RELIABLE NETWORKED DISTRIBUTED ON-BOARD DATA HANDLING USING A MODULAR APPROACH WITH HETEROGENEOUS COMPONENTS

Florian Kempf[(1)], Alexander Hilgarth[(2)], Ali Kheirkhah[(3)], Tobias Mikschl[(2)], Tristan Tzschichholz[(3)], Sergio Montenegro[(2)] and Klaus Schilling[(1)]

[(1)]Department of Robotics and Telematics, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany; {kempf,schi}@informatik.uni-wuerzburg.de

[(2)] Department of Aerospace Information Technology, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany; {alexander.hilgarth,tobias.mikschl,sergio.montenegro}@uni-wuerzburg.de

[(3)] Zentrum für Telematik, Allesgrundweg 12, D-97218 Gerbrunn, Germany ; {ali.kheirkhah,tristan.tzschichholz}@telematik-zentrum.de

ABSTRACT

Robustness is a key system design criterion for on-board data handling of modern space vehicles. This contribution emphasizes for these objectives a distributed networked approach, which introduces a high degree of flexibility and reliability, in particular also for vehicle formations. The architecture is based on universal wireless inter-system links, which can directly access internal subsystems from outside and thus supports sharing of distributed resources of processing and storage capacities. For intra- and inter-vehicle communication a related concept is proposed and simulated by Omnet++. At proof of concept level for system control the RODOS embedded operating system combined with MATLAB/Simulink and the "Building Blocks Execution Platform (BBEP)" was employed. On this basis future hardware demonstrations will be prepared.

1. INTRODUCTION

Performance increases of modern data handling systems support autonomous reaction capabilities on-board to optimize mission results. Nevertheless, the increasing on-board responsibilities in the challenging space environment require robust and fault-tolerant data processing approaches to guarantee reliable spacecraft operations. State of the art spacecraft technology for on-board data handling purposes is based on multiple specialized computers organized in a hierarchical way [3], [15]. Each sensor or actuator module brings its own processing power, and few central processing units connect the subsystems (Figure 1). The communication in this network of computing nodes is generally hardwired and various types of bus systems are used, like space-wire, SPI or MIL standards. While this system design is straight forward and well tested, it has several drawbacks. Every computing unit for a certain subsystem has to be designed powerful enough to handle subsystem peak usage, but will be idle most of the time. Thus the full available processing capacity is not exhausted and excess processing power is wasted, without the possibility to use it for other, more demanding tasks.

Another problem arises, if the central computing unit fails. As all other subsystems are hardwired to this unit and external access to sensors and actuators is not possible, the whole system fails. Therefore

satellite systems use redundant central units, decreasing the overall efficiency and not preventing failures if an inherent system error causes redundant central units to fail.

Decentralized approaches offer an alternative architecture onboard a single spacecraft [4], but are even more relevant for multi-spacecraft systems, in particular for the emerging satellite formations [2], [11]. At University Würzburg's Experimental satellite program (UWE) redundant micro-processor systems [1] for reliable on-board operations were successfully tested with UWE-3 since November 2013 in orbit. Future missions will address Pico-satellite formations with research on relevant technologies already on the way [10], [12]. Here it would be very attractive to use resources from other spacecraft of the formation, when needed.

In this framework the project YETE (ph**y**sical distribut**e**d con**t**rol in spac**e**), addresses a novel approach to on-board data relying on a decentralized system composed of distributed networked data processing modules at hardware and software level, to achieve highly reliable operations [8].

The remainder of the work is organized as follows. First the general YETE concept and a suitable operating system for the nodes inside YETE are introduced. Subsequent an idea to simulate communication among the individual nodes in Omnet++ is proposed. Afterwards a solution to integrate Simulink control into the system as well as a proof of concept is presented. Finally the results are summarized and planned future work is discussed.

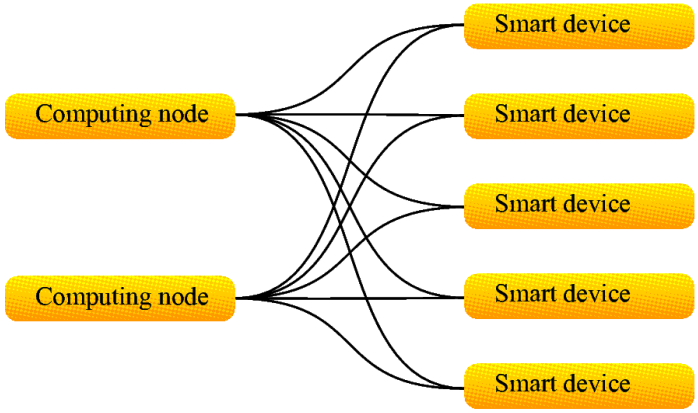

Figure 1: State of the art: One or more board computers are connected to "smart" sensors and actuators (such as GPS receivers, reaction wheels, star trackers, etc.) by a bus system or individual wires.

2. YETE OBJECTIVES

In the project YETE, the rigid hardwiring is replaced by wireless communication (inside the spacecraft as well as towards other companion satellites), and the computational capacity is concentrated into so-called *computing nodes*. The individual subsystems are reduced in complexity to the absolute minimum; sensor-/ actor-specific processing now takes place on the computing nodes (figure 2).

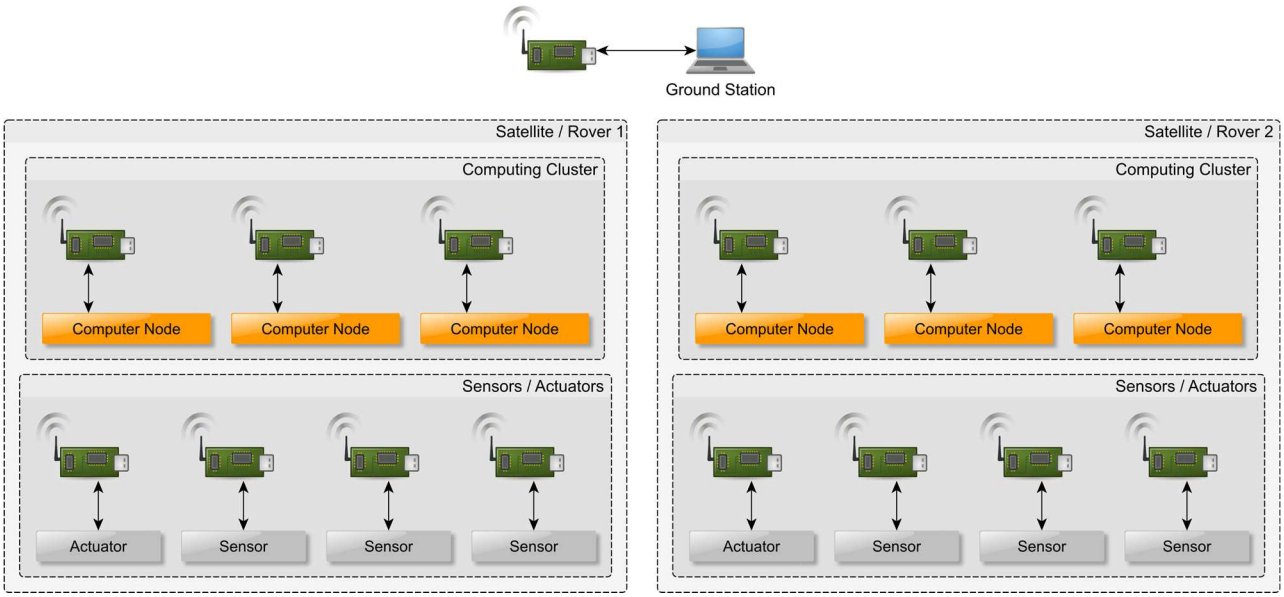

Figure 2: The YETE concept: Wireless links are used to connect a cluster of computing nodes to very simple sensors/ actuators. All sensor- or actuator-specific processing now takes place in one or more computing nodes.

This approach has several advantages. First of all, it provides modularity at the software- and hardware level. On the hardware level, the modularity is achieved by the wireless links, which make it unnecessary to think about physical connectors. Components and modules can be exchanged without problems. On the software level, the Building Blocks Execution Platform (BBEP) is a set of functionally independent software modules (e.g., IMU driver, reaction wheel driver, star sensor buffer, etc.). This modular structure allows for task migration among the nodes regarding intra system connections as well as inter system connections. Overall, this facilitates a fast adaption to changing mission requirements.

The most prominent advantages resulting from this modular architecture are: capability of hot-swapping of all nodes in the system (actuators, computing nodes, sensors), the sharing of resources (most importantly, computational power), higher fault tolerance (due to redundant systems and the ability to access sensors and actuators of a system from the outside when required), and sensor data fusion from multiple systems.

In the YETE project, a demonstrator is realized for this vision, which will be built in five steps:

1. Implementation of sensors/ actuators (wired connection), reflecting the state-of-the-art.
2. The local processing power is removed from the sensors/ actuators and moved into a computing cluster. This will demonstrate the capability of the system to distribute tasks, as well as fail-over in case a computing node fails.
3. The wired connection is replaced by a suitable wireless technology (such as Bluetooth, ZigBee, etc.) This will demonstrate the feasibility of the distributed system architecture on a single satellite/ system level.
4. A distributed multi-satellite system is built which will demonstrate solving a problem in a distributed fashion. This will demonstrate the communication between two or more systems, and more importantly, the communication between sensors/ actuators and a set of foreign computing nodes (i.e., computing nodes of another satellite/ system).
5. The distributed system is enhanced by a control facility to command the distributed system from outside, in order to demonstrate seamless operation during remote control phases, as well as autonomous control.

2.1 RODOS Framework

The distributed system of the YETE project will be hardware independent. To achieve this goal an operating system is used. Our requirements for this system are:

1. The development boards / processors we want to use have to be supported.
2. The operating system has be robust out of the box and avoid unnecessary complexity to enable a simpler debugging process.
3. High level of hardware abstraction is desirable, as we want to develop our system hardware independent.
4. If possible the operating system shall be space proven.

After some consideration the real-time operating system RODOS was chosen, because it adheres to the requirements mentioned above and includes additional features, which benefit to our whole system and integrate nicely with our proposed system design.

RODOS is light-weight (size 1MB on x86 architecture) and developed by the means of “keep it simple”. But with its simplicity it also brings a high level of fault tolerance and flexibility [7]. It is space proven in different spacecraft and systems and is going to be used in various spacecraft within the next two years. As RODOS is open source, it can be freely modified and enhancements made within this project can be ported back to it, benefiting the open source project.

RODOS includes a preemptive scheduler for concurrent task execution. Tasks can have different priorities. The task with the highest priority is executed. If multiple tasks have the same priority a round-robin scheduling method is used, in which every task gets an equal share of processing time. To enable communication between different tasks RODOS includes a messaging system, which follows the publisher-subscriber principle. This means that a unique topic is attributed to all data that is exchanged between individual tasks. Tasks can subscribe to these topics or publish data on them.

The message-system which enables communication between tasks in a running system, can also be extended across system borders. The physical link between different systems or nodes is handled by so called link-interfaces. These interfaces are responsible for the data transmission over the hardware underlying the respective link, e.g. UART, CAN, UDP, etc.

RODOS also provides a Hardware Abstraction Layer (HAL) for typical embedded hardware interfaces. This layer enables us to program hardware drivers for external sensors and actuators target platform independent (figure 3).

To allow task-distribution over the different nodes and to encapsulate functionalities in exchangeable modules, a building block system on top of RODOS is used. Building Blocks are software components, which implement a certain functionality like sensor reading or fusion and communicate with other parts of the software via the messaging system. This encapsulation simplifies the development process, because building blocks can be developed independently of each other and can be reused more easily.

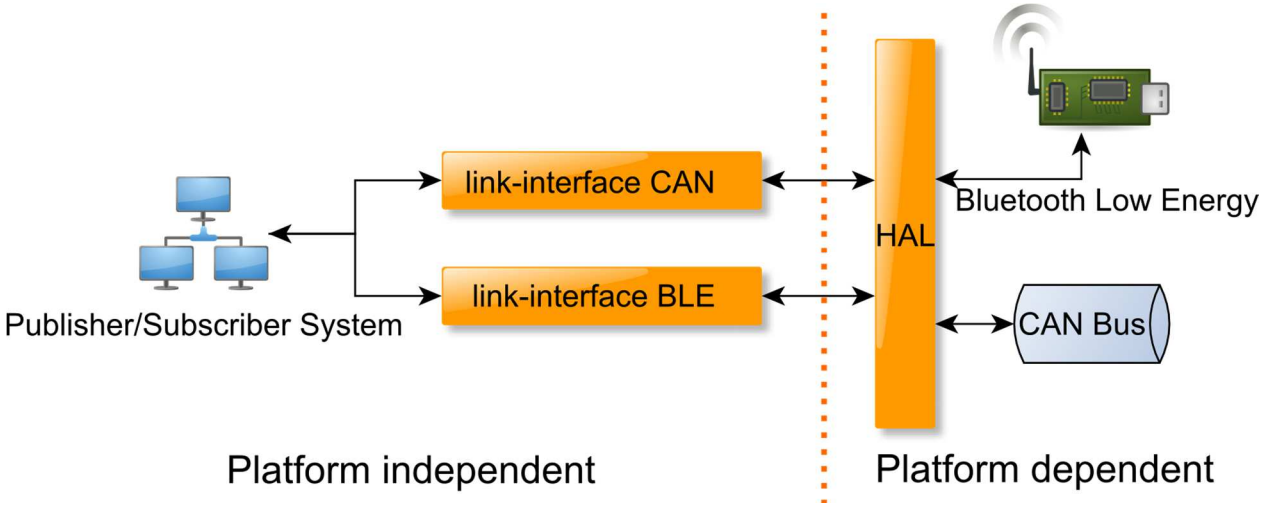

Figure 3: Relationship between RODOS platform independent link-interfaces and platform dependent HAL layer.

3. COMMUNICATION SIMULATION CONCEPT

As YETE is a highly distributed concept, communication between the individual system nodes plays a major role in the overall system behavior. The transmission channel properties, such as delay, bit error rate or packet loss greatly influence the ability to perform distributed system control and the way in which tasks can be distributed or shared among nodes in the network. Therefore, to allow the testing of different control algorithms, task distribution concepts and RF link hardware under controlled conditions, a communication simulation concept for YETE was developed. We decided to use the discrete event simulator Omnet++ [14], [13] as simulation environment, because it satisfies the following necessary requirements.

3.1. Simulation requirements

The first requirement we imposed on the simulation is, that it should substitute the simulated real system components as accurately as possible. One prerequisite for that is, that the simulation is transparent for all not simulated components in the system. In a later project stage the simulation can then just be substituted by real hardware without the need for changes to the not simulated system components. To achieve this we integrated the Omnet++ simulation seamlessly into the RODOS communication layer, this can be seen in figure 4 and figure 5.

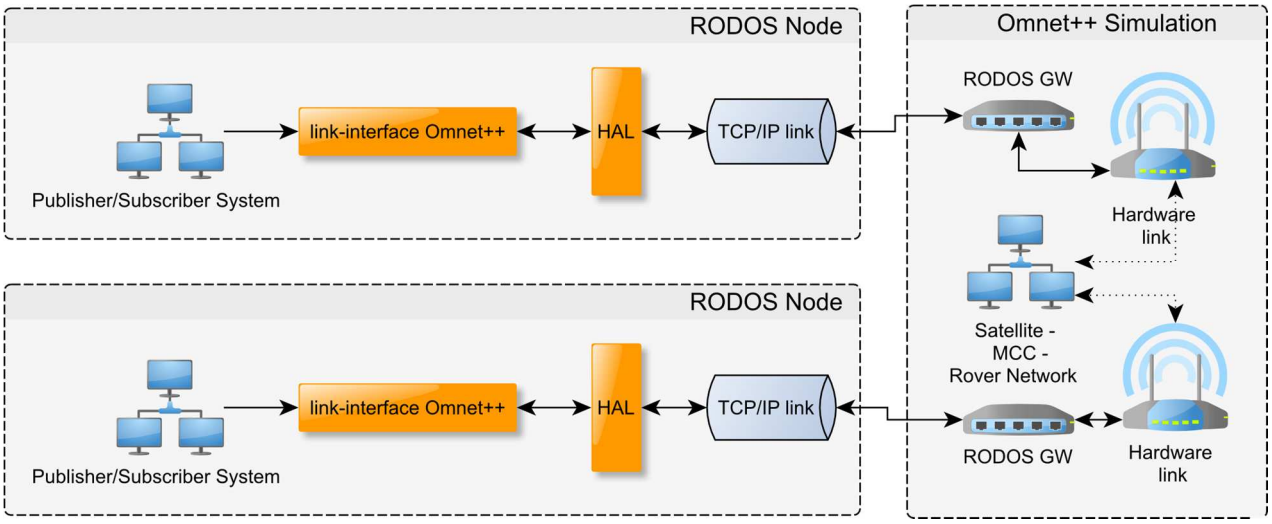

Figure 4: Integration of Omnet++ into the RODOS node communication layer.

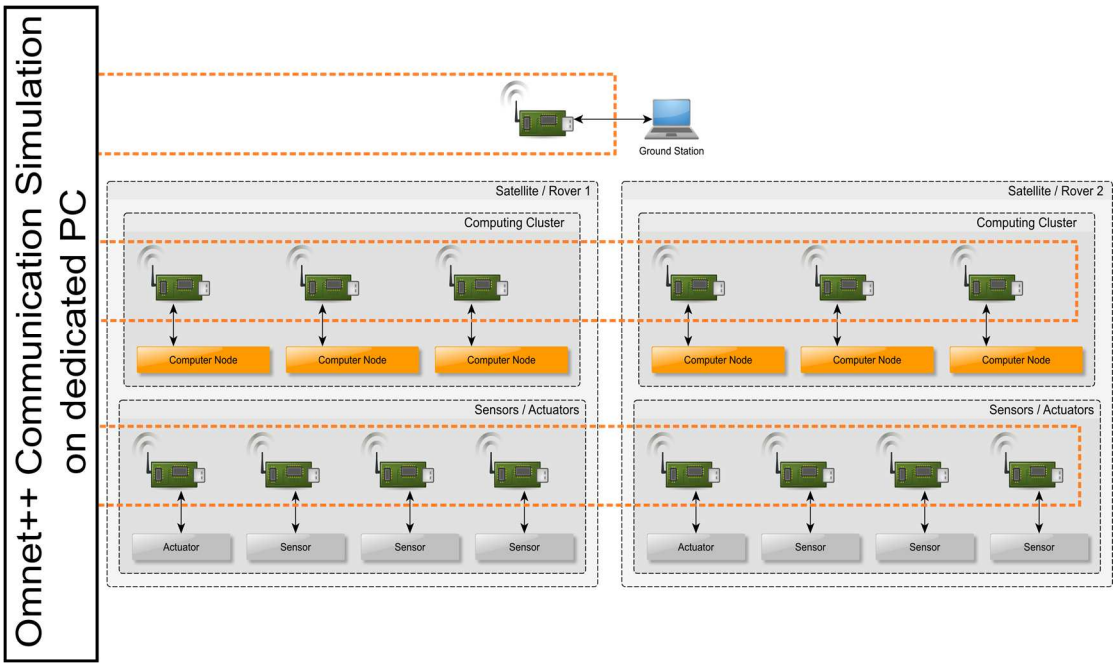

Figure 5: Integration of Omnet++ into the YETE system.

Furthermore the simulation itself should introduce minimal simulation overhead into the system. For this purpose we designed the simulation to be run on a powerful external dedicated PC, to limit the influence of simulation calculations on the simulated system process. We also extended the Omnet++ Real Time Event Scheduler to handle real time events coming from a RODOS-to-Omnet++ Gateway, which will be further explained later on.

The second requirement for the simulation is that it should speed up the design ←→ evaluation cycle throughout the project duration. Omnet++'s ability to reuse simulation components, to structure them in a hierarchical way and to easily define large complex nets of connected entities (e.g. spacecraft, Mission Control Centers (MCC) or rovers) via its meta description language NED is essential in this regard [13]. By using Omnet++ we also have several extensive component libraries (e.g. INet) at our disposal, which contain many commonly used network components (e.g. WLAN links, Ad-Hoc routing components, OSI Layer implementations, etc.).
The last requirement we regarded as important is the ability to perform online and offline data visualization and collection. The simulation concept supports both, visualization of the current satellite and MCC state (e.g. position, connectivity, etc.) live in a map, as depicted in figure 6, as well as using Omnet++'s event logging facility to collect and store simulation statistics for later use.

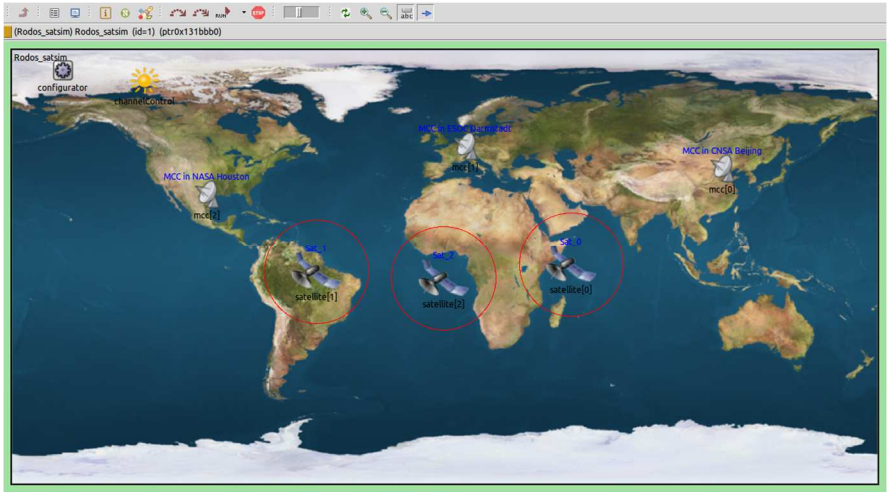

Figure 6: Visualization of the spacecraft system network inside Omnet++. Satellite to MCC link cover-age of the satellites is marked by a red circle.

3.2. RODOS to Omnet++ Gateway

Integration of external real-time applications into Omnet++ has already been done before [6], however none of the proposed concepts apply in this case. The reason is, that the external application, RODOS in this case, will be running on several distributed nodes, some of which are embedded devices. Therefore the link between the RODOS Operating System and the Omnet++ communication simulation consists of two parts, one is platform dependent and is running on the RODOS side and one is platform independent and is running on the Omnet++ side. The communication between the two parts is done via the TCP/IP protocol, preferably over an Ethernet cable connection. The part on the RODOS side is implemented as a RODOS hardware link-interface (omnetpp-linkinterface) over which RODOS topics can traverse. This link-interface can later be exchanged by link-interfaces of real hardware, like one of a Bluetooth Low Energy (BLE) connection. This way the simulation appears transparent to the RODOS system and to the other software Building Blocks (BBs) running on the node. On the Omnet++ side a RODOS gateway module receives the topic data from one or more RODOS nodes via a two way tcp connection and forwards the topics to exactly one simulated hardware link inside the simulation environment. If the RODOS gateway module represents an intra-satellite link it also broadcasts a special satellite state topic to selected modules inside the satellite module which need the current satellite state, e.g. the satellite position or attitude, for their operation. One example for such a module is the satellite mobility module, which updates the satellite position inside the visualization map and which is also used in the signal strength calculations of the radio transmission links (e.g. inter satellite and satellite to MCC links). The connection concept is exemplary shown in figure 7, in which three nodes of a simple satellite, a Simulink RODOS node, a sensor and an actuator node communicate over a simulated intra satellite radio link.

To facilitate realtime event processing and to handle new received RODOS data inside Omnet++ we extended the realtime socket scheduler of Omnet++, so that between two scheduled events, it performs a blocking select operation on the sockets of all RODOS gateways in the network and lets them handle newly received topic data if a socket event occurred.

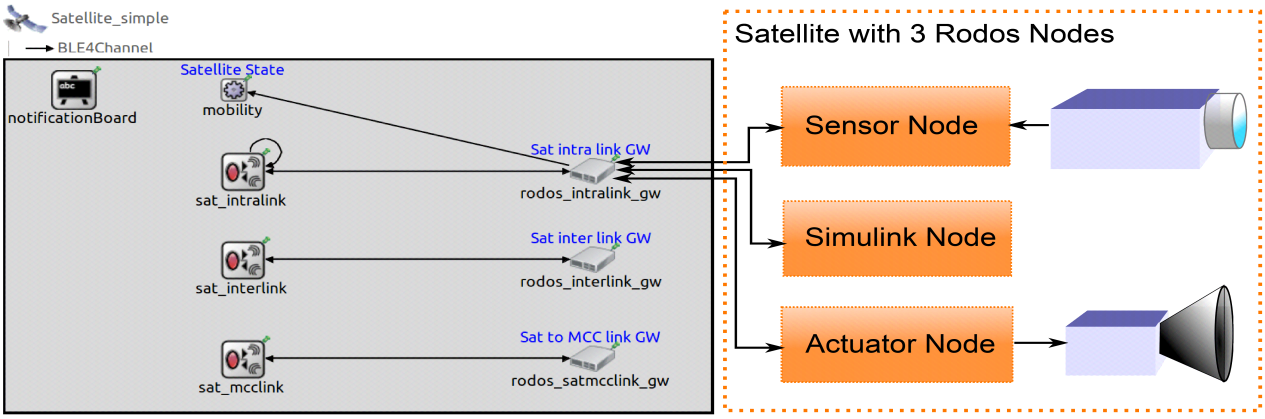

Figure 7: Omnet++ Satellite module (on the left) and 3 RODOS nodes belonging to the same satellite which are connected to the RODOS gateway module of the satellite intra-sat link.

The Omnet++ satellite module shown in figure 7 contains three RODOS gateways and the three corresponding radio links, an intra-satellite link, an inter-satellite link and a satellite to MCC link. The intra satellite link has a simulated Bluetooth Low Energy transmission channel connecting the module to itself. This simulates the intra satellite communication, i.e. the communication among the RODOS nodes which belong to a single satellite. Inter satellite and Satellite to MCC radio links rely on a free space transmission model and the distance between sender and receiver to determine the received signal strength and transmission delay.

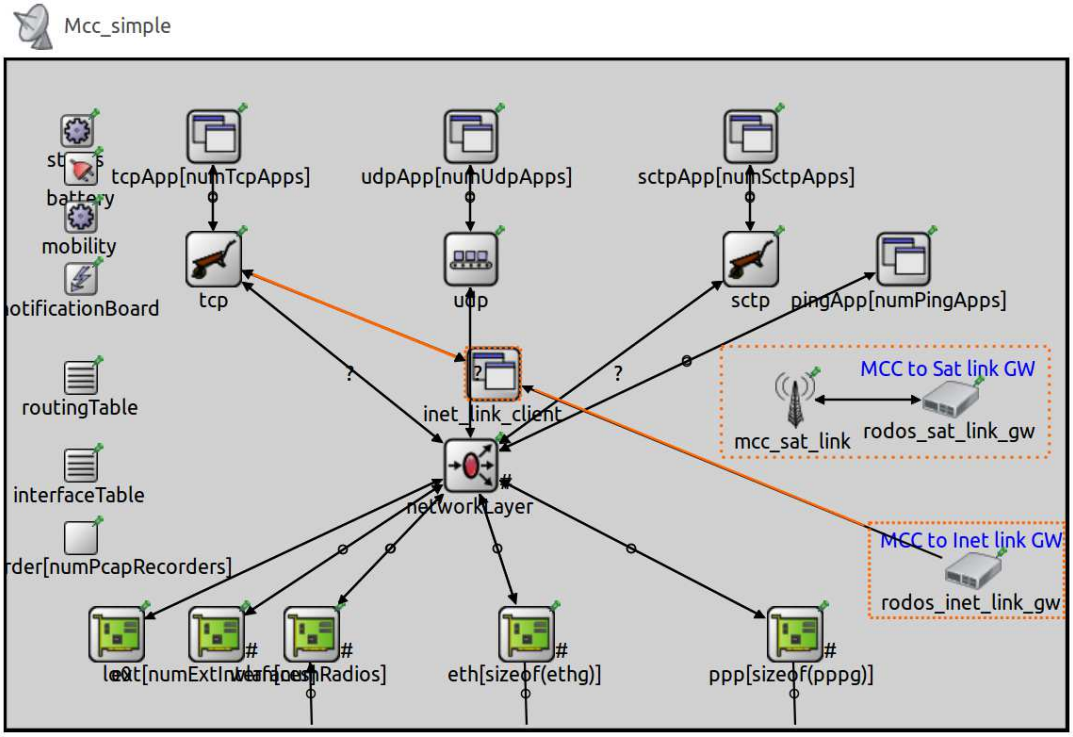

Figure 8: Mission Control Center Module in Omnet++.Our extensions to the standard INet host

module of Omnet++ are marked in orange.

The MCC Omnet++ module is depicted in figure 8. It has a static map position and contains two RODOS gateways, one is connected to a MCC-to/from-satellite link, and another one is connected to an Omnet++ TCP application module which sends/receives RODOS topics over/from a simulated Internet connection to/from other simulated MCCs. This should pave the way for MCC network simulations and the testing of different routing protocols in a later stage of the project. Another interactive Omnet++ module represents a planetary rover, which has a dynamic position and contains a single RODOS gateway connected to a Wireless LAN module, which is in turn is connected to the simulated Internet. This allows the rover to receive and transmit topic data from/to the MCCs. For example a control command topic for the rover could be relayed by one of the satellites to a MCC which then sends the control topic directly to the rover.

4. SIMULINK CONTROL

4.1. Simulink integration

For facilitating the development of control algorithms and high-level tasks, we developed a Simulink toolbox targeting RODOS applications. The toolbox consists of a support package (SP) which basically contains the glue code and interfaces to RODOS, and the block library, which provides specific interface blocks for connecting Simulink models with sensors or RODOS internals.

We have chosen Simulink, since this allows us to re-use existing models and develop new models very quickly. Also, by using Simulink we can extend the models with other toolboxes provided by Matlab and interface the controller to real hardware, when required.

Depending on the application, the controller requires measurements at a specific rate. The sensor responsible for these controller inputs must provide the samples at the required rate and send them to the controller via the RODOS middleware.

Using our support package, code can be directly generated from Simulink. This code can then be compiled into a standalone binary containing a complete instance of RODOS and the model designed with Simulink. Later on, this binary can be copied into the flash memory of a microprocessor and can then be run on dedicated hardware.

The interface between the sensors, actuators and the model (i.e., the controller) is made by the RODOS middleware and its publisher/ subscriber architecture (figure 9).

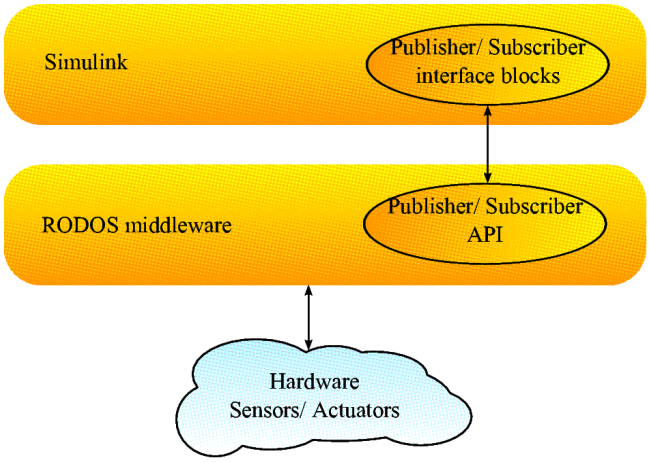

Figure 9: The interface between the Simulink model and the hardware is provided by a set of flexible

RODOS middleware interface blocks.

4.2. Simulation results

To investigate the integrability of the simulation concept and to prove the compatibility of Matlab/Simulink with RODOS and Omnet++, a Simulink model is generated which contains a simplified control loop. The model consists of a doubled integrator as the plant, representing a point mass moving on a surface with two degrees of freedom which are both actuated using external forces as input. The coordinates of the position are taken to build the output.

$$m\ddot{x} = u; \;\; y = x \tag{1}$$

with

x: $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ the position coordinates

u: $\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$ the forces as input

y: The output

m: Mass

To control the position of the point mass, a linear PD-controller is applied. With the control input from (2), the closed loop can be described by the Equation (3) which has a stable equilibrium at the origin. The stability of the closed loop can be proven using e.g. the Routh-Hurwiz criterion [5].

$$u_c = -D\dot{x} - Px; P, D > 0 \tag{2}$$

$$m\ddot{x} + D\dot{x} + Px = 0 \tag{3}$$

The corresponding C++ code is used to generate the required binary file to run within RODOS. Finally, the results are compared to show the proper working of the generated simulation on a hardware with RODOS running.

Assuming the scenario, where the point mass having an initial velocity of $\dot{x}_0 = [0.1,0.3]^T$ should move from the origin $x_0 = [0,0]^T$ to the point $x^* = [-3,0]^T$, the simulation results are shown in figure 10.

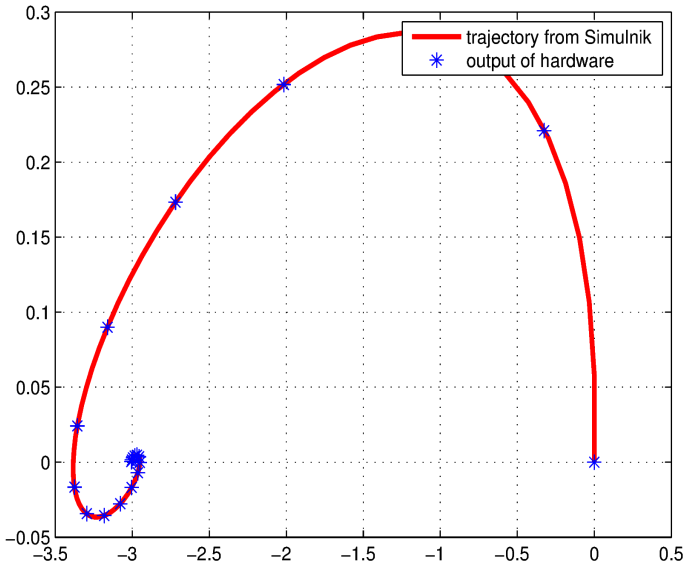

Figure 10: Simulation results from running a stand-alone Simulink simulation and the output from the same Simulink simulation inside a hardware RODOS node sampled with $f_{sampling}$= 1 Hz

The fact that both simulation results align perfectly demonstrates that, by using the proposed concept, Simulink control from within a hardware RODOS node can be done successfully.

5. CONCLUSION

With YETE a highly distributed spacecraft on-board data handling concept was introduced as well as a five step transition to reach this from the current state of the art data handling was presented. As suitable operating system for the YETE objectives RODOS was identified. To pioneer the development of the five models of YETE in the future, Simulink control and Omnet++ network simulation was integrated into the RODOS environment. For the Omnet++ integration we focused on a fast testing capability of new/individual system components, easy substitution of simulated hardware by real one and the ability to simulate big satellite - rover - MCC networks. Simulink control from within a RODOS hardware node proved to be feasible and delivered accurate results.

In the future we plan on doing extensive system tests with the Omnet++ communication simulation and to further extend it. This includes employing a system tailored variant of Delay-Tolerant-Network (DTN) protocols for inter-sat and satellite to MCC communication and testing different short range wireless solutions (BLE, ZigBee, Wlan, etc.) for the intra-sat link. The satellite to MCC link can also be extended to account for atmospheric perturbations (e.g. current weather above the MCC) and to incorporate more detailed receiver/sender and antenna properties into the signal strength calculations. For these calculations we plan to use the well proven CNI-OS3 framework [9].

On the control side the control algorithm needs to account for delays during control/sensor data transmissions. One way to do this is to include the current expected system delay into the state of the control process and to base control calculations on this extended system state. A delay tolerant control is also a necessity for another important step, the separation of the controller and the control process model into individual nodes, which are solely connected by the intra satellite link. We are also currently designing a satellite hardware demonstration platform, on which we plan to implement our concepts and run hardware in the loop tests.

One of our major goals is to demonstrate the benefits of the YETE approach in a future network of satellites in orbit, which are controlled in a distributed way.

ACKNOWLEDGEMENTS

The authors appreciated the support for YETE by the German national space agency DLR by funding from the Federal Ministry of Economics and Technology by approval from German Parliament with reference 50RA1332.

REFERENCES

[1] Busch, S., Schilling, K.; Robust and Efficient OBDH Core Module for the Flexible Picosatellite Bus UWE-3, 19th IFAC Symposium on Automatic Control in Aerospace, Würzburg 2013, p. 218 – 223

[2] *D'Errico, M. (ed.), Distributed Space Missions for Earth System Monitoring*. Springer Verlag 2012

[3] Fortescue, P., G. Swinerd, and J. Stark.. *Spacecraft Systems Engineering*. Wiley 2011.

[4] Lyke, J. L., *Plug-and-Play Satellites*, IEEE Spectrum August 2012, p. 30-35

[5] Lunze, Jan.. *Regelungstechnik 1*. Springer 2008.

[6] Mayer, Christoph P, and Thomas Gamer. 2008. "Integrating Real World Applications into OMNeT++." *Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2*.

[7] Montenegro, S., V. Petrovic, and G. Schoof. 2010. "Network Centric Systems for Space Application." In *IEEE Conference SPACOM* 2010

[8] Montenegro, S., K. Schilling, YETE: Distributed, Networked Embedded Control Approaches for Efficient, Reliable Mobile Systems, Proceedings Embedded World 2014 Exhibition and Conference, Nürnberg 2014

[9] Niehoefer, Brian, Sebastian Šubik, and Christian Wietfeld. 2013. "The CNI Open Source Satellite Simulator Based on OMNeT++." In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, 314–321. ICST (Institute for Computer Sciences, Social-Informatics; Telecommunications Engineering).

[10] Schilling Klaus, Networked Control of Cooperating Distributed Pico-Satellites, Proceedings IFAC World Congress, Cape Town 2014, WeC19.6

[11] Schilling, K., F. Pranajaya, E. Gill, A. Tsourdos; *Small Satellite Formations for Distributed Surveillance: System Design and Optimal Control Considerations*. NATO RTO Lecture Series SCI-231, 2011.

[12] Schilling, K., Schmidt, M.; *Communication in Distributed Satellite Systems*; In D'Errico, *Distributed Space Missions for Earth System Monitoring*. Springer Verlag 2012, pp. 345 -354

[13] Varga, Andras. 2010. "OMNeT++." In *Modeling and Tools for Network Simulation*, 35–59. Springer.

[14] Varga, András, and Rudolf Hornig. 2008. "An Overview of the OMNeT++ Simulation Environment." In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, 60. ICST (Institute for Computer Sciences, Social-Informatics; Telecommunications Engineering).

[15] Wertz, J. R., D. F. Everett, J. J. Puschell, *Space Mission Engineering*, Microcosm Press 2011.