DREAMS about reconfiguration and adaptation in avionics

Guy Durrieu, Gerhard Fohler, Gautam Gala, Sylvain Girbal, Daniel Gracia Pérez, Eric Noulard, Claire Pagetti, Simara Pérez

▶ **To cite this version:**

Guy Durrieu, Gerhard Fohler, Gautam Gala, Sylvain Girbal, Daniel Gracia Pérez, et al.. DREAMS about reconfiguration and adaptation in avionics. ERTS 2016, Jan 2016, Toulouse, France. hal-01258701

HAL Id: hal-01258701
https://hal.science/hal-01258701

Submitted on 19 Jan 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

DREAMS about reconfiguration and adaptation in avionics

Guy Durrieu* Gerhard Fohler† Gautam Gala† Sylvain Girbal ‡ Daniel Gracia Pérez‡ Eric Noulard*
Claire Pagetti* Simara Pérez Zurita †
*ONERA - France †Technische Universität Kaiserslautern - Germany ‡Thales TRT - France

***Abstract*—The paper describes the reconfiguration approach implemented in the DREAMS middleware to cope with failures and how the concepts are tested on an avionic demonstrator.** [1]

I. Introduction

The DREAMS [Oa13] (Distributed REal-Time Architecture for Mixed Criticality Systems) FP7 project addresses the design of a cross-domain architecture for executing applications of different criticality levels in networked multicore embedded systems.

A. General overview of DREAMS

A DREAMS architecture is composed of several multi-core chips (such as Freescale T4240 [Fre14]) connected through a TTEthernet network [KAGS05]. The DREAMS middleware is in charge of:

1) Ensuring strong temporal and spatial partitioning;
2) Supporting adaptation strategies for mixed-criticality systems to deal with unpredictable environment situations, changes in resource availability, and occurrence of faults;
3) Delivering virtualization technologies for ease of designing.

The DREAMS development methodology and tools are based on model-driven engineering enabling mapping and scheduling of mixed-criticality applications. Three demonstrators, that encompass a broad range of application domains (namely avionics, wind power and healthcare) will be developed and will highlight the DREAMS results.

The project, started in October 2013 with a duration of 4 years, is in its mid-term progress. At this stage, the basic software blocks have been developed and will be integrated next year in the demonstrators.

B. Objective and contributions

This paper focuses on reconfiguration and adaptation strategies and their implementation in the avionic demonstrator. Those strategies only take place upon failures, with the purpose to bring the system back to a safe functioning state. We consider two types of failures:

1) **A permanent core failure**. Intensive integration of small devices on chip increases the *permanent failures* occurrence due to various phenomena such as aging, wear-out or infant mortality [Bor05]. When a core is halted, the partitions executing on the failed core are re-allocated according to pre-computed configurations. We then speak of *reconfiguration*;
2) **A temporal overload situation**, resulting in deadline miss without corrective action. Such a situation may occur because the resources are over-utilized in the nominal mode. However, the timing constraints are respected in degraded modes, that consist in interrupting or degrading the execution of *best-effort applications*. When a *critical application* (i.e. that is not best-effort) detects an internal deadline overrun, the execution moves temporarily the best-effort applications to a degraded mode. We then speak of *adaptation*.

In the following, we describe the resource management proposed in the DREAMS middleware and we define formally the notions of reconfiguration and adaptation (see Section II). We then detail the reconfiguration strategies defined for mitigating the core failures (see Section III) and the adaptation approach for mitigating the temporal overload situations (see Section IV). Finally we give the main ideas of the implementation for the avionic demonstrator and the results obtained by simulation (see Section V). Related works are discussed in Section VI.

II. Resource management in DREAMS

Resource management is a core service provided in the DREAMS middleware for system wide adaptability of mixed criticality applications. The approach is based on the Matrix framework [RF07], but adapted to platforms in which multiple multi-core chips exist and applications can have several criticality levels. Furthermore, the concept of service levels in ACTORS [BBE+11] is extended in DREAMS for its application on virtualized hardware resources instead of applications. The main goals of the integrated resource management are:

- Reconfiguration of a mixed-criticality system upon foreseen and unforeseen changes in its operational and environmental conditions.
- Adaptability mechanisms for securely modifying over the system without interrupting or interfering with its execution.

A. Structure of resource managers

Practically, the resource management services are realized by a Global Resource Manager (GRM) in combination with a set of Local Resource Managers (LRM). The GRM gathers information from the LRMs and provides new configurations

[1]The research leading to these results has received funding from the European FP7-ICT project DREAMS under reference n° 610640.

for the virtualization of resources (e.g., partition scheduling tables or resource budgets). The GRM configuration can include different pre-computed configurations of resources (e.g., time-triggered schedules) or parameter ranges (e.g., resource budgets).

Local resource management services consist of three major parts: Resource Monitors (MONs), Local Resource Schedulers (LRSs) and Local Resource Managers (LRMs). The MON monitors the resource availability and timing of components (e.g., detection of deadline violations). The LRS performs the runtime scheduling of resource requests (e.g., execution of tasks on processor, I/O requests) based on the configuration set by the LRM. The LRM either adopts the configuration from the GRM to particular resources (e.g., processor core, memory, I/O) or selects a new configuration from the ones available and reports state of the resource (from MON) to the GRM.

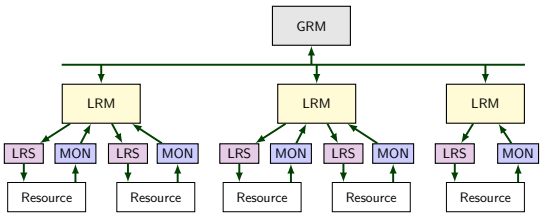

Figure 1. Interaction between resource managers

The LRM and the GRM can be organized in a hierarchical or flat architecture. The flat architecture, shown in Figure 1, consists of a GRM which controls and supervises all LRMs and has a complete view of the system. All LRMs are placed at the same level and they communicate directly to GRM regardless of which resource they monitor or where they are physically located. In the hierachical architecture the LRMs can control underlying LRMs.

B. Implementation choices

DREAMS middleware relies on time and space partitioning principles [Rad05]. In this paper, we consider that those principles are implemented at the chip level by the XtratuM hypervisor [MRC+09], which is a technology involved in the project. Therefore, applications will be executed by a set of partitions. A partition is defined by one or multiple slots, each with a start time and a length. Inside a slot, several tasks can be executed. In the sequel, we will use the color code shown in Figure 2.

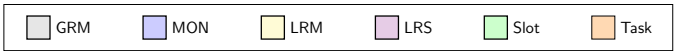

Figure 2. Legend

We consider mixed-critical systems where we differentiate two types of application.

Definition 1 (Application model)**.** An application can be a:

- *critical application*. Such an application must respect its timing constraints and in particular the WCET must fit in the allocated slots. Moreover, it cannot be stopped apart if the application encounters an internal error or if the executive layer fails. A critical application app is defined as a set of periodic or sporadic tasks $app = \{\tau_i = (C_i, AET_i, T_i)\}$ where C_i is the WCET, AET_i is the average execution time and T_i is the period or minimal inter-arrival time;
- *best-effort application*. Such an application has less strong constraints. We accept to interrupt them as long as a minimal QoS (quality of service) is ensured. A best-effort application is defined as $app = (U_i, AU_i)$ where U_i is the worst-case asked utilization and AU_i is the average utilization.

A configuration consists in defining temporal slots on the multi-core and mapping the applications in the slots.

Definition 2 (Configuration)**.** A configuration (also denoted *plan* in the hypervisor terminology) consists of:

- a *major cycle* (MaC), the length of which is denoted *MaC_length*;
- a set of slots sl_i distributed over the cores and the MaC. A slot is defined as $sl_i = ([s_i, e_i], n_i)$ where s_i is the start time, e_i is the end time and n_i is the number of core where the slot is allocated;
- a mapping of the jobs of critical applications in the slots. Jobs are unrolled on the MaC and we know for all job $\tau_{i,j}$ in which slot sl_k it belongs to. We know moreover in which order are executed the jobs inside a slot;
- a mapping of best-effort applications in the slots. For instance, app_i is executed in the slots $sl_{j_1}, \ldots, sl_{j_p}$.

C. Definition of the notions of reconfiguration and adaptation

A reconfiguration consists in moving from one configuration to another and this happens when a core has failed. An adaptation consists in degrading a configuration and this occurs when a temporal overload situation happens. Adaptions are handled locally by the LRM whereas core failures may be recovered locally by the LRM or globally by the GRM.

a) Permanent core failures: When a core has failed, the partitions hosted on it are no longer executed. Such a situation can be mitigated by an active redundancy (if some other resource executes the same partitions) or by applying a reconfiguration. Due to the high number of cores provided by a DREAMS platform and the overall resource managements, we decide to incorporate reconfiguration capabilities.

b) Temporal overload situations: The chapter 8 of [But97] focuses on the overload conditions, that are *critical situations in which the computational demand requested by the task set exceeds the time available on the processors, and hence not all tasks can complete within their deadlines*. Such a situation can result for several reasons, e.g. environmental solicitations or fault of peripheral devices or cohabiting applications.

In the DREAMS project, we consider IMA platforms where such problematic situations are usually contained thanks to the temporal isolation. However, we decided to leverage this

restriction in order to increase the overall utilization of the multi-core chips. Indeed, we observed that when computing an upper bound of applications WCET on a multi-core chip and reserving this amount of time for all of them leads to an over-provisioning of the platform. As a matter of fact, this WCET is rarely reached and most of the time, the average execution time (AET) is much below the capacity of the platform. This is the reason why we accept a multi-core to be over-utilized by the applications. Our model is detailed in the definition below.

Definition 3 (Under-provisioned platform)**.** Any multi-core can be over-utilized in the following way:

- $\sum_i \frac{C_i}{T_i} + \sum_j U_j >$ *number of cores*: the overall utilization exceeds the multi-core capacity;
- $\sum_i \frac{AET_i}{T_i} + \sum AU_j \ll$ *number of cores*: the overall average utilization is much below the multi-core capacity;
- $\sum_i \frac{C_i}{T_i} <$ *number of cores*: the overall utilization for the critical applications fits the multi-core capacity.

This means that the best-effort applications are those leading to the overtaking of the provisioning. This situation will be handled as proposed in [KPR+14], which means that we will monitor regularly the critical applications and if an internal deadline is exceeded, the best-effort applications will be interrupted and resumed once the critical applications are not any longer endangered.

Note that GRM only makes global reconfiguration decisions when necessary, but it is not required for the continuous operation of the system. The unique failure mode considered for the GRM is the *loss*, due to the permanent failure of the hosting core. Thus, in case of GRM failure, the overall system dependability is not compromised as the system will still keep on executing; just no new global reconfigurations will be possible.

D. Interaction between GRM and LRMs

The interaction between resource management components takes place via Sampling and Queuing ports (provided by the hypervisor). As shown in figure 3, three channels are created between each GRM-LRM pair:

1) Updates channel: For LRM to send resource status updates to the GRM and request for global reconfiguration.
2) Orders channel: For GRM to send reconfiguration messages to the LRM.
3) Membership channel: Each LRM periodically sends a live-signal to the GRM via this channels for the membership purposes.

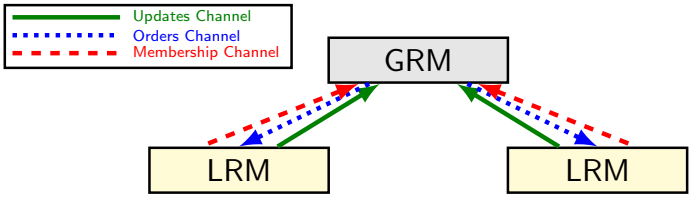

Figure 3. Comunication between RM components

The properties of the RM communication channels are summarized in table I. We remind that TTEthernet protocol [KAGS05] allows several types of traffic:

- rate constraint (RC) traffic: bandwidth guarantee for each application is predefined and delays and temporal deviations have defined limits.
- time triggered (TT) traffic: messages are sent over the network at predefined times and take precedence over all other traffic types.
- best-effort (BE) traffic: it follows the methods of classical Ethernet networks. There is no guarantee whether and when the BE messages can be transmitted, what delays might occur and if they arrive at the recipient.

Table I
SUMMARY OF RESOURCE MANAGEMENT COMMUNICATION CHANNELS

Communication Channel	Port Type	TTEthernet Traffic	Source	Destination
Updates	Queuing	TT	LRM	GRM
Orders	Sampling	TT	GRM	LRM
Membership	Sampling	TT	LRM	GRM

III. RECONFIGURATION STRATEGY IN CASE OF A CORE FAILURE

When a failure occurs, a detection mechanism must detect the problem and a system recovery procedure must bring the system to a correct state. In the DREAMS project, the detection is based on monitoring (MON) at the multi-core level and recovery at the LRM or GRM level. The recovery procedure is based on pre-defined mode changes executed by the LRM. This entails that a set of possible configurations is computed off-line and that a reconfiguration consists in moving from one configuration to another. The transition steps between mode changes must be safe.

A. Reconfiguration graphs

Since core failures may be recovered locally by the LRM or globally by the GRM, we need to distribute the view of the current configuration between the different stakeholders. The current configuration is represented as the combination of the local configurations. The GRM has an up-to-date system wide vision of the current configuration and stores all the admissible reconfigurations. Each LRM and each switch store a local reconfiguration graph detailing local reconfiguration and global mode changes asked by the GRM.

Definition 4 (Local reconfiguration graphs)**.** A (local) reconfiguration graph is a tuple $\langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ where:

- Q is a finite set of configurations;
- q_0 is the initial configuration;
- $\rightarrow \subseteq Q \times Q$ is the set of local transitions from one configuration to another;
- $\dashrightarrow \subseteq Q \times Q$ is the set of transitions from one configuration to another requested by an other entity.

A reconfiguration graph is stored in each LRM, in each switch and in the GRM. In an LRM, plain arrows represent local

decisions while dashed arrows represent decisions provided by the GRM. In the GRM, plain arrows represent local decisions while dashed arrows represent decisions made by some LRM. A switch graph only contains dashed arrows and reconfiguration requests are triggered by the GRM or some LRM.

Since network switch routing tables must be reconfigurable, we must define the reconfiguration strategies for the different types of traffic:

- For rate constraint (RC) traffic, the VLs are defined with their BAG and maximal packet size. Therefore, if an application is reconfigured on the same multi-core by a local reconfiguration then it has no impact on the routing table. If the reconfiguration is global, then several routing tables must be pre defined.
- For time triggered (TT) traffic, it depends whether the instant of emission of packets is related to the offset of the partition slot. If not, then the same reasoning as for RC traffic applies. Otherwise, we must consider the link between offsets of local reconfigurations and network TT messages scheduling.
- For best-effort (BE) traffic, the same reasoning as RC traffic applies.

The GRM stores the complete view of the system which is represented as a global reconfiguration graph.

Definition 5 (Global reconfiguration graph)**.** A global reconfiguration graph is a tuple $\langle Q, \rightarrow, \dashrightarrow, q_0 \rangle$ which consists of the product of all local reconfiguration graphs $\langle Q^i, \rightarrow^i, \dashrightarrow^i, q_0^i \rangle$ from the LRMs and the switches together with the GRM local graph $\langle Q^G, \rightarrow^G, \dashrightarrow^G, q_0^G \rangle$. More precisely:

- $Q = Q^1 \times \ldots \times Q^n \times Q^G$,
- $q_0 = (q_0^1, \ldots, q_0^n, q_0^G)$,
- $\rightarrow \subseteq Q \times Q$ is defined as

$$((q^1, \ldots, q^n, q^G), (p^1, \ldots, p^n, p^G)) \in \rightarrow$$
$$\Longleftrightarrow$$
$$\begin{cases} \exists i \in \{1, \ldots, n\}, \\ \quad (q^i, p^i) \in \dashrightarrow^i \wedge \forall j \neq i, q^j = p^j \wedge q^G = p^G \\ or\ (q^G, p^G) \in \rightarrow^G \wedge \forall j, q^j = p^j \end{cases}$$

- $\dashrightarrow \subseteq Q \times Q$ is defined in a similar way than $\rightarrow$ by replacing $\dashrightarrow^i$ with $\rightarrow^i$ and $\rightarrow^G$ with $\dashrightarrow^G$.

Figure 4 illustrates the reconfiguration graphs stored by the different resource managers and switches. The chip on the right hand side has several pre-defined configurations named from C1 to C7. The switch on the right hand side has several pre-defined configurations named from S1 to S3. For the GRM, we only show the global reconfiguration graph as the product of all local reconfiguration graphs.

B. Local vs. global decisions

Example 1 (of local reconfiguration)**.** Failure f1 (a core halt) occurs in the multi-core T1. According to the reconfiguration graph of T1, the LRM will move to configuration C2. A message must be sent to the GRM so the latter can maintain an updated configuration. This is shown in Figure 5.

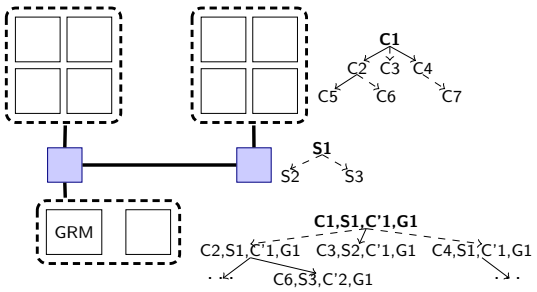

Figure 4. Distributed reconfiguration graphs

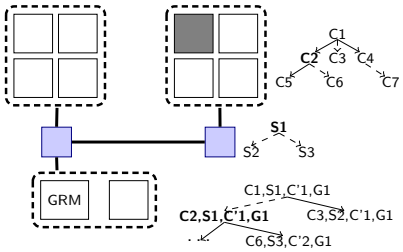

Figure 5. Local reconfiguration

Example 2 (of global reconfiguration)**.** Failure f5 (a core halt) occurs in the multi-core T1. According to the reconfiguration graph of T1, the LRM has no solution. Thus it informs the GRM. The GRM can apply a global reconfiguration: applications running on the failed core of T1 will be reconfigured in T2. The GRM informs (1) T2 to load and execute the applications, (2) T1 that a reconfiguration is applied, (3) the switches to reconfigure the routing tables (messages are emitted by T2 and not T1). An ack by T2 may be expected. This is shown in Figure 6.

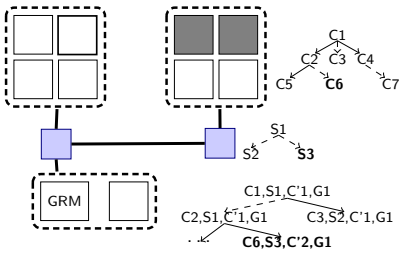

Figure 6. Global reconfiguration

C. Detailed specification

In this section, we explain how the resource management services are implemented at the chip level.

1) MON: executes a service regularly in each core to detect the core's health. If the core is working correctly, the service writes to a shared structure that everything is fine. Otherwise, if the core has failed, the service is not activated and is not able to update the shared structure.

The cores update asynchronously the structure at distinct pre-defined times and check the other cores status at that

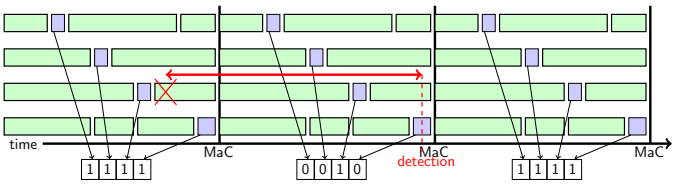

Figure 7. Example of core failure detection on a quad-core.

moment. For example let us consider a quad-core where the MON service is executed in each core only once per major cycle (MaC), see Figure 7. Let us suppose that one of the cores fails just after the MON execution (represented as a red cross on core 3). The detection will be done by the core 4 in the next MaC (the time needed for the detection is shown as a red arrow).

2) LRM: Once a core failure has been detected by the MON, the latter informs the LRM. The time between the detection by the MON and the execution of the LRM has a direct influence on the response time for reconfiguration. This is the reason why we impose the MON and LRM to have pre-defined slots next to each other in order to minimize the delay between the detection and decision. Figure 8 gives an example of a decision of the LRM after the detection of the failure of core 2.

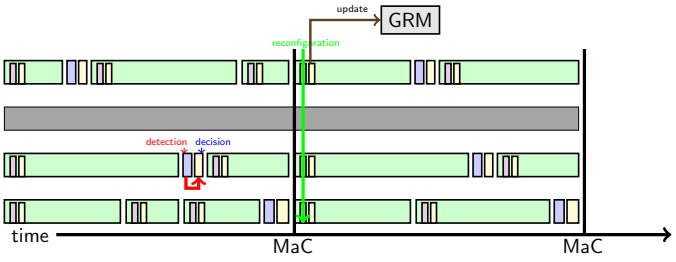

Figure 8. Example of LRM decision after a core failure detection.

Once the LRM is informed by the MON service of a core failure, it has two possibilities:

- a local reconfiguration is possible according to its local reconfiguration. In that case, it asks the LRS to change the plan at the end of the MaC for the new configuration one. The reallocated partitions are re-started in a default state, no context has been stored from the previous executions. The unchanged partitions continue their execution transparently. This transition step is then safe;
- no local reconfiguration can recover from the situation. In that case, the critical tasks are locally reconfigured in priority if possible (pre-computed configuration) while some best-effort applications may be removed. Then, the LRM informs the GRM that some applications cannot be hosted any longer on the multi-core platform and it is up to the GRM to find a global reconfiguration.

The DREAMS project requirements state that a critical application cannot split onto different cores of a multi-core. Thus, when a global reconfiguration must be taken, complete applications are thus reconfigured on different cores. A future work could consider to parallelize the applicative code onto different cores, but at the price of modifying the applicative code.

3) LRS: The LRS is more detailed in section IV-B3, because it plays a more important role for the temporal overload situations. The LRS is in charge of scheduling the tasks inside the slots. For the core failure case, it just reads the current configuration and applies it.

IV. Adaptation strategy in case of a temporal overload situations

In the DREAMS project, we under-provision the platform to increase the average performance. Such an approach can in some cases lead to problematic situations where critical applications may overrun their deadlines. To forbid this timing failures, a detection mechanism is in charge of analyzing intermediate deadlines and adapt the processor demands by interrupting the best-effort applications.

A. Adaptation tables

An adaption consists simply in interrupting the best-effort applications. It is therefore sufficient to store statically the partition identifier of the best-effort applications. Since the adaptation mechanisms are combined with the reconfiguration capabilities due to the core failures management, those identifiers must be stored for all reachable configurations.

Definition 6 (Adaptation table)**.** An adaptation table consists, for each configuration defined in the reconfiguration graph, of a list of applications.

B. Detailed specification

In this section, we explain how the resource management services are implemented at the chip level.

1) MON: extends the deadline warning detection method described in [KPR+14]. In this initial work, only standard tasks sets were considered and the schedule consisted in executing a task alone on a core. In the DREAMS project, we consider partitions slots and the MON/LRM/LRS components. The idea is that each critical application monitors its execution and checks if the application is in danger of overrunning its deadline. If it is the case, then the MON service signals to the LRM that a deadline overrun will probably occur.

The partition slots for critical applications contain internal *observation points* which are defined off-line and correspond to the moments where the MON is executed. We choose to monitor the temporal behaviour between tasks in the slot. This way we do not modify the partition code. This illustrated in Figure 9.

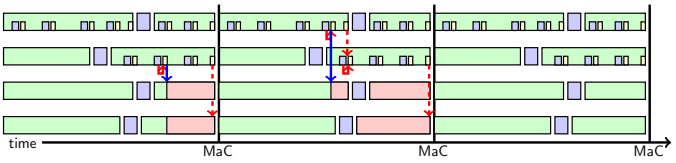

Figure 9. Example of internal deadline failure adaptation when LRM inside critical tasks.

The monitoring checks if the interferences of the low criticality tasks can be tolerated by verifying a safety condition. The safety condition in the initial work [KPR+14] consisted in checking that in the next observation point we would still have time to switch to the *degraded mode* (or isolated mode, in the sense that only critical applications may run). This required numerous information, such as the remaining WCET of the partition *Part* in isolated execution from the observation point x until the end. Thanks to the positioning of observation points between tasks, the safety condition can drastically be simplified as shown in Eq. 1.

$$\mathsf{ET}(x) \leq \textit{internal deadline}(x) \quad (1)$$

where $\mathsf{ET}(x)$ is the monitored execution time of *Part* until point x and *internal deadline*(x) is a pre-computed constant giving the maximal possible internal deadline.

2) LRM: stores the adaptation graphs and knows which applications must be suspended. This action is immediate (compare to the reconfiguration which occurs at the next MaC). Suspended applications are re-started once all running critical tasks have not asked to move to the degraded mode.

3) LRS: The LRS starts its execution as soon as a partition slot starts. The first time the LRS is executed (typically during plan 0 schedule of the hypervisor) it launches the application initialization, which sets up its internal state for execution. After that, the LRS initializes the application tasks schedule during the different slots and plan configuration.

Afterwards during the major cycles the LRS is executed at the beginning of each slot and it launches a predefined and sequential list of partition/application tasks for that slot, and once all the tasks have been executed the LRS stops its execution, even if time remains in the current partition slot. Note that an LRS execution can span multiple partition slots, but to facilitate the LRS for critical partitions comprehension we will always suppose that a LRS execution starts and finishes in the same partition slot.

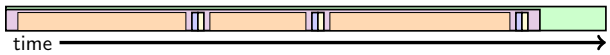

Figure 10. Example of critical partition slot execution.

Figure 10 shows an example of critical partition slot execution under the control of the LRS. In between the execution of two tasks the MON and the LRM are executed to:

- the MON execution collects the performance monitors of the just executed task and the current execution time of the slot,
- the LRM execution determines if an adaptation is needed.

V. Avionic demonstrator

The DREAMS architecture avionic demonstrator will highlight the reconfiguration capabilities of the middleware. The demonstrator combines critical applications with non-critical applications using heterogeneous multi-core platforms, connected using a wired network.

A. Applications involved in the demonstrator

Figure 11 shows the five applications/functions deployed in the avionics demonstrator, three critical ones and two non-critical. The critical applications are: (1) a Flight Management System (FMS, previously described in [DFG+14]), (2) a Display Management System (DMS), and (3) a Sensors Data Provider (SDP). The non-critical applications are: (1) an In-Flight Entertainment (IFE), and (2) the panels.

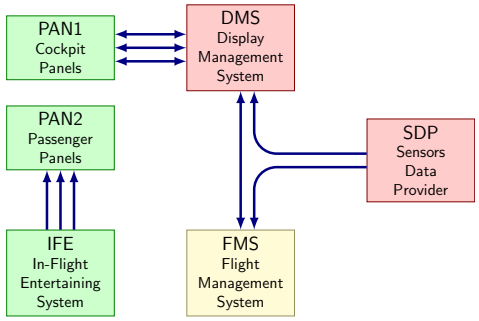

Figure 11. Inter-function communication in the avionic use-case

The FMS aims at performing in-flight guidance of aircrafts, which is based on the use of *flight plans* selected before departure, either by the pilot or a dispatcher for airliners. A flight plan includes basic information such as departure and arrival points, in-flight waypoints, estimated time en route, alternate landing airports, expected weather conditions, and so on. The SDP is the application responsible of collecting the sensors signals such as the GPS or the anemo-barometric probes. The SDP packetizes the sensors data and sends them to the FMS and the DMS. The DMS manages the information to be displayed on the cockpit panels for the pilots. The DMS also takes care of sending pilots commands to the FMS. While the cockpit panels are typically highly critical applications, their study is out of the scope of the DREAMS avionics demonstrator, so they are considered as non-critical. The IFE is connected to the passenger panels to broadcast video streams.

Table II
AVIONIC USE CASE SPECIFICATIONS

Function	DAL	Max unavailability	Number of tasks (periodic, aperiodic)
FMS	B	600ms	26 (10, 16)
SDP	A	600ms	5 (4, 1)
DMS	A	1000ms	7 (6, 1)
IFE	E	∞	NC
Panels	E	∞	NC

Table II summarizes the specification of each function in terms of criticality level (DAL), Maximal Unavailability, and a brief task set description. The Maximal Unvailabilty corresponds to the maximum allowed time to perform reconfiguration (maximum suspended time for the task due to a failure). The In-Flight Entertainment and the panels are implemented in commodity computers with standard OS (i.e., Linux) or in

non-critical partitions, and as such the task description is not considered (NC) in this study.

B. Demonstrator platform

Three different computing platforms are used for the deployment of the different applications:

- **Freescale T4240** [Fre14] (see Figure 13): The T4240 is a 64bit PPC architecture with 12 cores organized in 3 clusters of 4 cores interconnected connected through a propietary NoC to 3 different memory controllers, each one with a dedicated L3 memory cache. A PCIe TTEthernet card is also attached to the T4240 to satisfy the network requirements.

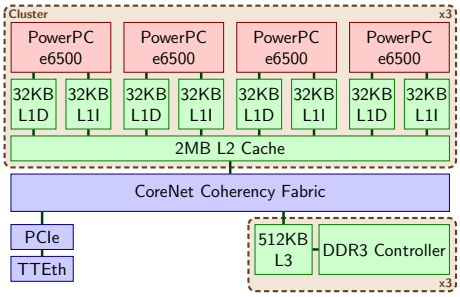

Figure 13. PowerPC T4240 architecture

- **DREAMS Harmonized Platform** [Oa13]: The Harmonized Platform is a development board with Xilinx Zynq-7000 SoC containing ARM Cortex-A9 cores and an FPGA. The DREAMS hardware solution like the Spidergon NoC, enhanced NoC interfaces and TTEthernet controller are implemented on the FPGA. Additionally, three Microblaze soft processor cores are connected to the NoC. The DDR memory controller can also be accessed by all the computing resources via the NoC.
- Regular PC: Regular PCs are used to deploy non-critical applications like the panels.

The applications are run on top of the XtratuM hypervisor enhanced with the DREAMS solutions, i.e., the MON, LRM and the GRM among others. Figure 12 shows one of the targeted deployments of the applications over the aforementioned hardware platforms.

The communications between the different computing platforms are ensured by: two TTEthernet switches, two PCIe TTEthernet cards and the TTEthernet controller embedded in the DREAMS Harmonized Platform. Regular ethernet cards are used in the PCs, as the applications on those systems don't require any safety level communication (i.e., time triggered or rate constrained). The demonstrator mixes the three types of traffic supported by TTEthernet:

- best effort for the communication from/to non-critical applications,
- and time triggered and/or rate constrained for the communication between the critical applications and the communication between the DREAMS services, as the communication between GRM and LRMs.

C. Results

Currently, the fault tolerance mechanisms have been implemented in XtratuM. But the hypervisor has not yet been ported on the T4240. Therefore, we could not run experiments on the avionic demonstrator. Instead, we made several simulations and prepared a series of fault-injection scenarios to validate the approach.

1) Reconfiguration graph and adaptation table: The initial configuration q_0 in the T4240QS of the left hand side of Figure 12 is defined as $\langle MaC\_length = 200ms, \{sl_i\}, alloc\rangle$ as shown in Figure 14 whereas TTE stands the TTEthernet driver.

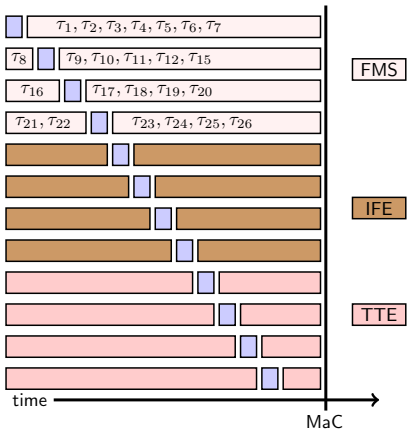

Figure 14. Initial configuration

The adaptation table for this configuration is given by

```
const int adaptation_point[NB_PARTITION][NB_MAX_POINT]={
  {20,  45, 100, 130, 170, 190},
  /* partition 0: 6 observation points,
     max time to reach point 1 = 20 */
  {-1,  -1,  -1,  -1,  -1,  -1},
  /* partition 1: 0 observation point */
  {40,  70,   90, 130,  -1,  -1},
  {-1,  -1,   -1,  -1,  -1,  -1},
  {80, 110,  150,  -1,  -1,  -1},
  {20,  -1,   -1,  -1,  -1,  -1},
  {90, 120,  180,  -1,  -1,  -1}
};
```

2) Fault-injection scenario: We have defined some scenarios that will be used to evaluate the DREAMS Local Resource Management services. The target architecture in all the scenarios will be based on the avionic demonstrator.

a) Scenario of double core failures: The purpose of the scenario is to test the local and global reconfiguration capabilities. The scenario is similar to the one detailed in examples 1 and 2 of Section III-B. Two faults are injected: (1) a core fails on the multi-core 2 leading to a local reconfiguration; (2) a second core fails on the same multi-core leading to a global reconfiguration whereas DMS is reconfigured on the second multi-core. To inject the fault, the MON will be modified not to update the share structure. The objective of this scenario is to:

- check that the LRM adaptation capabilities do not affect the safety of critical applications which were not hosted on failed cores;

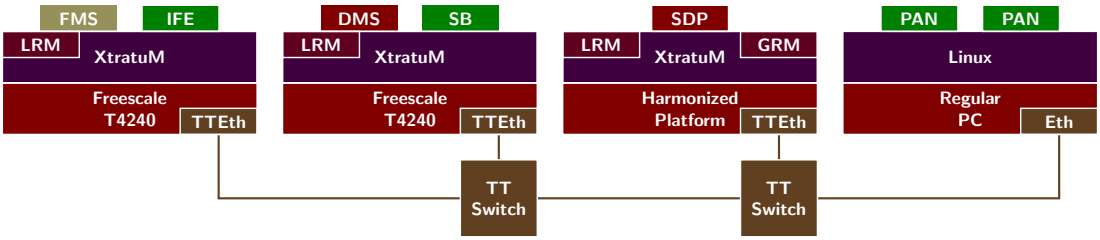

Figure 12. Avionic demonstrator architecture

- determine the gain due to local reconfigurations versus global ones;
- compare several timing parameters to improve the reconfiguration response times.

b) Scenario of temporal overload situation: The second scenario aims at testing the LRM capacity to interrupt low criticality tasks. The scenario consists in (1) detecting a deadline overrun in the FMS partition of multi-core 1; (2) interrupting the IFE execution after the detection until the end of the FMS slot. The objective of this scenario is to:

- check that the LRM adaptation capabilities maintain the predictability of the safety of critical applications;
- compare several timing parameters to assess the interruption response times and variability of the execution time of the safety critical applications.

3) Simulation: Since XTRATUM is not ported yet on the T4240QS, we run the scenarios with the QEMU simulator. The observed behaviours were those expected. In the next months, we will port the work on the real target.

VI. RELATED WORK

Reconfiguration for avionic platform has been proposed in Asaac [ASA04] project for military aircrafts; Diana [EJS+10] and Scarlett [PBB+12] for the civil domain. In all cases, reconfigurable IMA was able to change the configuration of the platform by moving applications hosted on a faulty computing module to spare computing modules. The main objective of such an extension was to reduce the cost of unscheduled maintenance and to improve the operational reliability of the aircraft while preserving current safety levels. In Diana the approach was distributed while in Scarlett the reconfiguration was centralized. In DREAMS, a module is based on a multi-core architecture and failures depend on this new hardware, and while the global reconfiguration of the system is centralized like in Scarlett the modules can perform local reconfigurations when one of the cores fails.

a) Permanent failures: The standard approach to deal with permanent failures is based on replicating applications and components into multiple copies. Such redundancy can be achieved via hardware or software mechanisms. Since the avionic demonstrator relies on COTS multi-core, only the second case could be considered. Most existing approaches target anomalous behaviours. For instance, the authors of [SHLR+09] present the *symptom based detection and diagnosis* principle, developed during the SWAT (SoftWare Anomaly Treatment) project, to manage faults in multi-core architectures running multi-threaded software. For permanent fault, the detection algorithm is based on a deterministic replay to diagnose the faulty core and the run-time is in charge of isolating the failed core.

The authors of [GLSS01] describe a fault-tolerant scheduling approach to support permanent core failures but they do not target a real software implementation.

b) Temporal overload situation: The objective of run-time monitoring is to check on-line, during the real execution, the timing behaviors of the system and verify if they are compliant with an abstract view of the expected behavior. If the system diverges from the specification, then a recovery may be applied.

In [BLS06] and [RRF10], the timing specifications are expressed with Timed Linear Temporal Logic (TLTL) formulas. Practically, timed automata are used to implement the valid behaviors and the decision layer stores the automaton description including the location invariants and the transition table. The verification function works for each event as follows: either it is valid and the execution continues or the event is invalid, in which case a recovery procedure or an error is called. Such an implementation requires a strong synchronization between the application and the monitor, and in particular it is necessary to ensure mutual exclusion. In [BFR13], the monitoring strategy has been extended for multi-threaded code. The monitor is decomposed into pieces that apply local detection while exchanging messages to ensure a coherent checking.

In the automotive domain, tasks can exhibit a dynamic real-time behavior, e.g. the period depends on the engine speed. This variability leads to a continuous change in the configurations taken into account by the OS schedule on the processors. We then speak of multi-mode applications that can switch between different operational modes at run-time. Such a change of rate may make the system unschedulable and the engine control inefficient or even unstable. The authors of [NES12] propose mode changes without violating timing constraint by pre-computating the possible behaviours. The approach consists in analyzing all potential run-time scenarios and study in details the critical ones. The possible recoveries are the following: degraded functional execution (with restricted WCET), abort or suspend low criticality tasks.

Several approaches propose resources reallocation based on information derived from monitoring their utilization, e.g. the memory accesses. For instance, in [NPB+14] interference-

sensitive WCETs are computed based on a preliminary analysis of the resource usage of tasks. The shared resources are off-line partitioned among tasks. A run-time monitoring device observes the resource usage of each task and suspends the task that overtakes the allocated capacity. In [NP13] the approach is extended by allowing safe dynamic changes in the resource partitioning, when resources are underutilized. In [YYP+13] an approach has been developed to reserve memory accesses for critical tasks. A run-time controller has been implemented which regulates the accesses to the shared memory and ensures temporal isolation among tasks. An off-line profiling technique has been proposed in [MDB+13] which finds the most frequently accessed memory pages in a task. Then, this information is used to modify the variables position in the shared caches in order to reduce the interferences.

VII. Conclusion

We have described the main ideas of the reconfiguration and adaptations strategies proposed in the DREAMS middleware. In the next year, the building blocks will be ported on the avionic demonstrator and the fault injection scenarios will be run on it.

The implemented monitoring techniques are simple and can be improved by defining an adapted set of rules depending on the current configuration. This is the idea developed in [GPBB08] where a safety mode automaton is constructed from the system and the environment. We will study how such ideas could be applied to increase the quality of the reconfiguration. Concerning the adaptation, the QoS for the best-effort applications has not been investigated yet and this will be also an axis of future work.

Finally, this article has addressed the technical aspects of the reconfiguration and adaptation strategies. However, in order to be applied to industrial solutions, the certifiability of the approach requires further study.

References

[ASA04] ASAAC. ASAAC final draft of proposed guidelines for system issues - volaume 4 : System configuration and reconfiguration, 2004. Aeronautical Radio INC.

[BBE+11] Enrico Bini, Giorgio C. Buttazzo, Johan Eker, Stefan Schorr, Raphael Guerra, Gerhard Fohler, Karl-Erik Årzén, Vanessa Romero, and Claudio Scordino. Resource management on multicore systems: The ACTORS approach. *IEEE Micro*, 31(3):72–81, 2011.

[BFR13] Olivier Baldellon, Jean-Charles Fabre, and Matthieu Roy. Minotor: Monitoring timing and behavioral properties for dependable distributed systems. In *19th Pacific Rim International Symposium on Dependable Computing, PRDC'13*, pages 206–215, 2013.

[BLS06] Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337, pages 260–272, 2006.

[Bor05] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25(6):10–16, November 2005.

[But97] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, volume 24 of *Real-Time Systems Series*. Springer, 1997.

[DFG+14] Guy Durrieu, Madeleine Faugère, Sylvain Girbal, Daniel Gracia Pérez, Claire Pagetti, and Wolfgang Puffitsch. Predictable flight management system implementation on a multicore processor. In *Proceedings of the 7th Conference on Embedded Real Time Software and Systems (ERTS'14)*, 2014.

[EJS+10] Christian Engel, Eric Jenn, Peter H. Schmitt, Rodrigo Coutinho, and Tobias Schoofs. Enhanced dispatchability of aircrafts using multi-static configurations. In *Embedded Real Time Software and Systems Congress (ERTS 2010)*, Toulouse, France, 2010.

[Fre14] Freescale. T4240 QorIQ: Integrated multicore communications processor family reference manual, 2014.

[GLSS01] Alain Girault, Christophe Lavarenne, Mihaela Sighireanu, and Yves Sorel. Generation of fault-tolerant static scheduling for real-time distributed embedded systems with multi-point links. In *15th International Parallel & Distributed Processing Symposium (IPDPS-01)*, page 125, 2001.

[GPBB08] Jérémie Guiochet, David Powell, Etienne Baudin, and Jean-Paul Blanquart. Online Safety Monitoring Using Safety Modes. In *Workshop on Technical Challenges for Dependable Robots in Human Environments*, pages 1–13, May 2008.

[KAGS05] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The time-triggered ethernet (TTE) design. In *8th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, pages 22–33, 2005.

[KPR+14] Angeliki Kritikakou, Claire Pagetti, Christine Rochange, Matthieu Roy, Madeleine Faugère, Sylvain Girbal, and Daniel Gracia Pérez. Distributed run-time wcet controller for concurrent critical tasks in mixed-critical systems. In *Proceedings of the 22th International Conference on Real-Time and Network Systems (RTNS'14)*, pages 139–148, 2014.

[MDB+13] Renato Mancuso, Roman Dudko, Emiliano Betti, Marco Cesati, Marco Caccamo, and Rodolfo Pellizzoni. Real-time cache management framework for multi-core architectures. In *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*, pages 45–54, 2013.

[MRC+09] Miguel Masmano, Ismael Ripoll, Alfons Crespo, J.J. Metge, and Paul Arberet. Xtratum: An open source hypervisor for TSP embedded systems in aerospace. In *DASIA 2009. DAta Systems In Aerospace.*, May. Istanbul 2009.

[NES12] Mircea Negrean, Rolf Ernst, , and Simon Schliecker. Mastering timing challenges for the design of multi-mode applications on multi-core real-time embedded systems. In *Proceedings of the 6th Conference on Embedded Real Time Software and Systems (ERTS'12)*, 2012.

[NP13] Jan Nowotsch and Michael Paulitsch. Quality of service capabilities for hard real-time applications on multi-core processors. In *21st International Conference on Real-Time Networks and Systems (RTNS'13)*, pages 151–160, 2013.

[NPB+14] Jan Nowotsch, Michael Paulitsch, Daniel Bühler, Henrik Theiling, Simon Wegener, and Michael Schmidt. Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement. In *26th Euromicro Conference on Real-Time Systems (ECRTS'14)*, 2014.

[Oa13] Roman Obermaisser and al. DREAMS: Distributed REal-time Architecture for Mixed Criticality Systems. http://dreams-project.eu, 2013.

[PBB+12] Claire Pagetti, Pierre Bieber, Julien Brunel, Kushal Gupta, Eric Noulard, Thierry Planche, Francois Vialard, Clément Ketchedji, Bernard Bésinet, and Philippe Despres. Reconfigurable ima platform: from safety assessment to test scenarios on the scarlett demonstrator. In *Proceedings of the 6th Conference on Embedded Real Time Software and Systems and Software (ERTS'12)*, 2012.

[Rad05] Radio Technical Commission for Aeronautics (RTCA) and EUROpean Organisation for Civil Aviation Equipment (EUROCAE). DO-297: Software, electronic, integrated modular avionics (ima) development guidance and certification considerations, 2005.

[RF07] Larisa Rizvanovic and Gerhard Fohler. The matrix - a framework for real-time resource management for video streaming in networks of heterogenous devices. In *The International Conference on Consumer Electronics 2007*, January 2007.

[RRF10] Thomas Robert, Matthieu Roy, and Jean-Charles Fabre. Early Error Detection for Fault Tolerance Strategies. In *18th International Conference on Real-Time and Network Systems (RTNS'10)*, pages 159–168, Toulouse, France, 2010.

[SHLR+09] Siva Kumar Sastry Hari, Man-Lap Li, Pradeep Ramachandran, Byn Choi, and Sarita V. Adve. mSWAT: Low-cost Hardware Fault Detection and Diagnosis for Multicore Systems. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 122–132, 2009.

[YYP+13] Heechul Yun, Gang Yao, R. Pellizzoni, M. Caccamo, and Lui Sha. MemGuard: Memory bandwidth reservation system for efficient performance isolation in multi-core platforms. In *19th Real-Time and Embedded Technology and Applications Symposium (RTAS'13)*, pages 55–64, 2013.