A SW MIDDLEWARE TO PROVIDE ULTRA HIGH DEPENDABILITY IN SPACE USING COTS-COMPUTERS

Raffaele Vitulli [(1)], **Sergio Montenegro** [(2)]

[(1)] *ESA/ESTEC, Keplerlaan 1, 2201AZ Noordwijk, The Netherlands, Email: Raffaele.Vitulli@esa.int*
[(2)] *FIRST, Kekuléstraße 7, 12489 Berlin, Germany, Email: sergio.montenegro@first.fraunhofer.de*

ABSTRACT

Much more than fault tolerance, space applications require dependability, which is the combination of availability, reliability and safety. (The main paper will present the relationship between these properties and the roles of robustness and fault tolerance).
How to reach high dependability has been an important research theme for a long time, and still today, after 20 or 30 Years of research, there is not THE ONE solution. Big efforts go to improve reliability, for example using reliable radiation hardened components. Many other efforts go to improve robustness by means of fault tolerance.
But now, we join efforts from reliability and robustness to reach real dependability. The ESA-funded HiPeRCAR (High Performance Resilient Computer for Autonomous Robotics) project shows how to combine reliability and robustness in an optimal way to get the highest possible dependability using limited resources. This is important because space missions have to be sparing with resources.

Keywords: robotics, fault-tolerance, dependability, robustness, resilience, middleware, FDIR.

1. INTRODUCTION

From our experience we can identify at least 4 important very common conceptual obstacles when designing a fault tolerant system.

1) Any digital control system is a team work of software and hardware. Trying to solve any problem or challenge using only one of them means to loosing a lot of possibilities, like a team (Software-Hardware) where one player in software, but very few try to face the challenge in a team work of software and hardware.

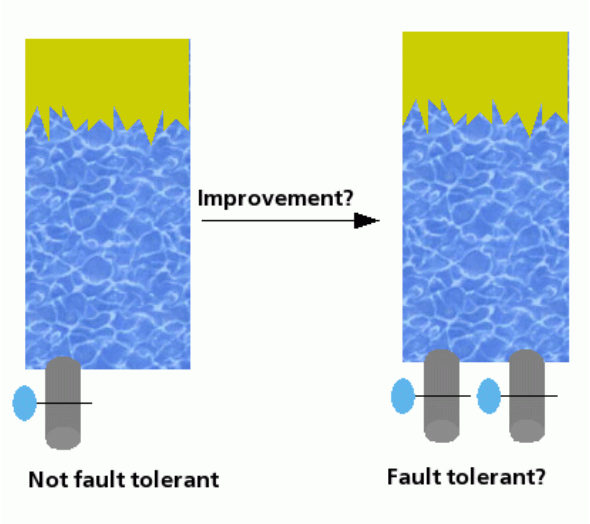


Figure 1: Implementing fault tolerance by replication?

2) Another wrong assumption is to think that to provide fault tolerance you just have to replicate resources. For example if you can control a water tank level opening and closing a valve, like in Fig. 1.

Then you may think: The valve can fail; let's add another valve (Fig. 1). Now one of the redundant valves fails in open position, you can not control the tank any longer. What you got is a higher possibility of failure and not more dependability. This was not right the solution!

3) Another wrong assumption is to think you can reach an absolute failure free system. Then an enormous effort is invested to reach what you can not reach. See

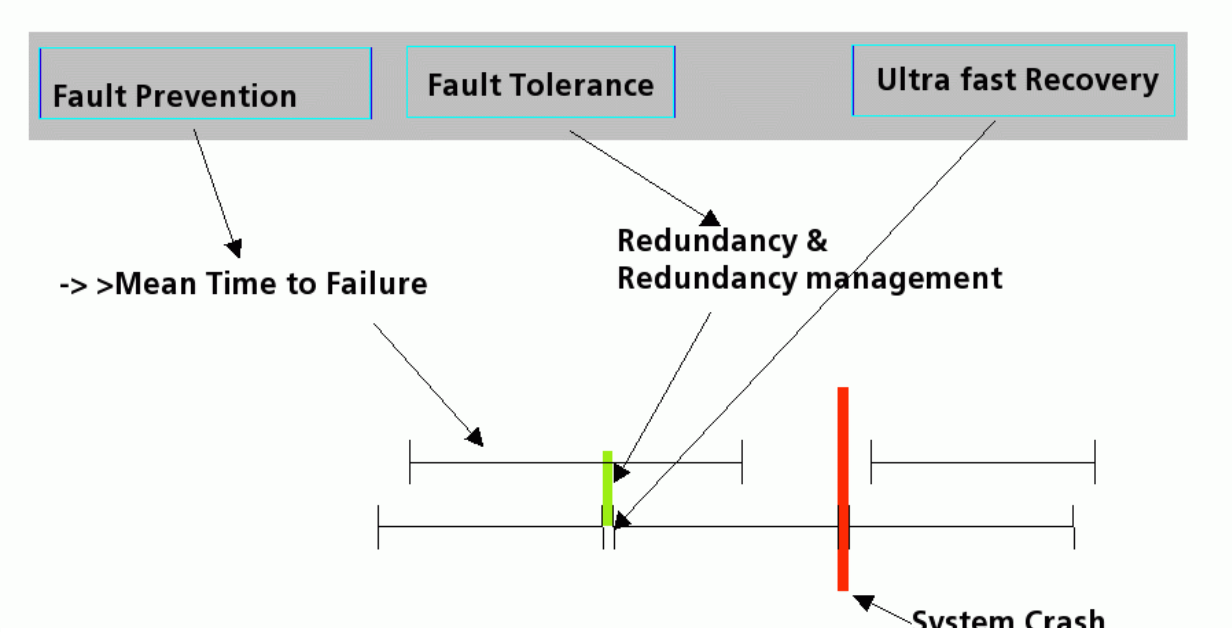


Figure5: Increasing dependability (2)

Fig. 2. You can do what you want, but any system will crash some day.

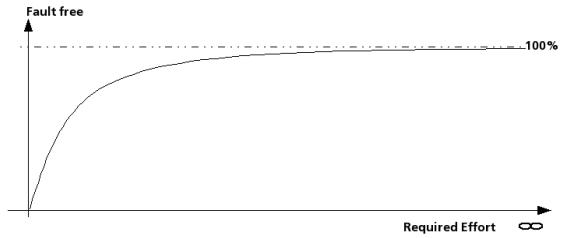
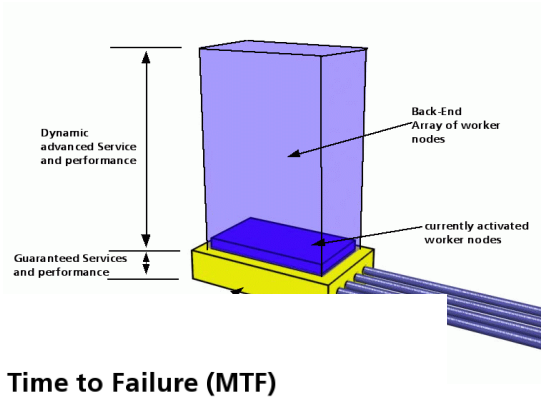


Figure 2: Infinite effort to reach a total fault free

4) Another important consideration is what to do after so many resources are permanently damaged, that no normal operation is possible any longer.



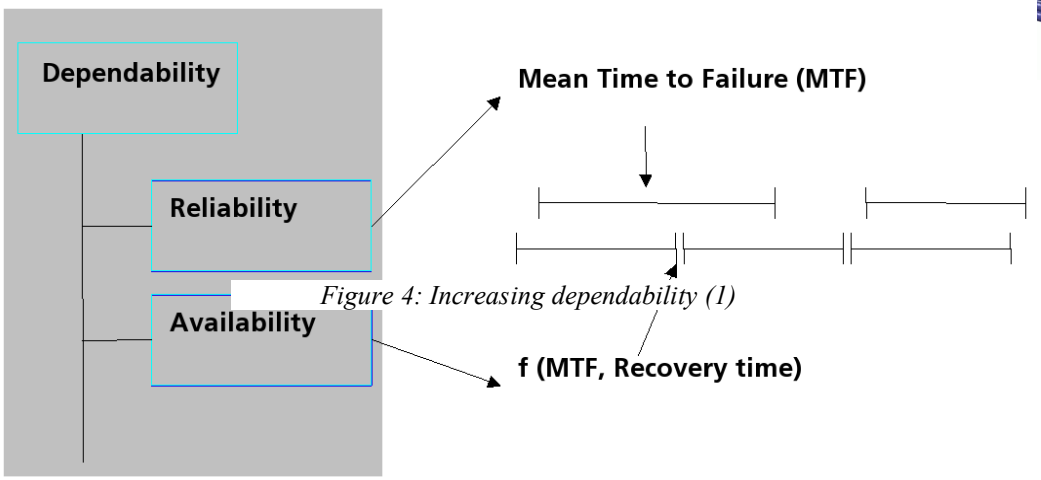


Figure 4: Increasing dependability (1)

system

2. A NON CONVENTIONAL APPROACH TO DEPENDABILITY

Let's consider (in this paper) 4 (from 0 to 3) stages of fault tolerance:

0: No fault tolerance. One failure is enough to cause a system failure or system crash
1: Any single failure can collapse some critical functions, but the system remains in a safe operation. (Gracefully degradation)
2: Like 1, but after a short time period the system can reconfigure itself to go back to normal operation (resilience)
3: No single failure can disturb the normal operation of the system.

Stage 3 sounds very nice, but it requires at least a 3 fold replication of (almost) all resources, including volume, mass, power consumption, heat production, etc. For some space applications this can be a prohibitive condition. Furthermore after the first permanent failure, the system will go down to stage 2 or 1. For long operations where permanent failures are expected, stage 3 will be possible only at the beginning of the mission.

It is therefore advisable to create a resources economical system which provides stage 2 for the whole mission even after several failures. Our target is to get the highest possible dependability using very limited resources. The system has to be operable always in a safe way, but temporally loose of advanced features is allowed. Thereby it was very important not to do again the common mistakes listed in chapter 1.

2.1. Step 1: Let it crash!

A realistic node computer operation can be represented like in Fig. 3.

Figure 3 realistic node computer operation.

After start up a node computer will work properly for a time period (may be years) but some day it will crash and no one can impede this. Better than trying to create a (sub)system which shall never fail or where a failure is an extraordinary exception, we count with failures, they are not exceptions, but expected events which will be handles smoothly. The paradox situation is: to be safe, be ready to crash at any time. To increase the dependability of the system we can increase the reliability of each node (eg. mean time to failure MFTF) and reducing the recovery time after crashes (increasing Availability). See Fig. 4.

Much more important than try to avoid crashes (we can not) it to provide an ultra fast recovery. Recovery includes reboot, restore context of applications and update the context to the current situation. Having a very short recovery time, we can tolerate very easily multiple node crashes, and if we use some kind of redundancy, the probability of two redundant nodes to be down at the same time becomes lower. See Fig. 5.

Today the recovery time is very long, depending on the operating system it can range from 10 seconds to some minutes. This long period increase the probability of a total system failure if redundant nodes are down at the same time (see Fig. 5). Or target is to have a recovery time of a few milliseconds. If so, we can tolerate several crashes per second without problems.

2.2. Step 2: Do not paddle only on one side, use both: Software and Hardware.

Better than trying to have fault tolerance in Hardware and to make it transparent to the software (e.g. Maxwell approach), we implements robustness as a harmonized team work of software and hardware. If you have a legacy or extreme complex software, which you cannot modify (or even understand) then you have to make fault tolerance transparent to the software, but doing so you lose an enormous potential from HW-SW team work: Hardware provides redundancy and software provides an intelligent redundancy management.

Using an intelligent dynamic redundancy management it is possible to adapt the redundancy degree to the current situation and mission phase. The resources usage can vary to provide high performance (no/low redundancy) or high dependability (high redundancy). See Fig. 6.

3. HARDWARE APPROACH

In order to reach a level of reliability suitable for critical space mission, several architectures have been proposed that exploit a full HW approach to reach dependability. An example is the Maxwell Board, that is a Power-PC based board with an embedded FPGA that implements TMR (Triple Modular Redundancy) voting in order to prevent failures. The drawback is the quite relevant cost, restriction of purchase abroad, resource demand.

Figure 6: Dynamic usage of resources

European industry is, on the other hand, mainly focusing on rad-hard component like the LEON family. But in this case the drawback is the limited processing power, which is quite relevant in the case of the Maxwell board.

All these limitations makes important to find an alternative solution based on Hybrid SW-HW architecture.

4. HiPeRCAR (SW-HW) APPROACH

HiPeRCAR aims to provide a dependable high performance control system for space applications, which guarantees continuity of services despite of failures and anomalies.
It was required:

- High performance in space and low cost
- Fault tolerance and no replication of resources

- To tolerate crashes but to do not discontinue the provided services

For the first glance all of them appear to be in contradictions but HiPeRCAR has a way to combine COTS and Radiation-Hard Hardware to fulfil all these requirements.

The solution places a Radiation-Hard node as front-end of a pool of high performance COTS nodes. The Front-End computer guarantees the safety of the system and the continuity of basic services. The Back-End high performance pool of COTS nodes provides not-dependable high end computations (nominal operation mode). See Fig. 7.

Services provided by the system are produced by a (software) network of co-operating tasks which are distributed in a (hardware) network of computing nodes. The safety, performance and continuity of service do not depend on computer nodes, it relays on the properties of the SW-HW network. The network is the computer.

Each service has a basic mode and an advanced nominal mode. The basic mode guaranties only the safe operation of the system. It is very simple and not resources demanding. The basic mode is implemented in tasks running on master nodes. The nominal mode includes complex and resources intensive computations, which are provided by a network of tasks distributed in the worker nodes. The most important property of the (SW) network is to allow task to smoothly flow from one node to another. At any time any task can disappear from one node (e.g. node crashes) and reappear in another node (task distribution). This is a normal system execution and do not require any type of reconfiguration, because it is not reconfiguration, the system relays on a continuous flow of tasks.

This approach requires new and innovative software concepts, which go beyond standard procedural operating systems and middleware. In the project we develop the HiPeRCAR framework, which is based on the BOSS (BIRD Operating System) real time kennel and middleware. The HiPeRCAR framework provides an object oriented framework application interface. Its middleware distributes messages according to a simple producer / subscriber protocol. Message distribution is position independent and connectionless. This is important to allow an effective task flow in the system. To support fault tolerance and redundancy management the middleware is able to replicate messages without extra handling of the control tasks.

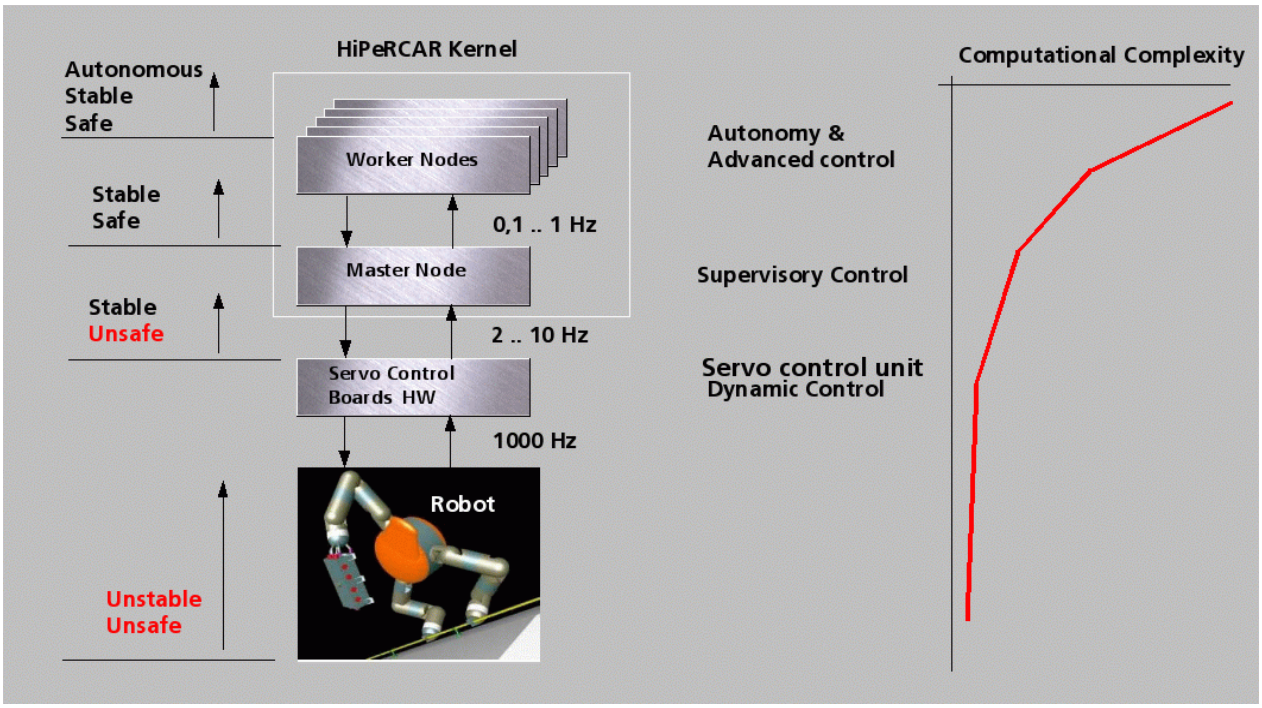


Figure 7: Layered services

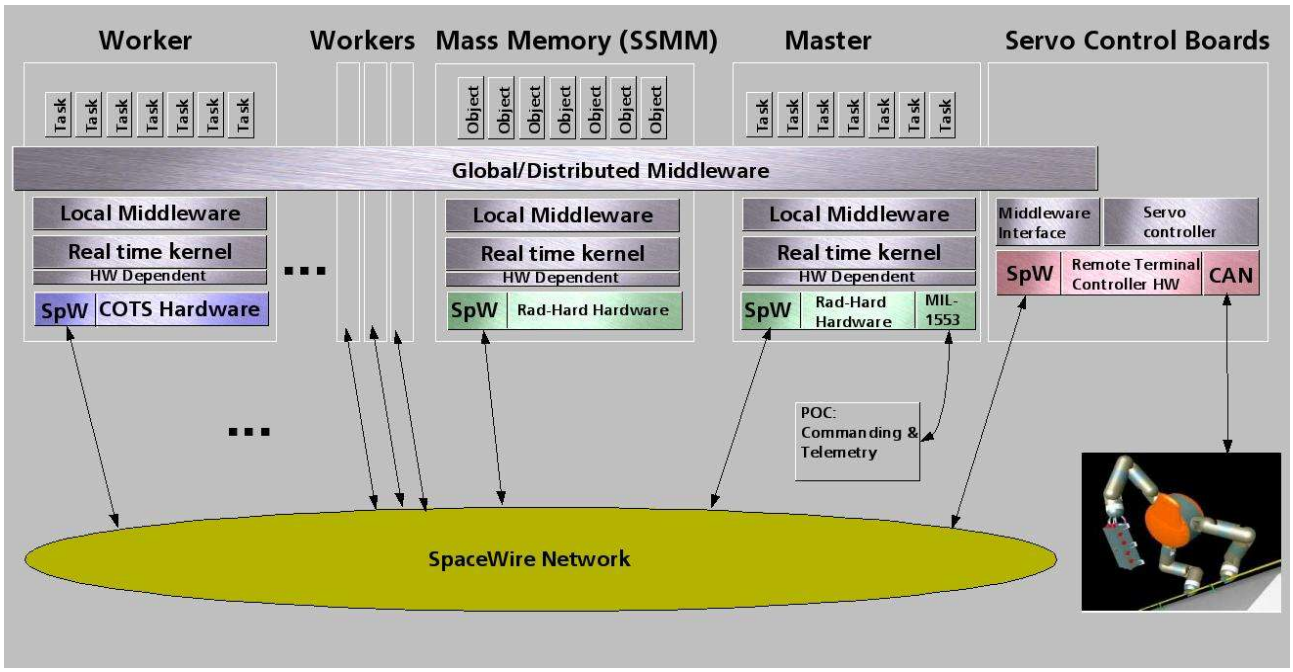


Figure 9: HIPERCAR System Configuration

5. HiPeRCAR-FRAMEWORK KERNEL AND MIDDLEWARE

The HiPeRCAR real time kernel and middleware offer an integrated framework (OO) interface. There is no need to differentiate what is a RT kernel and what is middleware functionality. Both together will be called the HiPeRCAR-framework. The HiPeRCAR framework aims to offer the most simple and small possible interface to users tasks, which still provides all required functionality and flexibility.

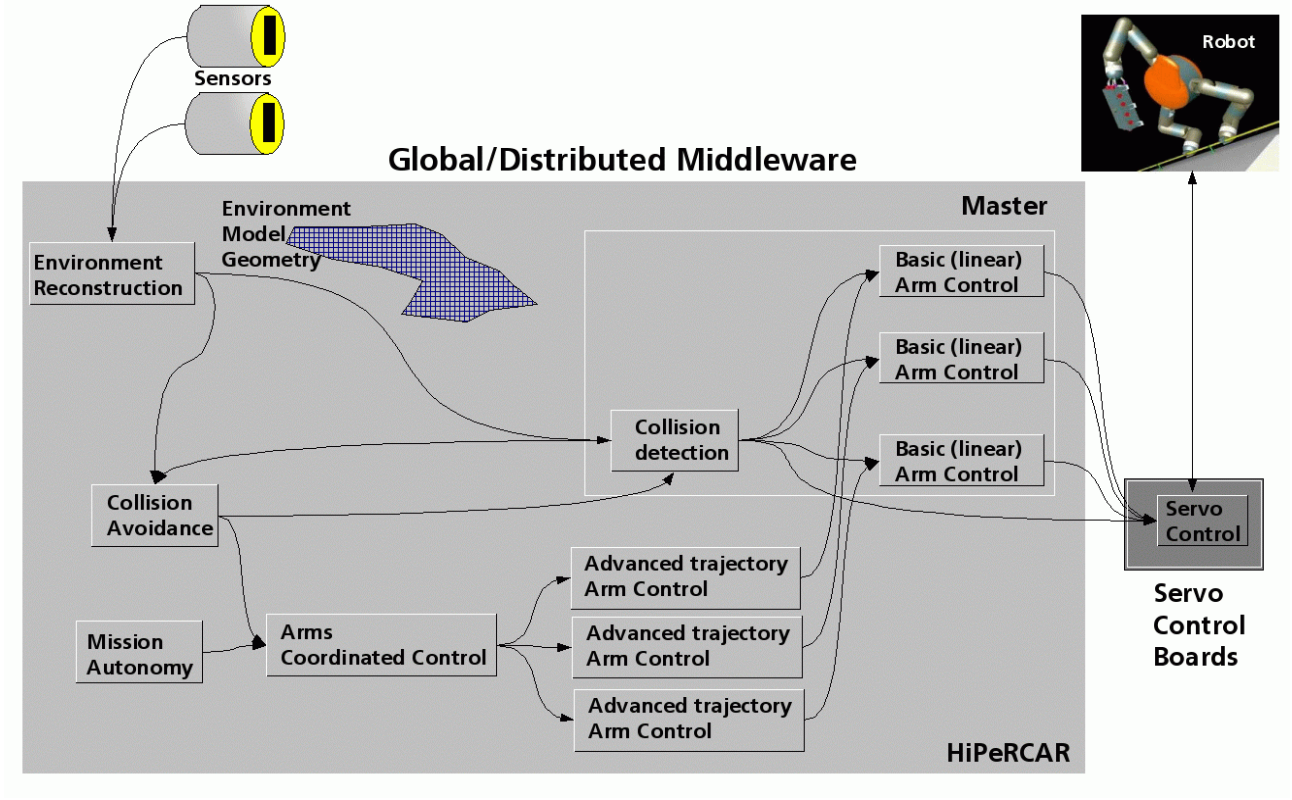


Figure 8: Hypothetical Example of Task Distribution

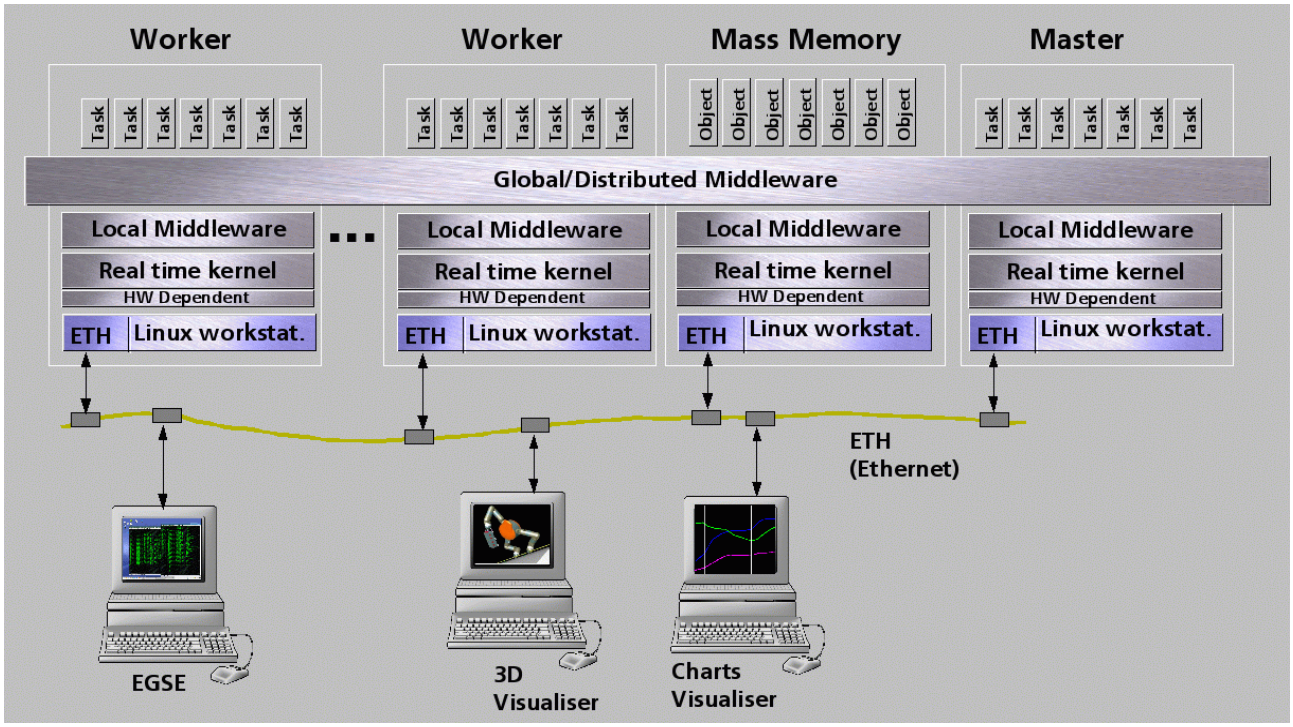


Figure 11: HiPeRCAR Simulator

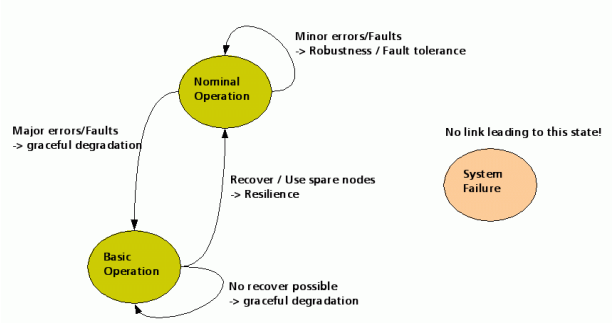


Figure 10: Fault tolerance and resilience

The HiPeRCAR-framework includes time management, resource management and communication functionality. Without an application the framework is inactive, it just reacts to interrupts doing nothing. An application can add actions to the HiPeRCAR-framework by inheriting classes and creating active objects. These objects will be integrated automatically in the framework. In this way the framework will be extended with user functionality.

5.1. Communicating tasks

Let's take as example a task network to control 3 arms using an environment model. The task distribution is shown in Fig. 8.

For the implementation there is no difference where the tasks will be deployed. The position of tasks can even change at run time, without requiring any explicit reaction of the other involved tasks. Task communicate using the HiPeRCAR Middleware, but its only interface are messages, the rest is transparent. Tasks just define input and output messages. The rest is some one else job (Middleware and real time kernel).

5.2. Structure of the HiPeRCAR framework

A HiPeRCAR controller is implemented as a collection of building blocks on the top of a middleware, which creates a virtual totally interconnection network for all tasks running on it. The middleware allows every task to communicate with any other task independent of their position. It imposes no limitations to communication paths, but the application shall use/create a meaningful, reasonable and efficient inter-task communication structure.

There may be different system configurations for HiPeRCAR applications. The system supports to migrate from any of these configurations to another without having to change the application. This will be achieved by implementing a layered system like in Fig. 9.

The application tasks run on the top of a virtual global middleware which cross node boundaries and interconnect all threads/tasks in the system. They do not need to know the details of lowered layers. Applications can interact without having to know about locations and hardware architectures.

The global middleware is implemented on local middleware running on each node. This local

middleware runs on the top of a real time kernel or operating system which administrates the time and local resources in the node. The real time kernel is implemented using a virtual hardware abstraction, which is implemented by the hardware dependent layer. To move from any hardware platform to another one, only this small layer has to be re-implemented.

5.3. Fault Detection and Fault Recovery

HiPeRCAR provides continuity of services despite of failures in Worker nodes (the Master node is supposed to be dependable). In our approach it is possible to recover (reboot and initialise) a node in less than one seconds, but this is still too long in order to provide real time continuity of service. Therefore the safe control do not relay on tasks running on Workers. The basic tasks allocated in the Worker node can control the devices alone, but if advanced tasks are running, the results of such advanced services will be used to improve the movements and autonomy.

In this way, if a Worker node fails just the advanced commands will be missing, but the basic control task stays operable. From outside of the HiPeRCAR kernel there is no discontinuity of service (control). The System goes instantaneously and automatically from nominal operation to basic operation. See Fig. 10.

In the meantime, a monitor tasks will detect the failure of the Worker node. They report it to a recovery task which will try to reboot the faulty node. If this does not help, the faulty node will be turned off and eventually a spare node will be activated. After the recovery task, a configurator task will reallocate advanced tasks on running workers to go back to nominal operation (resilience). If this fails, then the system will stay in the basic operation mode (graceful degradation). In any case there will be no interruption of service.

6. VERIFICATION TEST BED

Before the Demonstrator can be built, the concepts will be tested on a simulator running on normal workstations like in Fig. 11.

The real robot will be substituted by a simulation, the SpaceWire Network will be substituted by a fast Ethernet Network, and the flight hardware will be substituted by normal office PCs.

For the applications it will make no difference on which configuration they are running! We aim to make this test environment totally transparent for any application running on the top of the HiPeRCAR framework.

7. ACKNOLEDGMENTS

This work has been performed in the frame of an ESA-ESTEC Contract nr. 18750. The prime contractor is Alcatel Alenia Space Italy (Milan). The subcontractors are Galileo Avionica (Italy), FIRST Institute Berlin (Germany) and Syderal (Switzerland).

8. REFERENCES

1. Marra, et al. - *"HiPeRCAR: the High Performance Resilient Computer for Autonomous Robotics"* - DASIA 2006, Berlin.
2. Jameux, D. – *"Application of the Payload Data Processing and Storage System to MOSREM Multi-processor On-board System for Robotic Exploration Missions"*
3. Astrium/Scisys – *"UNIONICS Advanced On-Board Distributed Processing Architecture"* - Initial results from the Bench-Top Demonstrator"
4. M. Johnson, B. Green, O. Eman - EADS Astrium *"Improving the efficiency of spacecraft systems with the Unionics Advanced On-board Architecture"* – SDSS 2005
5. Guy Estaves, Alcatel Alenia Space – *Super Computers for Space Applications*- SDSS 2005, 17-20 October 2005, ESTEC, Noordwijk
6. *SpaceWire – Links, Nodes Routers and Networks*, ECSS-E-50-12A at ECSS-E-50-12 A
7. ESA ITT AO/1-4497/NL/AG – *SpaceWire Remote Terminal Interface ASIC.*