Automatically RELAXing a Goal Model to Cope with Uncertainty

Andres J. Ramirez, Erik M. Fredericks,
Adam C. Jensen, and Betty H.C. Cheng

Michigan State University,
East Lansing, MI, 48823, USA
{ramir105,freder99,acj,chengb}@cse.msu.edu

Abstract. Dynamically adaptive systems (DAS) must cope with changing system and environmental conditions that may not have been fully understood or anticipated during development time. RELAX is a fuzzy logic-based specification language for making DAS requirements more tolerable to unanticipated environmental conditions. This paper presents AutoRELAX, an approach that generates RELAXed goal models that address environmental uncertainty by identifying which goals to RELAX, which RELAX operators to apply, and the shape of the fuzzy logic function that defines the goal satisfaction criteria. AutoRELAX searches for RELAXed goal models that enable a DAS to satisfy its functional requirements while balancing tradeoffs between minimizing the number of RELAXed goals and minimizing the number of adaptations triggered by minor and adverse environmental conditions. We apply AutoRELAX to an industry-provided network application that self-reconfigures in response to adverse environmental conditions, such as link failures.

1 Introduction

A dynamically adaptive system (DAS) must identify and respond to changing system and environmental conditions that may not have been fully understood or anticipated during requirements analysis and design time. Within a DAS, this contextual *uncertainty* arises from a combination of unpredictable environmental conditions [1,2,5,24] that can limit the adaptation capabilities of a DAS. RELAX [2,24] is a specification language that can be used in goal-oriented modeling approaches for specifying and mitigating sources of uncertainty in a DAS. This paper presents an approach for automatically RELAXing a goal-oriented model to account for environmental uncertainty, thus potentially decreasing the number of dynamic reconfigurations needed at run time.

It is unlikely for a DAS to always satisfy its requirements since it is often infeasible for a requirements engineer or a developer to identify all possible environmental conditions that the DAS may encounter throughout its lifetime [2,24]. In light of this implication, RELAX extends goal-oriented requirements modeling approaches, such as KAOS [4,14], with fuzzy logic-based operators that specify the extent to which a goal can become temporarily unsatisfied and yet deliver acceptable behavior. For instance, the "AS EARLY AS POSSIBLE" RELAX operator

G. Fraser (Ed.): SSBSE 2012, LNCS 7515, pp. 198–212, 2012.

enables a DAS to satisfy a goal over a longer period of time [24]. Nevertheless, it is difficult for a requirements engineer to assess, at design time, which goals to RELAX, what RELAX operators to apply, and how a goal's RELAXation will affect the overall behavior of the DAS at run time.

This paper introduces AutoRELAX, an approach that extends and automates an approach previously presented by Cheng *et al.* [2] for modeling sources of uncertainty in a DAS with RELAX. AutoRELAX explicitly handles environmental uncertainty in a DAS by automatically RELAXing goals in a KAOS goal model. In particular, AutoRELAX specifies whether a goal should be RELAXed, and if so, which RELAX operator to apply, and to what degree to lessen the constraints or bounds that define a goal's satisfaction criteria. AutoRELAX can be applied to automatically generate one or more RELAXed goal models, each of which enables a DAS to cope with specific manifestations of system and environmental uncertainty while reducing the number of adaptations performed.

AutoRELAX leverages a genetic algorithm [9] as a search heuristic to efficiently explore parts of the solution space comprising all possible RELAXed goal models. Throughout the search process, AutoRELAX uses an executable specification of the DAS to measure how candidate RELAXed goal models handle the effects of system and environmental uncertainty. AutoRELAX applies a set of fitness subfunctions that use this information to reward candidate RELAXed goal models that enable a DAS to satisfy its functional requirements while also reducing the number of adaptations the DAS performs and, consequently, the impact of a dynamic reconfiguration at run time.

We demonstrate AutoRELAX by applying it to an industry-provided application that handles the dynamic reconfiguration of a remote data mirroring (RDM) network [11,12] that improves data availability and protection by replicating and storing data at physically isolated locations. In particular, the RDM network must distribute data even under adverse system and environmental conditions, such as faulty network links and dropped messages. Experimental results show that AutoRELAX can automatically generate RELAXed goal models that are as good, if not better, than those manually created by a requirements engineer. Moreover, experimental results also demonstrate that RELAXing the satisfaction criteria of goals affected by uncertainty can reduce both the number of adaptations and the level of disruption of an adaptation.

The remainder of this paper is organized as follows. Section 2 provides background material on remote data mirroring, goal-oriented requirements modeling, RELAX, and genetic algorithms. Next, Section 3 presents the AutoRELAX process, followed by an experimental evaluation in Section 4. Section 5 overviews related work. Lastly, Section 6 summarizes findings and presents future directions.

2 Background

This section presents background material on remote data mirroring, goal-oriented modeling, the RELAX specification language, and genetic algorithms.

2.1 Remote Data Mirroring

Remote data mirroring (RDM) [11,12] is a data protection technique that prevents data loss and unavailability by storing replicates at physically remote locations. An RDM can be configured in terms of its network topology, such as a minimum spanning tree or a redundant topology, as well as the method and timing of data distribution among nodes. In particular, synchronous propagation automatically distributes every data modification to all other nodes. In contrast, asynchronous propagation batches data modifications to coalesce edits to the same data. While asynchronous propagation provides better network performance than synchronous propagation, it also provides a weaker form of data protection because batched data could be lost in the event of a site failure.

2.2 Goal-oriented Requirements Modeling

A goal declaratively specifies the objectives and constraints that a system must provide and satisfy, respectively. A functional goal specifies a service that the system must provide to its stakeholders. A goal can also be classified either as an invariant or a non-invariant. While a system must always satisfy invariant goals, a system may tolerate the temporary violation of a non-invariant goal.

A goal-oriented requirements model visually captures relationships between goals by using a directed acyclic graph where a node represents a goal and an edge represents a type of goal refinement [14]. For instance, KAOS [4,14] provides a framework for systematically refining high-level *functional* goals into finer-grained goals that are more amenable to analysis. Within KAOS, a goal can be refined via an AND or OR refinement. A goal that has been AND-refined can only be satisfied if every subgoal is also satisfied. Conversely, a goal that has been OR-refined is satisfied if at least one subgoal has been satisfied. Goal refinement continues until each leaf-level goal is assigned to an agent responsible for satisfying that goal, which then defines a requirement.

The KAOS goal model in Figure 1 captures functional requirements of the RDM application. RDMs must (A) maintain remotely stored copies of data. To this end, RDMs must (B) maintain operational costs within the allocated budget while (C) achieving acceptable levels of risk and (D) distributing data to all other nodes. Before distributing data, the RDM must (E) measure network properties and (F) construct a network by (P) activating and (Q) deactivating network links. The system must then (I) send and (J) receive data, using either (G) synchronous or (H) asynchronous propagation methods.

2.3 RELAX Specification Language

RELAX [24] is a language for specifying how sources of uncertainty that arise at the shared boundary [10] between the system and its environment affects requirements. RELAX organizes information about these sources of uncertainty into ENV, MON and REL elements: ENV specifies environmental properties that

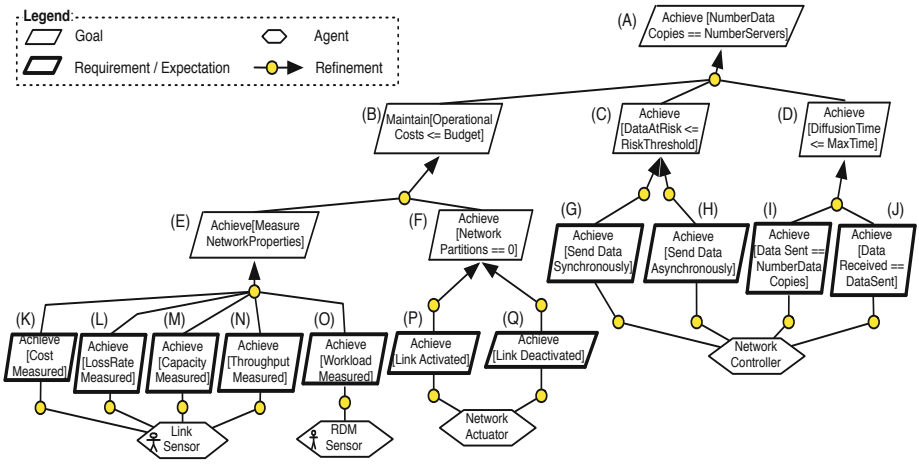

Fig. 1. KAOS goal model for the remote data mirroring application

can be observed by the DAS; MON indicates sensors in the monitoring infrastructure of the DAS; and REL establishes mathematical relationships for computing the values of ENV properties by aggregating values from MON elements.

Table 1 presents the fuzzy logic operators that RELAX provides for capturing uncertainty in requirements. These ordinal and temporal operators add flexibility in terms of how and when a requirement must be satisfied, respectively. For example, goal (F) in Figure 1 specifies that the RDM shall "achieve zero network partitions". Nevertheless, unpredictable network link failures can temporarily obstruct this goal. This goal can be RELAXed to state that the RDM shall "achieve AS FEW network partitions AS POSSIBLE", thus providing flexibility to account for unanticipated events while distributing data. In this manner, RELAX facilitates designing more flexible systems that might potentially require fewer dynamic reconfigurations.

Table 1. RELAX operators

Temporal Operators	Ordinal Operators
AS EARLY AS POSSIBLE	AS FEW AS POSSIBLE
AS CLOSE AS POSSIBLE TO [frequency]	AS CLOSE AS POSSIBLE TO [quantity]
AS LATE AS POSSIBLE	AS MANY AS POSSIBLE

2.4 Genetic Algorithms

A genetic algorithm [9] is a stochastic search-based heuristic for efficiently solving complex optimization problems. In a genetic algorithm, a population comprises a set of individuals, each encoding a candidate solution. A fitness function evaluates the quality of each individual, thereby guiding the search process towards promising areas in the solution space. New solutions can be generated with crossover and mutation operators. The crossover operator exchanges parts

of existing solutions to form new individuals with, ideally, higher fitness values, and the mutation operator randomly modifies an individual to maintain diverse solution elements in the population. These operations are executed until the maximum number of generations, or iterations, are exhausted.

3 Approach

This section introduces the AutoRELAX approach. First, we state the assumptions, inputs, and outputs of AutoRELAX. We then describe how AutoRELAX can be applied to automatically generate RELAXed goal models.

3.1 Assumptions, Inputs, and Outputs

AutoRELAX needs three key input elements: a goal model, a set of utility functions for requirements monitoring, and an executable specification or prototype of the DAS. Next, we briefly describe the contents and purpose of each element.

Goal Model. AutoRELAX requires a goal model of the functional requirements that the DAS must satisfy. Currently, we target KAOS goal models [4,14]. Each goal must be designated as invariant or non-invariant.

Utility Function. A requirements engineer must derive utility functions that can monitor the satisfaction of requirements in a DAS [8,17,22]. Each utility function comprises mathematical relationships that map monitoring data to a scalar value between zero and one. This value is proportional to how well a given goal or requirement is satisfied at run time. For example, satisfaction of goal (B) in the RDM application (see Figure 1) can be evaluated with a utility function that returns one if operational costs have always been less than or equal to the allocated budget, and zero otherwise. AutoRELAX uses these utility functions to evaluate how goal RELAXations can affect DAS behavior.

Executable Specification. AutoRELAX requires an executable specification of the DAS, such as a simulation or a prototype, that applies the set of utility functions to measure how well the DAS satisfies its requirements in response to adverse conditions. In addition, a requirements engineer must also specify possible sources of uncertainty to which the DAS will be exposed. Ideally, these sources of uncertainty will exercise the adaptation logic of the DAS by subjecting it to unpredictable and adverse environmental conditions that can lead to a requirements violation. For example, in our remote data mirroring application, we configure the probability and frequency that a network link can fail or a message can be dropped. Changing either the sources of uncertainty, their likelihood, or frequency can lead to different types of RELAXed goal models.

3.2 AutoRELAX Process

Figure 2 presents a data flow diagram (DFD) that overviews the AutoRELAX process. We now present each step in the AutoRELAX process in detail:

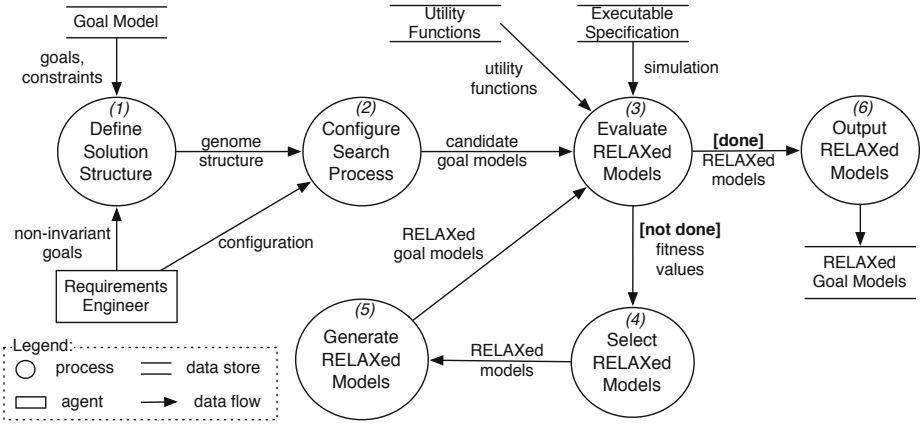

Fig. 2. DFD diagram of AutoRELAX process

(1) Define Solution Structure. Each candidate solution in AutoRELAX comprises a vector of n elements or *genes*, where n is equal to the total number of non-invariant goals in the KAOS goal model of the DAS. Figure 3(A), in turn, shows the structure of each gene. As this figure illustrates, each gene comprises a boolean variable that specifies whether a non-invariant goal will be RELAXed, a corresponding RELAX operator (see Table 1), and two floating point values that define the left and right boundaries of the fuzzy logic function, respectively.

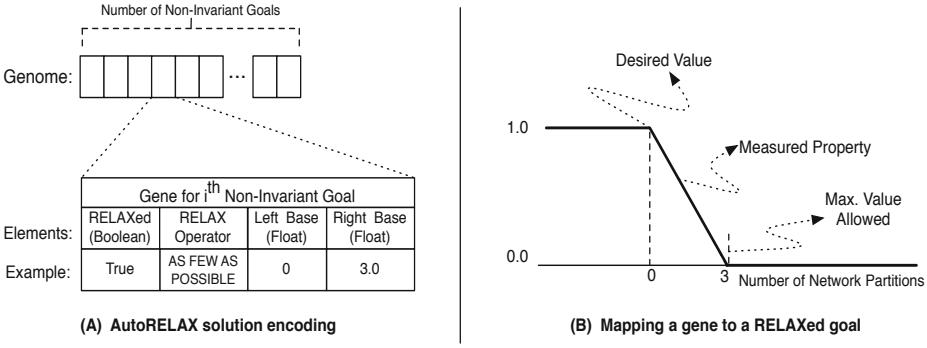

Fig. 3. Encoding a candidate solution in AutoRELAX

Figure 3(B) illustrates how each gene is mapped to a fuzzy logic function that can be used to evaluate the satisfaction of a goal. In this example, the unRELAXed satisfaction criteria (i.e., utility function) for goal (F) in Figure 1 returns 1.0 if the network is connected and 0.0 otherwise. Nevertheless, if the network partition is transient, then it may be possible to continue the data replication process amongst those nodes that are connected while the network is reconfigured. Thus, as the bolded lines show in Figure 3(B), the satisfaction criteria of this goal can be RELAXed by applying the "AS FEW AS POSSIBLE" ordinal operator that maps to a *left shoulder* fuzzy logic function shape [24].

For this RELAXed goal, the apex is still centered upon the ideal value of a system or environmental property. In this case, zero network partitions and the downward slope from the apex to the right endpoint reflects values that, while not ideal, might be tolerated at run time. Note that the *left endpoint* value encoded in the gene is not used for this particular fuzzy logic shape.

(2) Configure Search Process. A requirements engineer must configure AutoRELAX by specifying a population size, crossover and mutation rates, and a termination criterion. The population size determines how many candidate RELAXed goal models AutoRELAX can explore in parallel during each generation; the crossover and mutation rates specify how AutoRELAX will generate new RELAXed goal models; and the termination criteria specifies when AutoRELAX will stop searching for new solutions and output the resulting RELAXed goal models.

(3) Evaluate RELAXed Models. To evaluate the quality of a RELAXed goal model, AutoRELAX first maps the RELAX operators encoded in an individual to their corresponding utility functions for requirements monitoring in the executable specification (see Step 1). Next, AutoRELAX simulates the executable specification and records the satisfaction of each goal as well as the number of adaptations performed by the DAS. Two fitness sub-functions use this information to reward candidate RELAXed goal models for minimizing both the number of RELAXed goals as well as how many adaptations are triggered by minor environmental conditions.

The first fitness sub-function, $\mathrm{FF}_{\mathrm{nrg}}$, rewards candidate solutions that minimize the number of RELAXed goals:

$$\mathrm{FF}_{\mathrm{nrg}} = 1.0 - \left(\frac{|\mathrm{relaxed}|}{|\mathrm{Goals}_{\text{non-invariant}}|}\right)$$

where $|\mathrm{relaxed}|$ and $|\mathrm{Goals}_{\text{non-invariant}|}$ are the number of RELAXed and non-invariant goals in the goal model, respectively. The intent of this fitness sub-function is to preserve the intent of the original goal model by discouraging AutoRELAX from unnecessarily introducing RELAX operators.

The second fitness sub-function, $\mathrm{FF}_{\mathrm{na}}$, rewards candidate solutions that minimize the number of adaptations performed by the DAS in response to minor and transient environmental conditions:

$$\mathrm{FF}_{\mathrm{na}} = 1.0 - \left(\frac{|\mathrm{adaptations}|}{|\mathrm{faults}|}\right)$$

where $|\mathrm{adaptations}|$ represents the total number of adaptations performed by the DAS, and $|\mathrm{faults}|$ measures the total number of adverse environmental conditions introduced throughout a simulation. This fitness sub-function rewards RELAXed goal models that tolerate unanticipated environmental conditions and reduce the number of adaptations a DAS performs, thereby reducing the number of passive and quiescent components at run time [13]. While a *passive* component may service transactions from other components, it may not initiate transactions. In contrast, a *quiescent* component cannot initiate new transactions nor service transactions from other components. Reducing the number of passive and quiescent components minimizes the impact of adaptation upon the DAS behavior.

These two fitness sub-functions can be combined into a linear weighted sum:

$$\text{Fitness Value} = \begin{cases} \alpha_{\text{nrg}} * \text{FF}_{\text{nrg}} + \alpha_{\text{na}} * \text{FF}_{\text{na}} & \text{iff invariants true} \\ 0.0 & \text{otherwise} \end{cases}$$

where α_{nrg} and α_{na} coefficients reflect the relative importance of each fitness sub-function, the sum of which must equal 1.0.[1] The fitness value of a RELAXed goal model depends upon the satisfaction of all invariant goals. For example, if a RELAXed goal model in our RDM application does not replicate every data item, then its fitness value is 0.0. This penalty ensures only *viable* RELAXed goal models, where all invariant goals are satisfied, are output as solutions.

(4) Select RELAXed Models. Using the fitness value associated with each evaluated RELAXed goal model, AutoRELAX *selects* the most promising individuals from the population to guide the search process towards that area of the solution space. To this end, AutoRELAX applies tournament selection [9], a technique that randomly selects k individuals from the population and *competes* them against one another. The RELAXed goal model with the highest fitness value amongst these k solutions survives onto the next generation.

(5) Generate RELAXed Models. AutoRELAX uses two-point crossover and single-point mutation to generate new RELAXed goal models, which were set to 50% and 40% for this work, respectively. As Figure 4(A) shows, two-point crossover takes two individuals from the population as *parents* and produces two new RELAXed goal models as *offspring*. As this figure illustrates with different shading, two-point crossover exchanges the genes between two randomly chosen indices. In contrast, Figure 4(B) shows how single-point mutation takes an individual from the population and randomly modifies the values of a single gene. In this particular example, the effect of the mutation operator is to change a gene such that its corresponding non-invariant goal is now RELAXed with the "AS MANY AS POSSIBLE" RELAX operator. In this manner, while crossover attempts to construct better solutions by combining good elements from existing RELAXed goal models, mutation introduces diverse sets of goal RELAXations that might not be obtainable otherwise.

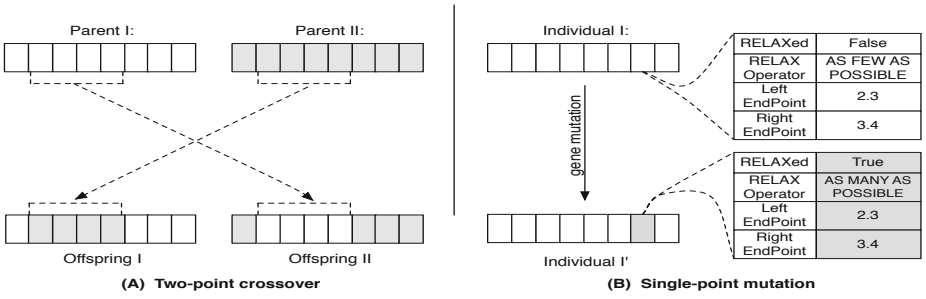

Fig. 4. Generating new RELAXed goal models with crossover and mutation operators

[1] Although fitness sub-functions can be combined in different ways, we find that a linear-weighed sum facilitates the balancing of competing concerns.

(6) Output RELAXed Models. AutoRELAX iteratively applies steps (3) through (5) until it reaches its generational limit. Then, AutoRELAX outputs one or more RELAXed goal models with the highest fitness values in the population.

4 Experimental Results

This section describes our experimental setup and discusses the experimental results where we apply AutoRELAX to a RDM application.

4.1 Experimental Setup

For this work, we modeled the RDM network as a completely connected graph where a node represents an RDM and an edge represents a network link. In particular, the network consists of 25 RDMs and 300 network links that can be activated and used to transfer data between RDMs. We leverage an RDM operational model previously presented by Keeton *et al.* [12] to generate performance attributes of each RDM and network link. Each network simulation executes for 150 time steps. Throughout each simulation, 20 data items are randomly inserted at different RDMs that are then responsible for distributing those data items to all other RDMs.

The RDM network is subject to environmental uncertainty in the form of unpredictable network link failures and dropped messages. As such, the RDM network might need to self-adapt in response to these adverse system and environmental conditions. To this end, each RDM implements the dynamic change management (DCM) protocol previously introduced by Kramer and Magee [13]. Furthermore, we implemented a rule-based adaptation engine that monitors the satisfaction of each goal to determine if the network structure and propagation parameters need to be reconfigured. If an adaptation is warranted, then Plato [18] and the DCM protocol are executed to generate a target system configuration and a series of reconfiguration steps that safely transitions the executing system from its current configuration to its target configuration, respectively.

We compare and evaluate the resulting RELAXed goal models produced by AutoRELAX with two different goal models of the same RDM application, the unRELAXed goal model shown in Figure 1 and a goal model manually RELAXed by a requirements engineer. The manually RELAXed goal model consists of five goal RELAXations: goal (C) is RELAXed to allow larger exposures to data loss; goal (D) is RELAXed to add temporal flexibility when diffusing data; goal (F) is RELAXed to allow up to three simultaneous network partitions; and goals (I) and (J) were RELAXed to tolerate dropped messages.

We use the fitness functions presented in Section 3 to compare these models and illustrate the benefits of RELAXing a goal model to address uncertainty as well as demonstrate that AutoRELAX is capable of generating RELAXed goal models that are as good, if not better, than those manually created by a requirements engineer. We set α_{nrg} to 0.3 and α_{na} to 0.7, thereby emphasizing the reduction in the number of adaptations performed. For statistical purposes, we conducted 50 trials of each experiment and, where applicable, plot or report the mean values with corresponding error bars or deviations.

4.2 Uncertain Environment

For this experiment, we define the first null hypothesis, $H1_0$, to state that there is no difference between a RELAXed and an unRELAXed goal model. In addition, we also define a second null hypothesis, $H2_0$, to state that there is no difference between RELAXed goal models generated by AutoRELAX and those manually created by a requirements engineer.

Figure 5 presents three box plots with the fitness values obtained by generated AutoRELAX models, a manually created RELAXed goal model, and an unRELAXed goal model, respectively, with the latter two being conducted only once. As these box plots illustrate, despite the fitness boost unRELAXed goal models obtain by not introducing any goal RELAXations (see Section 3, FF_{nrg}), RELAXed goal models achieved statistically significant higher fitness values than unRELAXed goal models ($p < 0.001$, Welch Two Sample t-test). These results enable us to reject our first null hypothesis, $H1_0$, as well as conclude that RELAX does indeed reduce the number of adaptations when addressing system and environmental uncertainty.

The box plots in Figure 5 also demonstrate that AutoRELAX generated RELAXed goal models achieved statistically significant higher fitness values than those manually created by a requirements engineer ($p < 0.001$, Welch Two Sample t-test). As a result, we also reject our second null hypothesis, $H2_0$ and conclude that AutoRELAX is capable of generating RELAXed goal models that better address specific sources of uncertainty than manually RELAXed goal models.

Figure 6 presents three sets of box plots that capture the adaptation costs incurred by generated AutoRELAX models, a manually created RELAXed goal model, and an unRELAXed goal model, respectively. Specifically, each set of box plots measures the amount of time components in the RDM network were in active, passive, and quiescent modes *during* a reconfiguration (these plots do not include time outside of a reconfiguration). As this figure illustrates, a key

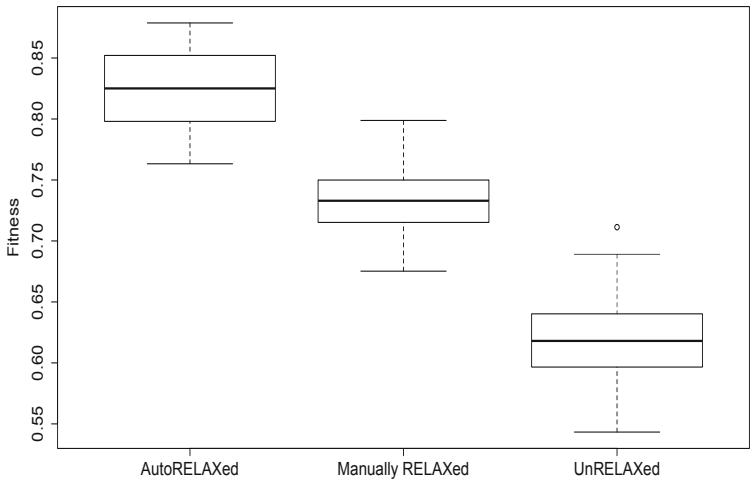

Fig. 5. Fitness values comparison between RELAXed and unRELAXed goal models.

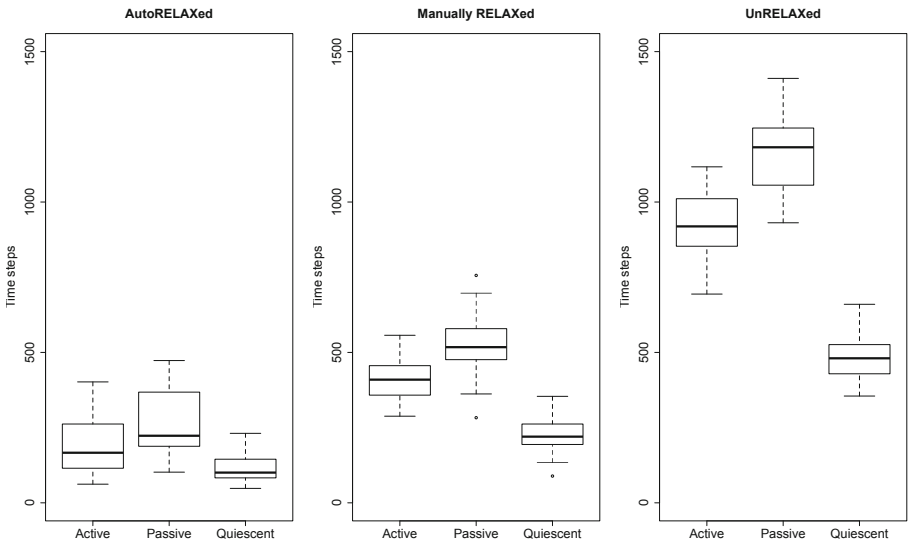

Fig. 6. Adaptation costs comparison between RELAXed and unRELAXed goal models

reason for why RELAXed goal models outperform unRELAXed goal models is that by carefully lessening the satisfaction criteria of non-invariant goals, the number of adaptations decrease and so does the amount of time components spend in passive and quiescent modes during a reconfiguration.

Both Figures 5 and 6 show that AutoRELAX is able to generate RELAXed goal models that perform better than manually RELAXed goal models. Examining the automatically generated RELAXed goal models suggests two primary reasons for this difference in fitness values and, consequently, in adaptation costs. First, while the manually RELAXed goal model introduced RELAXations to goals (C), (D), (F), (I), and (J), AutoRELAX mostly introduced RELAX operators to goals (F), (I), and (J), thereby slightly boosting its fitness value in comparison. Second, the manually RELAXed goal model contained some goal RELAXations that were too constrained. For instance, AutoRELAX was able to extend the goal satisfaction boundary of goal (F) beyond the bounds applied in the manually RELAXed goal model. As a result, the goal model produced by AutoRELAX was able to tolerate a greater number of temporary network partitions while allowing components to remain actively distributing data throughout the network.

Threat to Validity. This research was a proof of concept study to determine if it is *possible* to automatically RELAX goal models to produce viable goal models. We applied the technique on a problem provided to us from industrial collaborators. As a point of reference, we compared the AutoRELAXed goal models to those developed by a requirements engineer. Threats to validity include whether this technique will achieve similar results with other goal models and other applications. While, we did not intend to study human effectiveness [20], the positive results motivate its study explicitly as part of future work.

5 Related Work

This section presents related work on obstacle mitigation, expressing uncertainty in requirements, and requirements monitoring.

Obstacle Mitigation. van Lamsweerde *et al.* [14,15] proposed a set of strategies to facilitate the systematic identification, analysis, and resolution of obstacles, or conditions that prevent a system from satisfying its objectives. If an obstacle cannot be prevented, then one of their proposed mitigation strategies consists of tolerating the violation of a goal. This strategy, however, does not specify the extent to which a goal can become unsatisfied without adversely affecting other goals. From this perspective, AutoRELAX complements their proposed mitigation strategies by automatically determining if, and to what extent, a goal can become unsatisfied at run time.

Although the heuristics proposed by van Lamsweerde *et al.* [14,15] facilitate the systematic identification and analysis of obstacles, unpredictable environments may still prevent a DAS from satisfying its requirements. Letier and van Lamsweerde [16] also introduced a probabilistic framework for specifying the probability of a goal being satisfied. These probabilities, which can be obtained from a domain expert or derived from actual system usage data, can be used to identify previously unknown obstacles. In contrast to AutoRELAX, however, their probabilistic framework treats requirements as being either strictly satisfied or not. AutoRELAX could leverage the probability of a goal being satisfied when identifying goals that might benefit from RELAXation.

Expressing Uncertainty in Requirements. Fuzzy set theory has been recently applied to represent and analyze the effects of uncertainty in requirements. For instance, Whittle *et al.* [24] introduced RELAX to facilitate the identification and analysis of sources of uncertainty in a DAS. Cheng *et al.* [2] extended RELAX to support the modeling of RELAXed goals in a KAOS goal model [4,14]. Similarly, Baresi *et al.* [1] presented FLAGS, a KAOS goal modeling framework that introduces the concept of a fuzzy goal whose satisfaction is evaluated through fuzzy logic functions. Both RELAX and FLAGS depend on a requirements engineer to manually determine goals that may become unsatisfied, as well as how much flexibility to introduce for each goal's satisfaction criteria. AutoRELAX automates this process.

Welsh *et al.* [23] introduced Claims as markers of uncertainty that capture doubts about how a given goal realization strategy contributes to the satisficement [3], or satisfaction of a degree, of a soft goal. If a Claim is proven false at run time, then the DAS self-reconfigures towards a more desirable goal realization strategy. Although AutoRELAX focuses on RELAXing the satisfaction of functional non-invariant goals, it could be extended to also automatically RELAX the satisfaction criteria of soft goals.

Requirements Monitoring and Reflection. Feather *et al.* [6,7] developed requirements monitoring frameworks that can detect the occurrence of obstacles and reconfigure the system in response if necessary. More recently, Sawyer *et al.* [19]

suggested promoting requirements to live run-time entities whose satisfaction can be evaluated in support of adaptation decisions. Their concept is similar to the feedback-loop Awareness Requirements (AwReqs) construct proposed by Souza *et al.* [21] where meta-level requirements manage the satisfaction of other requirements. None of these approaches, however, support the management or run-time monitoring of RELAXed requirements. If these requirements monitoring and management frameworks were extended to support RELAXed requirements, then AutoRELAX could be applied to automatically specify the satisfaction criteria of RELAXed goals while these frameworks handle the run-time logistics of monitoring their satisfaction at run time.

6 Conclusions

In this paper we presented AutoRELAX, an approach that applies a genetic algorithm to automatically generate RELAXed goal models that can mitigate the effects of uncertainty upon the self-assessment capabilities of a DAS. By providing automated tool support, AutoRELAX relieves requirements engineers from the daunting task of considering a large number of strategies for dealing with uncertainty. We applied AutoRELAX to a RDM application that must distribute data to all nodes within the network while self-reconfiguring in response to network link failures and dropped messages. When compared with an unRELAXed goal model, experimental results show that RELAX is able to reduce the number of adaptations, and therefore adaptation costs, that would otherwise be incurred by minor and transient environmental conditions. Results also show that AutoRELAX can generate RELAXed goal models of equal or greater quality than those manually created by a requirements engineer.

AutoRELAX automatically provides feedback about sources of uncertainty. In these experiments, the goal RELAXations introduced by AutoRELAX concurred with the specific sources of uncertainty. Specifically, AutoRELAX introduced goal RELAXations to goals (F) and (J) depending on whether network links were more likely to fail than messages dropped, and vice-versa. Interestingly enough, AutoRELAX always RELAXed goal (J). By analyzing our goal models with this information we noted that, in contrast to goals (F) and (J), the satisfaction of goal (I) is affected by both network failures and dropped messages. Thus, the environmental uncertainty introduced in our experiments would frequently cause goal (I) to be violated and the network to self-reconfigure in response. As such, AutoRELAX automatically identified how sources of uncertainty affected this specific goal and then introduced RELAXed operators to lessen its impact.

Future directions include extending the search parameters to also optimize the underlying shape (i.e., triangle versus trapezoid) of the fuzzy logic function that defines the satisfaction criteria of a RELAXed goal, as well as extending AutoRELAX to support the automatic RELAXation of non-functional goals.

Acknowledgements. This work has been supported in part by NSF grants CCF-0541131, IIP-0700329, CCF-0750787, CCF-0820220, DBI-0939454, CNS-0854931, Army Research Office grant W911NF-08-1-0495, Ford Motor Company.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Army, Ford, or other research sponsors.

References

1. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: Proceedings of the 18th IEEE International Requirements Engineering Conference, pp. 125–134. IEEE, Sydney (2010)
2. Cheng, B.H.C., Sawyer, P., Bencomo, N., Whittle, J.: A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 468–483. Springer, Heidelberg (2009)
3. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers (2000)
4. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1-2), 3–50 (1993)
5. Esfahani, N., Kouroshfar, E., Malek, S.: Taming uncertainty in self-adaptive software. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, Szeged, Hungary, pp. 234–244 (2011)
6. Feather, M.S., Fickas, S., van Lamsweerde, A., Ponsard, C.: Reconciling system requirements and runtime behavior. In: Proceedings of the 8th International Workshop on Software Specification and Design, pp. 50–59. IEEE Computer Society, Washington, DC (1998)
7. Fickas, S., Feather, M.S.: Requirements monitoring in dynamic environments. In: Proceedings of the Second IEEE International Symposium on Requirements Engineering, pp. 140–147. IEEE Computer Society, Washington, DC (1995)
8. de Grandis, P., Valetto, G.: Elicitation and utilization of application-level utility functions. In: The Proceedings of the Sixth International Conference on Autonomic Computing (ICAC 2009), June 2009, pp. 107–116. ACM, Barcelona (2009)
9. Holland, J.H.: Adaptation in Natural and Artificial Systems. MIT Press, Cambridge (1992)
10. Jackson, M., Zave, P.: Deriving specifications from requirements: an example. In: Proceedings of the 17th International Conference on Software Engineering, pp. 15–24. ACM, Seattle (1995)
11. Ji, M., Veitch, A., Wilkes, J.: Seneca: Remote mirroring done write. In: USENIX, 2003 Annual Technical Conference, pp. 253–268. USENIX Association, Berkeley (2003)
12. 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)
13. Kramer, J., Magee, J.: The evolving philosophers problem: Dynamic change management. IEEE Trans. on Soft. Eng. 16(11), 1293–1306 (1990)
14. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley (March 2009)
15. van Lamsweerde, A., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Transactions on Software Engineering 26(10), 978–1005 (2000)

16. Letier, E., van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. In: Proceedings of the 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 53–62. ACM, Newport Beach (2004)
17. 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, New Zealand, pp. 501–516 (2011)
18. 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 Sixth International Conference on Autonomic Computing, Barcelona, Spain, pp. 97–106 (June 2009)
19. Sawyer, P., Bencomo, N., Letier, E., Finkelstein, A.: Requirements-aware systems: A research agenda for re self-adaptive systems. In: Proceedings of the 18th IEEE International Requirements Engineering Conference, Sydney, Australia, September 2010, pp. 95–103 (2010)
20. de Souza, J.T., Maia, C.L.B., de Freitas, F.G., Coutinho, D.P.: The human competitiveness of search based software engineering. In: Proceedings of the 2nd International Symposium on Search Based Software Engineering (SSBSE 2010), September 7-9, pp. 143–152. IEEE, Benevento (2010)
21. Souza, V.E.S., Mylopoulos, J.: From awareness requirements to adaptive systems: A control-theoretic approach. In: Proceedings of the Second International Workshop on Requirements at Run Time, pp. 9–15. IEEE Computer Society Press, Trento (2011)
22. 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, New York (2004)
23. Welsh, K., Sawyer, P.: Understanding the Scope of Uncertainty in Dynamically Adaptive Systems. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 2–16. Springer, Heidelberg (2010)
24. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.M.: RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th International Requirements Engineering Conference (RE 2009), pp. 79–88. IEEE Computer Society, Atlanta (2009)