Model-driven reliability evaluation for MPSoC design

Tien Thanh Nguyen, Anthony Mouraud, Mathieu Thevenin, Gwenolé Corre, Olivier Pasquier, Sébastien Pillement

Model-driven reliability evaluation for MPSoC design

Tien Thanh Nguyen*, Anthony Mouraud*, Mathieu Thevenin†, Gwenole Corre†,
Olivier Pasquier‡ and Sebastien Pillement‡
* CEA/CTREG/DPLOIRE, 5 rue de l'Halbrane, 44340 Bouguenais, France; Email: firstname.lastname@cea.fr
† CEA Saclay, 91191 Gif-Sur-Yvette, Essonne Cedex, France; Email: firstname.lastname@cea.fr
‡ IETR, Ecole Polytechnique, 44306 Nantes Cedex, France; Email: firstname.lastname@univ-nantes.fr

***Abstract*—When designing a Multi-Processor System-on-Chip (MPSoC), a very large range of design alternatives arises from a huge space of possible design options and component choices. Literature proposes numerous Design-Space-Exploration (DSE) approaches thats mainly focus on cost optimization. In this paper, we present a DSE approach which focuses on the reliability of the whole design. This approach is based on a meta-model of Multi-Processor System-on-Chips (MPSoCs) integrated the reliability evaluation. We develop a tool that allows designers to describe and optimize their platform based on the proposed meta-model. The obtained results of an MPSoC is presented including the improved overall reliability of the system thanks to the automatic selection of the fault tolerance strategies for each component.**

I. Introduction

In the design process of a Multi-Processor System-on-Chip (MPSoC), a huge space of alternative solutions emerges from different design solutions. A design is built by a system architect to match specific objectives such as cost and/or performance. This work is based on the designer experience, consequently, the huge number of possibilities of the design space are not explored. The use of Design Space Exploration approaches (DSE) is mandatory to ensure an optimal exploration.

However, fast-growing technology scaling as well as working in harsh environments have made systems increasingly vulnerable to faults [1]. Beside the efforts to improve the hardware reliability, a need to use fault tolerance strategies to diminish the faults impact is rising. Moreover, reliability becomes one of the most important properties in embedded systems.

The fault tolerance requirements of the design imply a new dimension to the DSE, which makes almost impossible for a designer to manually select the optimal design under cost, performance and reliability constraints. Therefore, designers need a comprehensive methodology that defines the DSE of alternative solutions and integrates the system reliability evaluation into the early phase of design.

Model Driven Engineering (MDE) can provide effective ways to address needs of DSE. MDE approaches are at a high abstraction level and provide mechanisms to reuse, maintain and operate models [2]. A model represents an abstraction of a system as well as elements of that system from a particular design point of view. Mechanisms for the construction of valid models are specified in meta-models. The meta-model needs to be constructed in a suitable way at a high level of abstraction in order to cover the design alternatives as much as possible.

Many works provided meta-models for structures and behavior of MPSoC platforms [3]–[6]. However, these proposed meta-models are designed for specific purposes such as architectures with time-triggered execution paradigm, performance estimation or code generation for simulation, what make them difficult to reuse. In another aspect, some meta-models are defined at a low abstraction level [3], which limits the exploration capability. In any case, the meta-models presented in the literature are not developed for the reliability evaluation.

We propose in this work to develop a new platform meta-model that aims at integrating the reliability evaluation, a custom developed software which automatically performs the DSE and the optimization process of the platform design.

The paper is organized as follows: the next section introduces related works. Section III presents our proposal of meta-model which integrates the reliability elements. In Section IV, the modeling tool is introduced and its optimization abilities are illustrated through an example. Finally, Section V draws main conclusions and introduces further coming works.

II. Related Work

Several meta-models were proposed in the MPSoC design research effort using the MDE approach. Within the Graphical Array Specification for Parallel and Distributed Computing (GASPARD) framework [3], the authors propose meta-models corresponding to several design abstraction levels. The first one describes the hardware architecture at the Cycle-Accurate Bit-Accurate (CABA) level and the second is at the Timed Programmer's View (TPV) level. The CABA level meta-model is described at the signal level which hinders the discovery of new design options. The TPV meta-model is at a higher abstraction level which focuses on the performance estimation of a platform. Both meta-models are used to generate the simulation models while fault tolerance is not considered.

The Data Parallelism to Real Time (DaRT) [4] project also proposes meta-models for SoC design. It mainly focuses on code generation for simulation at Transaction-Level Model (TLM) and Register Transfer Levels (RTL) for the particular case of intensive signal processing applications. Moreover, only the hardware aspect of the DSE is taken into account and the fault tolerance is not considered.

In [5], the authors present a framework that provides a design flow for fault-tolerant embedded systems. A merged class-diagram of application and platform meta-models is proposed. The description is useful for modeling the dependencies between the application and the execution platform. In addition, their approach focuses on heterogeneous multiprocessor architectures with time-triggered execution paradigm. Thus, the meta-model aims at describing the relationships between application and the time-triggered bus while their reliability evaluation is not integrated on the meta-model.

The Model-driven Embedded System design (ModES) framework [6] provides meta-models representing MPSoC system concerns in specific aspects – application, platform, mapping, and implementation. These meta-models may partially match with our objectives despite they only focus on the mapping process and performance of a platform. The aspect of reliability is not considered in their meta-models.

III. MPSoC Platform Meta-Model

In this section, we firstly present a useful set of definitions, secondly, our platform meta-model which is built on the Unified Modeling Language (UML) syntax. Thirdly, we briefly introduce how to deal with different fault tolerance strategies.

A. Definitions and concepts

Before building a meta-model, basic concepts need to be defined [7] [8]. This helps to have a clear view of targeted architectures and to focus on our objectives.

Definition 1: An application is defined as $F = \{F_1, F_2, ..., F_k\}$ a set of k ($k \in \mathbf{N}$) functions that will have to be executed in a given order to produce desired outputs.

Definition 2: An MPSoC platform is composed of subsystems configured to provide a set of services (memorization, running, etc.). The connections between subsystems form the platform topology. The description of subsystems supports the redundancy modeling used in tolerance strategies. In addition, a subsystem is able to select many component options from available component libraries. Subsystem model is used in many works about redundancy allocation for the fault tolerance and reliability evaluation [9].

Definition 3: A hardware component can be:

- a processing element (PE) that can be hardwired (such as FPGA, ASIC), thus called a dedicated PE (DPE) (no software can be executed on it) or a general purpose processor (it can execute software), called a programmable PE (PPE). PE components are used to compute functions of a given application;
- a memory component is used to store data and source codes;
- a communication component is used to transfer data, signals between others hardware components.

Each component provides at least one service to implement requirements of a given application. A service is represented by metrics such as delay, cost, computing capacity.

Definition 4: A software component is an implementation of a function. Its source code is stored in a memory and a software component runs on a PPE.

Definition 5: A subsystem is composed of one type of hardware component and possibly several versions of software components.

B. Proposed Meta-models

The platform meta-model of the ModES framework partially matches our objectives as well as our definitions. Therefore, we develop our new platform derived from the proposition of the ModES meta-models. However, their MPSoC platform meta-model does not consider the fault tolerance. Thus, in this section, we present how to cover this gap. The Figure 1 depicts the architectural part of the proposed meta-model described below.

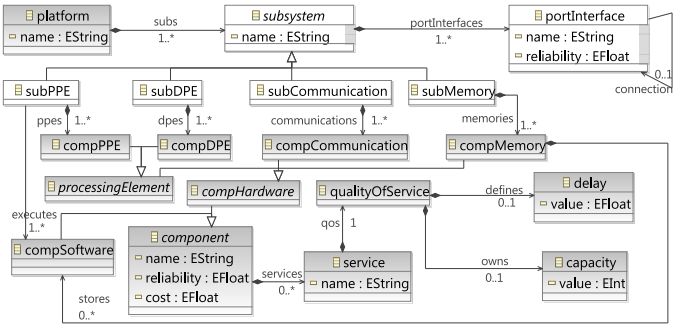

Fig. 1: Architectural part of the proposed platform meta-model. Gray elements refer to the ModES platform meta-model.

In the ModES, two basic classes are used to build an MPSoC platform meta-model: $platform$, $component$. A platform contains many components. Components are classified to hardware ($compHardware$) (Definition 3) and software ($compSoftware$) (Definition 4). $compHardware$ can be a communication component ($compCommunication$), a memory component ($compMemory$) or a processing component ($processingElement$). $processingElement$ may be a $comDPE$ (dedicated, an FPGA or ASIC) or a $compPPE$ (programmable, a GPP). A component can offer $services$.

TABLE I: Quality of service and properties specializations.

Component type	Service	Metric (quality of service)	
		Delay	Capacity
$compCommunication$	data transfer	N/P	bandwidth (Gb/s)
$compMemory$	storage	read/write delay	memory size (KB, GB...)
$compPPE$	instruction set	mean inst. execution time	inst. set size (number)
$compDPE$	logic block (LB) computation	mean LB execution time	number of logic blocks

For each service provided by a $compHardware$, $delay$ and $capacity$ come generic properties defined in the ModES meta-model $qualityOfService$ (bottom-right elements in Figure 1) are specialized (Table I) to match our objectives.

In our work, we also need to add supplementary architectural level: namely the $subsystem$ level. Indeed evaluating

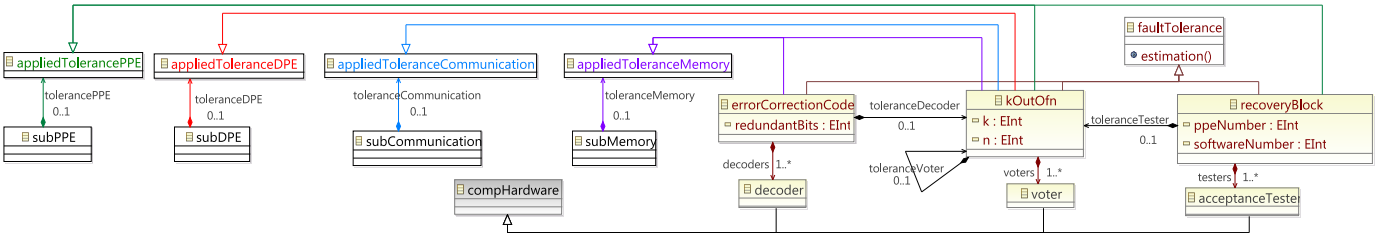

Fig. 2: Fault tolerance part of the proposed platform meta-model. The $subDPE$, $subPPE$, $subCommunication$, $subMemory$ and $compHardware$ elements are extracted from their Figure 1 counterparts.

whether requirements of a function are met may rely on the evaluation of several components as a whole. For example, a software component always runs on a GPP component for a given function. To know whether the requirements of the function are met, we must evaluate both of software component and PPE component in their common viewpoint. Furthermore, applying fault tolerance strategies like redundancy implies to evaluate the reliability of "group" of components. The $subsystem$ is defined (Definition 5, top-middle white element in Figure 1) in the scope of our platform meta-model as follow:

- a $platform$ contains different $subsystems$;
- a $subsystem$ may be one of the four types corresponding to components: $subPPE$, $subDPE$, $subCommunication$, $subMemory$;
- a $subsystem$ contains at least one component; all hardware component in a subsystem are of the same type;
- $portInterface$ represents the bounding of a subsystem.

We also implement the platform topology, which is required to calculate the connection delay between subsystems. The $connection$ reference defines connection wires between subsystems. Moreover, for a given function, $portInterface$ and $connection$ blocks allow to define the relationship between elements of a platform upon which the function relies. For example, when a $subPPE$ computes a function, we need to know in which $subMemory$ the data required by the execution is saved. Therefore, we can evaluate if the function's reliability requirements are met.

Behaviors of a $compSoftware$ depends on its supporting hardware components. For this reason, the relation between a $compSoftware$ and $compHardware$ needs to be defined. In the ModES, authors admit this but it is not defined on their platform meta-model. To evaluate the subsystem reliability and performance, we need to know all of the component relationships in a subsystem. A software is executed on a $compPPE$, consequently, it only belongs to the $subPPE$. A $compMemory$ can store one or many $compSoftware$.

C. Fault Tolerance Strategies

After having presented the architecture part of the meta-model, now we can introduce the tolerance strategies ($faultTolerance$). Several fault tolerance strategies can be applied to a $subsystem$ (redundancies, re-execution, correction codes etc.) [10] as shown in Figure 2.

A tolerance strategy performance is evaluated by computing the $estimation()$ method of a given subsystem. This method computes the reliability of the subsystem through a set of probability formulas taking into account each of the subsystem's component reliability. These formulas parameters are declared for each component as $reliability$. Of course, the $cost$ parameter needs to be defined also for each component to evaluate the fault tolerance strategy cost for each subsystem. Each strategy has its own typical parameters. The error code correction strategy make the fault tolerance for a memory with redundant bits ($redundantBits$). With the k-out-of-n strategy, a subsystem is functional when there are equal to or greater than k working components (n is a redundant number of components in the subsystem). The $recoveryBlock$ strategy uses a redundant number of software versions ($softwareNumber$) and a redundant number of PPE components ($ppeNumber$) for the fault tolerance in a $subPPE$. In the tolerance strategies, there are additional dedicated hardware components required by each specific strategy such as $voter$, $decoder$, $acceptanceTester$. These components can affect the reliability and cost of subsystems as well as the entire platform. Therefore, we can apply the redundancy tolerance strategy also for these components ($toleranceDecoder$, $toleranceVoter$, $toleranceTester$).

The $appliedTolerancePPE$, $appliedToleranceDPE$, $appliedToleranceCommunication$ and $appliedToleranceMemory$ classes represent the tolerance strategy applied on the corresponding subsystem. Appropriate strategies corresponding to each subsystem type are selected from the library. For example, a tolerance strategy applied for a memory subsystem ($appliedToleranceMemory$) can be $errorCodeCorrection$ or $kOutOfn$.

The UML view of the meta-model is given in Figure 1 and Figure 2. Figure 1 represents the architectural part and the Figure 2 represents the fault tolerance part in the proposed platform meta-model.

IV. Supporting Tool

The models built by our framework are based on the meta-model proposed and described in the previous section. The tool is constructed over the Sirius environment which is an Eclipse project allowing the creation of graphical modeling workbench. We create a workbench which allows designers to describe their platform through a graphical user interface.

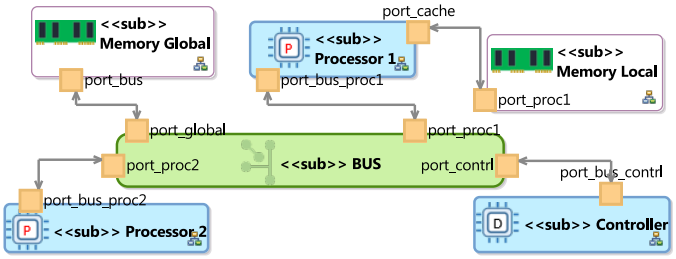

Fig. 3: Illustration of the workbench for the modeling of a 6-subsystem platform.

Figure 3 illustrates an example of such platform comprising 6 subsystems which hold different component types: 2 PPE subsystems, 1 DPE subsystem, 2 memory subsystems and 1 communication subsystem. They connect together through ports and wires. Components and their parameters are declared inside corresponding subsystems. Since the contribution of this paper is the adjudication of reliability to DSE approaches, this section firstly presents the reliability model, followed by a brief description of the optimization algorithm and the presentation of the results obtained with our approach.

A. Reliability Model

We develop a model to evaluate the subsystem reliability as well as the platform reliability. The component-level reliability is given by the user as an input data. Regarding hardware components, it is obtained using fault models derived from the physical failure mechanisms [11]. Concerning software components, it is acknowledged that reliability depends on numerous factors such as function complexity, programming language, programmer skill, etc [12]. In real cases, designers must have knowledge of their component reliability.

Specifically, the reliability of a platform is formulated with the following assumptions, originally proposed in [13]:

- hardware components in a subsystem are identical and software components in a subsystem are identical;
- each component and the platform have 2 states: functional or failed;
- component reliability is the probability that the component does not fail in a fixed mission time interval from the beginning;
- a platform is functioning if and only if all of its subsystems are functioning.

The number of available tolerance strategies which can be applied on a subsystem is noted NtS . We defined a variable x_i which represents the tolerance strategy index on the subsystem i: $x_i \in \{1, 2, ..., NtS\}$. r_{comp_i} is the reliability of a single component in the subsystem i. r_{comp_i} can be $r_{comp-soft_i}$, $r_{comp-PPE_i}$, $r_{comp-DPE_i}$, $r_{comp-memory_i}$, $r_{comp-bus_i}$ corresponding respectively to the reliability of software, PPE, DPE, memory and bus components. Respectively, the cost of a single component may be $c_{comp-soft_i}$, $c_{comp-PPE_i}$, $c_{comp-DPE_i}$, $c_{comp-memory_i}$, $c_{comp-bus_i}$.

Therefore, with the tolerance strategy x_i, the reliability level of the subsystem i: $R_{sub_i}(x_i, r_{comp_i})$ is the probability that the subsystem i does not fail in its mission time interval from the beginning. Respectively, the cost of the subsystem i is $C_{sub_i}(x_i, c_{comp_i})$. Note that a $subPPE$ has $compSoftware$ and $compPPE$, so that the reliability and cost of a $subPPE$ are respectively $R_{sub_i}(x_i, r_{comp-soft_i}, r_{comp-PPE_i})$ and $C_{sub_i}(x_i, r_{comp-soft_i}, r_{comp-PPE_i})$.

A platform has N subsystems which owns a vector $x = [x_1, x_2, ..., x_N]$. Each x vector defines a tolerance solution for a platform. Therefore, the reliability of a platform is the probability that all subsystems are functioning, the product of all the reliability of subsystems [14], is represented by:

$$R_{\text{platform}}(x; N) = \prod_{i=1}^{N} R_{sub_i} \tag{1}$$

and respectively the cost of a platform is calculated by:

$$C_{\text{platform}}(x; N) = \sum_{i=1}^{N} C_{sub_i} \tag{2}$$

Reliability optimization model:

The objective is to look for a vector x to reach a maximum platform reliability level under a cost constraint. Each solution corresponding to a definite vector x have to respect the cost constraint. It means that the platform cost has to be less than or equal to the maximum cost $Cost_{max}$ pre-defined by designers. The reliability optimization model hence is given by:

$$\begin{aligned} &max\Big(R_{\text{platform}}(x; N)\Big) \\ &\text{with } C_{\text{platform}}(x; N) \leq Cost_{max} \end{aligned} \tag{3}$$

To illustrate the reliability model, we use 2 tolerance strategies: Triple Modular Redundancy (TMR) and 3-out-of-5. In TMR, a subsystem is functioning if 2 out of 3 components are functioning. In 3-out-of-5, a subsystem is functioning if 3 out of 5 component are functioning. Specially, if TMR is applied on a PPE subsystem, there are three independent software components, each running on a separate PPE component. The PPE subsystem is functioning if 2 out of 3 software components (on working PPE component) are operating. Similarly, when 3-out-of-5 is applied on a PPE subsystem, the PPE subsystem is functioning if 3 out of 5 software components (on working PPE component) are operating. Thus, the variable x_i gets value in $\{1, 2, 3\}$ – 1 means the no-tolerance, 2 stands for TMR and 3 for 3-out-of-5. Both strategies use one voter with its reliability r_{voter} and its cost c_{voter} for selecting correct outputs.

The reliability and cost evaluation of a subsystem corresponding to each value of x_i is given by Table II. The original formulas are retrieved from [13]. However, in a PPE subsystem, there are hardware and software components. So the formulas of PPE subsystems have been modified to evaluate the impact of both hardware and software components on PPE subsystems for each strategy.

All input parameters of components are declared for our example, as depicted in Table III. The **Component reliability**

TABLE II: Formulas for the subsystem cost and reliability estimation corresponding to tolerance strategies [13].

Name	x_i	Subsystem type	R_{sub_i} and C_{sub_i}
No tolerance	1	PPE	$R_{sub_i}(1, r_{comp-soft_i}, r_{comp-PPE_i}) = r_{comp-soft_i}.r_{comp-PPE_i}$
			$C_{sub_i}(1, c_{comp-soft_i}, c_{comp-PPE_i}) = c_{comp-soft_i} + c_{comp-PPE_i}$
		Others	$R_{sub_i}(1, r_{comp_i}) = r_{comp_i}$
			$C_{sub_i}(1, c_{comp_i}) = c_{comp_i}$
TMR	2	PPE	$R_{sub_i}(2, r_{comp-soft_i}, r_{comp-PPE_i}) = r_{voter}.(3.(r_{comp-soft_i}.r_{comp-PPE_i})^2 - 2.(r_{comp-soft_i}.r_{comp-PPE_i})^3)$
			$C_{sub_i}(2, c_{comp-soft_i}, c_{comp-PPE_i}) = c_{voter} + 3.(c_{comp-soft_i} + c_{comp-PPE_i})$
		Others	$R_{sub_i}(2, r_{comp_i}) = r_{voter}.(3.r_{comp_i}^2 - 2.r_{comp_i}^3)$
			$C_{sub_i}(2, c_{comp_i}) = c_{voter} + 3.c_{comp_i}$
3-out-of-5	3	PPE	$R_{sub_i}(3, r_{comp-soft_i}, r_{comp-PPE_i}) = r_{voter}.(10.(r_{comp-soft_i}.r_{comp-PPE_i})^3.(1 - r_{comp-soft_i}.r_{comp-PPE_i})^2 + 5.(r_{comp-soft_i}.r_{comp-PPE_i})^4.(1 - r_{comp-soft_i}.r_{comp-PPE_i}) + (r_{comp-soft_i}.r_{comp-PPE_i})^5)$
			$C_{sub_i}(3, c_{comp-soft_i}, c_{comp-PPE_i}) = c_{voter} + 5.(c_{comp-soft_i} + c_{comp-PPE_i})$
		Others	$R_{sub_i}(3, r_{comp_i}) = r_{voter}.(10.r_{comp_i}^3.(1 - r_{comp_i})^2 + 5.r_{comp_i}^4.(1 - r_{comp_i}) + r_{comp_i}^5)$
			$C_{sub_i}(3, c_{comp_i}) = c_{voter} + 5.c_{comp_i}$

and **Component cost** columns declare the input parameters of a single component in the corresponding subsystem. The tolerance strategies use one voter defined with $r_{voter} = 0.999$ and $c_{voter} = 50$. There is no tolerance for the voter.

TABLE III: Component input parameters for the platform given in Figure 3.

Subsystem	Component name	Component reliability	Component cost
PPE Processor 1	GPP 1	0.9850	40
	Software 1	0.9760	35
PPE Processor 2	GPP 2	0.9940	35
	Software 2	0.9700	35
Memory Local	SRAM 1	0.9860	40
Memory Global	SRAM 2	0.9870	45
DPE Controller	FPGA	0.9900	60
Communication Bus	Bus	0.9960	80

B. Optimization algorithm

We use Simulated Annealing (SA) to solve the system reliability optimization problem, by modifying the tolerance strategy. SA parameters are:

Objective function: the objective is to find the best solution through the decision vector x with the evaluation function $R_{platform}(x; N)$. A reliability value equal to 1 is the maximum reachable (higher is better).

Initial solution: the initial solution for each subsystem is selected randomly. In a particular case, if designers have predefined knowledge of some particular subsystems, they can select a more reasonable initial solution.

Cooling function: the cooling schedule is implemented by using the function proposed in [15]: $T(j) = T_0.\alpha^j$, with: $T(j)$ the temperature at j^{th} step, j the temperature step number, T_0 an initial temperature, and α a factor to decrease the temperature in each step. Based on previous work [16], we can take $T_0 = \Delta R_{max}$ with ΔR_{max} the maximal objective function value difference between any two neighboring solutions. Since the reliability value difference between two solutions is small compared to 1 so that it also makes T_0 small. And the cooling schedule does not have practical sense. Therefore, the objective function is transferred to $1000 \times R_{platform}(x; N)$. This does not alter the nature of the algorithm. In our example, T_0 ranges from 45 to 55. In the literature, the cooling factor (α) is between 0 and 1. But, according to [15], α can be around of 0.85-0.96, to gain a optimal solution. Moreover, SA produces better results when the neighbor-compare-move process is carried out many times at each temperature step. The number of iteration per temperature is a function of the number of neighborhood solutions of a given solution [17]. In our example, each subsystem has 3 tolerance options. Consequently, with 6 subsystems, a given solution has 17 neighborhood solutions. Thus, we choose the number of iteration per temperature equal to 17.

Constraint and next neighborhood move: the reliability optimization is under a cost constraint. A neighbor move is performed only to a new solution if the cost condition is satisfied. One subsystem is randomly selected at each iteration. A random tolerance strategy is applied at the subsystem.

Stopping condition: normally, the simulation process stops when the temperature reaches a value as close as possible to the zero. In our case, we terminate the optimization simulation when there is no more expected move. We observe that satisfied solutions were found after about 20 temperature steps.

C. Results

Our optimization model applied to the example previously presented is simulated with Scilab. We consider four cost constraints: 400, 750, 1100, 1800. The goal is to find a tolerance solution which has the highest reliability level in each constraint. Based on 10 simulation runs for each cost constraint, results with the highest reliability level represent our results in this constraint.

Furthermore, we also looked for all possible tolerance solutions for the platform. This has confirmed that the result obtained from the optimization algorithm is actually the optimal result for each constraint.

Table IV presents the optimization results. The first column shows the subsystems in the platform. Each cost constraint ($Cost_{max}$) has two columns. The **Solution** column indicates tolerance strategies used on the corresponding subsystem. The **Reliability** column gives the subsystem reliability that corresponds to the strategy which was applied. The platform reliability and the average computation time per simulation run of each cost constraint are shown in the lower half of the table. For example, no component redundancy is allowed for a system cost of 400. Such system overall reliability is 0.889473, it means that the system can be operating without

TABLE IV: A selection of optimal results obtained for different cost constraints.

	$Cost_{max} = 400$		$Cost_{max} = 750$		$Cost_{max} = 1100$		$Cost_{max} = 1800$	
Subsystem	**Solution**	**Reliability**	**Solution**	**Reliability**	**Solution**	**Reliability**	**Solution**	**Reliability**
PPE Processor 1	1	0,96136	2	0,994641	2	0,994641	3	0,998457
PPE Processor 2	1	0,96418	1	0,96418	2	0,995247	3	0,998565
Memory Local	1	0.9860	2	0,998418	2	0,998418	3	0,998973
Memory Global	1	0.9870	1	0.9870	2	0,998498	2	0,998498
DPE Controller	1	0.9900	1	0.9900	1	0,9900	2	0,998702
Communication Bus	1	0.9960	1	0.9960	1	0.9960	2	0,998952
Platform	Ave. time	Reliability	Ave. time	Reliability	Ave. time	Reliability	Ave. time	Reliability
	14,6235	0,889473	6,52861	0,931855	8,5676	0,973085	11,8545	0,992173

fault after a mission with a probability of 88.95%. As well as the average simulation time of this case is $14,62$ seconds. Note that the run time in the constraint of 400 is highest because the optimization process wastes many times to look for a neighborhood solution.

When the cost budget is increased to 750, this makes possible to use TMR in some subsystems, thus increasing the system reliability. When the cost budget is set to 1100, most of the subsystems are applied TMR and the platform reliability reaches 0.973085. With a cost budget of 1800, the results indicate that all subsystems are applied the fault tolerance and the most reliable strategy (3-out-of-5) is used. With this cost budget, the platform can reach more than 99% of reliability in a mission time from the beginning. Thus, the platform reliability is improved by 11% compared with the no-tolerance case. The run time increases when its cost constraint is widened. Because the smaller the cost constraint is, the smaller the space of satisfied designs is. So the best solution was found earlier.

In summary, results from the Table IV shows that, for each cost budgets, our tool finds a fault tolerance solution with the best level of reliability.

V. Conclusion and future works

Design space exploration under constraints is an important topic in computer architecture, especially with new MPSoCs which can comprise a large number of subsystems. However, up to now, no approach has taken into account the overall system reliability.

In this paper, we propose a DSE approach that is based on meta-model in which we have integrated the elements of the fault-tolerance. This enables the use of an SA optimization algorithm to select the most efficient architecture in terms of system cost and reliability. We apply our proposal on an example of SoC and present the results. We showed substantial (11%) improvement of the reliability of the considered platform. It can be concluded that our meta-model provides a satisfactory foundation for modeling an MPSoC platform. Besides, we also design a modeling tool integrating a reliability model to illustrate the ability of the meta-model in the reliability optimization process.

In this paper, we have focused on simple fault-tolerance strategies with a toy example to illustrate the novelty of the proposed meta-model. Future work will consider several other strategies in association with different types of subsystems (Recovery Block, Error Detection/Correction Coding, Hybrid). Moreover, the DSE process will also consider the application structure and optimization of the mapping between an application and a platform.

References

[1] S. Mittal and J. S. Vetter, "A survey of techniques for modeling and improving reliability of computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1226–1238, 2016.

[2] S. Assar, "Model driven requirements engineering: mapping the field and beyond," in *Model-Driven Requirements Engineering Workshop (MoDRE), 2014 IEEE 4th International*. IEEE, 2014, pp. 1–6.

[3] R. B. Atitallah, L. Bonde, S. Niar, S. Meftali, and J.-L. Dekeyser, "Multilevel mpsoc performance evaluation using mde approach," in *System-on-Chip, 2006. International Symposium on*. IEEE, 2006, pp. 1–4.

[4] L. Bondé, C. Dumoulin, and J.-L. Dekeyser, "Metamodels and mda transformations for embedded systems," in *Advances in design and specification languages for SoCs*. Springer, 2005, pp. 89–105.

[5] J. Huang, S. Barner, A. Raabe, C. Buckl, and A. Knoll, "A framework for reliability-aware embedded system design on multiprocessor platforms," *Microprocessors and Microsystems*, vol. 38, no. 6, pp. 539–551, 2014.

[6] M. F. d. S. Oliveira, "Model driven engineering methodology for design space exploration of embedded systems," 2013.

[7] Z. J. Jia, A. Núñez, T. Bautista, and A. D. Pimentel, "A two-phase design space exploration strategy for system-level real-time application mapping onto mpsoc," *Microprocessors and Microsystems*, vol. 38, no. 1, pp. 9–21, 2014.

[8] F. R. Wagner, F. A. Nascimento, and M. F. Oliveira, "Model-driven engineering of complex embedded systems: Concepts and tools," *Porto Alegre, RS: Institute of Informatics, Federal University of Rio Grande do Sul (UFRGS)/Cooperative Computing and Communication Laboratory (C-LAB), University of Paderborn, Paderborn, Germany*, 2011.

[9] R. Soltani, "Reliability optimization of binary state non-repairable systems: A state of the art survey," *International Journal of Industrial Engineering Computations*, vol. 5, no. 3, pp. 339–364, 2014.

[10] H. Mushtaq, Z. Al-Ars, and K. Bertels, "Survey of fault tolerance techniques for shared memory multicore/multiprocessor systems," in *Design and Test Workshop (IDT), 2011 IEEE 6th International*. IEEE, 2011, pp. 12–17.

[11] Y. Xiang, T. Chantem, R. P. Dick, X. S. Hu, and L. Shang, "System-level reliability modeling for mpsocs," in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010, pp. 297–306.

[12] P. Kapur, H. Pham, A. Gupta, and P. Jha, *Software reliability assessment with OR applications*. Springer, 2011.

[13] D. W. Coit, N. Chatwattanasiri, N. Wattanapongsakorn, and A. Konak, "Dynamic k-out-of-n system reliability with component partnership," *Reliability Engineering & System Safety*, vol. 138, pp. 82–92, 2015.

[14] M. Rausand, H. Arnljot *et al.*, *System reliability theory: models, statistical methods, and applications*. John Wiley & Sons, 2004, vol. 396.

[15] K.-L. Du and M. Swamy, *Search and Optimization by Metaheuristics*. Springer, 2016.

[16] W. Ben-Ameur, "Computing the initial temperature of simulated annealing," *Computational Optimization and Applications*, vol. 29, no. 3, pp. 369–385, 2004.

[17] M.-W. Park and Y.-D. Kim, "A systematic procedure for setting parameters in simulated annealing algorithms," *Computers & Operations Research*, vol. 25, no. 3, pp. 207–217, 1998.