Reconfiguration Strategies for Critical Adaptive Distributed Embedded Systems

Alberto Ballesteros*, Julián Proenza*, Manuel Barranco* and Luis Almeida†
*DMI, Universitat de les Illes Balears, Spain and †CISTER, Universidade do Porto, Portugal,
{a.ballesteros, julian.proenza, manuel.barranco}@uib.es and lda@fe.up.pt

***Abstract*—This paper describes the architecture and mechanisms proposed in the context of the DFT4FTT project for implementing Adaptive Distributed Embedded Systems (ADESs), that is, distributed systems with real-time, dependability and adaptivity requirements. The focus is on the reconfiguration strategies that allow, not only to change the system behaviour, but to improve its tolerance to permanent hardware faults.**

I. Introduction

An Adaptive Distributed Embedded System (ADES) is a type of Distributed Embedded System (DES) that has the ability to reconfigure itself dynamically in response to changing operational requirements and conditions. Some examples of potential applications of these systems are: autonomous vehicles, exploration vehicles, machinery in a smart factory and self-repairing devices.

Adaptivity brings interesting benefits from the point of view of the functionality, efficiency and dependability. First, ADESs can dynamically change their behaviour to cope with new demands. Second, ADESs can dynamically reserve the necessary resources at each instant thus avoiding the need to overprovision resources according to pessimistic worst-case assumptions. Third, ADESs can recover themselves by reallocating the tasks of a faulty node to a non-faulty one.

To properly implement an ADES it must be provided with the appropriate architecture and mechanisms, that make it possible to fulfil its real-time, dependability and adaptivity requirements. In this regard, the DFT4FTT (Dynamic Fault Tolerance for Flexible Time-Triggered Ethernet) project [1] proposes a complete infrastructure that addresses all these requirements both at the *network* and *node* level.

At the network level, DFT4FTT is based on FTTRS [2], a switched-Ethernet implementation of the Flexible Time-Triggered (FTT) communication paradigm. FTT allows to exchange real-time traffic, while providing the necessary flexibility to support the operation of an adaptable system. By flexibility we mean that FTT supports the transmission of periodic and aperiodic messages, with different and dynamic real-time requirements. To provide high reliability, FTTRS extends FTT with fault tolerance capabilities. Specifically, the network is replicated to tolerate permanent hardware faults, while messages are proactively retransmitted to tolerate transient faults. We are currently working in adding dynamic fault-tolerance capabilities to FTTRS, so that we can dynamically modify the number of proactive retransmissions to efficiently tolerate transient faults due to changing Bit Error Rates (BERs).

At the node level, DFT4FTT proposes a centralized approach in which the so-called *Node Manager* (NM) manages on-line the allocation of tasks in the nodes of the ADES. Specifically, the NM monitors the environment and the system itself, determines when and how to change the task allocation (from now on *configuration*) and starts/stops/reallocates the tasks in the nodes. To ensure that the tasks meet their real-time requirements, the NM performs a schedulability analysis before any allocation. To achieve high-reliability, active replication with majority voting is used. Specifically, each critical task is redundantly executed in several nodes and task replicas periodically vote to obtain a consensus result. This allows to tolerate permanent and temporary hardware faults affecting the nodes and the tasks. Moreover, DFT4FTT includes fault-diagnosis and reintegration mechanisms to prevent the redundancy attrition provoked by temporary hardware faults [3].

This paper describes the design of DFT4FTT, the first self-reconfigurable FTT-based infrastructure for highly-reliable ADES, which allows to change the allocation of tasks, as well as their real-time and reliability attributes in an automatic and dynamic manner, to fulfil the system requirements.

II. The Task model

DESs operate thanks to the execution of multiple *functionalities*, each of which implemented by one *application*. In turn, each application is a set of interconnected *tasks* that are executed in a sequential and/or parallel manner, for example, sensing, control and actuation tasks in a distributed feedback control loop. Note that distributed tasks communicate among them thanks to the exchange of messages through the network. Consequently, the operation of a an application is carried out as a sequence of task executions and message transmissions.

Functionalities can have different non-functional requirements. In this sense, tasks can have different real-time and reliability requirements, inherited from the application they belong to. To fulfil the real-time requirements, we rely on the work presented in [4], which allows to adequately determine the triggering instants of tasks and messages in a holistic and on-line manner. Moreover, from the reliability perspective, this work suits our task model as it seamlessly supports the replication of tasks. It is noteworthy, however, that we have to extend it to also consider the scheduling of several tasks being executed in a given node, as well as the fault tolerance mechanisms necessary to ensure the required level of reliability at the network level.

III. The System Architecture

The DFT4FTT architecture is composed of several components. The central element is an FTTRS network which interconnects all the other components. The Computational Nodes (CNs) are responsible for executing the tasks. Sensors and Actuators (SAs) provide the means for interacting with the environment. Finally, the Node Manager (NM) manages dynamically the allocation of tasks to the CNs.

In DFT4FTT the self-reconfiguration process [5] is carried out in three phases: *monitoring*, to obtain the system state; *decision*, to determine when and how to switch to a new configuration; and *configuration change* to carry out the system modifications. To explain how this complete process is carried out, the internals of the NM and a CN are shown in Fig. 1.

There are different levels in the software architecture. At the lowest level the *Communication Enabler* allows to interface with the network. Above that, we find the modules that give support to the reconfiguration process. The services provided by these modules can be accessed by means of the *Task Allocation Scheme (TAS) Service Interface*. At highest level we find the applications that can make use of these services.

The NM determines the *system state* [5] thanks to the *Monitoring Manager*, which gathers and processes the network traffic. The system state includes, for example, the list of faulty CNs and links, as well as the tasks executed in each CN.

The decision process is carried out automatically and collaboratively in the NM by the *Knowledge Entity* (KE), using predefined rules, and the *Wisdom Entity* (WE), using machine learning. Additionally, tasks in the CNs can also ask for changes. In this paper the focus is on the KE, which constantly verifies that the *system requirements*, that is, the set of tasks that have to be executed together with their real-time and reliability requirements, are fulfilled. If they are not met, due to a change in the system state or in the requirements themselves, the KE commands a configuration change.

When a configuration change is needed, the *Configuration Change Manager* orchestrates the appropriate changes that the *Main Communication Manager* and the *Main Task Manager* have to apply on the communications and tasks respectively. Note that the changes carried out by these components are supported by their counterparts in the CN side.

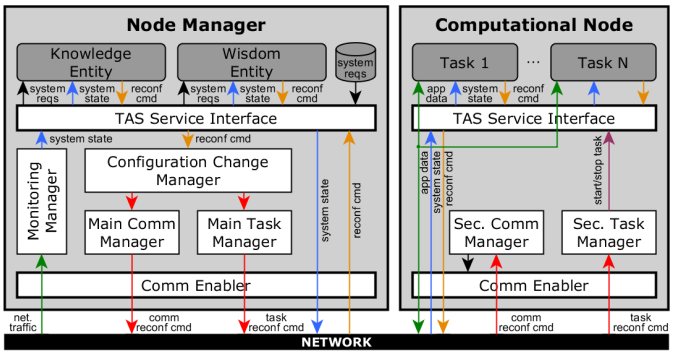

Fig. 1. Internals of the Node manager and a Computational Node.

IV. Reconfiguration Strategies

As introduced previously, the KE constantly compares the system state with the system requirements. In case the requirements are not met, due to a change in the state or in the requirements themselves, a new configuration is enforced.

The reconfiguration time has a great impact in the response time of the system to changes. That is why, we propose a two-step process that reduces the reconfiguration time. First, the KE provides, as fast as possible, a new configuration that meets the requirements. The policy for allocating the tasks is load balancing, that is, new tasks are allocated in CNs where the computational and network resources are underused. The selection of the CNs is fast since we can keep track of the available resources, and the resulting configuration can be still easily modified in case new tasks have to be allocated. Second, KE searches for the *best configuration* while the system is running. The definition of the best configuration depends on the goal of the system and is given by different configuration attributes like the performance or energy consumption.

From a reliability perspective, the reconfiguration capabilities of the NM allow us to reallocate the tasks being executed in one CN to another, when the first one suffers a permanent failure. For non-critical tasks this means we can continue delivering the service, after a minimum downtime. For critical (replicated) tasks this means we have redundancy preservation, that is, we can maintain the level of replication. This is equivalent to a N-Modular Redundancy scheme with several spares, where the spares are the available computational resources in the CNs. It is well know that this technique allows to significantly increase the reliability.

V. On-going Work

We are currently finishing the specification of the reconfiguration process. We will address issues like how to determine a valid configuration from all the possible ones.

Apart from that, we are evaluating the feasibility of dynamically changing the replication scheme and the number of replicas, to achieve the best reliability level while making use of the available resources in an efficient manner.

Acknowledgment

This work was supported by project TEC2015-70313-R (Spanish *Ministerio de economía y competividad*) and by FEDER funding.

References

[1] J. Proenza *et al.*, "DFT4FTT Project." [Online]. Available: http://srv.uib.es/dft4ftt/

[2] D. Gessner, "Adding Fault Tolerance To a Flexible Real-Time Ethernet Network for Embedded Systems," Ph.D. dissertation, University of the Balearic Islands, 2017.

[3] S. Derasevic, M. Barranco, and J. Proenza, "Designing fault-diagnosis and reintegration to prevent node redundancy attrition in highly reliable control systems based on FTT-Ethernet," in *Proc. 12th IEEE World Conf. on Factory Comm. Systems (WFCS)*, Aveiro, 2016.

[4] M. J. B. Calha, "A Holistic Approach Towards Flexible Distributed Systems," Ph.D. dissertation, Universidade de Aveiro, 2006.

[5] A. Ballesteros, J. Proenza, and P. Palmer, "Towards a Dynamic Task Allocation Scheme for Highly-Reliable Adaptive Distributed Embedded Systems," in *Proc. 22th IEEE Int. Conf. on Emerging Tech. and Factory Autom. (ETFA)*, Limassol, 2017.