From TET Data Management System (DMS) to RODOS generic DMS

Sergio Montenegro
Aerospace Informatics
University Wuerzburg
Am Hubland; 97074 Würzburg; Germany
Email: sergio.montenegro@uni-wuerzburg.de

Abstract: The TET data management system is mainly based on COTS components. To provide high reliability and live expectations (eg. BIRD is similar and is already been working for over 11 years) we employ redundancy and intelligent redundancy management implemented in software. The Software Architecture of TET is based on the Frame Work technology with an application discovery Protocol to interlink different applications. This technology was now extended with a middleware for real time communication and dynamic addressing to allow task migrations between nodes. This Middleware is integrated in the **RODOS** building blocks execution platform. TET was entirely programmed in C++. A big part of the code was automatic generated from the specification. From the Telemetry and Telecomand format definition which was used in the ground station. we generated code to collect and format data to fit the downlink spec and to extract commands and parameter from the telecommands. From a graphical Timing specification we generated code to control start and end of the different threads running on the TET Bus. From the hardware specification of analog channels we generated code to read, calibrate and distribute relevant values.

1. TET DATA MANAGEMENT SYSTEM

TET-1 (German: Technologieerprobungsträger 1, Technology Experiment Carrier) is a microsatellite operated by the German Aerospace Center. It is the centre of the OOV (On Orbit Verification) Program, initiated to offer on-orbit verification possibilities to the German industrial and scientific aerospace community. TET is based on the satellite bus used for the BIRD satellite. TET-1 Was successfully launched on July 2012.

Thet TET data management system is mainly based on COTS components. To provide high reliability and live expectations (eg. BIRD is similar and is already been working for over 11 years) we employ redundancy and intelligent redundancy management implemented in software. To achieve a high dependability, safety, and lifetime, the board computer is formed of four identical computers (nodes). As shown in the block diagram below (Figure 1), the redundant nodes and all the devices of the satellite that have to be controlled by the board computer are interconnected by several bus systems with different protocols. The architecture of the redundant control computer is totally symmetric, that means, each of the nodes is able to execute all control tasks. One node (the worker) is controling the satellite while a second node (supervisor) is supervising the correct operation of the worker node. The two other node computers are spare components and are disconnected. If an anomaly of the worker node is detected by the supervisor node 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 detected, it becomes the supervisor node. 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.

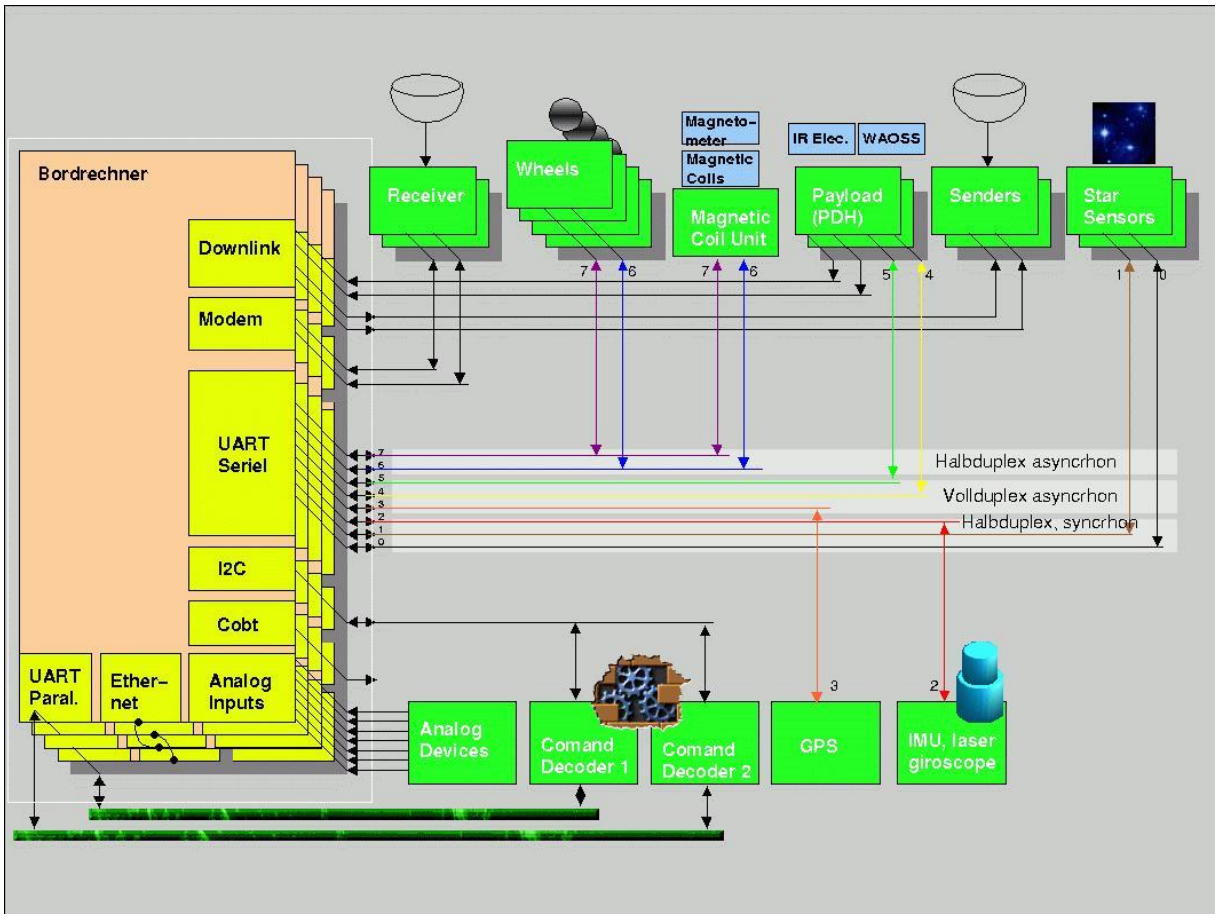

Figure 1: Hardware Configuration of the Bus Controller

The Software Architecture of TET is based on the Frame Work technology. TET was entirely programmed in C++. A big part of the code was automatic generated from the specification. From the Telemetry and Telecomand format definition which was used in the ground station. we generated code to collect and format data to fit the downlink spec and to extract commands and parameter from the telecommands. From a graphical Timing specification we generated code to control start and end of the different threads running on the TET Bus. From the hardware specification of analog channels we generated code to read, calibrate and distribute relevant values. From the Hardware specification of the power distribution we generated code to turn devices on and off and to test is the current power distribution to the expected configuration corresponds.

The different software task of the TET bus are implemented totally independent from each other. To make this possible we developed a task management systems which implements all data interchange and synchronization between applications. So is it very simple to remove, to add or to interchange applications without having to modify any thing else in the code. When started, each application reports it self to the task manager. There is no need of a central code

which allocates and calls all applications. The Task manager implements a task discovery protocol to identity the current software configuration, which tasks are available and which are ready. Using this same discovery protocol any task can find its corresponding working partners. All tasks are linked in a list which is managed by the task manager. At run tine any task may request contact to any other task using the Protocol showed in figured 2 and 3.

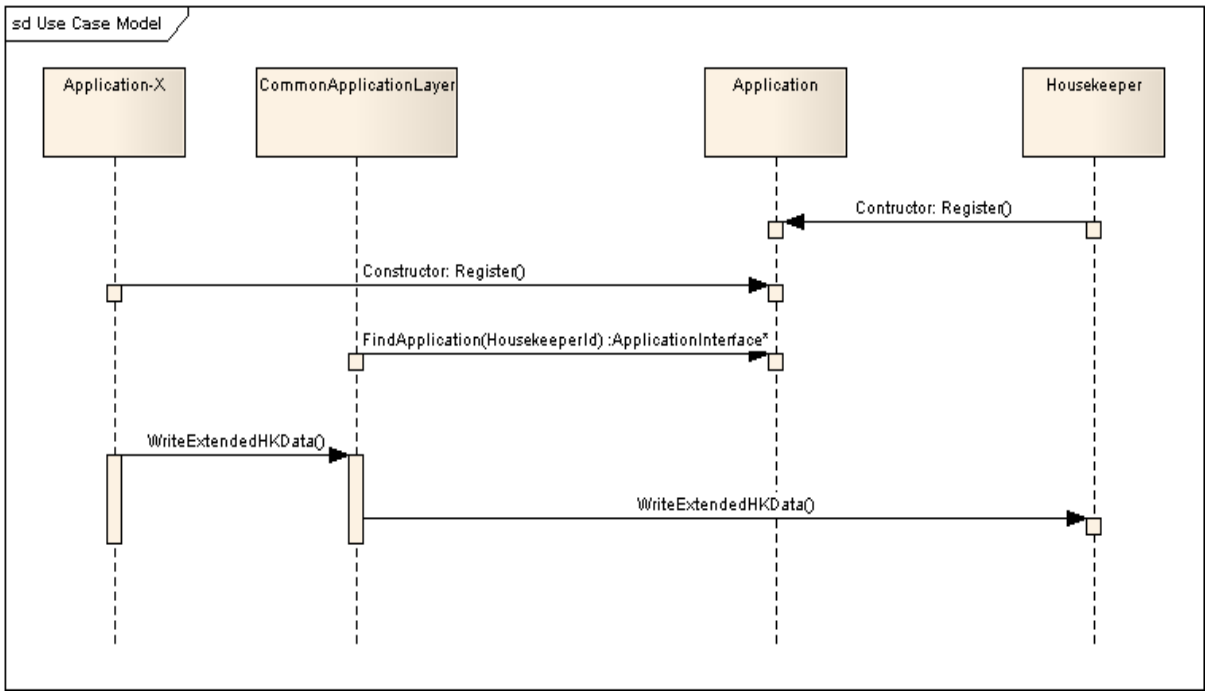

Figure 2: Registry of applications

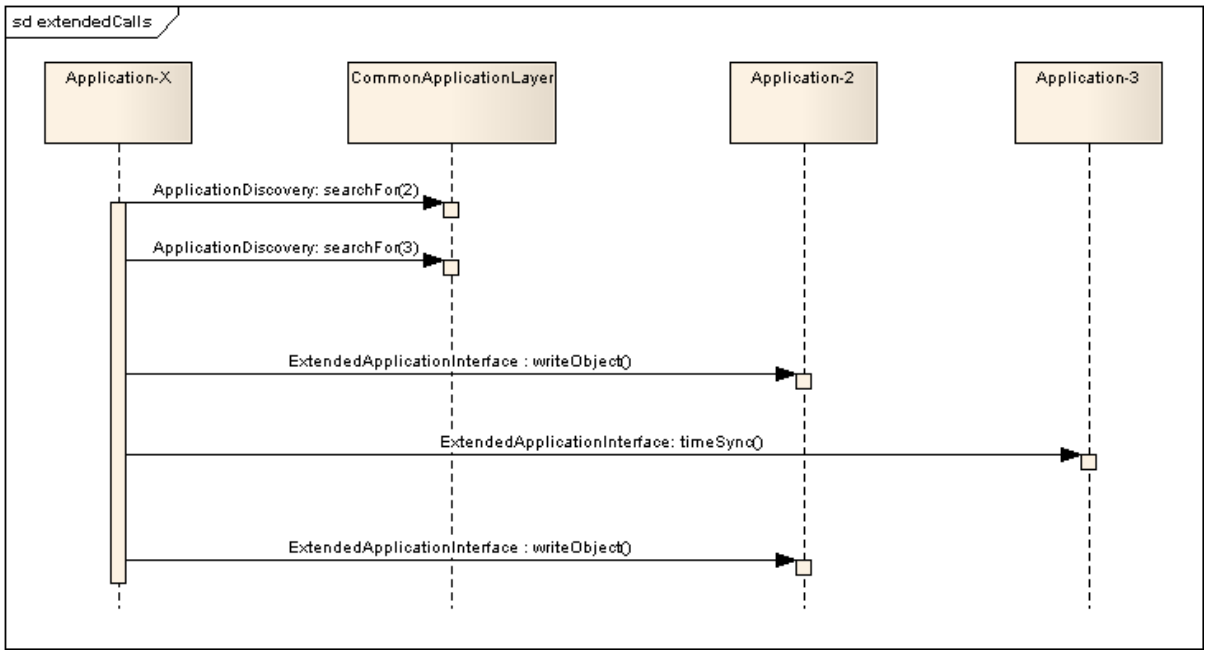

Figure 3: Finding Applications

2. RODOS GENERIC DATA MANAGEMENT SYSTEM

Now we are working an the next step on software technology. We are developing a middleware which was designed to support fault tolerance and task distribution on redundant systems. This Middleware in integrated in the Building Blocks execution Platform RODOS, which is being used in different space mission developments now.

The most effective and safe way to implement a complex parallel system is to compose it as a network of simple sequential tasks. 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).

RODOS implements a publisher / subscriber Middleware. Each message is routed and distributed without fixed sender/receiver In System we can have many communication topics for example attitude data, navigation data, temperatures, etc. See Figure 4.

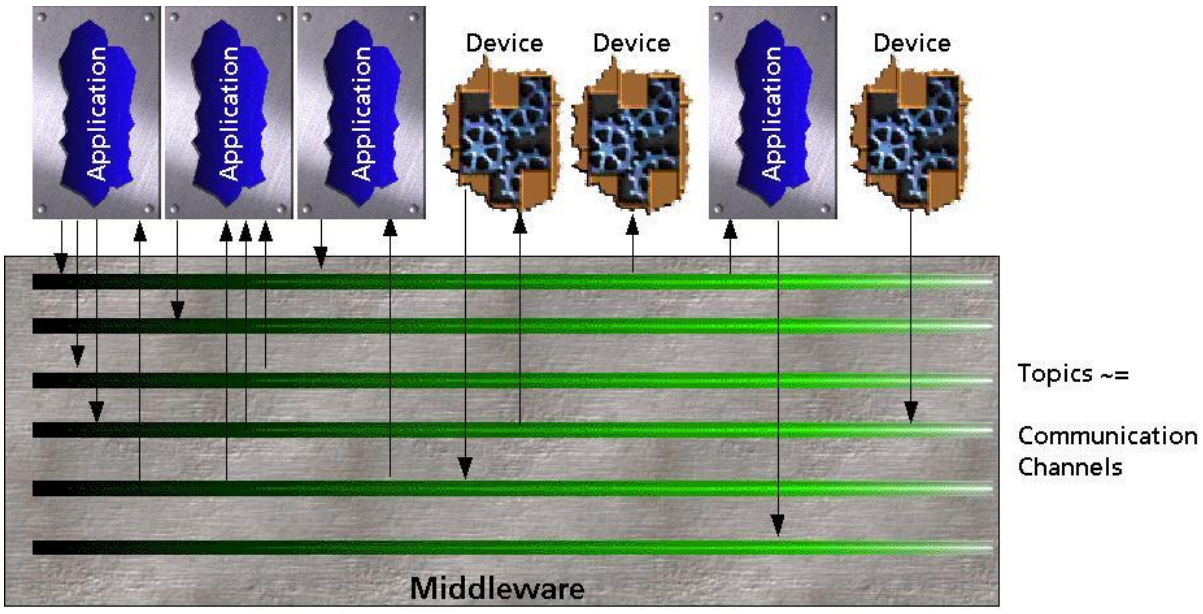

Figure 4: Communication Topics

Any task may publish data using different topics. 0 or more applications may subscribe any topic. So the sender does not needs to know who will receive its data, nor where is the receiver located or how many receivers we have. This makes it very simple to migrate applications from node to node. If a node crashes its applications may be distributed on other nodes. The communication partners do not need to be notify They may continue sending their data under the usual topic. At run time the middleware analyse each messages a determines how to forward, distribute and replicate it.

So the whole functionality of the spacecraft will be implemented as a distributed (software) network of relative simple software tasks.