FAA+RTS: Designing Fault-Aware Adaptive Real-Time Systems — From Specification to Execution —

Lukas MiedemaORCID[1], Dolly SapraORCID[1], Petr Novobilsky[2], Sebastian AltmeyerORCID[3], Clemens GrelckORCID[4], and Andy D. PimentelORCID[1]

[1] University of Amsterdam, Netherlands
{l.miedema, d.sapra, a.d.pimentel}@uva.nl
[2] Q-media s.r.o., Czech Republic
pno@qmediacz.onmicrosoft.com
[3] University of Augsburg, Germany
altmeyer@uni-augsburg.de
[4] Friedrich Schiller University Jena, Germany
clemens.grelck@uni-jena.de

Abstract. Large-scale cyber-physical systems, such as those for subway transportation or air traffic control, are becoming increasingly complex and often need to operate without human intervention. At the same time, these systems are subject to high requirements on the timing behavior and fault-tolerance. Consequently, the detection and mitigation of both hard and soft errors is of high importance in the already complex systems design process. The main challenges towards fault-aware real-time systems is the overall system design, in which the sheer size of the state-space and the system's complexity exceeds the capacity of today's development tools. In this paper, we present a new holistic methodology called FAA+RTS, for designing fault-aware adaptive real-time systems. We cover the entire path from system specification using a coordination language, via design-space exploration and task scheduling to the adaptive fault-aware runtime environment. Mitigating both hard and soft errors addresses competing requirements. Improving soft error tolerance (through redundant execution) may accelerate the aging process of silicon, thus expediting hard error failures. FAA+RTS is a novel solution as it integrates previously-isolated methods for dealing with multiple constraints into a single framework, presenting a single overview of all possible trade-offs to the application designer. This integration ensures that all aspects of system design, from specification to execution, are cohesively addressed, resulting in a robust and reliable system. We exemplify FAA+RTS using industrial-sized autonomous subway transportation system as a use-case.

1 Introduction

Complex cyber-physical, e.g. controlling critical infrastructure, are subject to a wide array of functional and non-functional requirements, ranging from reliabil-

ity and fault-tolerance to timeliness, energy usage and total system cost [15]. Availability combined with real-time reactivity is particularly challenging: Such systems must be available for years or even decades while always providing correct reactions within tight deadlines during their entire lifetime [7]. Yet, their long lifetime makes hardware faults, both soft and hard, inevitable [4]. Nevertheless, the system's functional and timing correctness must still persist.

Engineering correct and robust software systems is considered a challenging task in general. This general challenge is aggravated in our case in two ways. Firstly, the consequences of malfunctioning software in cyber-physical systems can be extremely serious up to the loss of lives. Secondly, the term correctness not only involves computational (or functional) correctness, i.e. computing correct results eventually, but expands to properties of program execution such as timing behavior and energy consumption. Computing the results within a given time period and energy budget can be as important for system correctness as their numerical values. Coordination languages such as TeamPlay coordination language [12] have been developed to alleviate the enormous software development burden by elevating coordination as a first-class citizen. The coordination layer enables the specification of application building blocks which can systematically be protected against hard and soft faults of hardware components, with the objective to ensure system integrity (functional results, time bounds, energy budgets) as long as possible despite faulting compute resources.

Autonomous adaptivity is a promising mitigation technique for hardware errors (both hard and soft) in the presence of real-time constraints. Limited over-provisioning of resources (extra cores, extra slack in the schedule) can be used effectively as the runtime system can respond to errors by rescheduling. However, competing constraints emerge, in particular as redundant execution to detect and mitigate soft errors increases system utilization, requires provisioning more hardware, increases energy consumption and accelerates silicon aging thus expediting the arrival of hard errors. To that end, Design-Space Exploration (DSE) offers a potent mechanism to help evaluate and reduce the number of design points. Many different System-on-Chip (SoC) designs can be explored, each offering a different amount of available computational resources to the application, and thus different degrees of room for adaptivity, fault-tolerance and thus different aging characteristics.

Contribution: In such a setting, we propose FAA+RTS: A new holistic methodology for designing fault-aware real-time systems. FAA+RTS encompasses the entire design flow from the application specification, multi-core system-on-a-chip (SoC) design, adaptive scheduling to mean time to failure (MTTF) analysis and the runtime environment. In FAA+RTS, adaptivity takes center stage as a driver of reliability to guarantee timing correctness even in case of partial system failure. The FAA+RTS runtime automatically reconfigures itself and reacts to hardware faults, using either re-execution (soft errors) or re-scheduling (hard errors) of critical components. FAA+RTS caters to the competing requirements of tolerance for soft errors and hard errors, of energy use and hardware utilization by recognizing that each solution is a compromise. Using design space

exploration (DSE), we give the application designer full insight in the trade-off between the various competing constraints and the resulting lifetime reliability as mean-time-to-failure (MTTF). This holistic view enables the application designer to select a compromise best tailored to the unique requirements of their application.

2 Industrial use case

We both motivate and validate our FAA+RTS methodology by a case study revolving around the communication subsystem of a railway control system developed by QMedia s.r.o., as illustrated in Figure 1. The application contains multiple tasks with precedence relations, has real-time constraints. It is a safety-critical application, thus reliability in the presence of both soft and hard errors must be considered. Consequently, the use-case exhibits a breadth of requirements benefiting from a holistic approach such as offered by FAA+RTS, making it ideally suited to evaluated our method.

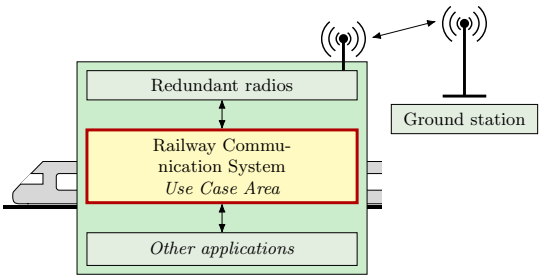


Fig. 1: Overview of the railway communication system

2.1 Description

The communication subsystem operates several different information flows between the ground system and the moving train(s). Each information flow is specific in purpose and criticality. It is necessary to establish independent communication channels for each of them. Under normal conditions each information flow has its own communication channel with allocated resources, but in case of incidents it is crucial to primarily keep up communication channels associated with the most critical information flows. Therefore, the system must be able to respond to various operating conditions (signal level, channel interference, HW-fault, line overload, etc.).

We show the parts of the software architecture of the use case in Figure 2. The railway communication subsystem is composed of 7 computational modules and 9 sinks, for a total of 16 real-time tasks. Each module is responsible for a part of the independent communication channels: *P1*, for instance, reads data from physical channels, and the data is further processed by the *AT Parser*.

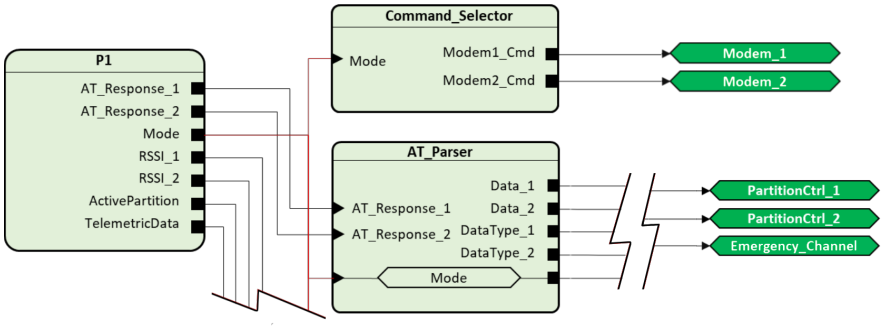


Fig. 2: Excerpt of the QMedia software architecture, showing 3 compute tasks and 5 sink tasks

2.2 Requirements

The timing requirements dictate a deadline and period of 100ms. Furthermore, the soft error and hard error tolerance must be maximized, while energy should be minimized. Desirable values for the energy use and reliability requirements are not static and instead are contingent on cost. Such an open formulation of requirements is ideal as it allows picking a compromise between the different factors. Through design space exploration, FAA+RTS gives the application designer a range of options to choose from, of which each such point can be subjected to cost-benefit analysis within the organization. The final process of narrowing the output of DSE down to a single option is out of scope for this paper.

3 Related Work

To the best of our knowledge, there is no other framework that addresses the fault-aware and adaptive aspect of real-time systems from specification all the way down to the execution and at the same time address both hard and soft errors. Several works address either soft errors [21] or hard errors [1] individually, but do not consider both together. Notably, some publications explore fault-tolerant scheduling techniques [18,19], but these efforts constitute only a fraction of the broader framework proposed in our work.

Coordination is a well established computing paradigm with a plethora of languages, abstractions and approaches, surveyed in [5]. Yet, it lacks adoption in mission-critical cyber-physical systems, even more so in fault-tolerance contexts. Hume [8], a resource-aware language for real-time systems, offers guarantees on time and space. Similarly, AADL, the Architecture Analysis & Design Language [6] supports performance and reliability analyses in real-time system design. However, both lack resilience features. Fault-tolerant Linda [3] extends the Linda coordination language [17], based on a tuple-space in which messages can be

shared between processes and the extensions focus mostly on making tuple-space operations safer and fault-tolerant.

The authors in [11] introduce utilization control as a technique to mitigate the core aging and core-failures (i.e. avoid hard-errors). This research focuses on the software aspect of embedded systems design and assumes the availability of specific hardware for improving lifetime reliability. Another similar work, [20] considers both hard and soft errors to extend the MTTF of a given hardware chip. In direct contrast, our framework delves into the bigger picture which include task scheduling and floorplan design of the SoC. A more powerful SoC design logically lowers utilization at the same redundancy levels, improving MTTF.

4 Integrated Methodology

We present an integrated methodology that cohesively links various stages of system design and execution, ranging from application specification to a fault-tolerant adaptive real-time runtime environment and a tailored multi-core SoC design.

Figure 3 outlines the proposed workflow.

Application Specification (Section 5) concerns the specification of the real-time application using the TeamPlay coordination language [12]. We leverage a coordination language to enable mechanical processing of the application code, its interdependencies and non-functional requirements in a robust and high-level manner.

Fault-tolerant Adaptive Real-time Scheduling (Section 6) is concerned with ensuring that all timing constraints are met under all conditions. Our approach is two-fold: (1) a scheduling bounds analyzer communicates with the iterative Design Space Exploration tool, while (2) offline schedules are produced for a range of degradation states from a particular design point.

Design Space Exploration (Section 7) identifies the SoC floorplan that optimizes lifetime reliability and power consumption in presence of errors. The tool generates a Pareto set from which an application designer selects a design point.

Our fault-tolerant, self-adaptive **Runtime Environment** *Artie* is linked with the compiled task code and runs the generated schedule set on the selected hardware. We discuss this aspect together with the validation of our industrial use-case in Section 8.

Each stage of our methodology is interconnected, ensuring that decisions made at one stage inform and enhance subsequent stages.

- The high-level application specification using the TeamPlay coordination language ensures robustness and clarity, feeding directly into the design space exploration.
- The scheduling component interacts continuously with design space exploration to ensure that timing constraints are met under all conditions.

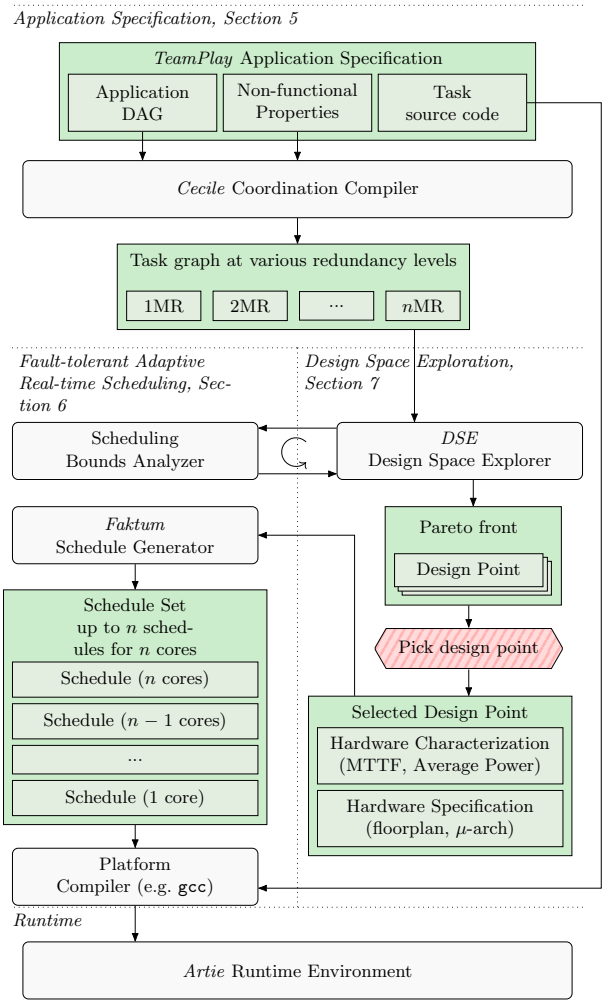


Fig. 3: FAA+RTS methodology overview

– The Design Space Exploration stage identifies optimal configurations, directly influencing the scheduling and runtime environment to balance reliability and power consumption effectively.
– The *Artie* runtime environment dynamically adapts based on the schedules and configurations identified, ensuring real-time system resilience.

This cohesive approach ensures that our methodology is more than just a collection of techniques; it is a unified framework designed to address the complexities of modern fault-tolerant real-time systems.

5 Application specification

The combination of functional and non-functional requirements plus resilience against hardware failure creates an enormous software engineering complexity problem that we address by leveraging the TeamPlay coordination language [12]. TeamPlay is an exogenous coordination language [2] that naturally induces a two-layer software architecture: On the *component implementation layer* we leverage a standard programming language (usually C/C++) to implement software components with defined but manageable size and complexity along with determined non-functional behavior. On the *coordination layer* the TeamPlay language permits the high-level specification of integrating individual components into a complete (cyber-physical) system that guarantees non-functional requirements and optimizes for non-functional objectives, e.g. saving energy, depending on the application. Communication between components is facilitated via *channels*.

Access to a channel is exclusive: a sending component must always be scheduled in its entirety before a receiving component, ensuring absence of contention. The explicit modeling of communication enables task-level redundancy, as multiple jobs can be spawned within one epoch. Non-functional properties of individual components are predetermined across the range of available micro architectures and, where applicable, specific voltage and frequency settings. The information is provided to TeamPlay via a data-base, coined the *non-functional properties file* (see Figure 3).

We leverage our two-layer, exogenous coordination design for further degrees of freedom in system design and integration. One example is *multi-versioning* where the component engineers provide multiple functionally exchangeable implementations of a component which differ in their non-functional properties, be it through competing implementation or through alternative compilation. Another example, prominently featuring in this work is the systematic and transparent introduction of redundancy to component execution for increased resilience against soft and hard hardware faults.

The FAA+RTS requires considerable flexibility of the application. Not only by requiring adaptivity to handle soft and hard errors, but also support for a range of platforms to enable effective Design Space Exploration. The component model, each with well-defined semantics, unlocks this flexibility without requiring dedicated support by the application developer. The Cecile coordination compiler serves as the front-end of the proposed FAA+RTS toolchain. As per Figure 3, it ingests the TeamPlay coordination specification together with the non-functional properties file, and it produces a number of task graphs with different component replication levels in machine-readable form to be further processed by our subsequent toolchain.

6 Fault-tolerant adaptive real-time Scheduler

We implement fault-tolerance via redundancy: redundant cores compensate for hard errors and redundant executions compensate soft errors. At runtime, we

adapt the schedule as dictated by the environment, i.e., the number of available cores and the presence of soft errors. In case of a hard error, we remap tasks of a failing core to available, healthy cores. Doing so decreases the number of available cores, and thus redundant execution may no longer be feasible. Consequently, hard errors may lead to an increased exposure to soft errors.

Soft errors. We consider triple and dual modular redundancy at the task level as primary means of mitigating soft errors, but other techniques are also compatible with FAA+RTS, provided a sufficient schedulability test is available. Triple modular redundancy mitigates errors by means of majority voting, while dual modular redundancy detects errors that then are mitigated by re-execution of the erroneous actors. Note that the system's response time increases with the level of redundancy, but the system response time can still be statically derived given an upper bound on the number of faults [9].

Hard errors. Adaptivity implies limited predictability due to system changes at run-time. Real-time scheduling, on the other hand, typically requires a high degree of predictability to enable derivation of real-time guarantees. Systems that are inherently unpredictable are unsuitable for a precise scheduling analysis. To overcome this contradiction, we resort to a set of statically precomputed schedules and a runtime environment that can dynamically switch between them.

To simplify remapping as a consequence of a hard error, we introduce *processing elements* as an abstract entity to which tasks are mapped. At runtime, cores are mapped to processing elements based on their availability. Hence, we do not need to compute schedule for all permutations of failed cores. Consequently, the number of schedules scales only linearly with the number of processing units. The size of each schedule scale linearly with the number of tasks. As such, for a system of n tasks and π processing units, we introduce a runtime memory complexity of only $\mathcal{O}(n \times \pi)$.

First, we prepare the task graph as encoded by the coordination language in two aspects: (a) transformation into a bipartite graph distinguishing between communication (data nodes) and computations (actor nodes); (b) partitioning the bipartite graph into sections that execute sequentially. Figure 4 shows such an extended graph, partitioned into two sections which can be scheduled independently. The sectioning of the graphs allows us to change redundancy levels

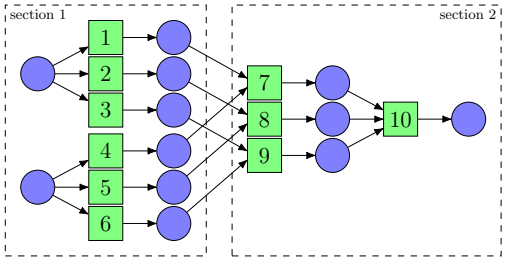


Fig. 4: Illustration of bipartite graph (example)

in between. For each section, we derive a set of static schedules via HEFT-scheduling[14]. The HEFT-scheduler is slightly adapted to correctly account for redundant execution and to schedule, if feasible, redundant actors on different processing elements. The precomputed schedules per section differ in the number of available processing elements and the redundancy level of the actor execution (TMR, DMR, no redundancy, see Figure 5).

Example. Figure 6 shows a sequence of hard errors. We assume a system of 6 cores. The actors are scheduled using triple-modular redundancy on 5 processing elements that are mapped to 5 of the 6 available cores, leaving one redundant core idle (a). In case of a hard error, i.e., a permanently failing core, processing element PE_3 is remapped to the spare core (b). In case a further core fails, the actors are mapped to only 4 processing elements using dual-modular redundancy. The switch from triple to dual-modular redundancy is necessary to meet the system requirements despite fewer processing elements. With the chosen scheduling, i.e., static section scheduling and adaptivity between sections, system's response time under x hard errors and y soft errors is merely a straight-forward combinatorial computation of maximal section lengths.

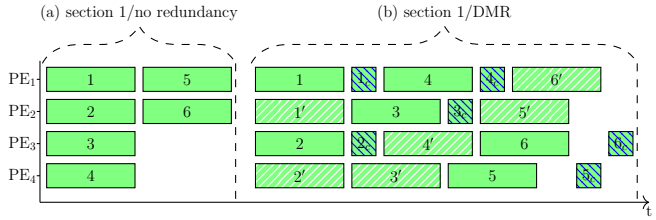


Fig. 5: Two schedules for Section 1 from Figure 4, both with five processing elements, (a) without redundancy and (b) with dual modular redundancy.

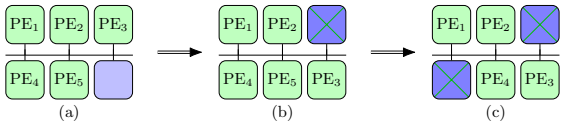


Fig. 6: Reaction to hard errors: Moving from TMR execution on 5 processing elements and one spare core (a) to TMR execution (b) to DMR execution on 4 processing elements.

Static analysis. *Faktum* implements the scheduler and scheduling analysis. It provides lower/upper bounds on needed processing elements, serves as a quick schedulability analysis to be used in DSE and generates section schedules to be used in the runtime environment.

7 Design-Space Exploration

The objective of the Design-Space Exploration (DSE) tool [13] is to help designing a specific Multi-core System-on-Chip (SoC) for an application, implementing core based redundancy to safeguard against hard errors. The redundant cores on the SoC offer a safety net to the application, allowing for the remapping of a workload in the event that one of the processing cores fails. This approach ensures that task deadlines continue to be met, even with reduced core resources. However, the decision to integrate extra cores at the onset is not without consequences. It introduces added costs, not only in terms of the monetary expenditure but also in the physical space occupied on the chip and the increased power consumption. This raises a pivotal question: how many extra cores are truly necessary to balance reliability and cost-effectiveness?

Our analysis indicates that the number of extra cores required is dependent on the specific system requirements and workload characteristics. Detailed experimentation and results demonstrate the trade-offs between reliability, cost, and power consumption, providing a clear guideline for determining the optimal number of extra cores.

The DSE tool employs a Genetic Algorithm (GA) to explore floorplan designs for a fixed grid size microchip and a specific workload. Each arrangement of (different types of) cores on a grid style placement forms a design point. The maximum size of the grid is predetermined (e.g., Figure 11). The GA is designed as a multi-objective search algorithm to optimize Mean-Time-To-Failure (MTTF) and average power consumption, which are inherently conflicting objectives. The optimization objective of average power consumption directly reflects the operational cost of the system and indirectly influences the physical cost through the required number of cores in the system. The framework produces a *Pareto Set* of design points, offering insights into floorplan choices that provide the trade-off between lifetime reliability (via MTTF) and power consumption. The framework evaluates design points using a simulator [16], which operates at a high abstraction level, estimating the chip's active lifespan and average power consumption for a specific workload. The simulator predicts core failures based on thermal, power and electro-migration ageing models, and redistributes workload until the application becomes unschedulable. The simulator employs Monte Carlo simulations with the stochastic fault models to predict the MTTF by averaging Time-To-Failures for each simulation and similarly computes the mean power usage.

Due to the significant computational requirements associated with performing numerous simulations per design point, the DSE tool continuously interacts with the *Scheduling Bounds Analyzer Faktum* to decrease the overall simulation count. When invoked, the *Scheduling Bounds Analyzer* provides information on the quality of the current floorplan specifically in terms of its potential to offer a valid and relaxed task schedule. The relaxation aspect suggests the capability of the floorplan to accommodate acceptable disruptions in the task schedule. Based on this analysis, the tool empirically categorizes the design point as good, bad or intermediate [13]. Given the costly nature of simulations, a higher number of

simulations is reserved for design points that fall in the intermediate range. The intuition here is that these specific floorplans effectively illustrate the trade-offs between power consumption and MTTF. They are valuable as they do not represent extremes, offering neither the flexibility for any relaxation in the schedule nor an excessively relaxed one. In other words, these floorplans represent a critical middle ground where the intricate relationship between power efficiency and reliability is evident.

The DSE tool predicts the SoC's reliability by considering core failures, i.e. hard errors. Soft errors are addressed via redundant execution. The DSE tool can be configured to account for soft errors by pre-processing the application to incorporate redundant tasks. It is important to note that lifetime reliability is significantly influenced by the presence of redundancy. Executing each task multiple times increases the computational demand, resulting in higher power consumption and elevated core temperatures. This elevated temperature accelerates the aging process of cores, emphasizing the trade-off between redundancy and the associated increase in power consumption and core aging.

8 Validation

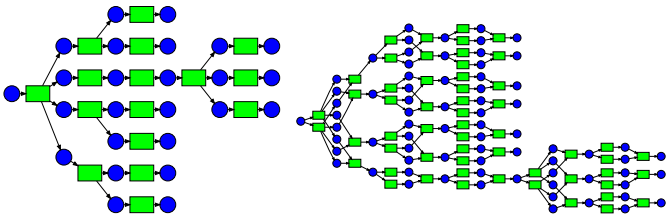
The first step is to model the application in the TeamPlay coordination language, as shown in Figure 7. Following the specification of the non-functional requirements *period* and *deadline*, we can easily identify two sections in the code that describe the application's *components* and their interactions through *channels*, respectively. The specification of a component consists of a set of typed *inports* and a set of typed *outports*, using a syntax reminiscent of C structs. Typed channels connect outport to inports. TeamPlay supports the annotation of various further properties to components, e.g. the redundancy level for resilience [10]. For space reasons we cannot show them all here. Note that we use the latest syntax of TeamPlay here, which slightly diverges from the syntax used in [12].

The strength of the TeamPlay exogenous coordination approach is that we can model our use case pretty much one-to-one according to the architecture specification in Figure 2. Component implementations do not need to be (re-)implemented, but can be adopted from existing software repositories. Port types, and thus channel types, are merely symbolic on the coordination layer, permitting integrity checks, but again are implemented at component implementation layer with programming language suited for that task, in the context of real-time software usually C.

This specification is converted to a bipartite graph split into sequential sections, composed of computing nodes (green) and data nodes (blue). Figure 8a shows one such section of the railway application. Next, we produce redundant variants of this graph to handle soft-errors. This means that we need to introduce replicas and comparators (dual-modular redundancy) or voters (triple-modular redundancy). Due to space limitations, we only show the bipartite graphs for dual-modular redundancy in Figure 8b.

```
app Railway {
  period = 100ms;
  deadline = 100ms;
  components {
    P1 { outports { Str_t AT_Response1;
                    Str_t AT_Response2;
                    u16 Mode;
                    PST_TelemetricData_t TelemetricData; ... }}
    CommandSelector { inports  {u16 Mode;}
                      outports {ET_ModemCmd_t Modem1CMD;
                                ET_ModemCmd_t Modem2CMD;}}
    ... }
  channels {
    P1.Mode -> CommandSelector.Mode;
    P1.AT_Response1 -> AT_Parser.AT_Response1;
    P1.AT_Response2 -> AT_Parser.AT_Response2;
    P1.Mode -> AT_Parser.ModeIn;
    ... }
}
```

Fig. 7: Excerpt from use case TeamPlay coordination code



(a) no redundancy (b) dual-modular redundancy

Fig. 8: Task graphs for one section of the railway use-case.

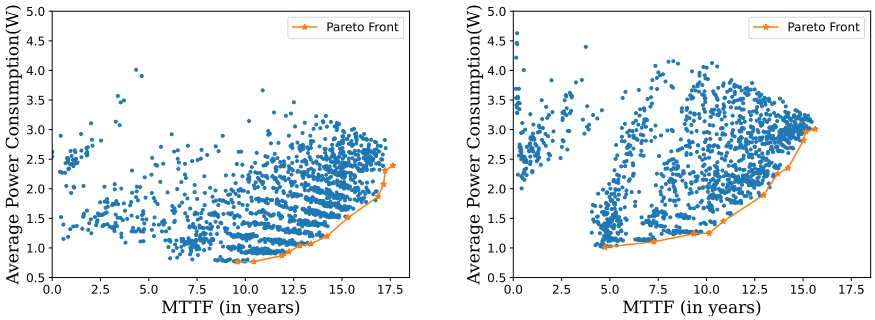
.

The Design Space Exploration (DSE) tool is applied to the QMedia use-case without redundancy, along with a dual-redundant task graph. Figures 9a and 9b highlights all the design points explored by the GA for no-redundancy and with dual-redundancy, respectively. Each floorplan is represented by its average power consumption and MTTF (as estimated by the simulator). The Pareto Set, denoted in orange, emerges as the GA converges in both scenarios.

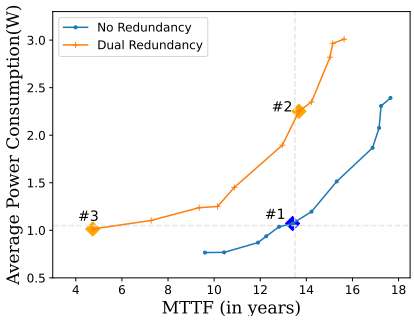
Furthermore, a direct comparison between the two Pareto Sets is demonstrated in 9c. It is noteworthy that the prospect of building safeguards against soft errors and hard errors incur a heavy cost in terms of operational power consumption of the SoC. Reduction in power consumption by curtailing the number of cores on the SoC shortens the lifetime reliability of the system.

Pareto Points #1, #2, and #3, illustrated in Figure 9c, highlight the cost of protecting the system against soft errors. Point #1, without redundancy, consumes 1.03W on average with a predicted lifetime of ≈13.5 years. After introducing redundancy, Point #2 offers a similar predicted lifetime but with an increased power consumption of ≈2.25W. Similarly, Point #3 operates at a com-

parable power level but is projected to have a shorter lifetime of ≈7 years. The significant contrast in lifetime reliability stems from the utilization of redundant schedules, wherein each task is executed multiple times. This leads to increased computational demands, higher power consumption, and overall, accelerates the aging process of the underlying microchip. Eventually, this framework provides valuable insights to the system designer to choose appropriate resources in creating a fault-aware adaptive real time system.



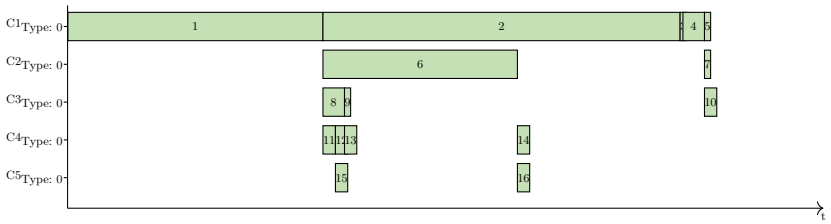
(a) No redundancy present (b) Dual redundancy present



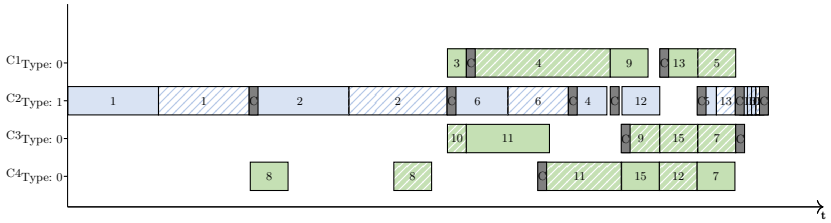
(c) Pareto Set comparison

Fig. 9: All design points explored by the DSE tool for no redundancy and dual redundancy modes. The orange line in (a) and (b) highlights the Pareto set.

The schedules corresponding to Pareto Points #1 and #3 are shown in Figure 10. Pareto Point #2 is omitted due to space limitations. In both the cases, the task graph offers only limited parallelism, see Figure 8, so that many cores remain idle throughout task execution. As can be seen by the complexity of the task graphs and the state-space, developing, optimizing and executing complex adaptive fault-aware real-time systems needs to be accommodated by suitable design-tools. Although all points from Figure 11 are Pareto optimal, the floorplans and schedules differ significantly.



(a) Schedule for Pareto Point #1 (no redundancy)



(b) Schedule for Pareto Point #3 (redundancy)

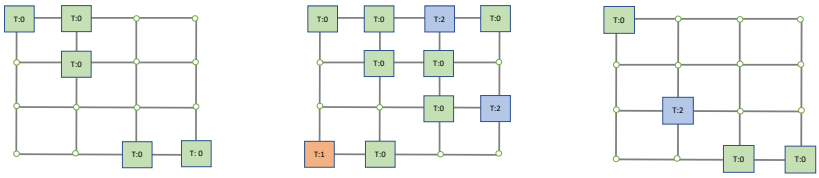
Fig. 10: Schedules of a single section of the railway use-case for Pareto Point #1 (no redundacy) and Pareto Point #3 (dual-modular redundancy) as marked in Figure 11. The hatched executions denote replaces, comparators are marked with **C**.

Furthermore, we have implemented the adaptive runtime environment *Artie*, short for *AROMA Runtime Environment*, to execute the schedules computed by *Faktum* on the selected platform. It support both bare-metal and pthread-based execution, for instance via Linux. This way, we support a large range of target architectures.

Our design methodology FAA+RTS not only enables us to specify the application and its non-functional requirements on a very high level of abstraction, but also to find Pareto-optimal design points and to execute the application with timing guarantees. The design is Pareto-optimal with respect to the mean-time-to-failure, cost and power consumption. Thus, we can execute the application in a fault-tolerant manner protecting us form hard and soft errors via self-adaptivity while meeting the stringent timing constraints.

9 Conclusion

In this paper, we have presented FAA+RTS, the first complete design methodology for the specification, development and execution of fault-tolerant adaptive real-time systems. We use a coordination language to model the application and its non-functional requirements. This specification is then input to Design-Space exploration, which searches for optimal system configurations and schedules. The scheduler provides a set of static schedules with varying levels of redundancy uses



(a) Floorplan for Pareto Point #1 (no redundancy)

(b) Floorplan for Pareto Point #2 (redundancy)

(c) Floorplan for Pareto Point #3 (redundancy)

Fig. 11: Pareto Points #1, #2 and #3 as marked in Figure 9c. Point #1 without redundancy, consumes ≈1.03W on average and has predicted lifetime of ≈13.5 years. When redundancy is required, Point #2 provides similar predicted lifetime consuming ≈2.25W. Similarly, Point #3 operates at similar power, but is predicted to have half the lifetime at ≈7 years.

a variant of HEFT scheduling. The runtime system then switches between these schedules at runtime depending on the current number of hard and soft-errors.

FAA+RTS targets highly complex self-adaptive systems featuring a wide array of contradicting requirements, foremost fault-tolerance via adaptivity timeliness, power consumption and costs. We have validated FAA+RTS on a real railway use-case. The results show the immense design space and complexity that can only be mastered using automated methodologies such as FAA+RTS.

References

1. Abdi, A., Zarandi, H.: A meta heuristic-based task scheduling and mapping method to optimize main design challenges of heterogeneous multiprocessor embedded systems. Microelectronics Journal **87** (2019)
2. Arbab, F.: Composition of interacting computations. In: Interactive Computation. Springer (2006)
3. Bakken, D., Schlichting, R.: Supporting fault-tolerant parallel programming in linda. IEEE Trans. on Parallel and Distributed Systems **6**(3) (1995)
4. Bansal, S., Bansal, R., Arora, K.: Energy conscious scheduling for fault-tolerant real-time distributed computing systems. In: Role of Data-Intensive Distributed Computing Systems in Designing Data Solutions. Springer (2022)
5. Ciatto, G., Mariani, S., Louvel, M., Omicini, A., Zambonelli, F.: Twenty years of coordination technologies: State-of-the-art and perspectives. In: International Conference on Coordination Languages and Models (COORD 2018). Springer (2018)
6. Feiler, P., Gluch, D., Hudak, J.: The architecture analysis and design language (AADL): An introduction. Tech. rep., Carnegie-Mellon University, Pittsburgh, USA (2006)
7. Gunes, V., Peter, S., Givargis, T., Vahid, F.: A survey on concepts, applications, and challenges in cyber-physical systems. KSII Transactions on Internet and Information Systems **8**(12) (2014)

8. Hammond, K., Michaelson, G.: Hume: a domain-specific language for real-time embedded systems. In: Generative Programming and Component Engineering (GPCE'03). Springer (2003)
9. Kühbacher, C., Ungerer, T., Altmeyer, S.: Redundant dataflow applications on clustered manycore architectures. In: Hong, J., Bures, M., Park, J.W., Cerny, T. (eds.) SAC '22: proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing, virtual event, April 25 - 29, 2022. pp. 226 – 235 (2022)
10. Loeve, W., Grelck, C.: Towards facilitating resilience in cyber-physical systems using coordination languages. In: Constantinou, E. (ed.) 13th Seminar on Advanced Techniques and Tools for Software Evolution (SATToSE 2020). vol. 2754. CEUR Workshop Proceedings (2020)
11. Ma, Y., Chantem, T., Dick, R., Hu, X.: Improving system-level lifetime reliability of multicore soft real-time systems. IEEE Trans. Very Large Scale Integr. VLSI Syst. **25**(6) (2017)
12. Roeder, J., Rouxel, B., Altmeyer, S., Grelck, C.: Towards energy-, time- and security-aware multi-core coordination. In: Bliudze, S., Bocchi, L. (eds.) 22nd International Conference on Coordination Models and Languages (COORD 2020), Malta. LNCS, vol. 12134, pp. 57–74. Springer (2020)
13. Sapra, D., Pimentel, A.D.: Exploring multi-core systems with lifetime reliability and power consumption trade-offs. In: Embedded Computer Systems: Architectures, Modeling, and Simulation: 23rd International Conference, SAMOS 2023. Springer (2023)
14. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3) (2002)
15. Villarreal Lozano, C., Vijayan, K.: Literature review on cyber physical systems design. Procedia Manufacturing **45** (2020)
16. Wasala, S.M., Niknam, S., Pathania, A., Grelck, C., Pimentel, A.D.: Lifetime estimation for core-failure resilient multi-core processors. In: 16th IEEE International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC 2023). IEEE (2023)
17. Wells, G.: Coordination languages: Back to the future with Linda. In: 2nd International Workshop on Coordination and Adaption Techniques for Software Entities (WCAT'05) (2005)
18. Youness, H., Omar, A., Moness, M.: An optimized weighted average makespan in fault-tolerant heterogeneous mpsocs. IEEE Trans. Parallel Distrib. Syst. **32**(8) (2021)
19. Zhang, L., Li, K., Li, C., Li, K.: Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems. Information Sciences **379** (2017)
20. Zhou, J., Hu, X., Ma, Y., Sun, J., Wei, T., Hu, S.: Improving availability of multicore real-time systems suffering both permanent and transient faults. IEEE Transactions on Computers **68**(12) (2019)
21. Zhou, J., Sun, J., Zhou, X., Wei, T., Chen, M., Hu, S., Hu, X.: Resource management for improving soft-error and lifetime reliability of real-time mpsocs. IEEE Trans. on Comp.-Aided Design of Integr. Circuits and Systems **38**(12) (2018)