Automated Optimization of Weighted Non-functional Objectives in Self-adaptive Systems

Kate M. Bowers[1(✉)], Erik M. Fredericks[1(✉)], and Betty H. C. Cheng[2(✉)]

[1] Oakland University, Rochester, MI 48309, USA
{kmlabell,fredericks}@oakland.edu
[2] Michigan State University, East Lansing, MI 48824, USA
chengb@cse.msu.edu

Abstract. A self-adaptive system (SAS) can reconfigure at run time in response to adverse combinations of system and environmental conditions in order to continuously satisfy its requirements. Moreover, SASs are subject to cross-cutting non-functional requirements (NFRs), such as performance, security, and usability, that collectively characterize *how* functional requirements (FRs) are to be satisfied. In many cases, the trigger for adapting an SAS may be due to a violation of one or more NFRs. For a given NFR, different combinations of hierarchically-organized FRs may yield varying degrees of satisfaction (i.e., satisficement). This paper presents `Providentia`, a search-based technique to optimize NFR satisficement when subjected to various sources of uncertainty (e.g., environment, interactions between system elements, etc.). `Providentia` searches for optimal combinations of FRs that, when considered with different subgoal decompositions and/or differential weights, provide optimal satisficement of NFR objectives. Experimental results suggest that using an SAS goal model enhanced with search-based optimization significantly improves system performance when compared with manually- and randomly-generated weights and subgoals.

Keywords: Search-based software engineering
Non-functional requirements · Self-adaptive systems
Evolutionary computation

1 Introduction

A self-adaptive system (SAS) provides adaptation strategies for reconfiguration at run time to mitigate unexpected issues that arise as a result of uncertainty (e.g., adverse environmental conditions or unexpected issues in the system itself) [15,19]. The SAS generally will use these adaptation strategies to select an optimal configuration that enables continuous requirements satisficement (i.e., degree of satisfaction) [5]. An SAS is governed by functional requirements (FRs) that can be mathematically quantified to monitor satisficement, as well as by non-functional requirements (NFRs) that tend to be qualitative and may not be

T. E. Colanzi and P. McMinn (Eds.): SSBSE 2018, LNCS 11036, pp. 182–197, 2018.
https://doi.org/10.1007/978-3-319-99241-9_9

easily mathematically quantifiable (e.g., resiliency and efficiency) [17,34]. However, this process relies on domain knowledge and may be sub-optimal given changing environmental conditions. This paper presents `Providentia`, a search-based technique to be used at design time to automatically determine an optimal set of FRs, including the level of impact of each, to support each NFR in an SAS.

NFRs can be modeled as behavioral goals (e.g., KAOS [17]) or soft goals (i.e., NFR framework [34] and iStar modeling language [36]). Soft goals describe preferences of system behaviors that tend to be qualitative in nature [23], thereby making the determination of an optimal reconfiguration strategy more challenging for SASs. Similar to `Providentia`, the Analytic Hierarchy Process (AHP) decomposes NFRs into one or more weighted FRs using an automated weighting scheme to prioritize FRs [27]. However, prioritizations in an SAS determined at design time may change drastically as the system experiences various forms of uncertainty due to changing environmental conditions and unexpected system changes, such as unwanted feature interactions.

We introduce `Providentia` to address the challenges with quantifying and analyzing NFRs at run time via a design-time technique that takes into account environmental and system uncertainty. `Providentia` uses a utility function that specifies a mathematical expression of goal satisficement for each FR. Each NFR has a linear-weighted expression that specifies the impact that a given FR has in satisficing the NFR objective [27]. For a given set of SAS FRs with corresponding metrics to assess their satisficement, `Providentia` explores different combinations of weights for a linear-weighted expression of FRs that contribute to the satisficement of their respective NFRs.

To make the overall system robust to adverse conditions, an evolutionary-based search process assesses the system's run-time behavior using an executable specification of the system that is subjected to randomly-generated sources of uncertainty in order to identify optimal goal model configurations for maximizing FR/NFR satisficement. `Providentia` optimizes the FR selection process using a genetic algorithm as a search heuristic, where the search space is the different goal model configurations that capture varying combinations of FRs. The genetic algorithm determines optimal weight assignments that result in the highest satisficement of the NFR when faced with uncertainty. The `Providentia`-optimized goal model is then applied to the SAS at run time. The correlated evaluation of NFR and FR satisficement enables traditionally soft goals to be evaluated with FR metrics during execution, thereby enabling online SAS reconfiguration in response to high-level non-functional and functional objectives. Furthermore, optimizing the weighted contributions of FRs to each NFR according to estimated sources of uncertainty enables the system to perform better at run time under actual sources of uncertainty, as a requirements engineer may not be able to foresee the diverse range and scope of cases when deriving their respective weights.

We illustrate the effectiveness of `Providentia` with an industry-provided application, namely, a remote data mirroring (RDM) network [13,14]. The RDM application must replicate and disseminate messages to each RDM within the

network and can experience uncertainty due to dropped or delayed messages, sensor noise, and unexpected server and network link failures. For run-time assessment purposes, we use a network simulator that conforms to the specifications provided by our industrial collaborator. Experimental results suggest that using `Providentia` to optimize NFRs based on simulated sources of uncertainty significantly improves overall requirement fitness as well as decreases the number of requirement violations of the RDM application when compared to NFRs using human-generated identification of contributing FRs, their weights, as well as those assigned by random search. The remainder of this paper is organized as follows. Section 2 provides relevant background information on the RDM application, goal-oriented requirements engineering (GORE), NFRs, and utility functions for assessing metrics. Section 3 details the implementation of `Providentia` for automatically determining FR selection and weight assignment for NFRs. Section 4 presents our experimental results and Sect. 5 discusses related work. Section 6 summarizes our results and overviews future directions.

2 Background

This section provides relevant background information on the RDM application, GORE, NFRs, and utility functions.

2.1 Remote Data Mirroring

RDM is a data protection technique for ensuring that data loss is minimized and data availability is maximized in the context of data replicates that are disseminated to other servers (i.e., data mirrors) in physically remote locations [13,15]. An RDM network can be modeled as an SAS [26], enabling reconfiguration in terms of network topology and data propagation parameters to enable continuous requirements satisficement. Uncertainty can impact the RDM in terms of unexpected dropped or delayed messages, random network link or data mirror failures, and noise applied to network links and data mirror sensors. These reconfiguration strategies can be fulfilled by downgrading the status of the affected data mirrors from active (i.e., can send and receive messages) to passive (i.e., can only receive messages) or quiescent (i.e., cannot send or receive messages).

2.2 Goal-Oriented Requirements Engineering

GORE is an approach for graphically specifying a system's key objectives and constraints using both *functional* and *non-functional* goals [8]. A goal is a system behavior achieved through the cooperation of its agents, where an agent is a system component that performs actions based on the behavior specified by goals. A requirement is a goal under the responsibility of a single agent. An expectation is a requirement whose agent is a part of the environment. Functional goals specify a service to be provided and non-functional goals impose a quality constraint on those functional services [17].

GORE enables goal decomposition using a directed acyclic graph, where each node represents a goal and each edge represents a goal refinement [17]. GORE has been extended with additional refinement strategies through KAOS [8,17] and iStar [35]. KAOS introduces AND- and OR-refinements for additional satisficement constraints, where an AND-refined goal is satisfied if *each sub-goal* is also satisfied and an OR-refined goal is satisfied if *at least one sub-goal* is satisfied. KAOS functional goals may be further categorized as *invariant* or *non-invariant*, where invariant goals must always be satisfied and non-invariant goals may be temporarily unsatisfied due to transient conditions. Invariant goals are denoted by the keywords "Maintain" or "Avoid" and non-invariant goals are denoted by the keyword "Achieve."

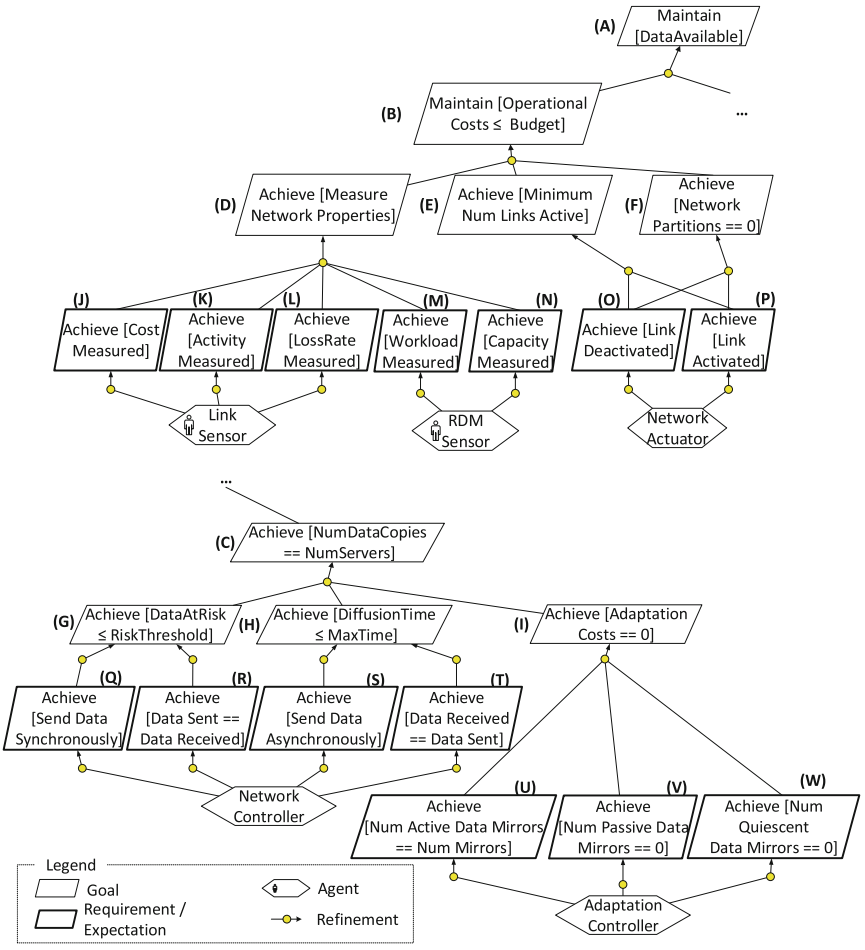


Fig. 1. RDM goal model.

Figure 1 presents the KAOS goal model of the RDM application that describes its hierarchical relationships between goals, requirements/expectations, and agents.[1]

2.3 Non-functional Requirements

NFRs impose a quality constraint on a system [6]. Such goals are often difficult to quantify, given their relative subjectivity. Moreover, cross-cutting concerns may manifest in NFRs, given their broad impacts on the overall system [6]. While rigorous mathematical models have been previously described for calculating requirements satisfaction [10,24], such models generally require a detailed understanding of the real-world environment that is often difficult or impossible to derive for NFRs. A sample NFR for the RDM application in Fig. 1 is `Minimize [Power]`, where many factors (e.g., Goals (A), (E), (I), (V), and (W)) could contribute to either increasing or decreasing power consumption over time, as illustrated in Fig. 2.

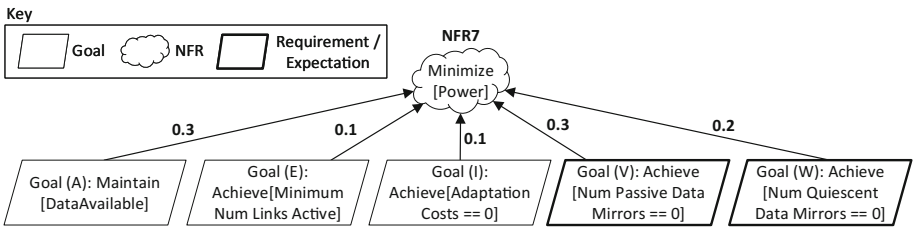


Fig. 2. NFR7: minimize[power].

We use the model in Fig. 2 as an illustrative example to demonstrate the effectiveness of `Providentia`, where all functional goals represent FRs and non-functional goals represent NFRs. Note that although Fig. 2 is presented in a separate diagram, for discussion, the NFR is intended to depict an extension of the input goal model shown in Fig. 1, where the NFRs are evaluated in conjunction with the FRs.

2.4 Utility Functions

A utility function can be used to quantify the degree of satisfaction (i.e., satisficement) of software requirements at run time in autonomic computing systems [9,24,30]. A utility value of 0.0 indicates a violation, 1.0 indicates complete satisfaction, and any value within range of (0.0, 1.0) indicates a degree of satisficement for that requirement [5]. For example, Expression 1 shows the utility value

[1] This work does not use the KAOS formal refinement infrastructure.

calculation for Goal (V) to `Achieve [Num Passive Data Mirrors == 0]`, as introduced in Fig. 1.

$$util(goal_V) = \begin{cases} 1.0 & \text{if Num Passive Data Mirrors} == 0 \\ x & \text{if } 0 < \text{Num Passive Data Mirrors} < 20\% \text{of total nodes} \\ 0.0 & \text{if Num Passive Data Mirrors} \geq 20\% \text{ of total nodes} \end{cases} \tag{1}$$

Goal (V) can be quantified by monitoring the state of each RDM within the network. If there are no RDMs in a passive state, then the utility value is 1.0. Otherwise, the utility value linearly decreases until a threshold (e.g., 20% of the total number of nodes for this paper) is met and then the utility value equals 0.0, indicating a requirement violation.

3 Approach

This section introduces `Providentia`, our technique for automatically optimizing the selection of FRs and their corresponding weights for satisficing NFR objectives. We first describe the inputs and outputs of `Providentia` and then present the approach.

3.1 Providentia: Inputs and Outputs

`Providentia` requires the following inputs: a goal model representing both FRs and NFRs of the SAS, a set of utility functions for run-time requirements monitoring, a set of applicable FRs for each NFR, and an executable specification or prototype of the SAS to be used for run-time simulation, including any defined sources of uncertainty (for this experiment, environmental and system uncertainty are used). The output of `Providentia` is a goal model with optimized FR/NFR relationships. Note that the success of `Providentia` relies on the accuracy of the input data. For example, if the set of applicable FRs for each NFR is inaccurate or if any sources of uncertainty are omitted, the effectiveness of the search-based heuristic may not necessarily be optimal. Note that the time to compute an optimal goal model increases as the size of the input goal model and requirements data increases.

Goal Model. A KAOS goal model is required to specify the FRs and NFRs of the SAS.

Utility Functions. A utility function shall be derived for each FR for run-time monitoring of SAS requirements [9,30]. Each utility function comprises a mathematical function that maps monitoring data to a scalar value within $[0.0, 1.0]$, demonstrating how well the FR is satisfied at run time.

Applicable Set of FRs. A requirements engineer shall provide an initial set of applicable FRs that can have an impact on an NFR. For example, a requirements engineer may specify that Goals (A), (E), (I), (V), and (W) most critically impact

the NFR for reducing power consumption, however that list may be further expanded at design time (e.g., due to uncertainty factors) to include Goals (B), (K), (M), (O), (P), and (U). Extending the list of possible FRs for each NFR allows a larger search space for `Providentia` to find an optimal solution that a requirements engineer may not be able to foresee.

Executable Specification. An executable specification, such as a simulation or prototype, of an SAS must also be provided as input. The specification applies the FR utility functions to measure how well SAS requirements are being satisfied at run time. The executable specification also applies different combinations of system and environmental parameters, including possible sources of uncertainty and their impact on the system (e.g., broken links, failed servers, etc.), to enable the SAS to experience a wide range of configuration states.

Output. The output of `Providentia` is (1) the NFR goal model, with (2) a set of FRs that collectively contribute to the satisficement of each NFR, and (3) an optimized weight value assigned to each FR. A weight value of 0.0 for an FR indicates that the FR does not contribute to the satisfaction of the NFR. For example, for NFR7 (`Minimize [Power]`), `Providentia` determined the following weights to be optimal for each Goal: B: 0.237144, E: 0.241000, K: 0.007185, M: 0.373794, O: 0.049442, U: 0.067218, V:0.024216. Although Goal (A) was included in the initial set of applicable FRs, its weight value was 0.0 to indicate that Goal A did not contribute to satisfying the requirements to minimize power.

3.2 Providentia Technique

This section overviews the `Providentia` technique, comprising a genetic algorithm [12] to search for optimal NFR weighting and requirements combinations. Figure 3 presents a data flow diagram that illustrates the process used by `Providentia`. Each step is next presented in detail.

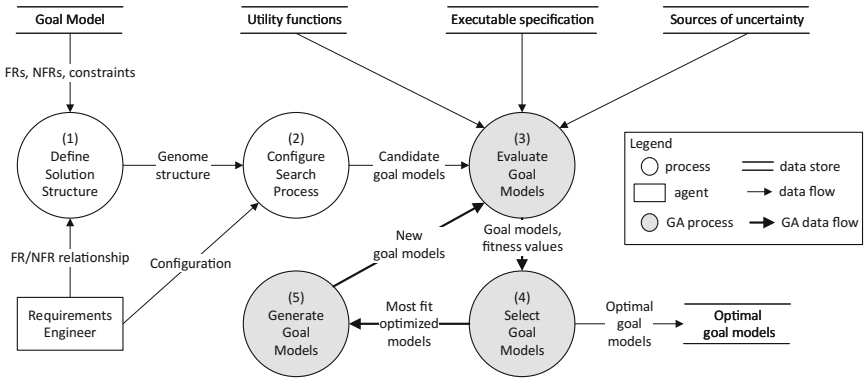


Fig. 3. Data flow diagram of providentia technique.

Fig. 4. Providentia sample genome.

(1) Define Solution Structure. Each candidate solution in `Providentia` is encoded in a fixed-length genome as shown in Fig. 4, where each gene corresponds to a floating-point weight specified for a supporting FR. Each set of weights that correspond to an NFR (i.e., sub-genome, denoted by bolded line) must sum to a value of 1.0. The entire genome comprises all sub-genomes that can be used to define each NFR, e.g., $[[weights_{nfr_1}],[weights_{nfr_2}],...,[weights_{nfr_n}]]$.

(2) Configure Search Process. The search process must be configured by specifying a population size, number of generations, crossover rate, mutation rate, and selection rate. Based on empirical evidence on convergence rates, this paper specifies a population size of 20, 50 generations, a crossover rate of 25%, and a mutation rate of 50%. For selection, we use the tournament selection approach [12] and set the tournament size to 3. While larger values for population size and generations were tested (e.g., populations of 25–50 and generations of 50–100), an optimal convergence was discovered on average at the specified values.

(3) Evaluate NFR Models. The simulation provided as input applies the goal model to randomized combinations of uncertainty in order to obtain a set of FRs with weights adjusted to be as robust as possible. To guide the search process, we maximize average FR/NFR satisficement as shown in Eqs. 2–4 and minimize the number of SAS adaptations to reduce overall network disruption as shown in Eq. 5. We collect these metrics in a linear weighted sum as shown in Eq. 6. We next describe each equation in turn.

The performance of each NFR as an aggregate utility function is defined as:

$$utility\_value_{nfr_n} = \sum_{i=1}^{|frs_{nfr_n}|} utility_{fr_i} * weight_{fr_i} \tag{2}$$

where $|frs_{nfr_n}|$ refers to the number of supporting FRs for nfr_n, $utility_{fr_i}$ refers to the calculated utility value for fr_i, and $weight_{fr_i}$ refers to the defined weight (i.e., relative importance) of fr_i. Based on Eq. 2, each NFR has a utility function that can be monitored to quantify performance at run time. If NFR7 (`Minimize [Power]`) becomes violated or satisficed to an unsatisfactory degree,[2] then the RDM application will self-reconfigure to perform an appropriate mitigation strategy.

[2] For this paper, we select a threshold of 0.4 to signify requirement non-satisfaction based on empirical evidence.

The fitness sub-function shown in Eq. 3 maximizes *FR* satisficement throughout execution, where $utility\_value_{functional}$ represents the calculated utility values for FRs and $timesteps_{sim}$ represents the number of simulation timesteps:

$$FF_{fr} = \frac{\sum utility\_value_{functional}}{|utility\_value_{functional}| * timesteps_{sim}} \tag{3}$$

The fitness sub-function shown in Eq. 4 maximizes *NFR* satisficement throughout execution, where $utility\_value_{non-functional}$ references the calculated utility values from Eq. 2:

$$FF_{nfr} = \frac{\sum utility\_value_{non-functional}}{|utility_value_{non-functional}| * timesteps_{sim}} \tag{4}$$

The fitness sub-function shown in Eq. 5 minimizes the number of adaptations performed by the SAS, where $|adaptations|$ reports the total number of reconfigurations performed by the SAS, and $|faults|$ reports the total number of adverse conditions introduced within the simulation.

$$FF_{na} = 1.0 - \frac{|adaptations|}{|faults|} \tag{5}$$

We aggregate FF_{nfr}, FF_{fr}, and FF_{na} into a linear weighted sum as shown in Eq. 6:

$$FF = \begin{cases} \alpha_{nfr} * FF_{nfr} + \alpha_{fr} * FF_{fr} + \alpha_{na} * FF_{na} & \text{iff invariants true} \\ 0.0 & \text{otherwise} \end{cases} \tag{6}$$

where α_{nfr}, α_{fr}, and α_{na} are manually set by a requirements engineer based on domain knowledge/empirical evidence, reflect the relative importance of each sub-FR, and must cumulatively sum to a value of 1.0. While many different approaches exist for combining fitness sub-functions, we find that a linear-weighted sum balances competing concerns adequately for this domain.

(4) Select NFR Models. `Providentia` selects genomes, using tournament selection, with the highest fitness values calculated from Eq. 6 to guide the search process towards promising areas of the search space. The remainder of the population is removed from consideration.

(5) Generate NFR Models. `Providentia` uses two-point crossover and single-point mutation to generate new solutions. Two-point crossover selects two indices to be used as crossover points, selects two candidate solutions as parents, and swaps genes between the crossover points to create two new child solutions. Single-point mutation randomly selects a single gene for mutation, where the floating-point weight value can be modified within ±20% of its original value.

Given that each genome comprises sets of weights for each NFR (i.e., sub-genomes), crossover and mutation are applied to internal sub-genomes. Furthermore, a process of normalization follows creation of child solutions. Specifically,

each value selected to participate in either crossover or mutation is retained, and the remaining genes for that particular NFR within a sub-genome are normalized to sum to 1.0. Steps (3)–(5) are applied iteratively (i.e., the genetic algorithm loop) until the number of generations is reached. `Providentia` then outputs a set of optimal weighted FRs for each NFR.

4 Experimental Results

This section describes our experimental setup and presents our experimental results from applying `Providentia` to the RDM application.

4.1 Experimental Setup

We modeled the RDM network application as a completely-connected graph, where each node represents an RDM and each edge represents a network link. System and environmental parameters were randomized for each trial and based on an operational model previously presented by Keeton *et al.* [13,14]. For each experimental trial, a given network comprised random number of RDMs (i.e., within $[15, 30]$), a random number of valid messages (i.e., $[100, 200]$) were inserted into RDMs throughout the network at random timesteps and were required to be replicated to all other RDMs. We examined seven NFRs specific to the system. The simulation was performed over 300 timesteps.

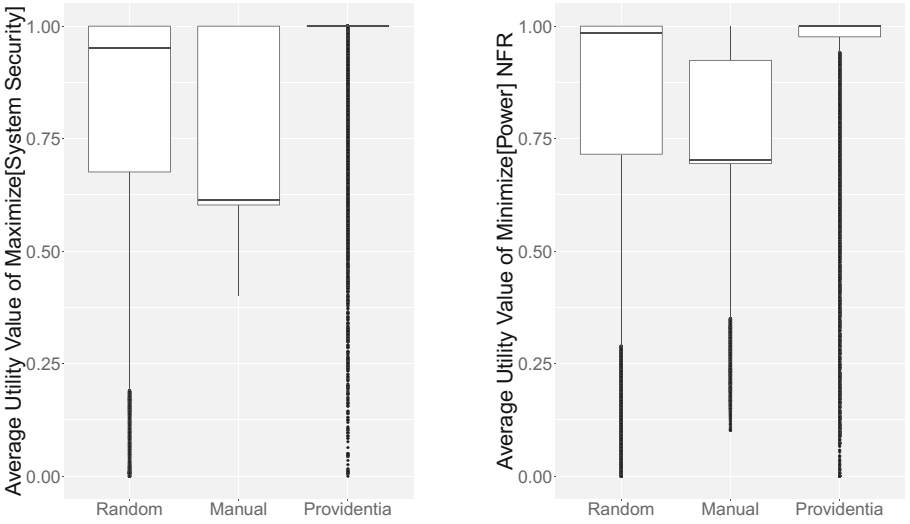
We compared and evaluated different combinations of supporting FR weights optimized by `Providentia`. The set of seven NFRs was applied to three types of treatments: (1) FRs and weights generated by random search [1], (2) manually-selected FRs and corresponding weights assigned by a requirements engineer, and (3) `Providentia`-optimized FRs and weights. We limit our discussion to NFR4 and NFR7 due to space constraints. The manually selected goals and weights for NFR4 are Goals A, B, D, G, H with corresponding weights 0.4, 0.2, 0.2, 0.1, 0.1, and for NFR7 manually selected Goals A, E, I, V, W with corresponding weights 0.3, 0.1, 0.1, 0.3, 0.2. Using the fitness functions defined in Eqs. 3-6, we demonstrate the benefits of using `Providentia` to both mitigate uncertainty (e.g., environmental and system) and reduce the impact of security threats against the RDM network. For this experiment, we set $\alpha_{fr} = 0.375$, $\alpha_{nfr} = 0.375$, and $\alpha_{na} = 0.25$ to emphasize minimization of network adaptations while considering maximization of FR/NFR satisficement. To demonstrate statistical significance, 50 trials were conducted for each experiment. Moreover, an equal number of experimental evaluations was performed per experiment.

4.2 Experimental Results

For this experiment, we define two null hypotheses. The first, $H1_0$, states that "there is no difference between `Providentia`-optimized NFRs and those that are unoptimized." The alternate hypothesis, $H1_1$, states that "there is a difference between `Providentia`-optimized NFRs and those that are unoptimized."

The second null hypothesis, $H2_0$, states that "there is no difference between `Providentia`-optimized NFRs and those that are optimized by a requirements engineer," with the corresponding alternate hypothesis, $H2_1$, stating that "there is a difference between `Providentia`-optimized NFRs and those that are optimized by a requirements engineer."

To demonstrate these hypotheses, Fig. 5(a) shows three boxplots with averaged fitness values calculated from `Providentia`-generated weights, from FR weights optimized by an engineer, and FR weights randomly selected, for NFR4. Similarly, Fig. 5(b) presents the averaged fitness values for NFR7.



(a) NFR4 fitness value comparison.

(b) NFR7 fitness value comparison.

Fig. 5. NFR fitness experimental results.

As the boxplots in Fig. 5 demonstrate, `Providentia`-optimized NFRs impact overall fitness significantly more than those set manually by a requirements engineer or randomly selected ($p < 0.05$[3]). The ideal utility value for a given NFR is 1.0 to indicate complete satisfaction and therefore the boxplot closest to 1.0 indicates optimal behavior. Table 1 provides the average utility values (μ) and standard deviation (σ) for each NFR. The genetic algorithm is able to effectively search for optimal FRs and weights when the system is subjected to randomized sources of uncertainty at design time to harden the system against uncertainty at run time, enabling a more robust set of NFRs in comparison to randomly- or manually-defined NFRs.

[3] The Wilcoxon-Mann-Whitney U-test was performed for all presented statistics.

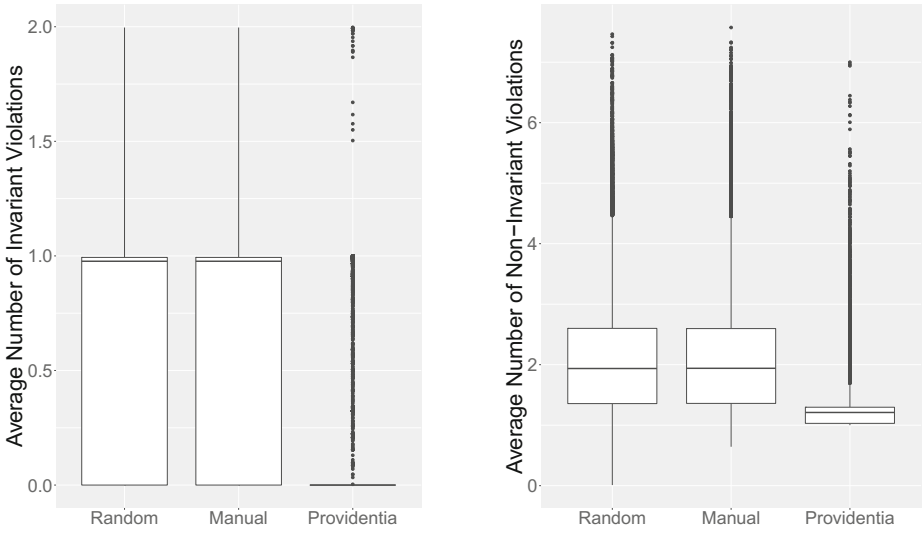
Table 1. NFR average utility values and standard deviations.

NFR	Random	Manual	Providentia
`NFR1: maximize`	μ: 0.654	μ: 0.615	μ: 0.905
`[Reliability]`	σ: 0.325	σ: 0.191	σ: 0.149
`NFR2: maximize`	μ: 0.655	μ: 0.666	μ: 0.882
`[Throughput]`	σ: 0.325	σ: 0.262	σ: 0.153
`NFR3: maximize`	μ: 0.875	μ: 0.743	μ: 0.975
`[Speed]`	σ: 0.207	σ: 0.148	σ: 0.085
`NFR4: maximize`	μ: 0.802	μ: 0.736	μ: 0.979
`[System security]`	σ: 0.273	σ: 0.177	σ: 0.085
`NFR5: maximize`	μ: 0.621	μ: 0.742	μ: 0.925
`[Secure communication]`	σ: 0.273	σ: 0.191	σ: 0.146
`NFR6: maximize`	μ: 0.921	μ: 0.919	μ: 0.980
`[Message security]`	σ: 0.181	σ: 0.072	σ: 0.069
`NFR7: minimize`	μ: 0.821	μ: 0.758	μ: 0.926
`[Power]`	σ: 0.270	σ: 0.172	σ: 0.188

`Providentia` also significantly decreased the amount of encountered FR violations when compared to manual and random search ($p < 0.05$) of FR combinations and their respective weights as seen in Fig. 6. These results further demonstrate the effectiveness of `Providentia`. The ideal number of FR violations is 0, and once again the difference between `Providentia` and random/manual results is significant. `Providentia` is able to not only significantly improve NFR satisficement, but is able to do so while significantly reducing the number of FR violations rather than creating extra overhead with additional functionality at run time.

The overall intent of `Providentia` is to ensure continuing requirements satisficement when faced with both uncertainty and NFR concerns. Given the overall success of `Providentia` when optimizing FR selection weights and minimizing violations, the presented results enable us to reject both $H1_0$ and $H2_0$, accept $H1_1$ and $H2_1$, and conclude that an optimized weighting scheme can significantly improve overall requirements satisficement when compared to random search or manually-derived weighting schemes, given that FF_{fr} and FF_{nfr} (c.f., Eqs. 3 and 4) form a major aspect of the overall fitness function.

Threats to Validity. This research has been a proof of concept to demonstrate how quantifying NFRs, elevating them to first-class entities, and automatically optimizing them can significantly improve overall requirements satisficement and minimize violations. One threat to validity includes the derivation of FRs that negatively impact the satisficement of an NFR, as `Providentia` currently only focuses on FRs that positively impact NFR satisficement. Additionally, the manual selection of the FR subset for each NFR could be argued to use better



(a) Invariant violation comparison.

(b) Noninvariant violation comparison.

Fig. 6. FR violation experimental results.

selections. The scalability of `Providentia` with respect to large numbers of goals and NFRs is a possible threat to validity as well.

5 Related Work

This section overviews related work in the areas of goal modeling, NFRs, and using functional and non-functional satisficement for guiding the adaptation of SASs.

Goal Modeling. Approaches similar to `Providentia` in goal modeling address dependencies between FRs [21], use probabilistic methods to improve NFR/FR satisficement [4,22] or optimize SAS satisficement [3,18,33], and represent NFRs as soft goals [11,35]. Our technique focuses solely on NFR/FR dependencies, optimizing for run-time performance without prior knowledge of system performance that most probabilistic methods require. We also do not discuss early-phase requirements engineering or high-level abstraction [7,20], but rather focus on a run-time model used by an SAS.

Non-functional Requirements. Other techniques have been introduced to quantify NFRs, generally representing NFRs as soft goals [16,32,34]. Our technique is independent of any framework (e.g., NFR Framework, iStar, and KAOS) and our weighted approach enables greater flexibility that an SAS can use to find an optimal reconfiguration strategy at run time rather than modeling NFRs at design time. Salehie *et al.* use a Goal-Action-Attribute Model (GAAM) and

an automated weighting scheme called Analytic Hierarchy Process to prioritize NFRs. `Providentia` uses a genetic algorithm to optimize goal and weight selection rather than prioritization, as priorities may shift due to uncertainty and requirement interactions. Contributing work has decomposed NFR behaviors into monitored patterns [28] and used quantifiable metrics to separate NFRs from the FR goal model [29]. `Providentia` monitors requirements at run time and does not separate NFRs from the goal model of FRs, as a separation does not necessarily allow the requirements engineer to identify cross-cutting concerns in NFRs.

6 Conclusion

This paper presented `Providentia`, a search-based technique for automatically quantifying NFRs at run time by optimizing FR and weight selections at design time. To demonstrate the effectiveness of `Providentia`, we used an industry-provided RDM application that must distribute messages amongst a network of RDMs that experienced uncertainty. Experimental results suggest that our approach significantly improves overall FR and NFR satisficement and decreases goal violations when compared to NFRs configured manually by a requirements engineer or configured by random search. Future directions for this research include performing the search process at run time while the system is subjected to uncertainty, exploring different search heuristics for `Providentia`, and applying `Providentia` to a real-world system. Furthermore, the RELAX language [25,31] and FLAGS [2] introduce flexibility into the satisfaction of selected requirements via fuzzy logic that can directly be applied to `Providentia` to better measure NFR satisfaction.

Acknowledgements. This work has been supported in part by grants from the NSF (CNS-1657061, CNS-1305358, and DBI-0939454), the Michigan Space Grant Consortium, the Comcast Innovation Fund, Oakland University, Ford Motor Company, General Motors Research, the Air Force Research Laboratory (AFRL) under agreement number FA8750-16-2-0284, and Michigan State University through the Institute for Cyber-Enabled Research. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and do not necessarily represent the opinions of the sponsors.

References

1. Arcuri, A., Briand, L.: A practical guide for using statistical tests to assess randomized algorithms in software engineering. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, pp. 1–10. ACM (2011)
2. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 18th IEEE International Requirements Engineering Conference (RE), 27 September 2010-1 October 2010, pp. 125–134 (2010)

3. Bencomo, N., Belaggoun, A.: A world full of surprises: Bayesian theory of surprise to quantify degrees of uncertainty. In: Companion Proceedings of the 36th International Conference on Software Engineering, pp. 460–463. ACM (2014)
4. Cailliau, A., van Lamsweerde, A.: Runtime monitoring and resolution of probabilistic obstacles to system goals. In: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 1–11. IEEE Press (2017)
5. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional requirements. Softw. Eng. (2000)
6. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-functional Requirements in Software Engineering. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-4615-5269-7
7. Dalpiaz, F., Borgida, A., Horkoff, J., Mylopoulos, J.: Runtime goal models: keynote. In: 2013 IEEE Seventh International Conference on Research Challenges in Information Science (RCIS), pp. 1–11. IEEE (2013)
8. Dardenne, A., Van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Sci. Comput. Program. **20**(1), 3–50 (1993)
9. deGrandis, P., Valetto, G.: Elicitation and utilization of application-level utility functions. In: Proceedings of the 6th International Conference on Autonomic Computing, ICAC 2009, pp. 107–116. ACM (2009)
10. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**(10), 46–54 (2004)
11. Giorgini, P., Mylopoulos, J., Sebastiani, R.: Goal-oriented requirements analysis and reasoning in the tropos methodology. Eng. Appl. Artif. Intell. **18**(2), 159–171 (2005)
12. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
13. Ji, M., Veitch, A., Wilkes, J.: Seneca: Remote mirroring done write. In: USENIX 2003 Annual Technical Conference, pp. 253–268. USENIX Association, Berkeley, June 2003
14. Keeton, K., Santos, C., Beyer, D., Chase, J., Wilkes, J.: Designing for disasters. In: Proceedings of the 3rd USENIX Conference on File and Storage Technologies, pp. 59–62. USENIX Association, Berkeley (2004)
15. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
16. Kobayashi, N., Morisaki, S., Atsumi, N., Yamamoto, S.: Quantitative non functional requirements evaluation using softgoal weight. J. Internet Serv. Inf. Secur. **6**(1), 37–46 (2016)
17. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, Hoboken (2009)
18. Letier, E., van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. In: Proceedings of the 12th ACM SIGSOFT Twelfth International Symposium on Foundations of Software Engineering, pp. 53–62 (2004)
19. McKinley, P., Sadjadi, S., Kasten, E., Cheng, B.H.C.: Composing adaptive software. Computer **37**(7), 56–64 (2004)
20. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: a process-oriented approach. IEEE Trans. Softw. Eng. **18**(6), 483–497 (1992)

21. Nagel, B., Gerth, C., Post, J., Engels, G.: Kaos4SOA-extending KAOS models with temporal and logical dependencies. In: CAiSE Forum, pp. 9–16 (2013)
22. Paucar, L.H.G., Bencomo, N.: The reassessment of preferences of non-functional requirements for better informed decision-making in self-adaptation. In: IEEE International Requirements Engineering Conference Workshops (REW), pp. 32–38. IEEE (2016)
23. Qureshi, N.A., Perini, A.: Engineering adaptive requirements. In: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-managing Systems, pp. 126–131, May 2009
24. Ramirez, A.J., Cheng, B.H.C.: Automatically deriving utility functions for monitoring software requirements. In: Proceedings of the 2011 International Conference on Model Driven Engineering Languages and Systems Conference, Wellington , pp. 501–516 (2011)
25. Ramirez, A.J., Fredericks, E.M., Jensen, A.C., Cheng, B.H.C.: Automatically RELAXing a goal model to cope with uncertainty. In: Fraser, G., Teixeira de Souza, J. (eds.) SSBSE 2012. LNCS, vol. 7515, pp. 198–212. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33119-0_15
26. Ramirez, A.J., Knoester, D.B., Cheng, B.H.C., McKinley, P.K.: Applying genetic algorithms to decision making in autonomic computing systems. In: Proceedings of the 6th International Conference on Autonomic Computing, pp. 97–106 (2009)
27. Salehie, M., Tahvildari, L.: Towards a goal-driven approach to action selection in self-adaptive software. Softw.: Pract. Exp. **42**(2), 211–233 (2012)
28. Supakkul, S., Hill, T., Chung, L., Tun, T.T., do Prado Leite, J.C.S.: An NFR pattern approach to dealing with NFRS. In: 2010 18th IEEE International Requirements Engineering Conference (RE), pp. 179–188. IEEE (2010)
29. Sykes, D., Heaven, W., Magee, J., Kramer, J.: Exploiting non-functional preferences in architectural adaptation for self-managed systems. In: Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 431–438. ACM (2010)
30. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility functions in autonomic systems. In: Proceedings of the First IEEE International Conference on Autonomic Computing, pp. 70–77. IEEE Computer Society (2004)
31. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.: Relax: Incorporating uncertainty into the specification of self-adaptive systems. In: 17th IEEE International Requirements Engineering Conference (RE 2009), pp. 79–88 (2009)
32. Yamamoto, S.: An approach for evaluating softgoals using weight. In: Khalil, I., Neuhold, E., Tjoa, A.M., Da Xu, L., You, I. (eds.) CONFENIS/ICT-EurAsia - 2015. LNCS, vol. 9357, pp. 203–212. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24315-3_20
33. Yang, Z., Jin, Z., Li, Z.: Achieving adaptation for adaptive systems via runtime verification: a model-driven approach. arXiv preprint arXiv:1704.00869 (2017)
34. Yrjönen, A., Merilinna, J.: Extending the NFR framework with measurable non-functional requirements. In: NFPinDSML@ MoDELS (2009)
35. Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, pp. 226–235 (1997)
36. Yu, E.: Social Modeling for Requirements Engineering. MIT, Cambridge (2011)