to the basic node.	

4. References

4.1. State-of-the-Art für Pico und Nanosatelliten

www.dk3wn.info/satellites.shtml,
www.ucsusa.org/nuclear_weapons_and_global_security/space_weapons/technical_issues/ucs-satellite-database.html,

http://www.brite-constellation.at/,

https://infeo1.eo.esa.int:8091/presentations/10000776/10002866.html

http://www.responsivespace.com/Papers/RS5%5CSESSION%20PAPERS%5CSESSION%205%5C5005_PRANAJAYA%5C5005P.PDF

4.2. Further References

RODOS: http://www.montenegros.de/sergio/rodos/index.html

Robust Intelligent Systems
Part IV, of Book Robust Intelligent Systems
Springer, Sept. 2008, ISBN: 978-1-84800-260-9

2012: Integration of Software and Hardware Building Blocks for Space Applications
Sergio Montenegro
Annual Computer Science Series 2012

Middleware Switch ASIC Implementation
Sergio Montenegro, Vladimir Petrovic
ICECS 2011: International Conference on Electronics, Circuits and Systems
Beirut, Lebanon from 11-14 December 2011.

Network Centric Systems for Space Applications
Sergio Montenegro: DLR Institut for Space Systems
Vladimir Petrovic, Gunter Schoof : IHP GmbH
SPACOMM 2010
June 13-19, 2010 - Athens, Greece
ISBN: 978-0-7695-4067-2