Applications of scenarios in early embedded system design space exploration
van Stralen, P.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (http://dare.uva.nl)
The *Sesame Automated Fault-tolerant Explorer (SAFE)* is a framework that takes fault tolerance into account while mapping the application onto the architecture. Fault tolerance patterns are automatically applied to the application, which makes an automatic exploration of the fault-tolerant design space possible. Such a fault-tolerant design space, however, involves a large degree of freedom with respect to the number of objectives that can be explored. Not only performance, power and cost are part of these objectives, but also metrics like frame drop ratio and additional buffer space that is required to store application checkpoints. For this frame drop ratio, it can even be analyzed which part is due to transient faults and which part is due to deadline misses. Such detailed statistics are extremely usable to extensively analyze a small set of fault-tolerant mappings by hand.

During early DSE, however, there are simply too many mapping possibilities to evaluate them by hand. This is already true for the traditional mapping problem that is handled by Sesame, but it becomes even worse when fault tolerance is taken into account. For a fault-tolerant mapping, not only binding needs to be done (like is the case with the traditional mapping problem), but also the patternization that divides the application into subnetworks and selects fault tolerance patterns. There are many potential ways of performing patternization and, due to the larger fault-tolerant application, there are even more possibilities to bind the application onto the architecture.

In Figure 7.1 the Ψ-chart is shown that was introduced in the previous chapter (Figure 6.8). This chart shows the general technique that can be applied to explore the fault-tolerant design space. Given the applications, the fault tolerance patterns and the architecture, a fault-tolerant mapping can be made that is evaluated using SAFE. Based on the performance numbers of the design instance (like performance, power or frame drop ratio of the given fault-tolerant mapping), the input of SAFE can be adapted. One can adapt the applications, the fault tolerance patterns or the architecture, but in this thesis we will only focus on changing the fault-tolerant mapping. The manipulation of the fault-tolerant mapping can be automated, whereas a change in the architecture (adding for example a processor, because all the processors are over-utilized) cannot easily be automated.
Apart from the automatic exploration of the fault-tolerant mapping, it is also important to focus on the dynamism of the architecture. The previous chapter only took the fault-tolerant mapping into account. For the SAFE simulations, the faults were injected randomly into the architecture. This sequence of injected faults was only generated once per generation and used for all the fault-tolerant mappings that needed to be compared. However, these injected faults are also dynamic in nature. As we already discussed in the related work of SAFE (Section 6.6), some approaches capture the dynamics of the transient faults into fault scenarios that are used to obtain the reliability of a system. On itself, SAFE cannot yet take reliability into account, apart from the frame drop ratio of a single specific fault scenario. Therefore, this chapter will utilize so-called architecture scenarios (i.e., fault scenarios) to describe how reliable the obtained objectives are of a given fault-tolerant mapping. This chapter starts by giving an overview of the fault-tolerant DSE framework. Next, the two main components of the fault-tolerant DSE framework are discussed in Sections 7.2 and 7.3. Finally, the chapter will be closed by two case studies and a conclusion.

7.1 Overview

In Section 5.1, we already introduced a framework to statically explore the design space to take the dynamism of application scenarios into account. In this chapter we do something similar, only in this case we are not dealing with application scenarios, but with architecture scenarios. An architecture scenario defines the sequence of independent transient faults of an architecture that will occur with the time and place of each individual fault. Not only influence architecture scenarios the quality of the design instances (as was the case with the application scenarios in Chapter 5), but a fault-tolerant mapping must also be able to cope with faults that are listed

Figure 7.1: The $\Psi$-chart that shows an exploration method the fault-tolerant design space can be explored by using SAFE.
Figure 7.2: The overview of the fault-tolerant DSE exploration framework.

in the architecture scenario. More detail on architecture scenarios will be given in Section 7.3.

Figure 7.2 shows the overview of the fault-tolerant DSE framework. Conceptually, this framework shows similarities to the framework of Chapter 5 that deals with application scenarios. There are two main components: the fault-tolerant mapping DSE (or shortly FMMapping DSE) and the subset DSE. Similarly to Chapter 5, the FMMapping DSE explores the different design instances given the subset of scenarios. This subset of scenarios is selected by the subset DSE that tries to select a representative subset of scenarios based on the current set of design instances in the FMMapping DSE.

The FMMapping DSE explores the complete range of possibilities of a fault-tolerant mapping. In contrast to a normal mapping, this not only includes the binding of the application processes onto the architecture, but also the patternization of the application that transforms the application into a fault-tolerant application (for further detail see Section 6.3). Corresponding to the two hierarchical steps during a fault-tolerant mapping, the FMMapping DSE uses two populations: a patternization population and a binding population. The patternization population defines the fault-tolerant application, whereas the binding population maps these fault-tolerant applications onto the architecture.

In the subset DSE, a representative subset of architecture scenarios is selected. Just as Chapter 5, this raises the question ”What is representative?”. To answer this question, one must consider what a designer wants to know when it designs a fault-tolerant embedded system. In our perspective, this is how resilient the embedded system is with respect to transient faults in the architecture (i.e., the architecture scenarios). Resilience in this case means that the rate of change of the objectives of a system when the number of transient faults increases. Ideally, one wants a
system that is affected by these transient faults as least as possible. Therefore, the fault-tolerant DSE will explore the design space given a subset of architecture scenarios with different rates of transient faults. The subset selector will identify these architecture scenarios such that the worst-case fitness is found for each transient fault rate (as will be discussed later in Section 7.2.3).

The complete framework is implemented using a combination of C++ and the Message Passing Interface (MPI). To evaluate the fault-tolerant mappings both the FMapping DSE and the subset DSE make use of a dedicated set of SAFE workers. These workers are implemented in a separate MPI process. Due to the coarse grained nature of SAFE jobs (processing a single job will normally take a couple of seconds); the combination of C++ and MPI is efficient enough to quickly explore the fault-tolerant design space.

As heuristic search method, both the FMapping DSE and the subset DSE use a modified GA. This modification is elaborated in the next two sections where the two main components are described in more detail. To clarify these explanations both of the sections will use the MJPEG application in Figure 7.3 as an example of an application that must mapped onto an architecture in a fault-tolerant fashion to optimize the different objectives of the embedded system. This example, which was used earlier in the previous chapter, searches for a fault-tolerant design of a MJPEG encoder that is mapped onto an architecture with two unreliable processors (PROC-1 and PROC-2). To achieve this, two fault tolerance patterns are available: a fault-tolerant processor (PROC-FT) or active redundancy that replicates the computation on the unreliable processors in time and / or space after which it checks the output of the different replicas with a majority voter (the NMR component).

### 7.2 Fault-Tolerant Mapping DSE

A fault-tolerant mapping describes the complete transformation of an application to a mapped fault-tolerant application. This involves three steps (Section 6.3): 1) patternization, 2) binding and 3) dispatch. The patternization segregates the applica-
Figure 7.4: The dimension oriented approach to co-explore the patternization and the binding of the fault-tolerant mappings. Gray scale is used to differentiate between the different patternizations of the mappings, whereas a dashed line type is used to emphasize the offspring mappings.

...tion into distinct subnetworks and applies fault tolerance patterns on the individual subnetworks. During the binding, these subnetworks are mapped onto the architecture together with the OWP processes of the application. Finally, the message dispatch determines the routing of the data that is exchanged between the different processes. As discussed earlier, the message dispatch remains fixed and, therefore, the FMapping DSE only needs to explore the patternization and the binding.

Basically, the patternization groups the processes of the application into sets to optimize the fault-tolerant mapping. To address these kind of grouping problems, traditional GAs do not suffice. Therefore, the exploration of patternizations uses a special representation for the chromosomes where the group part and the object part are separated [17]. The object part assigns the processes to the subnetworks, whereas the group part selects the fault tolerance patterns. For most of the grouping problems, the fitness of the partitioned objects can directly be obtained. One example is the bin-packing problem that can directly determine if the total size of the objects in a bin does not violate its constraints. Combined with a cost function that promotes the minimization of the number of bins, the search becomes straightforward. For our patternization problem, however, the fitness can not directly be obtained. Given a patternization, the application is already fault tolerant, but without a binding the quality (like energy, frame drop ratio, etc.) of the patternization cannot be evaluated. Therefore, the fault-tolerant DSE uses a dimension-oriented approach [33] to simultaneously co-explore both the patternization and the associated binding. This approach is illustrated in Figure 7.4 and it consists of five steps:
1) Search: The search step is responsible for creating offspring for the next generation of the genetic algorithm. It maintains a separate population for both the patternization and the binding. As a fault-tolerant mapping is a combination of both, each of the binding chromosomes is linked to a patternization chromosome. This patternization chromosome defines the transformed fault-tolerant application and, thus, the processes that need to be bound to the architecture. As it is hard to completely judge about a patternization using only a single binding, there is a one-to-many relation between the patternization and bindings. Therefore, a single patternization is quite likely to have several bindings in the binding population. This is also illustrated in Figure 7.4. Each of the patternizations has a different gray scale. Additionally, the binding population consists of multiple subpopulations: one for each patternization. Patternization chromosome A has only one associated binding chromosome (number 1) in its subpopulation, whereas the subpopulation of patternization chromosome B has five linked binding chromosomes (numbers 2, 3, 4, 6 and 7).

Creating offspring for the GA is done separately for both the patternization population and the binding populations that belongs to each of the patternizations. For this purpose, specialized genetic operators are used (as will be discussed later in Section 7.2.2) to create slightly manipulated individual chromosomes that are based on the selected chromosomes. For the binding chromosomes these genetic operators are sufficient to create the offspring, but to make the offspring of the patternization population complete a subpopulation of binding chromosomes must be generated. Without a subpopulation of binding chromosomes the patternization chromosome cannot be evaluated since only complete fault-tolerant mappings can be evaluated. One option would be to randomly generate matching binding chromosomes, but, quite likely, this will result into fault-tolerant mappings of a low quality. To improve the probability that an offspring patternization chromosome survives it to the next generation, the generated binding chromosome is based on an existing binding chromosome. This binding chromosome is selected using the binding chromosome that belong to the original patternization chromosome (the chromosome on which the patternization offspring is based). To make a valid binding for the new patternization, the changes in the patternization are reflected in the binding chromosome. If, for example, the patternization adds an additional replica to a fault-tolerant subnetwork, the binding is extended with an extra gene that maps this additional replica. Similarly, when a replica of a fault-tolerant subnetwork is removed, the gene binding of this replica will also be removed from the binding chromosome.

Figure 7.4 shows the offspring chromosomes using a dashed border. Chromosomes 6 and 7 are offspring chromosome from the binding population that are based on the parent chromosomes 2 and 4. There is only one offspring chromosome in the patternization population: chromosome C with parent chromosome A. The patternization chromosome also has an associated binding chromosome to complete the new fault-tolerant mapping. Adjusting the binding chromosome 1 of the parent chromosome
A creates the new binding chromosome 5.

2) **Combine:** The next step is to combine all the patternization and binding chromosomes to create fault-tolerant mappings that can be evaluated using SAFE. Since each of the binding chromosomes is linked to exactly one patternization chromosome, each of the binding chromosomes is transformed into a fault-tolerant mapping by combining it with the associated patternization chromosome.

3) **Evaluate:** Combined fault-tolerant mappings are evaluated with SAFE using a representative subset of architecture scenarios. Based on the architecture scenarios that are obtained from the subset DSE (see Figure 7.2), the fitness of each of the fault-tolerant mappings can be obtained.

4) **Select:** Using the fitness of the fault-tolerant mappings, the parent for the offspring of the next generation is selected. For this purpose, the NSGA-II [12] is used.

5) **Split:** As there are two separate populations, the selected parents cannot directly be used to select the offspring for the search step. Therefore, the evaluated fault-tolerant mapping population is split into two populations again. Each of the chromosomes is split in its patternization and binding part. The binding chromosomes can directly be used to produce the binding population. For the patternization chromosomes, however, the redundant copies of the patternizations are removed first. In Figure 7.4, for example, patternization chromosome B is only added once to the population.

Next, the selected parents for offspring are split into two parts. A small set of them will be used to create offspring in the patternization population and the rest will be used for the offspring in the binding population. In the offspring example given in the search step, fault-tolerant mapping chromosomes A1, B2 and B4 were selected as parents. Chromosome A1 was assigned to the patternization offspring (which applied genetic operators on patternization chromosome A and adjusted binding chromosome 1 to match the changes in A). The other two chromosomes B2 and B4 were used for the binding offspring. This means that patternization chromosome B is kept intact and the genetic operators are only applied on binding chromosomes 2 and 4.

### 7.2.1 Chromosome Representation

One of the aspects that affect the search efficiency of the GA is the chromosome design. In the introduction of this section, it was already emphasized that the grouping of the processes must carefully be encoded into the patternization chromosome. These patternization chromosomes, however, only describe the grouping of application processes into fault-tolerant subnetworks. A binding chromosome is required to map these fault-tolerant subnetworks onto the architecture. In this subsection, we will describe both of the patternization and binding chromosomes using a fault-tolerant mapping of the MJPEG application in Figure 7.3. More specifically, we
Figure 7.5: An illustration of a pattern chromosome for the MJPEG application.

describe the chromosomes that result into the fault-tolerant mapping that is shown in Figure 6.12 on page 142.

At first, the application is segregated into a set of fault-tolerant subnetworks. For each of these subnetworks, a fault tolerance pattern is selected. With respect to the design of the GA, this resembles a global grouping problem like bin packing or graph coloring [17]. Therefore, the chromosome consists of two parts: the object part and the group part. In the object part the application processes are assigned to subnetworks. To describe the properties of these subnetworks, the group part is used. An example of a patternization chromosome for the MJPEG application is given in Figure 7.5. The first six genes are the object part where, for each of the application processes (RGB2YUV, DCT, Q, VLE, DMUX and CONTROL), the number of the used subnetwork is encoded. This grouping must comply with the conditions of a fault-tolerant mapping that was given in Section 6.3. One of the most important conditions that must be enforced is that all subnetworks are weakly connected (Equation 6.7).

Properties of the subnetworks are encoded in the group part of the chromosome. Strictly seen, a single gene per group that encodes the fault tolerance pattern of the subnetwork (the pattern field in Figure 7.5) is sufficient. For efficiency purposes, however, some derived information of the fault tolerance pattern is also encoded in the patternization chromosome. This derived information includes: 1) the number of replicas of the fault tolerance patterns (#Replica), 2) the number of potential reliable architecture elements that can be used for mapping the voter process (#Reliable) and 3) the number of architectural processor elements that can be used to map the replica processes (#CPU). For every fault tolerance pattern \( f \in F \), these numbers are derived as follows:

\[
#\text{Replica}(f) = n_{\text{proc}}(f) \\
#\text{Reliable}(f) = |\{ r | r \in R_R \land \text{feasible}(f, r) \}| \\
#\text{CPU}(f) = |\{ r | r \in R_P \land \text{feasible}(f, r) \}|
\]

As an example, the patternization of the MJPEG application that is encoded in the chromosome in Figure 7.5 is the same as the patternization shown in Figure 6.9. In the object part the grouping into two subnetworks is encoded: subnetwork 1 (RGB2YUV, DCT and DMUX) and subnetwork 2 (Q, VLE and CONTROL). Subnetwork 1 is guarded using a DMR pattern. DMR has two replicas and for the
architecture in Figure 7.3b it has one feasible reliable architecture element for the voter and two feasible processor elements (for the definition of the feasible function see Table 6.1). Similarly, the second subnetwork uses a fault-tolerant processor that has one replica, one feasible reliable architecture element and one feasible processor architecture element.

One of the main purposes of including the derived properties of the subnetwork inside the patternization chromosome is that the binding chromosome can easily be manipulated. Only the linked patternization chromosome is required during manipulation. Just as the patternization chromosome, the binding chromosome consists of two parts. The first part encodes the IO binding, whereas the second part describes the computational binding. At first, the IO binding maps each OWP process onto the architecture. Therefore, it is completely independent of the patternization chromosome. The computational binding, on the other hand, is completely based on the group part of the patternization chromosome. For each subnetwork, a set of genes maps the voter process and the process replicas. From these genes, the first gene encodes the reliable component onto which the voter process is mapped using an integer that addresses one of the reliable components (Equation 7.2). All the remaining genes address one of the processor elements (Equation 7.3) that are used for binding the specific replica. Consequently, the total number of genes per subnetwork is one more than the number of replicas (Equation 7.1).

An example of the binding for the MJPEG application is given in Figure 7.6. Starting from the MJPEG patternization in Figure 7.5, a corresponding binding is created that is equal to the binding shown in Figure 6.10. For the IO binding there is only one feasible mapping: for OWP0 the CAMERA component is used and for OWP1 the DISK component is used. Since the first subnetwork has two replicas, the next three genes encode the binding of the first subnetwork. The first gene selects the NMR component as the reliable element, whereas the next two genes select processors PROC-1 and PROC-2 for the replica processes. In the second subnetwork, a fault-tolerant processor is used. The binding of such a pattern is quite straightforward. As the processor is already reliable, the pattern only requires a single element: the PROC-FT processor that is used both as reliable and as processor element.

Figure 7.6: An example of a binding chromosome based on the patternization shown in Figure 7.5
7.2.2 Genetic Operators

After defining the chromosome representations, the genetic operators need to be defined. For the binding chromosome, the traditional implementations can be used (see Section 2.2) as the chromosome uses a list of integers that can freely be manipulated. Within this chromosome, the alleles are defined by the feasibility function (for the IO binding) or by the linked patternization chromosome (for the computational binding). We should note that Figure 7.6 does not show integers, but the names of the architectural elements. This is purely done for visualization purposes.

For the patternization chromosomes specialized genetic operators are used. Important of these genetic operators is that they operate on the group part and not on the object part. Operating on the group part does not mean that the object part remains the same. A change in a group can result in the change of some of the references in the object part. The main reason for using a group-based genetic operator is that it improves the efficiency of the genetic algorithm [17]. Firstly, the group-based mutation operator can perform one of the following actions:

- Use a different fault tolerance pattern for a subnetwork.
- Move a subset of the nodes of a subnetwork to another subnetwork (existing or new).

Important is that after applying the specialized mutation operator the patternization chromosome may need to be repaired. At first, the derived group properties are updated. Secondly, there is verified if all the subnetworks are weakly connected. In case the nodes in a subnetwork are not weakly connected, the subnetwork is split into multiple subnetworks such that each of the individual subnetworks becomes weakly connected.

Our specialized uniform crossover operator is currently only defined to apply crossover for a multi-application workload. To be certain that the chromosome remains valid, only the complete set of subnetworks per application is exchanged during a crossover operation. Due to the weakly connected subnetworks, it is impossible that within a subnetwork processes of different applications are present. Therefore, on exchanging the complete set of subnetworks of an application the patternization chromosome will always remain valid.

7.2.3 Fitness Function

The fault-tolerant DSE provides a design space exploration for different rates of transient faults. To achieve this, the fitness function of a fault-tolerant mapping evaluates the fitness of different reliability classes. A reliability class characterizes the maximal probability of the architecture scenarios that are considered while determining the quality of the fault-tolerant mapping. For each of the different reliability classes, the worst-case objective is minimized. Consequently, the complete optimization problem can be posed as follows:
When a fault-tolerant mapping $m$ is optimized for $k$ different objectives (like energy or frame drop ratio), the worst-case objectives are obtained for each of the $n$ reliability classes independently (Equation 7.4). In order to obtain the worst-case objective $i$ for reliability class $c$, the fault-tolerant mapping $m$ is evaluated with $\text{SAFE}$ to find the maximal value of the $i$-th objective for all the architecture scenarios that belong to the reliability class $c$. As Equation 7.5 shows, this is done by simulating the fault-tolerant mapping $m$ with $\text{SAFE}$ for each of the architecture scenarios that have a probability lower or equal to the probability of reliability class $c$. After the $\text{SAFE}$ simulation, the maximal value of the objective is determined.

An example of a fitness calculation of a fault-tolerant mapping is given in Table 7.1. This mapping is evaluated for 6 architecture scenarios for the two objectives power consumption and frame drop ratio. Before starting the fault-tolerant DSE, it is decided that two reliability classes will be investigated: one for architecture scenarios with a probability of 50 percent or lower ($c_1$) and one for architecture scenarios with a probability of 90 percent or lower ($c_2$). By definition, reliability classes with a lower probability are contained in reliability classes with higher classes ($c_1 \leq c_2 \iff S^{c_1} \subseteq S^{c_2}$). As set of the architecture scenarios belonging to $c_2$ is a superset of the architecture scenarios of $c_1$, the minimal worst-case objectives for the reliability class $c_2$ is also always larger or equal to the objectives of reliability class $c_1$. For our example architecture scenarios $s_1$, $s_2$ and $s_3$ belong to reliability class $c_1$. The worst-case power for these scenarios is 0.7. Similarly, the worst-case frame drop ratio of reliability class $c_1$ is 0.1. Reliability class $c_2$ contains all the listed

<table>
<thead>
<tr>
<th>Architecture Scenario</th>
<th>Probability</th>
<th>Power</th>
<th>Dropping (ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>0.3</td>
<td>0.5</td>
<td>0.0</td>
</tr>
<tr>
<td>$s_2$</td>
<td>0.4</td>
<td>0.4</td>
<td>0.1</td>
</tr>
<tr>
<td>$s_3$</td>
<td>0.4</td>
<td>0.7</td>
<td>0.1</td>
</tr>
<tr>
<td>$s_4$</td>
<td>0.7</td>
<td>0.7</td>
<td>0.15</td>
</tr>
<tr>
<td>$s_5$</td>
<td>0.7</td>
<td>0.6</td>
<td>0.2</td>
</tr>
<tr>
<td>$s_6$</td>
<td>0.8</td>
<td>0.6</td>
<td>0.25</td>
</tr>
</tbody>
</table>

$c = [c_1, c_2] = [0.5, 0.9]$

$F(m) = [F_{\text{Power}}^{c_1}, F_{\text{Dropping}}^{c_1}, F_{\text{Power}}^{c_2}, F_{\text{Dropping}}^{c_2}]$

$= [0.7, 0.1, 0.7, 0.25]$

Table 7.1: An example of the fitness calculation of a fault-tolerant mapping for two reliability classes 0.5 and 0.9.

minimize $F(m) = [F_{i1}^{c_1}(m), ..., F_{ik}^{c_1}(m), ..., F_{i1}^{c_n}(m), ..., F_{ik}^{c_n}(m)]$ (7.4)

$F_{i}^{c}(m) := \max(\{\text{SAFE}_i(m,s) | s \in S \land \text{Prob}(s) \leq c\})$ (7.5)
architecture scenarios. As we will see in the next section, a higher probability of an architecture scenario corresponds to a higher fault rate. This means that, generally, the objective values are worse. In our example, the worst-case power is not affected and remains 0.7. The frame drop ratio, however, has grown to a ratio of 0.25.

7.3 Subset DSE

The fitness calculation of the fault-tolerant DSE shows a similar problem as the scenario-based DSE of Chapter 4 and Chapter 5 where the mapping fitness was calculated for the different application scenarios in the multi-application workload. During the early DSE, it is not feasible to evaluate all the possible scenarios. For the architecture scenarios it is not even possible to exhaustively evaluate a single fault-tolerant mapping for all architecture scenarios. There are simply too many scenarios in which transient faults can influence the architecture. That is why the fault-tolerant DSE also requires a simultaneous exploration of the design space of the architecture scenarios.

Before the exploration of the architecture scenarios can be described, a detailed definition of an architecture scenario must be given. An architecture scenario is a sequence of independent transient faults with a certain time and place. Currently, we do not consider dependent transient faults and permanent faults. For each of the architecture components that are susceptible to faults, a sequence of independent transient faults is defined. Within this fault sequence, the fault rate depends on the probability of the architecture scenario, whereas the placement of faults is completely random. Recall from the previous chapter (Section 6.4.1) that a Poisson distribution is used to model independent transient faults. This distribution can only provide the probability of a given number of events within an interval. Hence, the Poisson distribution can be used to obtain the expected waiting time on an event (as was done in the previous chapter) because waiting time corresponds to an interval in which no events occur. A Poisson distribution, however, cannot be used to model the way in which multiple faults are spread in an interval. Consequently, the probability of an architecture scenario is only dependent on the number of potential faults and not their placement:

\[
\text{Prob}(S) = \max_{r \in R_F} (\text{Prob}_r(x \leq |\text{faults}(S, r)|)) \tag{7.6}
\]

\[
\text{Prob}_r(x \leq X) = e^{-\lambda_r} \sum_{i=0}^{X} \frac{\lambda_r^i}{i!} \tag{7.7}
\]

To be precise, the probability of an architecture scenario is the maximum of all the probabilities of the transient faults in the individual architecture components (Equation 7.6). For each of the components \( r \in R_F \), the architecture scenario contains a number of potential transient faults (|faults(S, r)|). Using the Poisson distribution,
the probability of at most $X$ potential transient faults can be obtained by using the cumulative distribution function that is shown in Equation 7.7. Within this formula, $\lambda_r$ is the average number of transient faults of the architecture component $r$ during the simulation time. This average rate is obtained using the following formula:

$$\lambda_r = \frac{\text{time}}{\text{MTTF}_r}$$

(7.8)

For time, the deadline of the last frame in the multi-application workload is used. Together with the Mean Time To Failure (MTTF) of the architectural resource, the expected number of transient faults is obtained.

The FMapping DSE and the subset DSE are competing with each other. In the FMapping DSE the objective values are minimized, whereas the subset DSE tries to maximize the objective values. For the representative subset, this means that it contains architecture scenarios that challenge the fault-tolerant mapping as much as possible (i.e., make the worst-case objective values as large as possible), but still have a probability that is small enough to be contained in the associated reliability class.

In Figure 7.7, an overview is given of the exploration of the representative subset of architecture scenarios. For each of the reliability classes, a separate GA is used to identify the representative architecture scenarios. Generally speaking, the populations are training a fitness predictor for the worst-case fitness. This closely resembles the approach that is used in [63]. The GA will optimize the fitness predictors (i.e., architecture scenarios) such that the predicted fitness is as close to the real worst-case fitness as possible.

From each of the reliability class populations, the best architecture scenarios are selected as the representative subset of scenarios. This periodically updated representative subset of scenarios is used to predict the fitness of the fault-tolerant
mappings in the FMapping DSE. From the FMapping DSE, on the other hand, fault-tolerant mappings are obtained that are used for the trainer. This trainer contains a set of fault-tolerant mappings (including the current Pareto front) that is used to evaluate the quality of each individual architecture scenario.

### 7.3.1 Chromosome Representation

To represent an architecture scenario, a chromosome is used that encodes the time for each of the transient fault that occurs. This is done separately per architecture element that is susceptible to faults (i.e., the fault rate is larger than zero).

Figure 7.8 shows an example of an architecture scenario for the architecture of Figure 7.3b. In the first gene, the probability of the complete scenario is shown. This gene is static and it depends on the reliability class in which the architecture scenario is located. The shown chromosome is located in reliability class 0.25 (i.e., its probability of occurrence is at most 25 percent). After the first gene that encodes the reliability class, the transient faults of the different components are encoded. The first group of genes is for processor PROC-1 and the second group of genes is for processor PROC-2. As we are looking for the worst-case architecture scenario, the potential number of transient faults should be as large as possible. Therefore, the number of potential faults $X_r$ for architectural element $r$ in reliability class $c$ is the largest integer satisfying:

$$\text{Prob}_r(x \leq X_r) \leq c \quad (7.9)$$

In our example, this results into six potential transient faults for PROC-1 and four potential transient faults for processor PROC-2. The genes per group are chosen in such a way that it is a sample from all the integers in the range from zero to $\text{time}$ (where $\text{time}$ is the deadline of the last frame in the multi-application workload). Hence, multiple transient faults in the same component at the same time are not allowed. Whenever multiple things go wrong at the same time, it is still a single fault. There may, however, be a transient fault at multiple elements simultaneously.

### 7.3.2 Genetic Operators

The genetic operators for the architecture scenarios are rather straightforward. Upon mutation, the time of a transient fault is changed into a value that is not yet present
within the current group of transient faults of the specific architectural element. For the crossover, a one-point crossover is used (see Figure 2.8 at page 25): the first part of the chromosome is exchanged between the two scenario chromosomes of the same reliability class. In case the crossover results in a chromosome where a component has two transient faults at the same time, one of the transient faults is mutated.

7.3.3 Fitness Function

After manipulating an architecture scenario its fitness must be determined in order to know if the change to the architecture scenario was beneficial or not. Since the FMapping DSE tries to optimize the worst-case objectives, the predicted fitness of the architecture scenario must be as close to the worst-case objective as possible. Therefore, the fitness function $F_{\text{scen}}$ becomes as follows:

$$F_{\text{scen}}(s) = \frac{1}{|T|} \sum_{t \in T} d(\text{SAFE}(t, s), \text{WC}(t))$$  \hspace{1cm} (7.10)

To maximize the objective values of the training mappings, the fitness function tries to minimize the average normalized Euclidean distance $d$ (see Definition 5 on page 28) between the fitness that is obtained by the SAFE simulator and the worst-case (WC) objective values for the fault-tolerant mappings $t$.

In order to evaluate this fitness, two issues must be addressed: 1) how to determine the trainer $T$ and 2) how to obtain the real worst case objectives of each training mapping. To address these issues, we propose a modified GA procedure for the subset DSE that is shown in Figure 7.9. The gray steps in this picture show the
steps that are taken for the GA on all the reliability populations (as shown in Figure 7.7). These steps correspond to the steps of a generic GA: population initialization, fitness calculation of all the architecture scenarios, selection of the parents for the next generation and creating offspring from these parents. As there is one population per reliability class, the GA steps need to be done for each of the populations.

Three steps of our procedure are dedicated to the trainer. In the initialization, which is both gray and white as initialization is done both for the GA and the trainer, an initial trainer is created. At this point, the worst case fitness is not known. As it is not feasible to exhaustively evaluate training mappings with all possible architecture scenarios, the worst-case fitness will be estimated simultaneously with the search for the representative architecture scenarios. Before the fitness of an architecture scenario can be obtained, each of the training mappings must be evaluated with this architecture scenario (see equation 7.10). Possibly, the evaluation of the architecture scenario will update the worst-case fitness of one of the training mappings. Therefore, the longer a mapping is in the trainer, the more reliable its worst-case fitness is.

After the parent chromosomes of the architecture scenarios are selected for the next generation, the representative subset of scenarios is updated. At this point in time, the population only consists of scenarios with an evaluated fitness. Moreover, the NSGA-II procedure has removed the individuals with a low fitness. The update of the representative subset of scenarios also allows us to update the trainer with fault-tolerant mappings from the FMMapping DSE. These mappings are already evaluated and can, therefore, be added directly. From the imported fault-tolerant mappings, the mappings are removed that are already in the trainer. After that, the new fault-tolerant mappings are added.

The final step is to control the size of the trainer. As a SAFE simulation must be done for each training mapping, the trainer should be kept as small as possible. For this purpose, non-dominated sorting is used [12]. First, the training mappings are sorted on their Pareto dominance rank. If the ranks of two mappings are the same, the crowding distance is used. The crowding distance corresponds to the average distance of neighboring mappings (with respect to fitness value). A higher average distance means a better trainer as there is a higher diversity of mappings. After sorting the training mappings, the tail of the list of training mappings is removed to truncate the trainer.

This trainer is not only beneficial to evaluate the quality of the architecture scenarios to predict the worst-case fitness. It is also used as an elite population of the best fault-tolerant mappings that are found during the fault-tolerant DSE. As this elite population is kept in the subset DSE and not in the FMMapping DSE the high quality fault-tolerant mappings will be evaluated for a more diverse population of architecture scenarios than it would be the case in the FMMapping DSE. Therefore, the FMMapping DSE quickly selects promising fault-tolerant mappings using the representative subset of scenarios. This set of promising fault-tolerant mappings is evaluated in more detail by the subset DSE. As a result, the fault-tolerant DSE will
end up with a Pareto front that is as precise as possible.

## 7.4 Case Studies

Due to the limited amount of time that was left before the thesis deadline, this chapter only shows two case studies of the fault-tolerant DSE. Future work should incorporate more extensive experiments to fully analyze the fault-tolerant DSE framework. In the first case study of this chapter, a single fault-tolerant DSE of a multi-application workload is performed. This multi-application workload consists of two applications, namely a MJPEG encoder application and a Sobel edge detector. For a static multi-application workload, the resulting Pareto fronts of the different reliability classes are obtained and analyzed. We will also investigate the effect of the reliability classes on a single fault-tolerant mapping. Basically, this shows the resilience of the different fault-tolerant mappings to the potential faults in the architecture. For the second case study, the effect of the frame period on the drop ratio of both a Sobel edge detector and an MP3 decoder is investigated.

The three different applications (a MJPEG encoder, a Sobel edge detector and an MP3 decoder) are taken from the experiments in the previous chapter (Section 6.5). A fixed workload of 63 frames is used for the Sobel edge detector and the MP3 decoder. Our used workload for the MJPEG encoder consists of smaller images sizes than the Sobel edge detector and, therefore, 267 frames are modeled for the MJPEG encoder. Each of the frames starts eight periods before the final deadline of the specific frame. In the second case study, the frame period is a parameter of the experiment. Instead, the first case study fixates the frame period: the MJPEG encoder has a frame period of 40\(K\) cycles (i.e., with one million cycles per second this would be 25 frames per second). On the other hand, the Sobel edge detector uses a frame period of 160\(K\) cycles. As a result, the deadline of the final frame of the MJPEG encoder and the Sobel edge detector is equal.

All of these applications are mapped onto the architecture that is shown in Figure 7.10. This architecture has four general-purpose processors with a mean time to failure of 10^5. As was discussed in the previous chapter, such a fault rate is relatively large and will result very pessimistic frame drop ratios. However, the reliability classes that were introduced in this chapter make it possible to show the behavior
Table 7.2: Experimental settings

<table>
<thead>
<tr>
<th></th>
<th>Population size</th>
<th>Offspring size</th>
<th>P\text{crossover}</th>
<th>P\text{mutate}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Pattern population</td>
<td>30</td>
<td>24</td>
<td>0.8</td>
<td>0.02</td>
</tr>
<tr>
<td>(b) Binding population</td>
<td>120</td>
<td>300</td>
<td>0.8</td>
<td>0.02</td>
</tr>
</tbody>
</table>

of a single fault tolerant mapping for a wider range of fault ratios at the same time. In our experiments four reliability classes are used. At first, reliability class $C_0(0\%)$ corresponds to the situation without any transient faults. Next, reliability class $C_1(15\%)$ corresponds to a situation with a relative low number of transient faults. The other two reliability classes $C_2(85\%)$ and $C_3(99.9\%)$ correspond to situations where the application is stressed with a large number of transient faults.

Additionally, our architecture also contains a NMR component to implement active redundancy. As the architecture only has a NMR component, the set of fault tolerance patterns is limited to different flavors of DMR (active redundancy with two replicas) and TMR (active redundancy with three replicas). In the previous chapter, two parameters were used for the active redundancy pattern: the restart budget and the checkpoint budget. The restart budget specified the maximal number of times that a single frame was restarted. If a voter was out of the budget, the frame was dropped. For this chapter, we adapted our model: a frame will only be restarted if there is sufficient time left. In all other cases, it is dropped. As a result, for both the DMR and TMR we have seven different versions: 1) one pattern without restart capabilities, 2) one pattern with restart capabilities, but without explicit checkpoints and 3) five patterns with restarting capabilities and an explicit checkpoint budget (10, 20, 30, 40 and 50 explicit checkpoints per million cycles).

Table 7.2 shows some of the GA parameters of the FMMapping DSE component of the fault-tolerant DSE. In the subset DSE component a representative subset of architecture scenarios will be searched with at most three scenarios for each of the four reliability classes. This complete fault-tolerant DSE is run on the DAS-4 cluster [2] using five dedicated nodes with two 2.4GHz quad core Intel E5620 processors. On each node 16 MPI processes are placed to perform latency hiding (each node has eight hardware threads).

### 7.4.1 Pareto Front for a Fault-tolerant Multi-Application Workload

In the first case study, a fault-tolerant DSE of a multi-application workload with a MJPEG encoder and a Sobel edge detector is performed. During this fault-tolerant DSE four reliability classes were used: 0\%, 15\%, 85\% and 99.9\%. As explained
Figure 7.11: An excerpt of the Pareto front that is obtained from a DSE for a fault-tolerant implementation of a MJPEG application.

earlier, a higher probability in a reliability class corresponds to a higher number of potential faults. This means that the reliability class of 0% has no potential faults at all, whereas the reliability class with 99.9% has the most potential faults.

Figure 7.11a shows the trade-off between the frame drop ratio (X-axis) and the average power (Y-axis) of the Pareto optimal fault-tolerant designs. As there are two applications in our multi-application workload, the objectives contain two individual frame drop ratios. Together with the average power, this would result in a three-dimensional Pareto front in which it is quite hard to visually compare the different mappings. Instead, we have chosen to show the maximum frame drop ratio of the individual MJPEG and Sobel applications. As a result, a two-dimensional Pareto
front is obtained. To further improve the clarity of the Pareto front, there is also chosen to only show the Pareto front for designs with a frame drop ratio of 25 percent or lower. Low power designs with a higher frame drop ratio are left out. Apart from the fact that designs with a high frame drop ratio are not interesting as a final design, most of these designs achieve a low power by dropping frames early in the process. Hence, a lot of energy is spared, but this is done by completely ruining the QoS of the system.

A letter between A and H indicate all of the eight different non-dominated fault-tolerant mappings in the Pareto front. In Figure 7.11b a description is given of each of these mappings. At first, all of the non-dominated fault-tolerant mappings have a single subnetwork per application. Given the small architecture and the single shared communication bus, the fault tolerance overhead (especially, the communication part) is too large to support the communication overhead of more than two fault-tolerant subnetworks. Therefore, the "best" fault-tolerant mapping with more than one subnetwork per application has a frame drop ratio of more than 96 percent. Since each application has a single fault-tolerant subnetwork, Figure 7.11b can show the patternization of each fault-tolerant mapping by showing per application which fault tolerance pattern is used in the fault-tolerant mapping. As an example, mapping A has a DMR pattern for both the MJPEG and the Sobel application. For the Sobel application, the DMR in mapping A is extended with restart capabilities (as is shown by the -R postfix after DMR). None of the non-dominated fault-tolerant mappings used explicit checkpointing. This was due to two reasons. At first, the computational overhead was too large; our target architecture only contains four processors that are used to support two fault-tolerant applications. Secondly, fault-tolerant mappings where the frame drop ratio was improved by using explicit checkpoints resulted in a significant increase of power. Given our multi-application workload and target architecture, a DMR with a nonzero explicit checkpoint budget had both a larger frame drop ratio and power usage than a TMR alternative where no explicit checkpoints were used.

Pareto Fronts of the Different Reliability Classes

Within the Pareto front of Figure 7.11a, all of the fault-tolerant mappings are shown for each individual reliability classes \( C_0 \) to \( C_3 \). Additionally, the Pareto front of each individual reliability class is obtained by comparing the worst-case objective values for the specific class. For reliability class \( C_0 \) only the mappings A, B and C are optimal. All of these mappings use a DMR for both the MJPEG and Sobel application. The only difference is that mapping A has restart capabilities for the Sobel application and mapping B has restart capabilities for the MJPEG applications. As reliability class \( C_0 \) has no potential faults, restart capabilities do not affect the power of a fault-tolerant design. A restart will only be done after a non-maskable fault, which will never occur at reliability class \( C_0 \). Therefore, the frame drop ratio and average power of mapping A, B and mapping C are the same for reliability class
Reliability class $C_0$ purely measures the overhead of the fault tolerance: do all frames meet the deadline (which is the case for mapping $A$ to $H$) and how much power is used by the fault-tolerant design. In the given mappings, the only parameter that affects the power usage is the total number of replicas that is used for each application.

As a higher reliability class also includes the architecture scenarios of the lower reliability classes, the frame drop ratio and power of reliability classes $C_1$ to $C_3$ are always larger or equal to their preceding classes. Except for mapping $E$ and $F$, all mappings have a nonzero frame drop ratio when the architecture is affected by faults. The more potential faults there are (i.e., the higher the reliability class), the higher the frame drop ratio. One of the most obvious examples is mapping $B$ that is Pareto optimal for reliability classes $C_0$ up to $C_2$. Due to the large growth in frame drop ratio, however, the mapping is not Pareto optimal any more for reliability class $C_3$.

**Fault Sensitivity**

For some cases the fault tolerance of a mapping is sufficient to keep the frame drop ratio constant on an increasing number of potential faults. This is the case for mapping $G$ and mapping $H$ where both applications are using a TMR and, therefore, the fault-tolerant mapping is able to tolerate the increasing number of potential faults at the reliability classes of 15 ($C_1$) and 85 ($C_2$) percent. Similarly, fault-tolerant mapping $E$ does not need to drop frames when the reliability class is lower or equal to 15 percent. For mapping $F$, the frame drop ratio even remains zero for the highest reliability class $C_3$ (99.9 percent).

This fault sensitivity of the power of a fault-tolerant design depends on the fault tolerance patterns that are used. This is highlighted in Figure 7.11c where for each mapping the average power of the individual reliability classes are shown. Mapping $C$, $D$ and $H$ are insensitive to the faults with respect to power. As none of the used patterns have enabled restarting, the fault-tolerant design does not use fault correction. Without any fault correction, faults only lead to an increased ratio of dropped frames, but not to an increase of power usage. The more fault correction that is applied within the system, the more sensitive the power usage of the system is to faults. For mapping $B$, $E$ and $F$, where both applications have a fault tolerance pattern with an enabled restarting mechanism, the power usage is affected more heavily by the reliability classes than within mapping $A$ and mapping $G$ that only have a single application that uses fault correction.

Due to the fault sensitivity, the Pareto dominance differs over the reliability classes. This can be seen for the Pareto dominance relation of mapping $A$ and $D$. The power of mapping $D$ is fault insensitive, whereas the power of mapping $A$ is fault sensitive (due to the restart mechanism that is used in the Sobel application). As discussed earlier, the frame drop ratio is equal to the maximum of both applications. For these mappings, the frame drop ratio is equal to the frame drop ratio of the MJPEG application. As both mappings use the same fault tolerance pattern for the
MJPEG application, the frame drop ratio of both mappings is equal. Initially at reliability class $C_0$, the power of mapping $A$ is lower than the power of mapping $D$. With a higher reliability class, however, the power of mapping $A$ increases. As a result, mapping $A$ is dominated by mapping $D$ in reliability class $C_3$.

Pareto Dominance over Different Reliability Classes

For mapping $G$ and $H$ the Pareto dominance relation also differs per reliability class. In this case, however, it is caused by the advantages and disadvantages of the restart mechanism. Mapping $G$ and $H$ both use TMR for the two applications in the multi-application workload, but in mapping $G$ the fault tolerance pattern in the Sobel application has an additional restart mechanism. As a result, mapping $G$ takes more power for reliability classes $C_1$ to reliability class $C_3$. For reliability class $C_1$ and $C_2$, the restart mechanism still leads to a lower frame drop ratio by correcting non-maskable faults. In case of reliability class $C_3$, however, the computational overhead of the restart leads to more deadline misses (which results in a larger frame drop ratio).

Finally, not only the chosen pattern matters, but also the binding influences the effect of the reliability classes on the quality of the mapping. Mapping $E$ and $F$, for example, have exactly the same patternization, but a different frame drop ratio. As a result, mapping $E$ is still Pareto optimal in reliability class $C_1$, but for reliability class $C_2$ and $C_3$ mapping $F$ dominates mapping $E$ with a 9 percent lower frame drop ratio. The computational binding of the fault-tolerant mapping is the cause of this difference in frame drop ratio. As illustrated in Figure 7.12, mapping $E$ binds two of the replicas of MJPEG to processor CPU1 and one replica to processor CPU3. In mapping $F$ one of the replicas of the MJPEG application is bound to processor CPU1 and the other two replicas are bound to processor CPU2. For both mapping $E$ and $F$, the replicas of the Sobel application are bound to processor CPU1. The processors in our architecture are homogeneous, so there is no difference between processor CPU2 and CPU3. Only the number of replicas on processor CPU1 differs.
In mapping \( E \), there are four replicas that share the risk of being affected by a fault on processor CPU1. Mapping \( F \), on the other hand has a more even distribution of faults. Theoretically, the worst-case frame drop ratio of both mappings should be the same (as faults are independent with respect to place and time). In practice, however, the representative subset of architecture scenarios should contain worst-case scenarios that both stress the MJPEG and Sobel application as much as possible. Since the potential faults in mapping \( E \) need to be spread over four replicas, a single architecture scenario can only stress one application at the time. For mapping \( F \), however, a single architecture scenario can stress both applications. Consequently, the representative subset of architecture scenarios is harder to find for mapping \( E \) than it is for mapping \( F \).

Mapping \( E \) and \( F \) emphasize the need of a representative subset of scenarios. The more representative a subset of scenarios is the better the fitness prediction of the fault-tolerant mappings become. In our current case study, the representative subset of scenarios only contained 12 architecture scenarios to facilitate a quick exploration of the design space of potential fault-tolerant mappings. As a result, poor fault-tolerant mappings are easily identified, but it becomes harder to correctly differentiate between the good mappings.

### 7.4.2 Frame period versus frame drop ratio

In the second case study, we shifted our focus to the frame period. Up to now, a fixed frame period was picked and used as a basis for a fault-tolerant DSE. With
the introduction of the unreliable architecture, however, it becomes even harder to pick a frame period of an application. Therefore, the second case study shows the relation between the frame drop ratio and frame period for the different reliability classes that are explored in the fault-tolerant DSE. For this purposes, we performed several fault-tolerant DSEs for different frame periods for both the MP3 and Sobel application. Each of the applications is run in isolation and the only objective that is considered is the frame drop ratio.

Figure 7.13 shows the result of the experiment. The horizontal axis shows the frame drop ratio and the vertical axis shows the lowest frame drop ratio that was encountered during the fault-tolerant DSE. This optimal frame drop ratio is shown for four different reliability classes: $C_0$ (0%), $C_1$ (15%), $C_2$ (85%) and $C_3$ (99.9%). Each DSE is shown with a single point, the lines are added for visualization purposes.

As reliability class $C_0$ only contains architecture scenarios with a probability of 0% or lower, the architecture scenario contains no potential faults. Therefore, it shows if there is a possible fault-tolerant mapping that is able to meet the deadlines of the application given the overhead of the fault tolerance patterns. Take for example the Sobel edge detector (Figure 7.13a) where a frame period of at least $75K$ is required to meet the deadlines of all the frames. If the frame period is lower than $75K$, the frame drop ratio quickly grows: at $65K$ the frame drop ratio is 4.7 percent and at a frame period of $60K$ it is already 52.3 percent. The MP3 application can meet its deadlines with a much lower frame period. For all the shown frame periods in Figure 7.13b, the MP3 application is able to meet its deadlines. Below a frame period of $5K$, the frame drop ratio quickly grows to 66.6 percent for a frame period of $4K$ (for visualization purposes, the graph is cut off at $5K$).

The number of corrupt frames is relatively low for the MP3 application. In case of the highest reliability class, 3.1 percent of the frames are dropped (i.e., 2 out of the 63 frames). Up to a frame period of $15K$, the best fault-tolerant mapping for the highest reliability class $C_3$ uses TMR without a restart mechanism. With a frame period of $16K$ and higher, the best fault-tolerant mapping also uses TMR, but then a restart mechanism is available. To be able to successfully restart a frame, the frame period should be at least $17K$. In the same fashion, the fault-tolerant DSEs are able to find a fault-tolerant mapping for reliability classes $C_1$ and $C_2$ with a zero frame drop ratio when the frame period is large enough to facilitate the restart of a frame.

Our Sobel edge detector suffers from a larger percentage of frame drops than our MP3 application. However, as Figure 7.13a shows, for all of the reliability classes the frame drops are resolved at a frame period of $135K$. Similarly to the MP3 application, at this frame period it becomes possible to extend the used TMR pattern in the fault tolerance pattern with a restart mechanism and a nonzero explicit checkpoint budget. With this increased fault tolerance, all of the faults can be corrected.
7.5 Conclusion

In this chapter, a fault-tolerant DSE framework is presented that uses architecture scenarios to explore the design space of potential fault-tolerant mappings. These fault-tolerant mappings optimize the embedded system given the potential transient faults in the architecture. One of the challenges during such a DSE is the dynamic nature of the transient faults: faults are independent and the time and place of transient faults is unpredictable. The only thing that can be given is the probability that a system is hit by a given number of transient faults within a specific interval. To describe this dynamic nature of the occurrence of transient faults within the architecture, an architecture scenario is used. Basically, an architecture scenario is a sequence of transient faults with the time and the architectural resource that is hit by the fault. For the time and placement of these faults, there are a huge number of possibilities. It is absolutely infeasible to evaluate a fault-tolerant mapping for all possible architecture scenarios and, therefore, a scenario-based framework is used. This scenario-based framework is composed of two components: an FMapping DSE and a subset DSE. In the subset DSE, a representative subset of architecture scenarios is selected to quickly predict the fitness of a fault-tolerant mapping. The FMapping DSE, on the other hand, searches for optimal fault-tolerant mappings by using two different two steps: finding optimal patternizations to transform the application into a fault-tolerant application and, given a patternization, finding optimal bindings of a fault-tolerant application onto the target architecture. To both search for the optimal patternizations and bindings, the FMapping DSE contains two separate populations: a patternization population and a binding population. For each patternization in the patternization population, a subpopulation of bindings is present in the binding population. In contrast to a single population with fault-tolerant mappings, this approach is able to easily identify successful patternizations and to exploit this knowledge during the design space exploration.

One of the main features of the fault-tolerant DSE framework is the use of reliability classes that give the reliability of the fitness of a fault-tolerant mapping. More specifically, a reliability class defines the maximal probability of the architecture scenarios that are used to determine the worst-case fitness of the fault-tolerant mapping. The higher the probability used in a reliability class, the more potential faults are injected in the unreliable target architecture when determining the fitness of a fault-tolerant mapping. Another use of reliability classes is to observe the resilience of the fault-tolerant mappings to an increasing number of transient faults. These reliability classes are integrated in the fault-tolerant DSE by simultaneously exploring the design space for all the different reliability classes. During the exploration, the fitness of a fault-tolerant mapping is the combination of the fitness values of the individual reliability classes. Hence, the representative subset contains architecture scenarios for all the different reliability classes.

At the end of this chapter case studies showed how the fault-tolerant DSE framework can be used to explore the design space of fault-tolerant embedded systems.
First, a Pareto front of fault-tolerant designs for a multi-application workload was shown. Within this Pareto front, the fault sensitivity of the mappings differs. For fault-tolerant designs that do not apply any fault correction the power is fault insensitive. The frame drop ratio of these mappings, however, quickly grows with an increasing number of potential faults. Therefore, a restart mechanism is required to achieve a low frame drop ratio when the number of potential faults increases. This restart mechanism was also crucial to achieve a zero frame drop ratio in the second case study. In this case study the relation between the frame period and the frame drop ratio was investigated for two applications. The frame drop ratio only becomes zero, when the frame period is large enough to facilitate the restart of a frame.