Architecture-Driven Reliability and Energy Optimization for Complex Embedded Systems

Indika Meedeniya[1], Barbora Buhnova[2], Aldeida Aleti[1], and Lars Grunske[1]

[1] Faculty of ICT, Swinburne University of Technology
Hawthorn, VIC 3122, Australia
{imeedeniya,aaleti,lgrunske}@swin.edu.au
[2] Faculty of Informatics, Masaryk University
60200 Brno, Czech Republic
buhnova@fi.muni.cz

Abstract. The use of redundant computational nodes is a widely used design tactic to improve the reliability of complex embedded systems. However, this redundancy allocation has also an effect on other quality attributes, including energy consumption, as each of the redundant computational nodes requires additional energy. As a result, the two quality objectives are conflicting. The approach presented in this paper applies a multi-objective optimization strategy to find optimal redundancy levels for different architectural elements. It is implemented in the ArcheOpterix tool and validated based on a realistic case study from the automotive domain.

1 Introduction

Motivation. Reliability is one of the key quality attributes of complex embedded systems [1]. To increase reliability, replication of computational nodes (so-called *redundancy allocation*) is used, which however introduces additional life-cycle costs for manufacturing and usage of the system. One more drawback of introducing redundancy is that the system requires more energy to support the additional computational nodes. In most embedded systems, reducing energy consumption is an important design objective, because these systems must support their operation from a limited battery that is hard to recharge (e.g., in deep-sea or outer-space missions) or at least uncomfortable to be recharged very often (e.g., in mobile phones). This is further stressed in systems requiring the minimal size of the battery (e.g., in nano-devices).

State of the art. Research in both reliability and energy consumption for embedded systems is already well established. These two quality attributes are however rarely used in trade-off studies. Energy consumption is typically put in connection with performance [2, 3]. Reliability (when resolved using redundancy allocation) is typically put in connection with production costs [4–6]. The approaches balancing both reliability and energy consumption do not deal with architecture-level optimization, and are often strongly driven by energy consumption rather

then reliability. Such approaches typically examine low-level decision such as voltage reduction [7, 8] or channel coding techniques [9] to improve energy consumption without threatening reliability. Such techniques can be however hardly employed to maximize reliability and minimize energy consumption at the same time, since they have only a minor impact on reliability.

Aim of the paper. In this paper we aim to apply a trade-off analysis between reliability and energy consumption at an architectural level, and employ the technique of *redundancy allocation*, which has a significant effect on both the discussed quality attributes – reliability and energy consumption. To achieve this aim, we identify the main reliability- and energy-relevant attributes of embedded systems with respect to the redundancy allocation problem. We formalize the system model in terms of an annotated Markov Reward Model, formulate the optimization problem, and design an algorithm to resolve it. The whole approach is implemented within the ArcheOpterix framework [10], and illustrated on a realistic case study from the automotive domain.

Contribution of the paper. There are three main contributions of the paper: (i) architecture-level technique to optimize reliability and energy consumption, (ii) novel formalization of the problem and its solution, based on the expected number of visits of individual subsystems and links in between, (iii) employment of a meta-heuristic optimization algorithm, which reduces the likelihood to get stuck in local optima as the greedy algorithms used by related approaches.

Outline of the paper. The paper is structured as follows. After discussion of related work in Section 2, we summarize the essential definitions in Section 3 and present system model and its formalization in Section 4. Based on the model, Section 5 describes our technique of quantifying the quality of a single architectural alternative, from both a reliability and an energy-consumption point of view, and Section 6 designs an optimization algorithm to find the set of near-optimal candidates. Finally, Section 7 discusses our tool support, Section 8 illustrates the approach on a case study, and Section 9 concludes the paper.

2 Related Work

Estimation and optimization of the energy consumption of embedded systems has been the focus of many research groups. The application of energy optimization is evident in design, implementation and runtime [11]. Energy-aware compilation [12] and software design [13–15] has been addressed to achieve energy advantage independent from the hardware-level optimization. Apart from the limited optimizations in different parts of systems, a system-level energy consumption optimization has been proposed by Benini et al. [11]. Energy-efficient runtime adaptation of embedded systems has also been a primary approach, which can be broadly categorized as predictive, adaptive and stochastic control [2]. A key commonality of the approaches is that they use greedy heuristics

for the optimization and focus on the balance of energy consumption and performance.

A number of formal models have been adopted in the context of embedded-systems energy consumption estimation. *Continuous Time Markov Chains (CTMC)* have been widely used including the work of Qiu et al. [16]. Vijayakrishnan et al. [17] proposed to use the more powerful model of *Generalized Stochastic Petri Nets (GSPN)*. The use of *Markov Reward Models (MRM)* has gained visibility [18, 19] due to their power of modeling and expressiveness in the domain of energy consumption. Cloth et al. [20] presented the efficient use of MRMs in energy-consumption modeling for embedded systems.

On the reliability side, there is a considerable amount of approaches that address the *Redundancy Allocation Problem (RAP)* [4] at the system architecture design level [6]. Coit et al. [4] introduced an approach solving RAP defined as the use of functionally similar (but not necessarily identical) components in a way that if one component fails, the redundant part performs required functionality without a system failure. They have visualized the problem as the minimization of cost incurred for the redundancy allocation while satisfying a user defined system reliability level. In [4], *Genetic Algorithms (GA)* have been proposed for the optimization of component redundancy allocation, and *Neural networks (NN)* techniques have also been integrated in [21]. Kulturel-Konak et al. [5] has presented *Tabu Search* as the design space exploration strategy. The RAP has been adapted to *Ant Colony Optimization (ACO)* by Liang et al. [22]. Significant similarity of all the approaches is the view on RAP as cost minimization problem while satisfying the predefined reliability constraints. Grunske [23] addressed RAP by integrating multi-objective optimization of the reliability and weight.

Finally, the trade-off with respect to energy consumption and reliability has been the focus of a few research contributions. Negative implications on the reliability has been investigated due to energy optimization methods such as voltage reduction [7, 8] and channel coding techniques in the communication [9], which are however not connected to RAP. The work of Zhang et al. [24] on finding trade-offs of energy, reliability, and performance in redundant cache line allocation can be viewed as conceptually close to RAP context. Similarly, Perillo et al. [25] have presented an approach of finding the optimal energy management with redundant sensors. However, both these contributions observe only the static (hardware) architecture of the system, without taking the system execution and its control flow (software layer) into account. This allows them to disregard from the execution transfer among system components (which is crucial in software architectures), and to employ simple additive models.

In contrast to the above mentioned approaches, this paper describes a novel *architecture-level* approach of finding the *optimal redundancy levels* of system components (integrating both *software* and *hardware*) with respect to system *reliability* and *energy consumption*.

3 Preliminaries

This section outlines the definitions and preliminary formalizations used in the rest of the paper.

Definition 1 *A* Discrete Time Markov Chain (DTMC) *is a tuple* (S, P) *where* S *is a finite set of states, and* $P : S \times S \rightarrow [0, 1]$ *is a transition probability matrix.*

A DTMC is called absorbing when at least one of its states has no outgoing transition [18]. Such states are called absorbing states.

Definition 2 *A labeled discrete time* Markov Reward Model (MRM) *is a triple* $\mathcal{M} = ((S, P), \rho, \tau)$ *where* (S, P) *is an underlying DTMC,* $\rho : S \rightarrow \mathbb{R}_{\geq 0}$ *is a state reward structure and* $\tau : S \times S \rightarrow \mathbb{R}_{\geq 0}$ *is an impulse reward structure satisfying* $\forall s \in S : \tau(s, s) = 0$.

A path of an absorbing DTMC is a finite sequence $\sigma = s_0 s_1 s_2 ... s_n$ of states, where s_n is an absorbing state. Let $X_\sigma(s)$ denote the number of visits of state s in path σ. Similarly, let $XT_\sigma(s, s')$ represent the number of occurrences of transition (s, s') in σ. Then we can calculate the *accumulated reward of* σ as:

$$R_\sigma = \sum_{s \in S} (X_\sigma(s) \cdot \rho(s)) + \sum_{(s,s') \in (S \times S)} (XT_\sigma(s, s') \cdot \tau(s, s')) \tag{1}$$

Having the expected number of visits of each state and transition, the expected value of the *accumulated reward in all paths* can be computed as:

$$E[R] = \sum_{s \in S} (E[X(s)] \cdot \rho(s)) + \sum_{(s,s') \in (S \times S)} (E[XT(s, s')] \cdot \tau(s, s')) \tag{2}$$

In this paper, we use the method introduced by Kubat [26] to compute the expected number of visits of a state/transition and the above relationship in estimating the energy consumption, as described in Section 5.1.

4 System Model

In our approach, we target event-triggered embedded systems that are structured into interacting components (system elements), called special purpose microprocessors (MPs). The MPs are self-contained micro-computers along with programmed software, dedicated to fulfil a specific functionality. They have only one entry and exit point, and behave the same for each execution (visit of the MP by system control flow). For example, in an autonomous weather data gathering robot, these MPs are responsible for activities such as reading inputs from sensors and calculating the relative humidity of the environment. As the MPs need to communicate with each other during the operation, they are connected via communication channels forming a distributed architecture.

Inter-component communication is modeled as an execution transfer from one component to another. In the redundancy allocation domain, systems are modelled as *Series-Parallel (S-P)* systems [4, 5, 22, 23], with logical constructs for both serial and parallel execution. In the embedded systems domain, the models can be viewed as overlapped sets of S-P models (for individual system-level services[3]), because the execution can start in different components (triggering the services). The execution finishes in the components with no continuation of the execution transfer. The existence of such components is implied by the nature of services, which represent finite scenarios of system execution.

For the redundancy allocation, we use the *hot spare* design topology with *N-Modular Redundancy (NMR)* extension [27]. In hot sparing, each component in the system has a number of replicas (possibly zero), all of which are active at the same time, mimicking the execution of the original component. With the NMR extension the system employs a decision mechanism in the form of majority voting, applied on entry to a component (if multiple replicas deliver their results to the entry gate). See Figure 1 illustrating the concept. By merging the hot spare and NMR, the system can be configured to tolerate fail-stop, crash, commission and value failures. In this configuration, each component with its parallel replicas is considered as a single unit called *subsystem*.

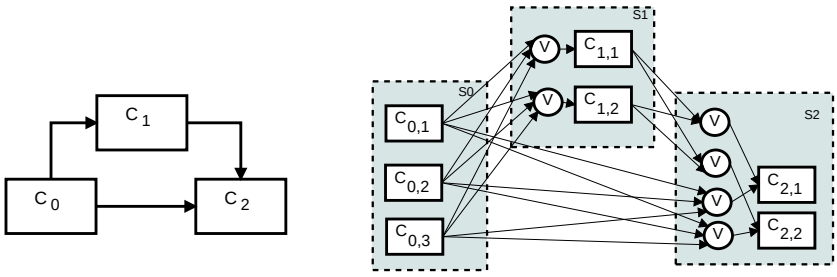

Fig. 1. Component-interaction model, without and with redundancies

4.1 Formalization of the Model

Let $C = \{c_1, c_2, ..., c_n\}$, $n \in \mathbb{N}$, denote the set of all components (before replication), and $\mathcal{I} = \{1, 2, ..., n\}$ the index set for components in C. The assignment of the redundancy level for all components is denoted a, and the set of all possible a is denoted $A = \{a \mid a : C \to N\}$, where $N = \{n \mid 0 \leq n \leq max, n \in \mathbb{N}_0\}$ delimits the redundancy level of a component[4]. Note that, since C and N are finite, A is also finite. A component c_i together with its redundancies form a subsystem S_i, which can be uniquely determined by c_i (what we do along the formalization).

[3] In embedded systems, a *service* is understood as a system-level functionality employing possibly many components.

[4] The original component is not counted as redundant, hence at least the one is always present in a subsystem.

Underlying DTMC model. The interaction among components without replication (Figure 1 on the left) can be formalized in terms of an absorbing DTMC, where nodes represent system components $c_i \in C$, and transitions the transfer of execution from one component to another (together with the probability of the transfer). Equivalently, the system with replication (Figure 1 on the right) can be formalized as an absorbing DTMC where nodes represent the whole subsystems S_i, and the transitions represent the transfer of execution in between with all the replicated communication links joined into a single transition (see Figure 2, ignoring the annotation for now). Note that since the replicated links are joined in the DTMC, the transfer probabilities remain unchanged with respect to redundancy allocation. In both cases, the DTMC represents a single execution of the system (its reaction to an external event), with possibly many execution scenarios (initiated in different nodes of the DTMC, based on the trigerring event). In summary, the underlying DTMC is determined by the following parameters:

- Execution initiation probability, $q_0 : C \to [0, 1]$, is the probability of initializing the execution in the component (or subsystem); $\sum_{c \in C} q_0(c) = 1$.
- Transfer probability, $p : C \times C \to [0, 1]$, is the probability that the execution continues to component (or subsystem) c_j after component c_i.

Energy and reliability annotation. The energy and reliability-relevant information is added to the DTMC model via model annotation. In case of energy consumption, the annotation is encoded in terms of rewards, and the DTMC extended to a discrete time *Markov Reward Model (MRM)* [28], as discussed below. In case of reliability, the annotation is directly connected to the parameters derived from the DTMC, and used for reliability evaluation, as discussed in Section 5.3. In both cases, the annotation is based on the following parameters of system architecture and execution flow:

- Failure rate $\lambda_c : C \to \mathbb{R}_{\geq 0}$, is the failure intensity of the exponential distribution of failure behavior of a component [29]. Component failures in the model are assumed independent and given per time unit.
- Failure rate $\lambda_l : C \times C \to \mathbb{R}_{\geq 0}$, is the failure intensity of the exponential distribution of failure behavior of a communication link between two components, assumed independent for different links and given per time unit.
- Processing time inside a component per visit, $t_c : C \to \mathbb{R}_{\geq 0}$, measured in a model with no redundancy, given in time units (ms).
- Transfer time for a link per visit, $t_l : C \times C \to \mathbb{R}_{\geq 0}$, measured in a model with no redundancy, given in time units (ms).
- Energy consumption of component processing per visit, $e_c : C \to \mathbb{R}_{\geq 0}$, is the estimated energy dissipation by the component during the execution per single visit of the component, given in Joules (J).
- Energy consumption of an idle component, $e_i : C \to \mathbb{R}_{\geq 0}$, is the estimated energy dissipation by the component when being in the idle state, given in Joules (J) per time unit.

– Energy consumption of a link transfer per visit, $e_l : C \times C \rightarrow \mathbb{R}_{\geq 0}$, is the estimated energy dissipation in communication between two components per single interaction. given in Joules (J).
– Trigger rate, $r \in \mathbb{R}$, is the expected number of system executions (occurrence of events trigerring system services) per time unit.

Energy annotated MRM. An example of a MRM (for the system in Figure 1) is in Figure 2. In the example, the nodes are annotated with state rewards $\rho(c_i)$, and transitions annotated with $p(c_i, c_j)/\tau(c_i, c_j)$ where p denotes the transition probabilities and τ the impulse rewards. Based on the above, the energy annotated MRM derives state rewards from the energy consumed in component processing e_c, and impulse rewards from the energy consumed in communication e_l. In both cases, the rewards are affected by the redundancy level of the relevant subsystem.

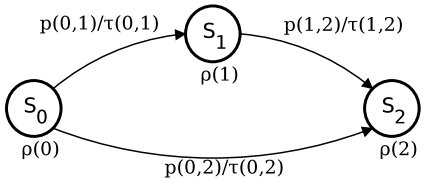

Fig. 2: Markov Reward Model

As the total number of identical components in subsystem S_i is given by $(a(c_i) + 1)$, the *energy consumed in component processing* for the subsystem S_i (node of the MRM) per visit is given by $e_c(c_i) \cdot (a(c_i) + 1)$. The *energy consumed in communication* from a sender subsystem S_i to a recipient S_j (transition in the MRM) is proportional to the number of senders (replicas in S_i), and hence given as $e_l(c_i, c_j) \cdot (a(c_i) + 1)$. In summary, if $c_i, c_j \in C$ are system components (subsystems), and $a \in A$ is a redundancy allocation, then:

– State reward of c_i is defined as $e_c(c_i) \cdot (a(c_i) + 1)$
– Impulse reward of (c_i, c_j) is defined as $e_l(c_i, c_j) \cdot (a(c_i) + 1)$

5 Evaluation of an Architectural Alternative

Each architectural alternative is determined by a single redundancy allocation $a \in A$ (defined in Section 4.1). To quantify the quality of a, we define a quality function $Q : A \rightarrow \mathbb{R}^2$, where $Q(a) = (E^a, R^a)$ for E^a quantifying the energy consumption of a (defined in Section 5.2), and R^a the reliability (probability of failure-free operation) of a (defined in Section 5.3). Both E^a and R^a are quantified per a single time unit, which is possible thanks to the the trigger rate r, and allows us to reflect also the energy consumed in the idle state. The computation of both E^a and R^a employs the information about the expected number of visits of system components and communication links during the execution (see Section 3 for explanation), which we compute first, in Section 5.1.

5.1 Expected number of visits

Expected number of visits of a component, $v_c : C \rightarrow \mathbb{R}_{\geq 0}$, quantifies the expected number of visits of a component (or subsystem) during system execution. Note

that $v_c(c)$ corresponds to the expected number of visits of state c_i in the underlying DTMC (as defined in Section 3), i.e. $E[X(c_i)]$. This can be computed by solving the following set of simultaneous equations [26]:

$$v_c(c_i) = q_0(c_i) + \sum_{j \in \mathcal{I}} (v_c(c_j) \cdot p(c_j, c_i)) \tag{3}$$

The formula (3) can be expanded to:

$$\begin{array}{l} v_c(c_0) = q_0(c_0) + v_c(c_0) \cdot p(c_0, c_0) + v_c(c_1) \cdot p(c_1, c_0) + v_c(c_2) \cdot p(c_2, c_0) + ... + v_c(c_n) \cdot p(c_n, c_0) \\ v_c(c_1) = q_0(c_1) + v_c(c_0) \cdot p(c_0, c_1) + v_c(c_1) \cdot p(c_1, c_1) + v_c(c_2) \cdot p(c_2, c_1) + ... + v_c(c_n) \cdot p(c_n, c_1) \\ v_c(c_2) = q_0(c_2) + v_c(c_0) \cdot p(c_0, c_2) + v_c(c_1) \cdot p(c_1, c_2) + v_c(c_2) \cdot p(c_2, c_2) + ... + v_c(c_n) \cdot p(c_n, c_2) \\ \vdots \\ v_c(c_n) = q_0(c_n) + v_c(c_0) \cdot p(c_0, c_1) + v_c(c_1) \cdot p(c_1, c_n) + v_c(c_2) \cdot p(c_2, c_n) + ... + v_c(c_n) \cdot p(c_n, c_n) \end{array}$$

In a matrix form, the transfer probabilities $p(c_i, c_j)$ can be written as $P_{n \times n}$, and execution initiation probabilities $q_0(c_i)$ as $Q_{n \times 1}$. The matrix of expected number of visits $V_{n \times 1}$ can be expressed as:

$$V = Q + P^T \cdot V$$

With the usual matrix operations, the above can be transformed into the solution format:

$$\begin{aligned} I \times V - P^T \times V &= Q \\ (I - P^T) \times V &= Q \\ V &= (I - P^T)^{-1} \times Q \end{aligned}$$

Expected number of visits of a communication link, $v_l : C \times C \to \mathbb{R}_{\geq 0}$, quantifies for each link $l_{ij} = (c_i, c_j)$ the expected number of occurrences of the transition (c_i, c_j) in the underlying DTMC (as defined in Section 3), i.e $E[XT(c_i, c_j)]$. To compute this value, we extend the work of Kubat et al. [26] for computing the expected number of visits of system components to communication links. In the extension, we understand communication links as first-class elements of the model, and view each probabilistic transition $c_i \xrightarrow{p(c_i,c_j)} c_j$ in the model as a tuple of transitions $c_i \xrightarrow{p(c_i,c_j)} l_{ij} \xrightarrow{1} c_j$, the first adopting the original probability and the second having probability $=1$. Then we can apply the above, and compute the expected number of visits of a communication link as:

$$\begin{aligned} v_l(l_{ij}) &= 0 + \sum_{x \in \{i\}} (v_c(c_x) \cdot p(c_x, l_{ij})) && (4) \\ &= v_c(c_i) \cdot p(c_i, c_j) && (5) \end{aligned}$$

since the execution is never initiated in l_{ij} and the only predecessor of link l_{ij} is component c_i.

5.2 Energy Consumption

The energy consumption of architectural alternative a is computed with respect to two contributing elements: (i) the energy consumed in system execution, and (ii) energy consumed in the idle state.

The *energy consumed in system execution* is represented by the *accumulated reward* defined in Section 3, whose computation employs the state and impulse rewards of the *energy annotated MRM* (defined in Section 4.1) and expected number of visits of both components/subsystems and communication links (defined in Section 5.1). In summary, the energy consumed in system execution is given as:

$$E^a_{exec} = \sum_{c_i \in C} e_c(c_i) \cdot (a(c_i)+1) \cdot v_c(c_i) + \sum_{c_i \in C} \sum_{c_j \in C} e_l(c_i, c_j) \cdot (a(c_i)+1) \cdot v_l(l_{ij}) \quad (6)$$

Since the MPs together with their redundancies consume certain amount of energy during their idle state as well, the total energy consumption takes into account also the *energy consumed in the idle state*, expressed for a single subsystem S_j and one time unit as $e_i(c_j) \cdot (a(c_j)+1)$.

Consequently, the total energy consumption of the system for a given redundancy allocation a and a single time unit can be expressed as:

$$E^a = E^a_{exec} \cdot r + \sum_{c_j \in C} e_i(c_j) \cdot (a(c_j)+1) \quad (7)$$

5.3 Reliability

Having the failure rate λ_c and processing time t_c defined in Section 4.1, the *reliability of a single component* c_i per visit can be computed as [29]:

$$R_c(c_i) = e^{-\lambda_c \cdot t_c(c_i)} \quad (8)$$

When the redundancy levels are employed, the *reliability of a subsystem* S_i (with identical replicas connected in parallel) for the architectural alternative a can be computed as:

$$R^a_c(c_i) = 1 - (1 - R_c(c_i))^{a(i)+1} \quad (9)$$

Similarly, the *reliability of a communication link* per visit is characterized by λ_l and t_l as:

$$R_l(c_i, c_j) = e^{-\lambda_l \cdot t_l(c_i, c_j)} \quad (10)$$

In consideration of a redundancy allocation a, the presence of multiple senders increases the reliability (thanks to the tolerance against commission and value failures) as follows:

$$R^a_l(c_i, c_j) = 1 - (1 - R_l(c_i, c_j))^{a(i)+1} \quad (11)$$

Having the reliabilities of individual system elements (subsystems and links) per a single visit, the reliability of the system execution can be computed analogically to the *accumulated reward* above, based on the expected number of visits,

with the difference that we employ multiplication instead of summation [1, 26]:

$$R^a_{exec} \approx \prod_{i \in \mathcal{I}} (R^a_c(c_i))^{v_c(c_i)} \cdot \prod_{i,j \in \mathcal{I}} (R^a_l(c_i, c_j))^{v_l(l_{ij})} \tag{12}$$

Finally, the system reliability for a given redundancy allocation a and a single time unit (with respect to trigger rate r) can be expressed as:

$$R^a = (R^a_{exec})^r \tag{13}$$

6 Architecture Optimization with Non-dominated Sorting Genetic Algorithm (NSGA)

The goal of our multi-objective optimization problem is to find the approximate set of solutions $A^* \subseteq A$ that represent a trade-off between the conflicting objectives in $Q : A \to \mathbb{R}^2$, i.e. the reliability of the system and the energy consumption, and satisfy the set of constraints Ω. In our case, Ω consists of only the constrains on the maximal redundancy levels of system MPs. Different algorithms can be used for solving the optimization problem. In our approach, we employ the *Non-dominated Sorting Genetic Algorithm (NSGA)* [30], which has shown to be robust and have a good performance in the settings related to ours.

For the optimization process, *NSGA* uses an initial *population* of randomly generated *chromosomes*, consisting of *alleles*. Each chromosome encodes a single redundancy allocation alternative $a \in A$. Each *allele* in a *chromosome* represents a redundancy level for a component $c_i \in C$.

The three genetic operators of the evolution process are *selection*, *cross-over* and *mutation*. NSGA varies from simple genetic algorithm only in the way the selection operator works. The three operators are adapted to the redundancy allocation problem as follows.

6.1 Selection

Before the *selection* operator is applied the population is ranked on the basis of an individual's non-domination. In a maximization problem a solution a^* is non-dominated if there exists no other a such that $Q(a) \leq Q(a^*)$ for all objectives, and $Q(x) < Q(x^*)$ for at least one objective. In other words, if there exists no other feasible variable a which would be superior in at least one objective while being no worse in all other objectives of a^*, then a^* is said to be non-dominated. The set of all non-dominated solutions is known as the non-dominated set.

First, the non-dominated solutions present in the population are identified and assigned a fitness value. These solutions will constitute the first non-dominated front in the population, which will be ignored temporarily to process the rest of the population in the same way, finding the solutions which belong to the second non-dominated front. These solution are assigned a lower fitness value. This process is continued until the entire population is classified into separate fronts.

A mating pool is then created with solutions selected from the population according to the fitness value that has been assigned to them during the ranking process. The solutions with a greater fitness value have a higher chance of being selected to be part of the mating pool than the ones with a lower fitness value. This helps the quick convergence of the algorithm towards the optimal solutions. The mating pool will then serve for the random selection of the individuals to reproduce using crossover and mutation.

6.2 Crossover

Crossover is the creation of new solutions $a'_i, a'_j \in A$ from two parents $a_i = [u_{i_1}, u_{i_2}, ..., u_{i_n}]$ and $a_j = [u_{j_1}, u_{j_2}, ..., u_{j_n}]$ coming from existing population by recombining the redundancy levels of components, i.e. for a random k: $a'_i = [u_{i_1}, ..., u_{i_{k-1}}, u_{j_k}, ..., u_{j_n}]$ and $a'_j = [u_{j_1}, ..., u_{j_{k-1}}, u_{i_k}, ..., u_{i_n}]$. After every crossover operation, the solutions are checked for constraints satisfaction in Ω, and repairing mechanisms are used to handle constraint violation.

6.3 Mutation

Mutation produces a new solution a'_i from existing a_i by switching the allocation of two components, i.e. for randomly selected k, l: $a'_i = [u_{i_1}, ..., u_{i_l}, ..., u_{i_k}, ..., u_{i_n}]$ while the original is $a_i = [u_{i_1}, ..., u_{i_k}, ..., u_{i_l}, ..., u_{i_n}]$. Each newly created chromosome is first checked for constraint satisfaction (for constraints in Ω) before it is allowed to become part of the population. This prevents us from the construction of infeasible solutions.

7 Tool Support

The presented approach, including the NSGA optimization algorithm, has been implemented within the *ArcheOpterix* framework [10], which has been developed with Java/Eclipse and provides a generic platform for specification, evaluation and optimization of embedded system architectures.

ArcheOpterix has a modular structure, with an entry module responsible for interpreting and extracting a system description from a specification, recognizing standard elements like components, services, processors, buses, etc., specified in AADL or XML. It also provides extensions to other elements and domain-specific parameters. The remaining modules allow for plug-in of different quality function evaluators, constraint validators, and optimization algorithms, which makes it a well fitting tool for multi-criteria quality optimization of embedded systems.

8 Validation on a Case Study

8.1 Automotive control system

An embedded system from the automotive domain is used as a case study for the demonstration of the approach. The case study has been designed based

on already published models [31, 32] and contains the *Anti-lock Brake System (ABS)* and *Adaptive Cruise Control (ACC)* functionality. System parameters required for the model are chosen to closely resemble the reality, including the component failure rates [33], and estimated execution time per visit [34].

Anti-lock Brake System (ABS): The ABS is currently used in most of modern cars to minimize hazards associated with skidding and loss of control due to locked wheels during braking. Proper rotation during brake operations allows better maneuverability and enhances the performance of braking.

Adaptive Cruise Control (ACC): Apart from usual automatic cruise control functionality, the main aim of the ACC is to avoid crashes by reducing speed once a slower vehicle in front is detected.

The main components used by the composite system and their interaction diagram are presented in Figure 3. The *ABS Main Unit* is the major decision-making unit regarding the braking levels for individual wheels, while the *Load Compensator* unit assists with computing adjustment factors from the wheel load sensor inputs. Components 4 and 5 represent the components that communicate with wheel sensors while components 7 and 8 represent the electronic control units that control the brake actuators. *Brake Paddle* is the component that reads from the paddle sensor and sends the data to the *Emergency Stop Detection* unit. Execution initialization is possible at the components that communicate with the sensors and user inputs. In this case study the *Wheel Sensors*, *Speed Limitter*, *Object Recognition*, *Mode Switch* and *Brake Paddle* components contribute to the triggering of the service. The captured data from the sensors, are processed by different components in the system and triggers are generated for the actuators like *Brake Actuators* and *Human Machine Interface*.

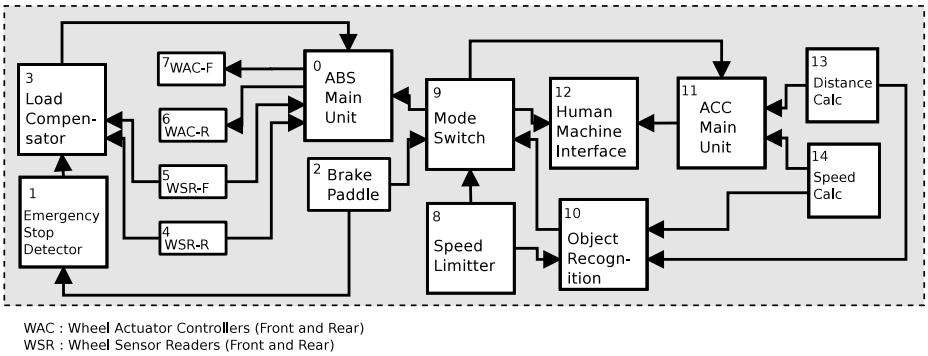

Fig. 3. Automotive composite system

Parameters of the elements of the considered system, and probabilities of transferring execution from one component to another are illustrated in Table 1. The trigger rate r of the composite system is assumed to be 1 trigger per second.

Comp ID	q_0	λ_c	e_c (mJ)	t_c (ms)	e_i (mW)
0	0	$4 \cdot 10^{-6}$	20	33	2
1	0	$6 \cdot 10^{-6}$	10	30	1
2	0.01	$5 \cdot 10^{-6}$	20	10	2
3	0	$8 \cdot 10^{-6}$	25	33	2.5
4	0.17	$8 \cdot 10^{-6}$	30	10	3
5	0.17	$8 \cdot 10^{-6}$	30	10	3
6	0.17	$8 \cdot 10^{-6}$	40	10	4
7	0.17	$8 \cdot 10^{-6}$	40	10	4
8	0.01	$5 \cdot 10^{-6}$	10	20	1
9	0	$5 \cdot 10^{-6}$	10	20	1
10	0	$5 \cdot 10^{-6}$	20	33	2
11	0	$4 \cdot 10^{-6}$	30	28	3
12	0	$7 \cdot 10^{-6}$	40	28	4
13	0.15	$3 \cdot 10^{-6}$	50	33	5
14	0.15	$3 \cdot 10^{-6}$	40	33	4

Trans $c_i \rightarrow c_j$	Prob. $p(c_i, c_j)$	λ_l	e_l (mJ)	t_l (ms)
$0 \rightarrow 7$	0.5	$4 \cdot 10^{-5}$	40	40
$0 \rightarrow 6$	0.5	$5 \cdot 10^{-5}$	40	40
$1 \rightarrow 3$	1	$6 \cdot 10^{-5}$	60	10
$2 \rightarrow 1$	0.75	$5 \cdot 10^{-5}$	60	30
$3 \rightarrow 0$	1	$4 \cdot 10^{-5}$	35	30
$4 \rightarrow 0$	0.7	$4 \cdot 10^{-5}$	60	30
$4 \rightarrow 3$	0.3	$5 \cdot 10^{-5}$	60	30
$5 \rightarrow 0$	0.7	$3 \cdot 10^{-5}$	40	30
$5 \rightarrow 3$	0.3	$5 \cdot 10^{-5}$	50	40
$2 \rightarrow 9$	0.25	$6 \cdot 10^{-5}$	30	40
$8 \rightarrow 9$	0.6	$8 \cdot 10^{-5}$	50	30
$8 \rightarrow 10$	0.4	$12 \cdot 10^{-5}$	40	30
$9 \rightarrow 0$	0.2	$4 \cdot 10^{-5}$	20	10
$9 \rightarrow 11$	0.4	$5 \cdot 10^{-5}$	20	10
$9 \rightarrow 12$	0.6	$5 \cdot 10^{-5}$	30	10
$10 \rightarrow 9$	1	$6 \cdot 10^{-5}$	50	20
$11 \rightarrow 12$	1	$8 \cdot 10^{-5}$	50	20
$13 \rightarrow 10$	0.5	$10 \cdot 10^{-5}$	20	40
$13 \rightarrow 11$	0.5	$12 \cdot 10^{-5}$	20	40
$14 \rightarrow 10$	0.5	$4 \cdot 10^{-5}$	30	40
$14 \rightarrow 11$	0.5	$5 \cdot 10^{-5}$	45	40

Table 1. Parameters of components and communication links

8.2 Results

Even though the presented case study is a comparatively small segment of an actual automotive system, the possible number of architectural alternatives is $4^{15} \approx 1.07 \cdot 10^9$ (assuming maximum redundancy level of 3), which is too large to be traversed with an exact algorithm. Therefore we employed the *NSGA* (see Section 6) as a meta-heuristic to obtain a near-optimal solutions in practically affordable time frame.

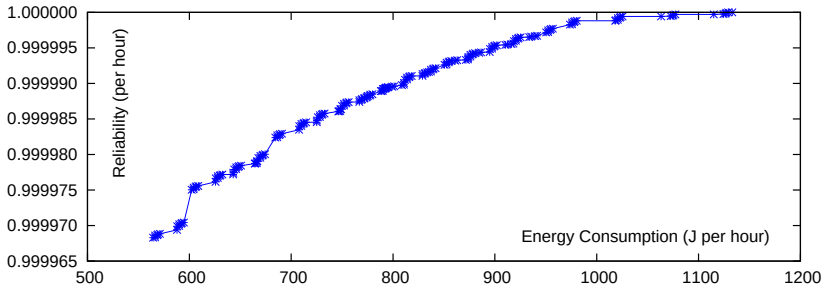

Fig. 4. Distribution of non-dominated solutions

The execution of the algorithm was set to 25 000 function evaluations, and performed under a settings on a dual-core 2.26 GHz processor computer. The algorithm took 7 seconds for the 25 000 function evaluations and generated 193 non-dominated solution architectures. The distribution of the near-optimal solutions is graphically represented in Figure 4. The prevalence of the solutions in the objective domain, together with their non-dominated trade-offs are depicted

in the graph. The designers are provided this set to choose the desired solution based on their utility (trade-off preference of reliability/energy consumption).

Solution	Redundancy Allocation	Reliability (h^{-1})	Energy Consumption(J/h)
A	[1,0,0,1,1,1,0,0,0,1,1,1,0,1,1]	0.99999828	973.79
B	[1,0,0,1,1,1,0,1,1,1,1,1,1,1,1]	0.99999948	1072.66

Table 2. Example of two non-dominated solutions

Table 2 illustrates two closely related non-dominated solutions generated by the optimization process. The arrays in the second column represent the redundancy levels for the components (subsystems) ordered by their ID. Note that the reliabilities of the two solutions are very similar while the energy consumption of B is approximately 100 J/h lower than of A. It should be highlighted that the non-obvious change from solution A to solution B has significantly reduced the energy consumption for a very small trade-off of reliability, which would definitely be an interesting information for the system designer.

9 Conclusions and Future Work

In this paper, we have formulated the models for estimating reliability and energy consumption at an architectural level, and combined the two quality attributes in optimizing the redundancy levels of system components. The energy consumption model, formulated in terms of a discrete time Markov Reward Model, builds on the expected number of visits in obtaining a point estimate for the accumulated state and impulse reward. The accumulated reward together with the energy consumed in components' idle states then characterizes system energy consumption. The reliability evaluation model extends the Kubat's model [26], applying the concept of subsystems and expected number of visits of system elements. The model is extended to consider also the impact of communication elements of the system. As a result, both estimation techniques enable quantification of the impact of critical design decision on reliability and energy consumption. We have employed this for automatically identifying architecture specification with optimal redundancy level to satisfy both quality attributes. For this identification, the redundancy allocation problem is solved with a multi-objective optimization strategy. We have implemented the architecture evaluation models and used NSGA to find near-optimal architecture solutions. An automotive case study of a composite system of *Anti-lock Brake System (ABS)* and *Adaptive Cruise Control (ACC)* has been used for the validation of our approach.

In future work, we would like to extend the set of investigated design decisions. In addition to the allocation of redundancy levels, also deployment decisions for software components and selection of appropriate architectural elements is interesting. Furthermore, we aim to investigate different optimization strategies such as Ant Colony Strategies, Tabu Search, etc., to compare which of them works better for which problem formulation.

Acknowledgment

This original research was proudly supported by the Commonwealth of Australia, through the Cooperative Research Center for Advanced Automotive Technology (projects C4-501: Safe and Reliable Integration and Deployment Architectures for Automotive Software Systems and C4-509: Dependability optimization on architectural level of system design).

References

1. Goševa-Popstojanova, K., Trivedi, K.S.: Architecture-based approach to reliability assessment of software systems. Performance Evaluation **45**(2-3) (2001) 179–204
2. Benini, L., Bogliolo, A., Micheli, G.D.: A survey of design techniques for system-level dynamic power management. IEEE Trans. VLSI Syst **8**(3) (2000) 299–316
3. Aydin, H., Melhem, R., Mossé, D., Mejía-Alvarez, P.: Dynamic and aggressive scheduling techniques for power-aware real-time systems. In: Real-Time Systems Symposium, IEEE (2001) 95–105
4. Coit, D.W., Smith, A.E.: Reliability optimization of series-parallel systems using a genetic algorithm. IEEE Transactions on Reliability **45**(2) (1996) 225–266
5. Kulturel-Konak, S., Coit, A.E.S.D.W.: Efficiently solving the redundancy allocation problem using tabu search. IIE Transactions **35**(6) (2003) 515–526
6. Grunske, L., Lindsay, P.A., Bondarev, E., Papadopoulos, Y., Parker, D.: An outline of an architecture-based method for optimizing dependability attributes of software-intensive systems. In: Workshops on Software Architectures for Dependable Systems. Volume 4615 of LNCS., Springer (2006) 188–209
7. Zhu, D., Melhem, R.G., Mossé, D.: The effects of energy management on reliability in real-time embedded systems. In: International Conference on Computer-Aided Design, IEEE Computer Society / ACM (2004) 35–40
8. Pop, P., Poulsen, K.H., Izosimov, V., Eles, P.: Scheduling and voltage scaling for energy/reliability trade-offs in fault-tolerant time-triggered embedded systems. In: International Conference on Hardware/Software Codesign and System Synthesis, ACM (2007) 233–238
9. Bertozzi, D., Benini, L., Micheli, G.D.: Energy-reliability trade-off for nocs. In: Networks on Chip, Springer US (2003) 107–129
10. Aleti, A., Björnander, S., Grunske, L., Meedeniya, I.: ArcheOpterix: An extendable tool for architecture optimization of AADL models. In: Model-based Methodologies for Pervasive and Embedded Software, ACM and IEEE Digital Libraries (2009) 61–71
11. Benini, L., Micheli, G.D.: Powering networks on chips. In: International Symposium on Systems Synthesis. (2001) 33–38
12. Simunic, T., Benini, L., Micheli, G.D.: Energy-efficient design of battery-powered embedded systems. IEEE Trans. VLSI Syst **9**(1) (2001) 15–28
13. Hong, I., Kirovski, D., Qu, G., Potkonjak, M., Srivastava, M.B.: Power optimization of variable-voltage core-based systems. IEEE Trans. on CAD of Integrated Circuits and Systems **18**(12) (1999) 1702–1714
14. Lu, Y.H., Simunic, T., Micheli, G.D.: Software controlled power management. In: International Workshop on Hardware/Software Codesign. (1999) 157–161
15. Seo, C., Edwards, G., Malek, S., Medvidovic, N.: A framework for estimating the impact of a distributed software system's architectural style on its energy consumption. In: Working IEEE/IFIP Conference on Software Architecture, IEEE Computer Society (2008) 277–280

16. Qiu, Q., Pedram, M.: Dynamic power management based on continuous-time markov decision processes. In: Design Automation Conference. (1999) 555–561
17. Vijaykrishnan, N., Kandemir, M.T., Irwin, M.J., Kim, H.S., Ye, W.: Energy-driven integrated hardware-software optimizations using simplepower. In: International Symposium on Computer Architecture. (2000) 95–106
18. Trivedi, K.S.: Probability and Statistics with Reliability, Queuing, and Computer Science Applications. Prentice-Hall, Englewood Cliffs, New Jersey (1982)
19. Cloth, L., Katoen, J.P., Khattri, M., Pulungan, R.: Model checking markov reward models with impulse rewards. In: Dependable Systems and Networks. (2005) 722–731
20. Cloth, L., Jongerden, M.R., Haverkort, B.R.: Computing battery lifetime distributions. In: Dependable Systems and Networks, IEEE Computer Society (2007) 780–789
21. Coit, D.W., Smith, A.E.: Reliability optimization of series-parallel systems using a genetic algorithm. IEEE Transactions on Reliability **45**(2) (1996) 254–260
22. Liang, Y.C., Smith, A.E.: An ant system approach to redundancy allocation. In: Congress on Evolutionary Computation, IEEE (1999) 1478–1484
23. Grunske, L.: Identifying "good" architectural design alternatives with multi-objective optimization strategies. In: International Conference on Software Engineering, ICSE, ACM (2006) 849–852
24. Zhang, W., Kandemir, M., Sivasubramaniam, A., Irwin, M.J.: Performance, energy, and reliability tradeoffs in replicating hot cache lines. In: Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES-03), ACM Press (2003) 309–317
25. Perillo, M.A., Heinzelman, W.B.: Optimal sensor management under energy and reliability constraints. In: IEEE Wireless Communications. (2003) 1621–1626
26. Kubat, P.: Assessing reliability of modular software. Operations Research Letters **8**(1) (1989) 35–41
27. Nelson, V.P., Carroll, B.: Fault-Tolerant Computing. IEEE Computer Society Press (1987)
28. Katoen, J.P., Khattri, M., S.Zapreev, I.: A markov reward model checker. In: QEST:International Conference on the Quantitative Evaluation of Systems, IEEE Computer Society (2005) 243–244
29. Shatz, S.M., Wang, J.P., Goto, M.: Task allocation for maximizing reliability of distributed computer systems. IEEE Transactions on Computers **41**(9) (1992) 1156–1168
30. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation **2**(3) (1995) 221–248
31. Fredriksson, J., Nolte, T., Nolin, M., Schmidt, H.: Contract-based reusable worst-case execution time estimate. In: The International Conference on Embedded and Real-Time Computing Systems and Applications. (2007) 39–46
32. Grunske, L.: Towards an Integration of Standard Component-Based Safety Evaluation Techniques with SaveCCM. In: Quality of Software Architectures. Volume 4214 of LNCS., Springer (2006) 199–213
33. Assayad, I., Girault, A., Kalla, H.: A bi-criteria scheduling heuristic for distributed embedded systems under reliability and real-time constraints. In: Dependable Systems and Networks, IEEE Computer Society (2004) 347–356
34. Florentz, B., Huhn, M.: Embedded systems architecture: Evaluation and analysis. In: Quality of Software Architectures, Second International Conference on Quality of Software Architectures. Volume 4214., Springer (2006) 145–162