Availability Analysis for Synchronous Data-Flow Graphs in Mixed-Criticality Systems

Roberto Medina, Etienne Borde, Laurent Pautet
LTCI, CNRS, Telecom ParisTech, Université Paris-Saclay.
Paris, France
Email: {firstname.lastname}@telecom-paristech.fr

***Abstract*—The safety-critical industry is compelled to continually increase the number of functionalities in embedded systems. These platforms tend to integrate software with various non-functional requirements, in particular different levels of criticality. As a consequence, Mixed-Criticality Systems emerged in order to assure robustness, safety and predictability for these embedded platforms. Although Mixed-Critcality Systems show promising results, formal methods to quantify availability are still missing for this type of systems and will most likely be required for deployment. This paper presents a transformation process that first produces a formal model of a Mixed-Criticality System. From this formal model, it generates a PRISM automaton in order to compute availability.**

I. Introduction

Due to safety requirements, certification processes overestimate execution times of software components deployed in safety-critical systems. Nonetheless, thanks to Mixed-Criticality Systems (MCS), redistribution of slack time caused by this overestimation is used to integrate more functionalities onto the same hardware platform. However, the processes deployed in these systems usually have different non-functional constraints, hence their levels of criticality are different as well (*e.g.* life-critical, mission-critical, non-critical).

The problem we tackle in this paper consists in modeling Mixed-Criticality (MC) applications in order to evaluate availability of tasks that might be stopped by the MC scheduling policy. To make this evaluation possible, we require these systems to be formally represented as Synchronous Data-Flow (SDF) graphs: taking advantage of SDF semantics, we propose a *fault model* and a *recovery process* that are suitable to compute functions availability.

We use the SDF model of computation in our research since the mathematical foundation of the model allows developers to verify execution properties (*e.g.* deadlock freedom, throughput, liveness, *etc*). Our fault model is defined by probabilities of timing failures that can occur during the execution of the SDF.

To overcome the complexity of availability computation, we build a model of the MCS using the Architecture Analysis and Design Language (AADL), gathering all the relevant information of the system. This AADL model is then translated to a probabilistic automaton, allowing us to compute availability for the MCS using adapted tools.

The remainder of this paper is organized as follows. Section II presents the preliminary notions considered in this paper. We establish a problem statement in Section III, demonstrating why availability for low criticality tasks is an important matter. Section IV describes the different models we consider to give a complete representation of a MCS. Our solution is described in section V: we model in AADL the MCS to produce a PRISM automaton used to verify availability properties. Our experimental results are discussed in section VI. Related work is reviewed in section VII. The final section concludes this paper with future research directions.

II. Preliminary Notions

In this section we present the different fields that are in the scope of our paper: Mixed-Criticality, Synchronous Data-Flow and reliability mechanisms.

A. Mixed Criticality Systems

Mixed-Criticality (MC) scheduling [2] is becoming an appealing solution to integrate various functions with different levels of criticality onto the same hardware platform.

MC scheduling was first presented by Vestal [3]. Vestal's task model considers various criticality levels for the system and is based on the fact that the higher the criticality level becomes, the more pessimistic the WCET is estimated. Each task belongs to a given criticality level and is coupled with various WCET estimations: one for each level. This model has been used across many other contributions on this topic, including our research presented in this paper.

For the remaining of this paper, we consider a two-level MCS: supporting HI and LO execution modes. When the system is in LO mode (initial mode), all tasks can be executed on the platform until their WCET in LO mode (noted $C_i(LO)$ for a task τ_i). WCET in LO mode are rather optimistic to leave processing time to execute more functions. For each task τ_i, $C_i(LO) \leq C_i(HI)$. A Timing Failure Event (TFE) can occur when a task τ_i runs for a time greater than its $C_i(LO)$ and triggers a mode switch from LO to HI mode. In HI mode, only tasks considered as HI-criticality are able to run until $C_i(HI)$ WCET.

B. Synchronous Data-Flow

For our study, we use the SDF model of computation to represent the software deployed in the MCS. In fact, data-flow graphs applications have been widely used in embedded systems due to their mathematical foundation: verify properties

like deadlock-freedom, soundness, boundedness of channels, makes this model appealing for real-time computing.

In a SDF, tasks have to communicate with each other in a specific order so the system can progress its execution. However if at least one task's inputs is not available, the task is blocked until the required data is available. Nonetheless, thanks to the information about the consumption and production rates of each task, SDF are amenable to compile-time construction of bounded and liveness guaranteeing static schedules [4]. Realizing a system with predictable performance properties is therefore possible.

Furthermore, recent efforts to adapt synchronous data-flows to mixed-criticality have shown great potential for uni-core [5] and multi-core [6], [7] architectures. We do not consider other types of task models since extracting an analyzable model might be too complex.

C. Reliability of Safety-Critical Systems

Another aspect to consider when it comes to safety-critical systems are fault tolerance mechanisms. Failures in these systems must be contained and corrected in order to have a reliable system.

To achieve the desired reliability, system designers usually replicate some sub-systems and use polling/voting mechanisms to detect, repair, or mask, the occurrence of faults in these replicas. For example, a typical architecture used in safety critical systems is the Triple Modular Redundancy (TMR).

When the inputs of a TMR are equivalent, the system is executing without errors. If one out of the three inputs of the TMR is not equivalent to the others, it is safe to assume that the component producing a different result is in an error state and needs to be fixed or ignored. If two (or three) of the replicas are in an erroneous state, the system is in an error state as well. The assumption of TMR is that having two replicas in an error state at the same time is extremely unlikely.

In the following, we base our availability analysis on these three preliminary notions : MCS, SDF and reliability mechanisms.

III. Problem statement

For the highest level of criticality, its components must be certified and for safety reasons WCET are overestimated. Thanks to MC approaches, when the system executes in normal conditions, *i.e.* no errors or timing failures occurred during execution, processing time is attributed to low criticality software components. This way the overestimation caused by the certification process is compensated and processor utilization is improved.

Most MC approaches do not consider a recovery mechanism after the system makes a transition to a high criticality mode. Once the system has switched to the high criticality mode, low tasks are stopped and only high criticality tasks are executed until their pessimistic WCET. The system is supposed to stop its mission as soon as possible by staying in this survival mode.

This is not ideal since services considered with low criticality may also play a major role in safety-critical system.

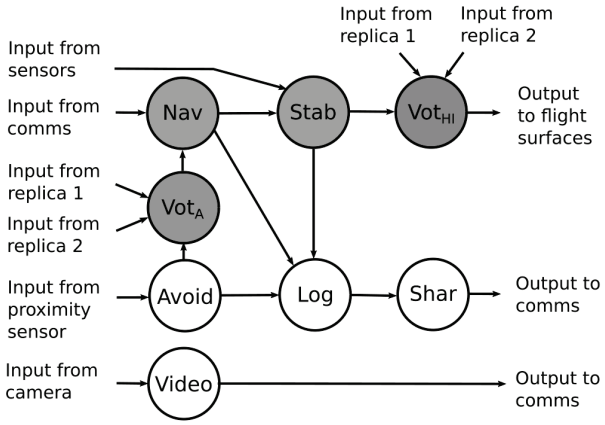

Fig. 1: **SDF Case Study for a UAV**

Shutting them down completely makes the system unusable: we need to reestablish low criticality tasks by introducing a recovery process for the MCS.

Availability of low-criticality services in MCS is therefore very important. We want to know how often the system fails and how long it takes to the system to come back to low mode.

We illustrate the different issues raised to analyze the availability of a safety-critical system on the MCS of Figure 1. This case study, adapted form the one from [6], represents software components deployed in an Unmanned Aerial Vehicle (UAV). Components are categorized as life-critical (grey circles) and mission-critical (white circles). We can clearly see that mission-critical components are essential for the quality of service for the UAV: the Avoidance mechanism ($Avoid$) makes the system safer, while the $Video$ component could be used for exploration purposes.

Computing the availability rate of an application becomes difficult when the application is designed using both SDF, MCS, and replication patterns such as TMR.

Replication patterns have to be introduced in the UAV in order improve the reliability of the system. Outputs of these functions are typically voted in a TMR before they are sent to actuators or to other computation units of the system. In addition, for our example, mission-critical components communicate with life-critical components which have a higher criticality level: the Avoidance mechanism sends data to the Navigation task. This communication process needs to be reliable since higher criticality tasks can be compromised if low tasks produce an erroneous output. Voting mechanisms are usually used in safety-critical systems to enable this type of communication. Nevertheless, synchronization of data sent by replicas is very important when using voting mechanisms: if the voter receives desynchronized data, it might consider such a situation as the occurrence of a fault, when in fact replicas are barely delayed.

To avoid desynchronization between voters and to assure deterministic communications, delayed communication mechanisms can be deployed between tasks. We consider a lock free implementation of M-to-N delayed communications [8] to

enable deterministic communication between mission-critical and life-critical tasks.

To satisfy availability requirements of low criticality tasks in MCS, we have to solve the following problems:

- Define a fault model, allowing us to represent the occurrence of TFEs.
- Provide the MCS with a recovery process that must be quantified in time.
- Build a representation of the MCS, that can be automatically analyzed and parametered in order to meet availability requirements.

The main contribution of this paper consists in obtaining a complete formal model of the MCS. By taking into account the different components: the SDF, the fault model, the recovery process and replication mechanisms into a single formalism. Then, we map the formal model into a PRISM [9] automaton used to compute an availability rate.

IV. Modeling Mixed-Criticality Synchronous Data-Flows

In this section we detail the different models that constitute a MCS for our analysis. The first model to consider is the SDF. This data-flow representation has to be extended, by including different execution times and criticality levels, in order to support task execution for different modes.

Moreover, a fault model needs to be defined for the system, so it can characterize the probability of occurrence of a mode switch from low criticality mode to high criticality mode.

Finally, evaluating an availability rate also requires the definition of a recovery process for the MCS and more precisely, it needs to be quantified in time. All these steps are necessary to compute an availability rate for a MCS and are finally collected in a formal model.

A. Application and task model

The SDF considered is represented as a graph (Fig. 1). HI tasks are represented with circles having a gray background while LO tasks are illustrated with white circles. Arcs between tasks represent communication channels, that can be delayed or immediate.

Names of task have been shortened (Nav stands for Nav, $Stab$ for $Stability$, $Shar$ for $Sharing$, etc.). The period given to the graph for executing one iteration in LO or HI mode is $50ms$. HI criticality tasks Nav and $Stab$ have the same $C_i(LO) = 10ms$. Vot_A and Vot_{HI} have their $C_i(LO) = 5ms$. LO criticality tasks have the same $C_i(LO) = 5ms$ as well. In HI mode, HI task can execute up to their $C_i(HI) = 20ms$ for tasks Nav and $Stab$, and $C_i(HI) = 5ms$ for tasks Vot_A and Vot_{HI} (without loss of generality, we used $C_i(HI) = C_i(LO)$ for Vot_A and Vot_{HI} to simplify the presentation). Period for the SDF in HI mode is therefore equal to $50ms$.

The first step of our transformation process to compute availability consists in obtaining the scheduling tables for the SDF application hosted in the MCS. Each mode has its own scheduling table containing a task set allowed to be executed. We compute a static scheduler for both LO and HI mode using the algorithm presented in [5]. In this case, the LO scheduling table, S_{LO}, is given by the sequence: $Vot_A, Nav, Stab, Vot_{HI}, Avoid, Video, Log, Shar$. For the HI scheduling table S_{HI}, it is also obtained using priority based scheduling but this time, HI tasks are assumed to be executed until their $C_i(HI)$. Therefore, S_{HI} is given by the sequence: $Vot_A, Nav, Stab, Vot_{HI}$.

For readability and space reasons, we assume the SDF is an Homogeneous SDF (HSDF): each task executes once per iteration and consumes only one *token* produced by their predecessors. Adapting the model to support multi-rate SDFs is possible, as long as a static scheduler can be found, the analysis of availability can be made.

B. Fault model

We characterize TFEs by determining their occurrence probability. To do so, estimation/computation of WCET is really important and it is not straightforward. Many methods are discussed in the literature (*e.g.* execution flow analysis, analysis of assembler instructions, experimental methods, etc.). We base our work on a probabilistic approach in order to specify *failure probabilities* for tasks.

It has been shown in [1], that probabilistic approaches can be used to estimate WCETs. For the two levels of criticality we consider, the more pessimistic WCET is used for high criticality mode, while the more optimistic WCET is used for low mode. A probabilistic WCET (pWCET) is categorized by a failure probability. A failure probability for HI tasks represents the probability of a HI task exceeding its $C_i(LO)$. Similarly, a failure probability for a LO task represents the probability that data was not produced after the task was executed for its $C_i(LO)$. For each task τ_i the function $f(\tau_i)$ gives the failure probability for τ_i with the associated optimistic $C_i(LO)$.

C. Recovery process

The availability computation requires a recovery process for the MCS to determine for how long the system is executing in HI mode. Without a recovery process, the system would remain in HI mode and availability would be unacceptable for industrial standards.

Let us first define the condition required to get back to LO mode. Only HI tasks are being executed by the scheduler in HI mode. However, if HI tasks do not exceed their optimistic WCET, $C_i(LO)$, processing time becomes available again for LO tasks. In addition, HI tasks are always executed at the beginning of the iteration (thanks to the scheduler of [5]). Doing so, we consider that the system has performed a mode switch in LO mode as soon as every task has received all the updated data it requires to be executed. However, updating data for components of the SDF can take some time.

The recovery process for the UAV system (Fig. 1) is presented in Figure 2. We assume the system has been executing in LO mode until the third iteration of the scheduler, where task $Avoid3$ did not produce its output before its $C_i(LO)$ (a warning sign represents the failure of the task). This triggers a

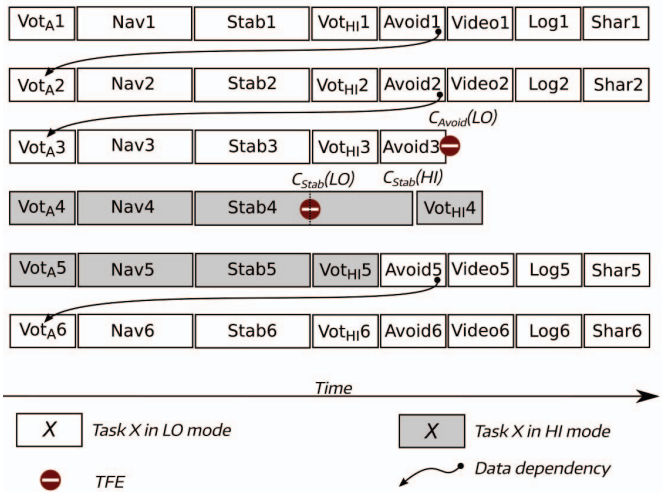

Fig. 2: **Recovery process example**

TFE and the system switches to HI mode. HI tasks start their execution during the next iteration: $Nav4$ finished before its $C_i(LO)$, however $Stab4$ exceeded its $C_i(LO)$ causing another TFE. The system stays in HI mode.

Iteration 5 starts by executing HI tasks, and they did not exceed their $C_i(LO)$ this time. LO tasks start to be executed and no TFE occurs in this iteration either. Since task Nav has a higher criticality level than task $Avoid$, a voting mechanism is used to allow this type of communication (we recall that task Vot_A implements this voting mechanism). The output produced by task Vot_{HI} at iteration 5 is therefore considered to happen in HI mode.

It is only at iteration 6, that the SDF has been brought back to LO mode: data was produced by task $Avoid5$ for the execution of $Nav6$, making the system safer. It is important to note that task $Nav5$ does not use the output that was produced by task $Avoid2$ since the safety-critical system might be in very different conditions: data produced by $Avoid2$ is considered to be outdated. In conclusion, the recovery process took three iterations to come back to LO mode.

We showed the different models considered for our availability computation of the MCS: the SDF task model, our fault model with failure probabilities and our recovery process. All these models are gathered in a formal model of the MCS using AADL. We then map the AADL model into a PRISM automaton to finally obtain an availability rate for the MCS.

V. Availability Computation

In this section we explain how we model a Mixed-Criticality SDF (MCSDF) in AADL and how we map this model into a PRISM automaton. AADL allows us to gather all the information relevant to the system: the SDF representation, the fault model and the recovery process. Mapping the obtained model to a PRISM automaton can then be performed thanks to this formal representation, in order to compute availability for the system.

A. AADL modeling of a MCSDF

We describe the modeling process of the MCSDF in this paragraph. The SDF graph is translated to a process component. Each task τ_i is represented as an AADL thread. Vertexes of the data-flow are translated to connections between threads which can contain more than one input/output port. These connections have a `Timing` AADL property: `Immediate` or `Delayed`, representing respectively synchronous and delayed communication in the MCSDF.

In an immediate connection, the receiver waits for the sender to complete its execution. The scheduler must ensure that the execution of the receiver is aligned with the completion of the sender. When connections use the delayed property, the sender always communicates with the recipient in the next periodic release of the recipient, *i.e.* after the deadline of the sender.

As explained before the voting mechanism needs to have its entries synchronized in order to be reliable. As a consequence, the delayed semantic is used for connections with output ports of $Avoid$ and $Stab$ that go to voting mechanisms.

AADL is capable of specifying different execution modes for all components. Within the process, we define two AADL execution modes: HI and LO. We specify WCET (AADL property: `compute_execution_time`) for threads in each mode, $C_i(LO)$ and $C_i(HI)$ for a HI task or only $C_i(LO)$ for a LO task. An event port, goes out from each thread and notifies the process to perform a mode switch from LO to HI mode when a TFE occurs. The recovery process is triggered when the process receives a notification from the scheduler in its event port `triggerrecover`, making the transition from HI to LO mode. The Error Modeling Annex V2 [10] allows us to describe our fault model in AADL by adding the failure probability: $f(\tau_i)$ to each thread of the AADL process.

B. Mapping the AADL MCSDF model to PRISM

We then propose to transform the AADL model into a PRISM automaton in order to quantify availability. Using PRISM to compute availability was experimented in the space domain [11] to quantify reliability, availability and maintenance properties. To the best of our knowledge, no such study was performed for MCS. Since the type of system we want to analyze is quite complex, we use a model checker to ease the computation of availability.

Our approach consists in translating the AADL model of the MCSDF into a PRISM automaton composed by various modules, synchronized between them thanks to transitions. A complete overview of PRISM automaton for our AADL MCSDF model is presented in Figure 3.

Each thread of AADL MCSDF model is mapped into one PRISM module (**Task module** in Fig. 3). Each PRISM module is an independent automaton and the parallel composition (similar to the parallel composition of communicating sequential processes) of all this modules represent our MCSDF.

The scheduler is represented with a PRISM module as well (**Scheduler module** in Fig. 3), it arbitrates task execution in function of the system mode and the scheduling tables for the

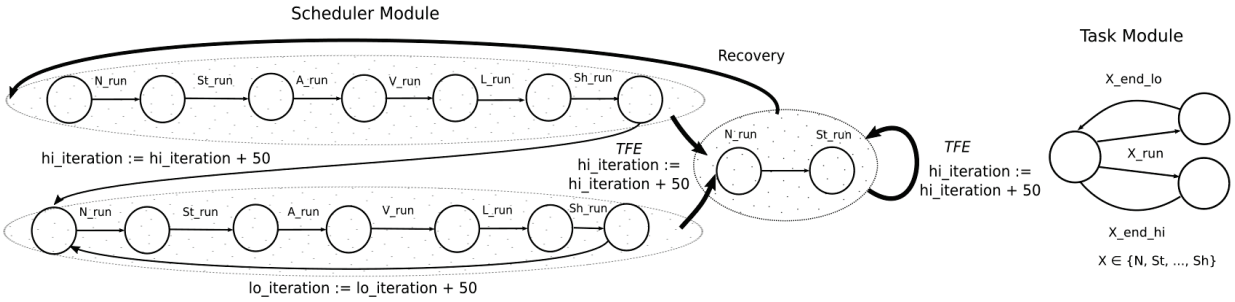

Fig. 3: **PRISM modules representing the MCSDF**

LO and HI modes. The scheduler module is also responsible for detecting timing failure events and triggering the recovery process. We have two types of transitions in the scheduler module:

1) The first one triggers the execution of a task and is represented as X_run (where X is the name of the task $Nav, Stab, \ldots$) in the Scheduler and Task module. When the Scheduler triggers a task execution, transition X_run, the task module triggers a probabilistic transition. There is a probability $f(\tau_i)$ that task τ_i can cause a TFE; and a probability $1 - f(\tau_i)$ that the system remains in LO mode after executing task τ_i. These failure probabilities are derived from the pWCET estimation as it was explained in Section II.
2) The second one is used to check if the task was executed without exceeding its $C_i(LO)$: X_end_lo. If a TFE occurs, the task module triggers a transition X_end_hi (`overrun` port of the AADL model) and the scheduler module reaches a state where only HI tasks are executed. These transitions are represented in our Figure 3. We simplified X_end_hi transitions in the scheduler module with a big arrow representing TFEs.

Voters Vot_A and Vot_{HI} have $C_i(LO) = C_i(HI)$, no TFE can be detected for these tasks and are simplified from the Scheduler module. The recovery process can be seen in the PRISM automaton (Fig. 3) as the first series of states in LO mode. After all these states have been executed without errors, the system goes back to the LO scheduling and remains there until a TFE occurs.

Therefore, all the information required to compute availability for a MCSDF is obtained with our transformation process. In the next section we explain our results to compute availability of our MCSDF presented in Figure 1.

VI. Experimental Setup

We go through our experimental setup for our case study illustrated by Figures 1 and 3. We only analyze availability in the case of TFE and do not consider other fault models like hardware errors.

After obtaining the PRISM automaton thanks to the AADL MCSDF model, we run simulations of the system for a certain number of steps. We assume that mission-critical tasks have a failure probability of 10^{-4}, mission-critical tasks a failure probability of 10^{-3} and non-critical tasks a failure probability of 10^{-2}. The PRISM automaton that was analyzed only represents one replica of the TMR, since we are only interested in evaluating availability in the presence of TFEs.

At each step of the simulation, the PRISM model checker triggers a transition of the Scheduler module. However, time spent in a given mode of the system needs to be measured. To reach this objective we use *reward functions* in PRISM. Two reward functions are used in our case study: one to keep track of time spent in LO mode (*lo_iteration*) and another for HI mode (*hi_iteration*). The counters are represented in Figure 3.

The availability formula used in PRISM to calculate availability equals the division of time the system executed in LO mode by the overall time the system was running in HI and LO mode. The simulation runs for a large number of iterations until the availability value does not have meaningful changes.

Figure 4 shows the results for 500 steps of the PRISM automaton (availability stagnates after these number of steps). As we can see, at the beginning of the system execution, availability is very variable. Difference between time spent in LO and HI mode is very variable. The availability rate calculated has a value of $A_R = 95.79\%$. To obtain the overall availability of the TMR we have to subtract the non-availability rate occurring on two and three components at the same time: $A_{TMR} = 1-((3\times(1-A_R)^2\times A_R)+(1-A_R)^3)) = 99.483\%$. This result clearly motivates that availability is better when replicas are used.

In this section we presented our experimental setup to compute availability for a MCSDF representing an UAV. The PRISM model checker was in fact capable of estimating the availability rate for the system. Our computation took into account many aspects related to safety-critical systems: different execution modes for a task set, a fault model, a recovery process and reliability mechanisms.

This result shows the capability of our approach to compute the availability of LO tasks in MCS. Having this result is of prime importance for designers of MCS, since availability is part of the quality attributes of such systems. The quantification of this quality attribute will help these designers to reach a trade-off among quality attributes of a MCS: certification costs, timing performances, availability, hardware platform cost, etc.

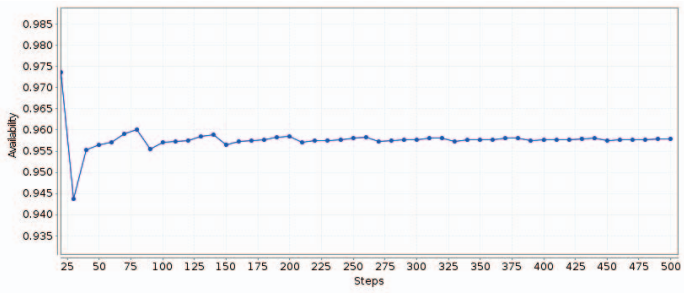

Fig. 4: **Availability results for the MCSDF**

VII. Related Works

As stated in Section II, our contribution uses different results coming from various research: mixed-criticality systems, data-flow, worst case execution time estimations and fault-tolerance mechanisms. Including all the related work for the scope of our paper would be too extensive, therefore we only focus on the key contributions that are related to our study.

Related work in the area of formal analysis in order to compute RAM properties using PRISM is presented in [11]. In this paper, authors evaluate these non-functional properties using the PRISM model checker as well. However, their analysis is focused on the overall platform for a satellite and does not consider MC scheduling for the platform.

Reliability computation for MCS using data-flows is presented in [12]. The authors' system is similar to ours: they consider an error, system and application model in addition to error handling mechanisms. Nonetheless their contribution is used to find the most reliable mapping of a task set onto the architecture and does not consider availability.

Other types of data-flow representations were extended to support different execution modes and dynamism of the application. For example Scenario-Aware Data-Flows [13] have specific tasks used to set up communication rates for the data-flow. Nevertheless, using SADF requires to explicitly define mode transitions for each actor of the SADF which can be tedious.

The Polychrony toolset is able to translate a sub-set of the AADL to components of a synchronous language, in order to schedule and verify timing properties [14]. This framework focuses on code generation and the respect of timing properties, but does not consider MCS or RAMS properties.

VIII. Conclusion and Future Work

This paper defines a transformation process that takes as input a SDF, probabilistic WCET estimations and a recovery process in order to compute availability of low criticality tasks of a MCS. This analysis is important in order to achieve the deployment of mixed-criticality based products and is necessary to ensure acceptable levels of Quality of Service in these systems.

Thanks to AADL we are able to overcome the complexity of availability computation for safety-critical systems. We extend a SDF model with a fault model and a recovery process in order to compute availability of low criticality tasks in a MCS.

The quantification of this property helps these designers to reach a trade-off among quality attributes of a MCS: certification costs, timing performances, availability, hardware platform cost, etc.

Our future works will consider availability analysis for other types of applications, *e.g.* different types of data-flow (Scenario-Aware, Affine), and other type of MC schedulers in the context of multi-processor architectures.

Acknowledgment

This research work has been funded by the academic and research chair Engineering of Complex Systems.

References

[1] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quiñones, and F. J. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," *Proceedings - Euromicro Conference on Real-Time Systems*, pp. 91–101, 2012.

[2] A. Burns and R. Davis, "Mixed Criticality Systems - A Review," *Department of Computer Science, University of York, Tech. Rep*, 2013.

[3] S. Vestal, "Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance," *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 239–243, 2007.

[4] E. Lee and D. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, 1987.

[5] S. K. Baruah, "Semantics-preserving implementation of multirate mixed-criticality synchronous programs," *Proceeding of the 20th International Conference on Real-Time and Network Systems (RTNS)*, pp. 11–19, 2012. [Online]. Available: papers/Baruah2012.pdf

[6] E. Yip and M. Kuo, "Relaxing the synchronous approach for mixed-criticality systems," *Proc. of the 20th IEEE ...*, pp. 89–100, 2014.

[7] S. B. M. B. Dario Socci Peter Poplavko, "Multiprocessor Scheduling of Precedence-constrained Mixed-Critical Jobs," no. TR-2014-11, 2014.

[8] F. Cadoret, T. Robert, E. Borde, L. Pautet, and F. Singhoff, "Deterministic Implementation of Periodic-Delayed Communications and Experimentation in AADL," *Isorc*, 2013.

[9] "PRISM - Probabilistic Symbolic Model Checker." http://www.prismmodelchecker.org/

[10] J. Delange and P. Feiler, "Architecture fault modeling with the AADL error-model annex," *Proceedings - 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014*, pp. 361–368, 2014.

[11] K. A. Hoque, O. A. Mohamed, C. Univeristy, and Y. Savaria, "Towards An Accurate Reliability , Availability and Maintainability Analysis Approach for Satellite Systems Based on Probabilistic Model Checking," pp. 1635–1640, 2015.

[12] P. Axer, M. Sebastian, and R. Ernst, "Reliability analysis for MPSoCs with mixed-critical, hard real-time constraints," *2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pp. 149–158, 2011.

[13] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," *Fourth ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2006. MEMOCODE '06. Proceedings.*, pp. 185–194, 2006.

[14] L. Besnard, A. Bouakaz, T. Gautier, P. Le Guernic, Y. Ma, J. P. Talpin, and H. Yu, "Timed behavioural modelling and affine scheduling of embedded software architectures in the AADL using Polychrony," *Science of Computer Programming*, 2014.