Towards a Dynamic Task Allocation Scheme for Highly-Reliable Adaptive Distributed Embedded Systems

Alberto Ballesteros, Julián Proenza and Pere Palmer
Dept. Matemàtiques i Informàtica, Universitat de les Illes Balears, Spain
{a.ballesteros, julian.proenza, pere.palmer}@uib.es

***Abstract*—An adaptive distributed embedded system is able to automatize some processes at the same time it modifies its behaviour autonomously and dynamically in response to changing operating conditions. To support adaptivity it is necessary that the underlying Distributed Embedded System (DES) is able to dynamically change the assignment of the processing and network resources. In this regard, the DFT4FTT project aims at providing a complete DES that can support applications with real-time, reliability and adaptivity requirements. This paper describes the first steps towards the design of the task allocation scheme used in the DFT4FTT architecture, responsible for dynamically distributing the workload among the nodes of the DES, taking into account the changes in the environment and in the system itself. This allocation scheme not only provides flexibility from a functional point of view, but also from a fault tolerance point of view. Moreover, its modular design makes it possible to tune the desired level of autonomy in the adaptivity, from a simple support for application reconfiguration to a complete automatic reconfiguration assisted with machine learning algorithms.**

I. Introduction

An Adaptive Distributed Embedded System (ADES) is a kind of Distributed Embedded System (DES) that, apart from automatizing some processes, can modify its behaviour autonomously and dynamically in response to unexpected requirements or operational conditions [1]. Some examples of these systems are: autonomous vehicles, exploration vehicles, machinery in a smart factory and self-repairing devices.

Adaptivity is an interesting feature in terms of functionality, efficiency and dependability. First, ADESs can dynamically change their behaviour to cope with changing operational requirements. Second, systems do not have to be dimensioned to execute all the needed functionalities. Instead, ADESs can load the necessary functionalities depending on the scenario. Third, ADESs also enable the use of adaptive fault tolerance. For instance, in a replicated system the ADES can dynamically change the number of replicas to maintain the level of reliability depending on the operational conditions.

To support adaptivity it is necessary that the underlying DES is able to dynamically change the assignment of the processing and network resources. The DFT4FTT (Dynamic Fault Tolerance for Flexible Time-Triggered Ethernet) project [2] aims at providing a complete DES which can support applications with real-time, reliability and adaptivity requirements, using commercial off-the-shelf components. This project addresses all these requirements both at the *network* and the *node* level.

At the network level, DFT4FTT is based on a switched Ethernet implementation of the Flexible Time-Triggered (FTT) communication paradigm [3]. FTT provides predictability and flexibility in the communications of mixed-criticality systems. Specifically, FTT ensures that messages are received before a certain deadline, while it allows to change online their real-time attributes. In this project we extend the work we already started [4] to provide FTT with dynamic fault-tolerance features. With this, we achieve a communication subsystem that is adaptive in terms of real-time and reliability.

At the node level, DFT4FTT proposes a novel dynamic task allocation scheme for DES, focused on providing high reliability. More precisely, this scheme defines: how to monitor the environment and the system itself, how to detect when a reconfiguration is needed, and how to carry out said reconfiguration. The objective is to decide online which tasks are allocated to each node of the DES while fulfilling the requirements of the tasks. To achieve the real-time behaviour, the allocation scheme ensures that each task has enough computation resources in its destination node. Moreover, it relies on the network level to ensure that its communication also fulfils the real-time constraints. Reliability requirements are fulfilled using fault tolerance mechanisms. Specifically, we tolerate hardware faults affecting the nodes by means of a task replication scheme based on one we already developed for control applications [5]. Finally, note that, thanks to the online allocation of tasks, we can provide adaptive fault-tolerance.

Some of the key features of this level are: the adaptivity it provides in terms of the allocation of tasks; the mixed-criticality support; the fault tolerance features that allows it to seamlessly tolerate node faults and to recover lost replicas; the dynamic fault tolerance provided by the adaptivity; the modular design that allows to select the desired level of autonomy in the adaptivity; and the possibility of using machine learning algorithms to enrich the decision making.

In this paper we describe the ongoing work we are carrying out to design the mechanisms responsible for the management of the node level. Note that, although this work is defined in the scope of the DFT4FTT project, here we present the main ideas and we do not assume any specific technology. Consequently, this dynamic task allocation scheme can be applied in any ADES, as long as the underlying communication layer provides the necessary flexibility services.

II. Related work

There have been several efforts to design architectures for highly-reliable distributed embedded systems like DFT4FTT. For instance, the Delta-4 [6] was a 5-nation collaborative project that designed and implemented an open dependable distributed computing architecture. In Delta-4 fault tolerance is achieved by replicating the software components, or *capsules*, on interconnected nodes. For this, the Delta-4 architecture includes a software infrastructure called *Deltase* that allows to generate the replicated capsules, as well as to coordinate and manage them. Moreover, it contains a mechanism called *cloning* to restore the level of replication in case of a failure. That is, if a node fails, the capsule replicas executed in said node are automatically migrated to other non-faulty node.

Nevertheless, Delta-4 presents some limitations. First of all, although Delta-4 has mechanisms to start, stop and migrate replicated capsules, it does not consider adaptivity. Moreover, the migration of a given replica is restricted to the type of node it was generated for.

There are also multiple middlewares focused on providing adaptivity in the scope of the DES [7]–[9]. This kind of middlewares, apart from providing the generic middleware features, ensure the end-to-end QoS. In this context, the attributes associated with the QoS could be the latency or the throughput. These middlewares include mechanisms to: monitor the resources used (memory, CPU and bandwidth), monitor the QoS attributes, diagnose the cause of any violation of the QoS and reallocate the resources to restore the QoS.

Middlewares for ADES are focused on managing the resources of the system to achieve a specific QoS. With these mechanisms they can restore the service if some node fails. However, in systems with highly-reliable requirements this could not be enough, as there is some downtime. The solution here proposed, includes a task replication scheme that allows it to seamlessly tolerate the hardware faults affecting the nodes. Moreover, similarly to these middlewares, the level of replication can be recovered by means of task migration.

Apart from the previous middlewares, there is a recent middleware called iLand [10] based on FTT, that supports time-deterministic reconfiguration in distributed soft real-time environments. The characteristic feature of this solution is that changes can be triggered by: a user action, or external request; internal system decision, that is, an automatic request based on system monitoring; and application decision, that is, an automatic request based on application-specific conditions or programmed events.

To the best knowledge of the authors, there is no solution that integrates all the features presented in the current work.

- Dynamically allocation of tasks in a DES.
- Modular design that allows to select the level of autonomy in the adaptivity.
- Possibility to enrich the automatic reconfiguration with machine learning algorithms.
- Inclusion of a task replication scheme that allows it to seamlessly tolerate hardware faults affecting the nodes.

III. System Architecture

The DFT4FTT architecture is composed of various components (see Fig. 1): a network, several sensors and actuators, several computational nodes and an *Node Manager*.

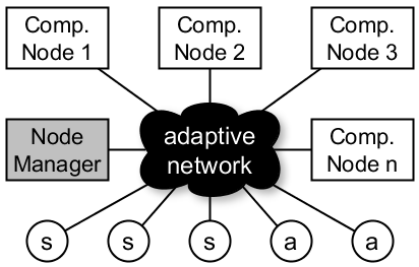

Fig. 1. System architecture.

The central point of the DFT4FTT architecture is the communication network, which interconnects all the other components. In order to properly support the dynamic allocation of tasks, this network should provide services for online managing the communications and their real-time attributes.

The Computational Nodes (CNs) are the components responsible for executing the tasks related to the purpose of the system. These nodes do not decide the tasks they execute. In contrast, is the Node Manager the one that determines online which tasks have to be executed in each of the CNs. From the point of view of the software, we implement a task replication scheme that uses active replication with distributed majority voting, in which critical tasks are replicated in different nodes and executed in a parallel and coordinated manner.

The sensors and the actuators (SAs) are the components responsible for providing information from the environment, as well as for acting on it. In this sense, they provide the CNs with the way to interact with the environment. It is noteworthy that, unlike many DES where SAs are connected to the nodes doing the processing, in this solution they are connected directly to the network. This makes SAs independent from the CNs which, in turn, makes the management of the tasks more flexible and the architecture more fault tolerant. On the one hand, now there is no restriction when allocating the tasks into the CNs since SAs can be accessed from any CN. On the other hand, SAs are now failure independent from the CNs.

The Node Manager (NM) is the component responsible for dynamically allocating the tasks into the CNs. The NM periodically monitors the environment and the system itself, identifies the situations where a reconfiguration in the system is needed, decides which tasks have to be started and/or stopped in each of the node and carries out these actions. All this operation will be further described in Sec. IV. Finally, note that, although the NM is represented as an independent physical component, it is possible to integrate it inside the network, for instance, inside a switch, similarly as we have done previously for other software components [4]. In fact, by doing this it can take advantage of its privileged position to better monitor the system.

IV. Design of the Task Allocation Scheme

As introduced previously, the core idea of this solution is the dynamic allocation of tasks, with different real-time and reliability requirements, into a set of available computational nodes, in response to changes in the environment or the system itself. The sources of changes we are interested in are: *environment*, operational conditions outside the system; *architecture*, operational conditions of the system; *phase*, part of the mission with associated list of necessary functionalities; and *human demands*, system requirements set by a human.

For this purpose, the task allocation scheme (TAS) provides applications with three services: *Monitoring*, to obtain the state of the system; *Decision*, to generate reconfiguration commands; and *Reconfiguration*, to implement the changes. As depicted at the bottom of Fig. 2, each of these services is provided by several processes, carried out by both the NM and the CNs in a distributed fashion.

Apart from that, we also propose the *Knowledge* and the *Wisdom Entities*, that is, two optional active entities that use and expand these services and processes to reconfigure the system in an autonomous manner, depending on application-dependent rules and with the assistance of machine learning algorithms, respectively. Next we describe more in depth the services provided by the TAS and the entities.

A. Task Allocation Scheme Services

The Monitoring service gives support to applications and the aforementioned entities to detect the relevant changes. The Monitoring service is provided by the NM and includes both the *Data Gathering* and the *Data Processing* processes.

The Data Gathering is responsible for extracting relevant data from the environment and the system itself. The sources of data we have considered are four. First, the values of the *application sensors*, that provide a view of the environment. Second, the values of *additional sensors*, installed in the system to complement the view provided by the application sensors. Third, the *network traffic*, from which we can determine the health of the hardware components conforming the system. Finally, the *votings* performed by the CNs, which provide a view of the health of the replicated software components.

Taking into account all these data, the Data Processing provides the *state of the system*. This state is defined by means of four different values. First, knowing the initial configuration of the system and observing the messages transmitted by the SAs and the CNs we can determine the *status of the architecture*, that is, which SAs and CNs are non-faulty and how they are interconnected. Second, sensor data allow us to know the environment and, thus, determine the *failure rate* of the hardware components. Third, by inspecting the network traffic we can determine the set of *tasks executed* by each of the CNs and their health, that is, if they are having problems voting. Finally, knowing the status of the architecture, the tasks being executed in each CN and the network traffic we can determine the *status of the resources*, that is, the amount of computational and network resources that are available.

The Decision service helps in analysing the state of the system provided by the Monitoring service, determining the changes to perform and, finally, instructing the proper reconfiguration to the NM. To support applications in this regard, the *Command Generation* process in the CNs allows to produce well-defined reconfiguration commands and then send them to the NM. Note that CNs must ensure that there are enough resources. The processes related to the analysis of the state of the system and the decision on the changes to perform will be discussed later, when describing the entities.

The Reconfiguration service allows to implement the changes and involves both the NM and the CNs. The *Configuration Updating* process in the NM accepts the reconfiguration commands and orchestrates the changes. The available operations are start or stop a given task in a specific CN, as well as to modify the real-time and fault-tolerance attributes of a given task. Note that, the modification of the real-time attributes make it possible to change the quality of service (QoS), or quality of control (QoC) in control systems, while the modification of the fault-tolerance attributes make it possible to change the number of task replicas. Note also that these changes not only imply changing the assigned computational resources, but also the associated communication resources. This is achieved with the assistance of the underlying communication network. The *Configuration Updating* process in the CNs support all this process at the CNs side.

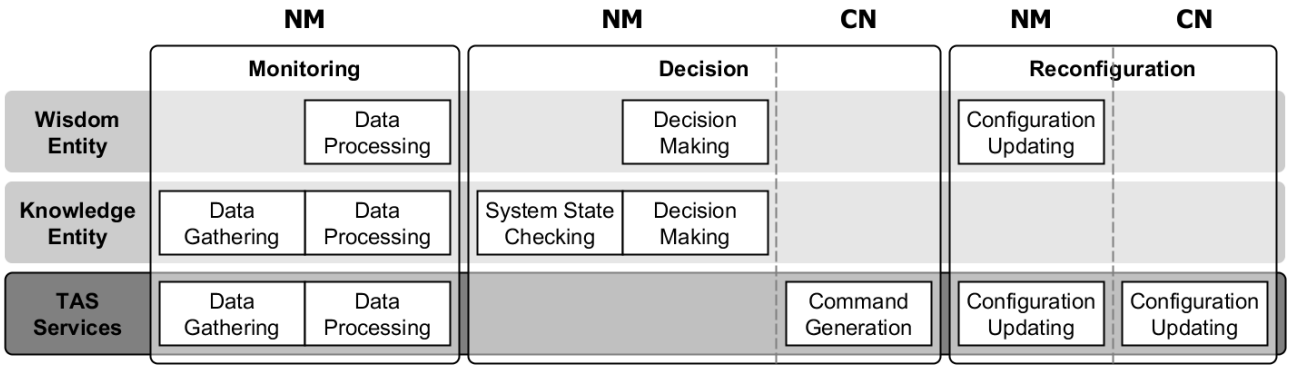

Fig. 2. Services provided by the task allocation scheme.

B. Knowledge Entity

The Knowledge Entity allows to carry out automatic reconfigurations to fulfil the requirements of the applications and to cope with predefined system changes like phase changes.

This entity extends the Data Gathering to also consider human instructions. This makes it possible for the Data Processing to automatically determine the phase of the mission and, thus, the specific needed functionalities at every instant.

In the Decision service the *System State Checking* process verifies that the *operational requirements* of the system are met, considering the current state of the system. The operational requirements are defined as the list of tasks, with their real-time and reliability requirements, that must be executed. This list contains the indispensable tasks of each phase plus the additional tasks triggered by the human instructions. In case the operational requirements are not met, for instance, due to a change in the phase, the *Decision Making* process is triggered. This process decides the sequence of changes that must be applied into the system to meet the new requirements. If the resources are not enough to meet all the requirements the process can decide to evict non-critical tasks. If this does not solve the problem, the process can enable a *degraded mode* in which the QoS or QoC of the tasks are reduced, upon a certain threshold, to make room for all the critical tasks.

C. Wisdom Entity

The Wisdom Entity plays a similar role as the Knowledge Entity but using machine learning mechanisms in order to improve the decisions taken for the reconfigurations [11]. This allows, for instance, to predict future needs of resources and, thus, achieve a better reactivity by allocating them in advance; or to infer the best configuration in situations beyond the ones that can be faced by the Knowledge Plane.

In the *Data Processing* process the data provided by the Data Gathering is further processed and stored to produce a proper input for the machine learning algorithm implemented in the *Decision Making* process.

The *Decision Making* process contains a self-supervised machine learning algorithm that learns from the information provided by the Monitoring service and the reconfiguration commands generated by the applications and the Knowledge Entity (if exists) [12], [13]. When the system starts running this algorithm just observes and tries to characterize the task allocation procedure. When its reconfiguration proposals start fitting the reconfiguration commands issued, its trustworthiness is considered to be high enough and, thus, they start be taken into account. This algorithm continues learning from its own proposals to permanently increase its trustworthiness.

Note at this point that the *Configuration Updating* process has to be slightly modified to take into account trustworthiness of the reconfiguration commands. More precisely, in case of a conflict in the reconfiguration commands, this process only implements the reconfiguration proposal of the learning algorithm if its trustworthiness is high enough.

V. Conclusions

In this paper we have described the general design of a dynamic task allocation scheme for highly-reliable adaptive distributed embedded systems, that integrates interesting features in terms of the fault tolerance and the adaptivity. On the one hand, this solution provides: a task replication scheme that allows to seamlessly tolerate permanent faults affecting the nodes; self-recovering mechanisms to reallocate tasks when a node fails; and support for adaptive fault tolerance to dynamically change the level of replication of the tasks in response to environmental changes. On the other hand, this solution is constructed following a modular design that allows to select at design time the desired level of autonomy in the adaptivity. Moreover, it enables the possibility of using machine learning algorithms to enrich the decision making.

As future work we will provide fault tolerance to the Node Manager by means of replication. Moreover, we will also study how to implement the algorithms for processing the data, checking the need for a reconfiguration and deciding on the reconfiguration. Finally, we plan to construct a prototype to prove the feasibility of this solution.

Acknowledgments

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

References

[1] F. D. Macías-Escrivá, R. Haber, R. del Toro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, 2013.

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

[3] P. Pedreiras and L. Almeida, "The flexible time-triggered (FTT) paradigm: an approach to QoS management in distributed real-time systems," in *Proc. Int. Parallel and Distributed Processing Symp.*, 2003.

[4] D. Gessner, J. Proenza, M. Barranco, and L. Almeida, "Towards a Flexible Time-Triggered Replicated Star for Ethernet," in *Proc. 18th IEEE Int. Conf. on Emerging Tech. and Factory Autom. (ETFA)*, 2013.

[5] S. Derasevic, M. Barranco, and J. Proenza, "Appropriate consistent replicated voting for increased reliability in a node replication scheme over FTT," in *Proc. 19th IEEE Int. Conf. on Emerging Tech. and Factory Autom. (ETFA)*, 2014.

[6] D. Powell, Ed., *Delta-4: A Generic Architecture for Dependable Distributed Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg.

[7] N. Shankaran and et al., "Design and performance evaluation of an adaptive resource management framework for distributed real-time and embedded systems," *Eurasip Journal on Embedded Systems*, 2008.

[8] B. Ravindran, L. Welch, and B. Shirazi, *Resource Management Middleware for Dynamic, Dependable Real-Time Systems*. Boston, MA: Springer US, pp. 69–82.

[9] Y. Zhang, C. Wandong, and Y. Wang, "Adaptive safety critical middleware for distributed and embedded safety critical system," *Proc. 4th Int. Conf. on Netw. Comp. and Adv. Inf. Manag. (NCM)*, 2008.

[10] M. Garcia Valls, I. R. Lopez, and L. F. Villar, "iLAND: An Enhanced Middleware for Real-Time Reconfiguration of Service Oriented Distributed Real-Time Systems," *IEEE Trans. on Ind. Inf.*, 2013.

[11] J. Davis, J. Hoffert, and E. Vanlandingham, "A Taxonomy of Artificial Intelligence Approaches for Adaptive Distributed Real-Time Embedded Systems," in *Int. Conf. on Electro Information Tech. (EIT)*, 2016.

[12] J. Hoffert, D. C. Schmidt, and A. Gokhale, "Evaluating Timeliness and Accuracy Trade-offs of Supervised Machine Learning for Adapting Enterprise DRE Systems in Dynamic Environments," *International Journal of Computational Intelligence Systems*, 2011.

[13] P. Vienne and J.-L. Sourrouille, "A Middleware for Autonomic QoS Management Based on Learning," in *Proc. 5th Int. Workshop on Software Engineering and Middleware*, 2005.