Empir Software Eng
DOI 10.1007/s10664-014-9305-0

AutoRELAX: automatically RELAXing a goal model to address uncertainty

Erik M. Fredericks · Byron DeVries · Betty H. C. Cheng

Abstract Dynamically adaptive systems (DAS) must cope with system and environmental conditions that may not have been fully understood or anticipated during development. RELAX is a fuzzy logic-based specification language for identifying and assessing sources of environmental uncertainty, thereby making DAS requirements more tolerant of unanticipated conditions. This paper presents AutoRELAX, an approach that automatically generates RELAXed goal models to address environmental uncertainty. Specifically, AutoRELAX identifies goals to RELAX, which RELAX operators to apply, and the shape of the fuzzy logic function that establishes the goal satisfaction criteria. AutoRELAX generates different solutions by making tradeoffs between minimizing the number of RELAXed goals and maximizing delivered functionality by reducing the number of adaptations triggered by minor and adverse environmental conditions. In a recent extension, AutoRELAX uses a stepwise adaptation of weights to balance these two competing concerns and thereby further improve the utility of AutoRELAX. We apply it to two industry-based applications involving network management and a robotic controller, respectively.

Keywords Uncertainty · Requirements engineering · Goal models · Genetic algorithms

1 Introduction

A dynamically adaptive system (DAS) may encounter operational contexts that may not have been fully understood or anticipated during requirements elicitation and system

Communicated by: Gordon Fraser and Jerffeson Teixeira de Souza

E. Fredericks (✉) · B. DeVries · B. H. C. Cheng
Department of Computer Science and Engineering, 3115 Engineering Building, 428 S. Shaw Lane, East Lansing, MI 48824-1226, USA
e-mail: freder99@cse.msu.edu

B. DeVries
e-mail: devri117@cse.msu.edu

B. H. C. Cheng
e-mail: chengb@cse.msu.edu

Published online: 12 March 2014

development. Contextual *uncertainty* (Cheng et al. 2009; Whittle et al. 2009; Baresi et al. 2010; Esfahani et al. 2011) may arise from a combination of unpredictable conditions that can limit a DAS's ability to adapt. Uncertainty may be mitigated through self-reconfiguration of a DAS, however introducing adaptation can incur costs in preparing and executing the adaptation. RELAX (Cheng et al. 2009; Whittle et al. 2009) is a requirements specification language for specifying and assessing sources of uncertainty in a DAS. RELAX can be used to make requirements more flexible and tolerant to different types of uncertainty, thereby potentially reducing the need for adaptation. Nevertheless, it can be a challenging task to manually identify during requirements analysis *which* requirements can be RELAXed, *how* to RELAX them, as well as determine their effects upon the behavior of the DAS. This paper presents an approach for automatically applying RELAX operators to a DAS goal model, thereby explicitly accounting for sources of system and environmental uncertainty, while decreasing the number of adaptations triggered by minor and transient contextual changes.

It is unlikely for a requirements engineer to identify all possible operational contexts that a DAS may encounter at run time during the requirements analysis and design phases of a DAS development cycle (Cheng et al. 2009; Whittle et al. 2009). A goal-oriented requirements modeling approach, such as KAOS (Dardenne et al. 1993; van Lamsweerde 2009), can be extended with RELAX operators defined in terms of fuzzy logic functions to mitigate contextual and environmental uncertainty (Cheng et al. 2009). This mitigation is handled by defining limits to which a goal may be temporarily unsatisfied while still delivering acceptable behavior. For example, the RELAX operator "AS EARLY AS POSSIBLE" can be used to make a rigid deadline for a given goal more flexible (Whittle et al. 2009) in a goal model. Unfortunately it can still be challenging for a requirements engineer to manually RELAX the requirements of a DAS since there may be numerous combinations of RELAXed goals for a given model, with each goal having a set of RELAX operators that can be applied. Furthermore, it is often difficult to evaluate how a particular goal RELAXation affects other requirements.

This paper describes AutoRELAX (Ramirez et al. 2012), an approach for automatically applying the RELAX process to goal-oriented requirements models for computing-based systems. Previously, Cheng et al. (2009) specified a manual process for identifying and modeling sources of environmental uncertainty in a goal model with RELAX operators, but they did not specify the process for selecting and applying fuzzy logic functions to increase the flexibility of a goal's satisfaction criteria. To this end, AutoRELAX automatically applies RELAX operators to a KAOS goal model in order to handle environmental uncertainty in a DAS. Specifically, AutoRELAX identifies goals that should be RELAXed, and specifies fuzzy logic function boundaries applied to each operator in order to weaken the constraints that define each goal's satisfaction criteria. Further, AutoRELAX generates a set of RELAXed goal models that enable a DAS to cope with varying system and environmental contexts while minimizing the number of adaptations required.

AutoRELAX applies a genetic algorithm (Holland 1992) as a search heuristic to explore all possible configurations of RELAXed goal models. In order to facilitate the search process, an executable specification of a DAS is used to measure how candidate RELAXed goal models are able to perform under system and environmental uncertainty. AutoRELAX applies two fitness sub-functions that evaluate the performance of the executable specification. These sub-functions reward candidates by measuring the satisfaction of functional requirements while minimizing the number of adaptations performed by the DAS and the number of applied goal RELAXations.

Since it is often difficult to determine an optimal weighting scheme for balancing fitness sub-functions, such as minimizing the number of RELAXed goals versus minimizing the number of triggered adaptations, this paper also extends earlier work by introducing a stepwise adaptation of weights (SAW) technique (Craenen and Eiben 2001; Eiben and van der Hauw 1997, 1998) that automatically adjusts the weights of each fitness sub-function in order to optimize the overall fitness value for a given set of environmental conditions. An algorithmic approach such as SAW can reduce the "trial and error" process when attempting to define an appropriate weighting scheme for elements of a fitness function. As the genetic algorithm determines the optimal goal model configuration, a meta-genetic algorithm, SAW, runs in tandem in order to determine an appropriate weighting scheme to yield a good fitness value.

We illustrate the use of AutoRELAX and SAW-AutoRELAX with two case studies: a remote data mirroring (RDM) network and an intelligent robotic vacuum. The RDM is an industrial application that handles dynamic reconfiguration of a remote data mirroring (RDM) network (Ji et al. 2003; Keeton et al. 2004). An RDM implementation improves data protection and availability by distributing data to physically remote servers, where data replication occurs when adverse events such as dropped messages, unforeseen message delays, or faults within a network link take place. Experimental results from the original AutoRELAX experiment (Ramirez et al. 2012) have shown that AutoRELAXed goal models compare favorably to those manually created by a requirements engineer. Furthermore, results also indicate that the number of adaptations and an adaptation's level of disruption may be reduced by RELAXing goal satisfaction criteria. We developed a second case study, a smart vacuum system (SVS), to demonstrate the application of AutoRELAX on a different application domain, namely an onboard control system. The SVS is an autonomous robotic vacuum tasked with cleaning a given environment and is modeled as a DAS via a set of configuration modes that are dynamically configurable at run time (Bencomo et al. 2010; Bencomo and Belaggoun 2013). Experimental results from the SVS indicate that AutoRELAX does indeed perform better than those that are manually RELAXed or unRELAXed. Finally, results from both studies also demonstrate that the overall fitness of an optimized goal model may be further improved by applying SAW to AutoRELAX.

The remainder of this paper is organized as follows. Section 2 provides background information on the RDM application used as a running example throughout the paper, as well as the enabling techniques used by AutoRELAX. Next, Section 3 presents the AutoRELAX process. The experimental evaluation is provided in Section 4; the specific application details and experimental evaluation of the SVS case study are also included. Section 5 discusses our experimental results, and then Section 6 overviews related work. Lastly, Section 7 summarizes findings and briefly discusses future directions.

2 Background

This section presents supporting background material for the AutoRELAX technique and its use. First, we introduce the remote data mirroring application domain used as a running example to illustrate the use of AutoRELAX. We then introduce enabling technologies used by AutoRELAX to address sources of environmental uncertainty, including goal-oriented requirements modeling, the RELAX requirements specification language, and genetic algorithms.

2.1 Remote Data Mirroring

Remote data mirroring (RDM) is a data protection technique that replicates copies of data at one or more secondary sites (Witty and Scott 2001; Ji et al. 2003; Keeton et al. 2004). By isolating data from failures that may affect the primary copy, RDM provides continuous access to important data across a network of data mirrors. In the event of a failure, recovery typically involves either a site failover to another data mirror or data reconstruction. For this work, the RDM network shall remain connected at all times while never exceeding an allocated monetary budget. In addition, the RDM network shall also replicate and distribute data as efficiently as possible by minimizing the amount of bandwidth consumed while ensuring data is neither lost nor corrupted.

At run time, the RDM network can self-reconfigure in response to network link failures, as well as dropped and delayed messages. In particular, each network link incurs an operational cost and has a measurable throughput, latency, and loss rate that collectively determine the overall RDM performance and reliability. In addition to reconfiguring the network topology by activating and deactivating operational network links, the RDM network can also modify its remote data mirroring protocols. Remote mirroring protocols, classified as synchronous or asynchronous, affect both network performance and reliability. In *synchronous propagation*, the secondary site receives and applies each write before it completes at the primary site. In batched *asynchronous propagation*, updates accumulate at the primary site and are periodically propagated to the secondary site, which then applies each batch atomically.

2.2 Goal-Oriented Requirements Modeling

As an integral part of software engineering, requirements engineering comprises activities such as eliciting, analyzing, and documenting objectives, constraints, and assumptions that a software system-to-be must meet to solve a specific stakeholder problem (van Lamsweerde 2009). In the 4-variable model (Jackson and Zave 1995), the system-to-be solves a problem within an organizational, technical, or physical world context. As a result, the system-to-be shares some phenomena with the problem world that defines the interface through which the system-to-be interacts with the world and its stakeholders.

Goal-oriented requirements engineering (GORE) extends the 4-variable model developed by Jackson with the concept of a goal that guides the elicitation and analysis of requirements based on the objectives of the system-to-be. In particular, a goal captures the intentions of a stakeholder on the system-to-be, as well as the assumptions and expectations upon its execution environment (van Lamsweerde 2009). A *functional* goal declares a service that the system-to-be must provide to its stakeholders, and a *non-functional* goal imposes a quality constraint or criterion upon the delivery of those services. A goal may also be classified either as a hard or soft goal. In addition, a functional goal may also be an invariant goal (denoted by keyword “Maintain”) that must always be satisfied by the system-to-be, or a non-invariant goal (denoted by keyword “Achieve”) that may become temporarily unsatisfied at run time. Moreover, goals can also be satisficed, or satisfied to a certain degree, possibly based on subjective preference (Chung et al. 2000).

The GORE process gradually decomposes high-level functional and non-functional goals into finer-grained subgoals (van Lamsweerde 2009). The semantics of goal decomposition are formally and graphically captured in a directed acyclic graph where each node represents a goal and each edge represents a goal refinement. Figure 1 presents a goal model for the RDM application in the KAOS modeling language (Dardenne et al. 1993; van Lamsweerde

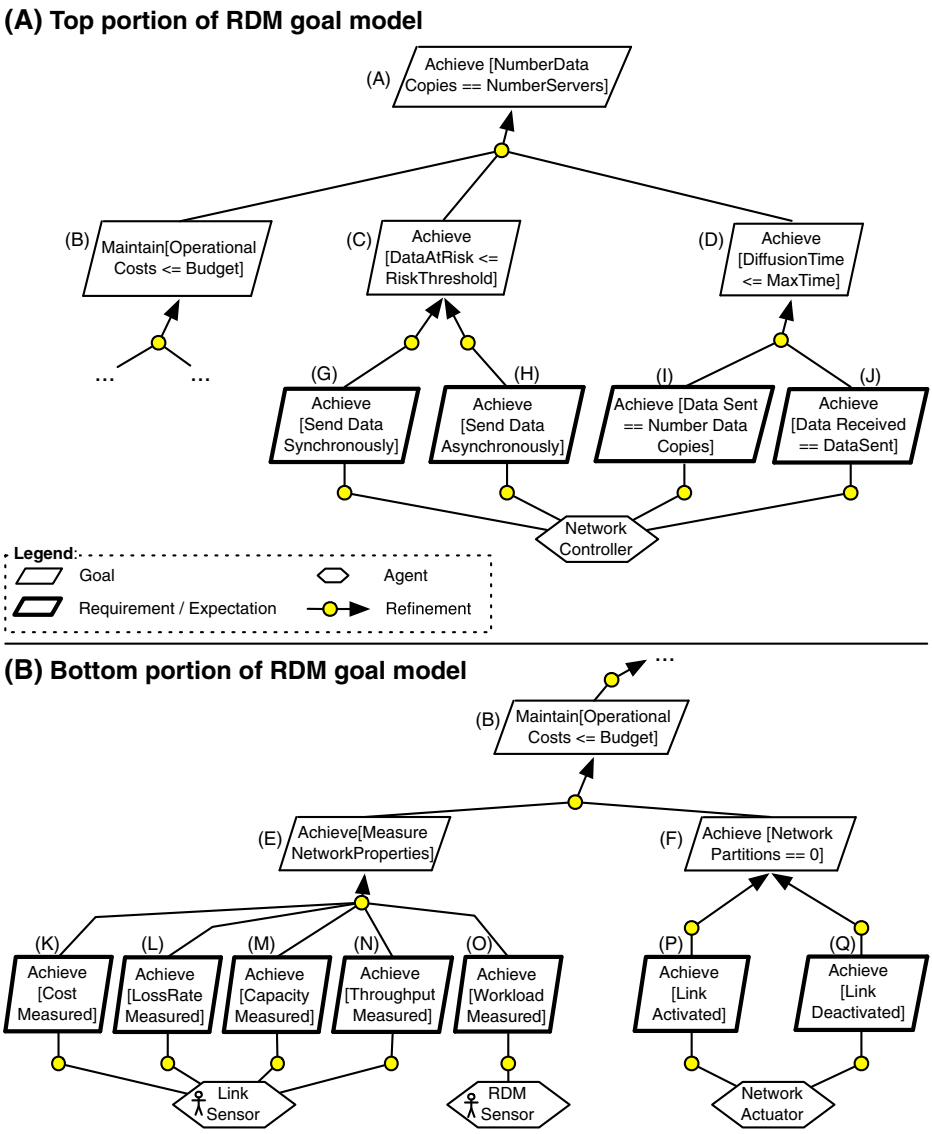

Fig. 1 KAOS goal model for the remote data mirroring application

2009). As this figure illustrates, KAOS depicts goals and refinements as parallelograms with directed arrows to point in the direction of their higher-level or parent goal. KAOS supports AND/OR refinements, where an AND-decomposed can only be satisfied if all of its subgoals are satisfied, and an OR-decomposed can be satisfied if at least one of its subgoals is satisfied. In general, AND-refinements capture milestones that must be performed in order to satisfy a single higher-level goal, whereas an OR-refinement provides alternative paths for achieving a given goal. For discussion and presentation purposes, Fig. 1 has been split into two portions, joined on Goal (B).

Goal decomposition continues until each goal is assigned to a single agent capable of fully achieving that goal. An agent is an active system component that restricts its behavior to fulfill leaf-goals in a goal model (van Lamsweerde 2009). Two different types of agents can fulfill goals: system and environmental agents. While a system agent is an automated component controllable by the system-to-be, an environmental agent is often a human being or some automated component that cannot be controlled by the system-to-be. As Fig. 1 illustrates, the KAOS modeling language depicts an agent with a hexagon connected to a bolded leaf-goal via an *assignment* link. KAOS also denotes the difference between a system and an environmental agent by including a stick figure icon within the hexagon of an environmental agent, such as agents "RDM Sensor" and "Link Sensor" in Fig. 1.

2.3 RELAX Specification Language

RELAX (Whittle et al. 2009) is a requirements specification language for identifying and assessing sources of environmental uncertainty in a DAS. In particular, RELAX *declaratively* specifies the sources and impacts of uncertainty at the shared boundary between the system-to-be and its execution environment (Jackson and Zave 1995). A requirements engineer organizes this information into three elements. ENV specifies environmental properties observable by the monitoring infrastructure of a DAS; MON specifies the elements that make up the DAS's monitoring infrastructure; and REL defines how to compute the values of ENV properties from MON elements.

RELAX operator semantics are defined in terms of fuzzy logic in order to specify the extent to which a non-invariant goal may become temporarily unsatisfied at run time (Whittle et al. 2009). Table 1 briefly describes the intent of each RELAX operator. For instance, if the Goal (F) in Fig. 1 is RELAXed into "Achieve[AS FEW Network Partitions AS POSSIBLE]", then this goal now states that it is *temporarily* acceptable for the network to become partitioned while data messages continue to be replicated and distributed. As this table also illustrates, fuzzy logic underpinnings (Whittle et al. 2009) of RELAX support both ordinal and temporal constraints upon goal satisfaction.

Figure 2 demonstrates the fuzzy logic membership functions that have been implemented for AutoRELAX. The AS EARLY AS POSSIBLE and AS FEW AS POSSIBLE use the left shoulder function in Fig. 2a, AS LATE AS POSSIBLE and AS MANY AS POSSIBLE

Table 1 RELAX operators (Whittle et al. 2009)

RELAX Operator	Informal Description
AS EARLY AS POSSIBLE ϕ	ϕ becomes true as close to the current time as possible.
AS LATE AS POSSIBLE ϕ	ϕ becomes true as close to time $t = \infty$ as possible.
AS CLOSE AS POSSIBLE TO [$frequency\ \phi$]	ϕ is true at periodic intervals as close to $frequency$ as possible.
AS FEW AS POSSIBLE ϕ	The value of a quantifiable property ϕ is as close as possible to 0.
AS MANY AS POSSIBLE ϕ	The value of a quantifiable property ϕ is maximized.
AS CLOSE AS POSSIBLE TO [$quantity\ \phi$]	The value of a quantifiable property ϕ approximates a desired target value $quantity$.

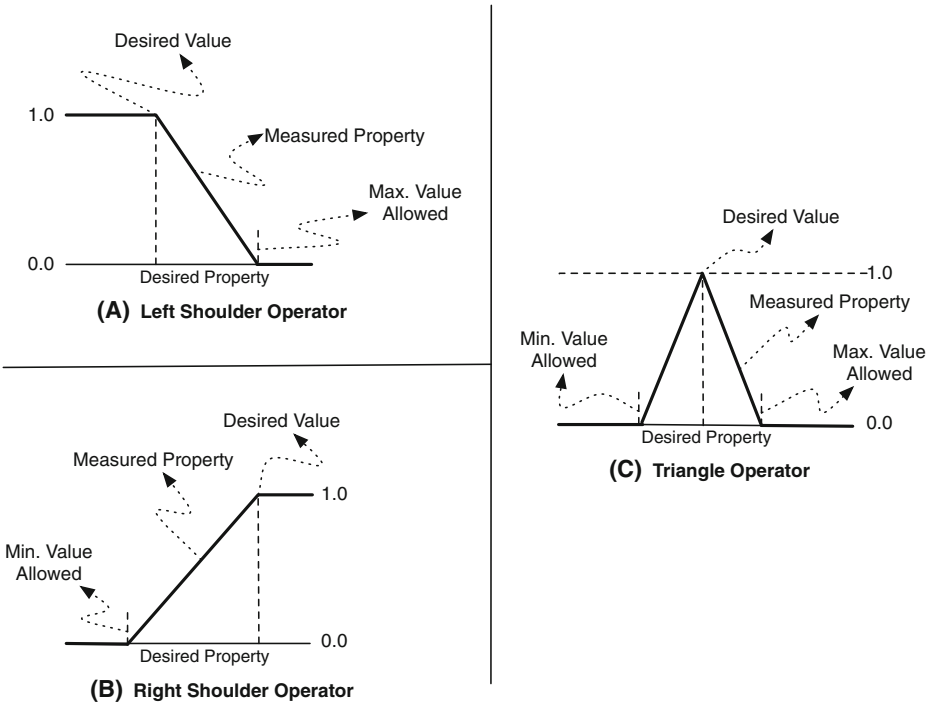

Fig. 2 Fuzzy logic membership functions

implement the right shoulder function from Fig. 2b, and the AS CLOSE AS POSSIBLE TO operators use the triangle shape function seen in Fig. 2c.

Cheng et al. (2009) proposed a manual approach for applying RELAX operators to non-invariant requirements in a KAOS goal model. First, a requirements engineer specifies the various ENV, MON, and REL elements necessary to define how environmental properties may be observed and quantified. Next, each goal in the model must be classified either as an invariant or non-invariant goal. For a non-invariant goal, a requirements engineer must then determine if environmental uncertainty may cause that goal to become unsatisfied. If so, then a requirements engineer must apply a corresponding RELAX operator to restrict how that goal may be temporarily violated. This last step must be manually repeated for each non-invariant goal in the goal model, leading to an explosion in possible combinations of goal RELAXations for a specified goal model. This paper, in contrast, describes a technique for automating this process and producing multiple RELAXed goal models, each of which provides a different goal model that captures different tradeoffs for satisfying functional and non-functional objectives.

2.4 Genetic Algorithms

A genetic algorithm (Holland 1992) is a stochastic, search-based heuristic that can be used to examine the space of possible solutions for complex optimization problems. Within a

genetic algorithm, a population of candidate solutions undergoes an evolutionary process in order to guide the search towards an optimal solution. To do so, a set of operations is executed until a specific number of iterations, or generations, are performed. These operations include generating a population of individuals, performing crossover and mutation operations, and evaluating each individual's fitness with respect to predefined fitness criteria. Each of these terms is described in turn.

Population generation At the start of a genetic algorithm, a completely randomized population of individuals is generated. With each successive generation, new individuals are created through genetic operations, such as crossover and mutation. The highest performing individuals may be retained at each generation in order to protect "successful" individuals.

Crossover and mutation The crossover and mutation operators generate new individuals from existing individuals at each generation. Crossover selects two candidate individuals and creates a new child individual by combining genes from each parent. The mutation operator selects an individual and then creates a new individual by randomly mutating a gene within its genome. Mutation aids the search process by exploring different areas of the solution space, helping to ensure that the algorithm does not end in local optima, and instead finds an optimal global solution.

Fitness evaluation Performance of individuals is evaluated at the end of each generation by applying a fitness function to each. The fitness function distills performance into a quantifiable value that is assigned to each individual. This value, or set of values, can then be used to identify individuals that perform better than others in deciding which to retain at the end of each generation.

3 Approach

AutoRELAX processes a goal model to generate multiple RELAXed goal models, each of which represents a balance of adaptation alternatives and different combinations of RELAXed operators mapped to different fuzzy logic functions. These AutoRELAX-generated goal models are best suited for handling different combinations of environmental conditions. In this section, we present the base AutoRELAX approach within the context of the RDM case study. The SVS case study, as described in Section 4.2, extends the base AutoRELAX approach. First, we introduce the AutoRELAX process and describe its key elements. Next, we state the assumptions, inputs, and expected outputs of AutoRELAX. We then describe how AutoRELAX may be applied to generate RELAXed goal models for a DAS that will execute in uncertain execution environments. Finally, we describe how the stepwise adaptation of weights (SAW) automatically balances competing concerns between minimizing the number of goal RELAXations and the number of triggered adaptations.

3.1 Assumptions, Inputs, and Expected Outputs

AutoRELAX requires three elements as input: a goal model of the DAS, a set of utility functions for requirements monitoring (Walsh et al. 2004; de Grandis and Valetto 2009; Ramirez and Cheng 2011), and an executable specification or prototype of the DAS that

can simulate different system and environmental conditions. Next, we briefly describe the purpose of each element and how they fit together within the AutoRELAX approach.

Goal Model AutoRELAX requires a goal model that captures the functional requirements that the DAS must satisfy, as well as their hierarchical relationships and constraints. Currently, AutoRELAX targets the KAOS goal modeling language (Dardenne et al. 1993; van Lamsweerde 2009) due to its emphasis on capturing functional requirements and its support for identifying and resolving obstacles. Moreover, since AutoRELAX can only introduce RELAX operators to non-invariant goals, each goal in the model must be identified either as an invariant or non-invariant goal.

Utility Functions Utility functions can be applied to monitor requirements satisfaction in a DAS at run time (Walsh et al. 2004; de Grandis and Valetto 2009; Ramirez and Cheng 2011). Each utility function comprises a mathematical relationship that maps gathered monitoring data to a scalar between zero and one, inclusive. For example, the satisfaction of Goal (B) in the RDM goal model in Fig. 1 can be evaluated with a utility function that returns 1.0 if operational costs have always been less than or equal to the allocated operational budget and 0.0 otherwise. AutoRELAX uses these utility functions, in conjunction with the executable prototype, to evaluate how goal RELAXations affect the behaviors of a DAS throughout a given simulation. These utility functions for requirements monitoring can be derived either manually (Walsh et al. 2004), automatically from a goal model (Ramirez and Cheng 2011), or statistically inferred based on observed DAS behaviors (de Grandis and Valetto 2009).

For this study, we manually derived the utility functions. More specifically, the goals specified in a KAOS goal model can be used to systematically derive utility functions using Athena, an approach developed by Ramirez and Cheng (2011). The Athena approach uses both the goal model and a mapping of the environmental conditions to the elements that monitor those conditions to derive three types of utility functions: state, metric, and fuzzy logic. The derivation process uses keywords in the individual goals to differentiate the type of utility function and the environmental condition mapping to identify observable goals, and a utility function can then be described based on the goal type. State-based utility functions monitor goals that are boolean in nature, and metric- and fuzzy logic-based utility functions monitor the *satisficement* of goals. The RELAX operator is used to specify the appropriate membership function for fuzzy logic goals in deriving the appropriate utility function. The resulting utility functions can then be used to detect run-time requirements violations. Furthermore, monitoring data provided by environmental agents are used to determine the *satisficement* of metric and fuzzy utility functions.

Executable Specification AutoRELAX uses an executable specification of the DAS, such as a simulation or prototype, to evaluate the effects of different system and environmental conditions upon the behavior of the DAS. To this end, a requirements engineer first identifies possible sources of environmental uncertainty that can affect the DAS at run time. The executable specification is then initialized based on the identified sources of uncertainty. For example, our RDM simulation can be initialized according to the likelihood of a network link failing at run time. Changing either the sources of uncertainty, their likelihood, or impact will ideally exercise the adaptive capabilities of the DAS, thereby leading to different types of RELAXed goal models. Note that this executable specification must incorporate the utility functions for requirements monitoring such that AutoRELAX can trace how well the DAS satisfied each goal throughout the simulation in response to different conditions.

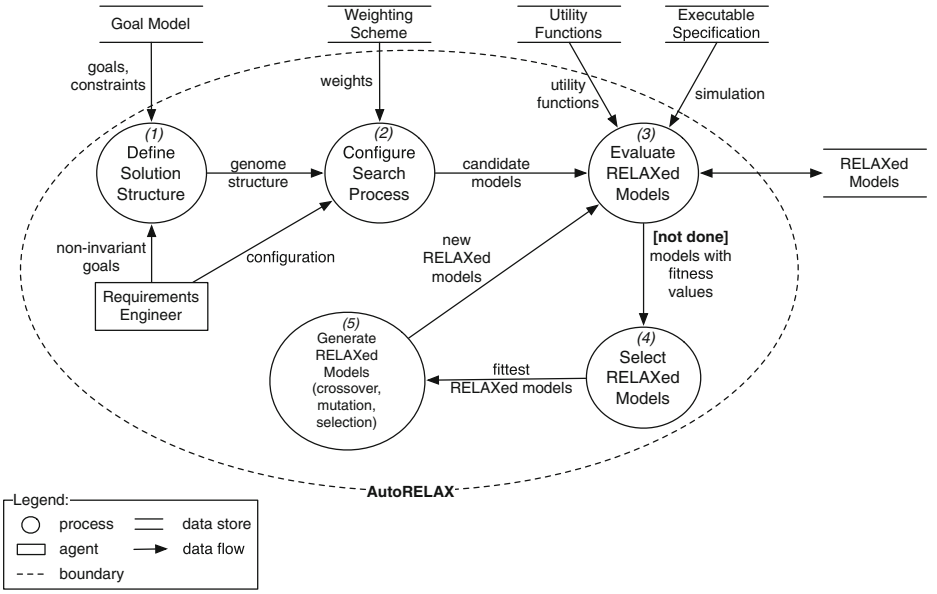

Fig. 3 DFD diagram of AutoRELAX process

3.2 AutoRELAX Process

Figure 3 presents a data flow diagram (DFD) that overviews the AutoRELAX process. We present each step in detail next.

(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 4a 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.

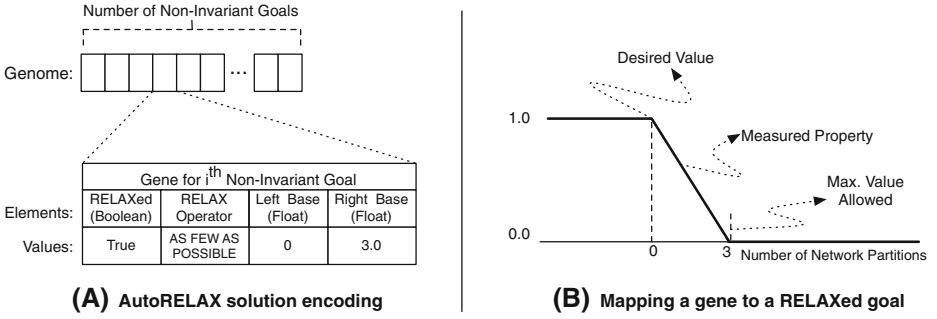

Fig. 4 Encoding a candidate solution in AutoRELAX

Figure 4b shows how each gene is mapped to a corresponding fuzzy logic function that can evaluate the satisfaction of a goal. In this example, the unRELAXed satisfaction criteria for Goal (F) in Fig. 1, as defined by its utility function, returns 1.0 if the network is connected (i.e., number of network partitions equals zero) and 0.0 otherwise. However, as long as the network partition is transient (i.e. minor and temporary), then it may be possible to continue diffusing data amongst connected data mirrors while the network topology is reconfigured. As the bolded line in Fig. 4b depicts, this goal can be made more flexible by introducing the "AS FEW AS POSSIBLE" ordinal RELAX operator that maps to a *left shoulder*-shaped fuzzy logic function (Whittle et al. 2009). For this RELAXed goal, the apex is centered upon the ideal value of a system or environmental property, zero network partitions in this case, and the downward slope from the apex to the right endpoint reflects values that are not ideal but might be temporarily tolerated at run time.

(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 criterion, such as a maximum amount of generations, specifies when AutoRELAX will stop searching for new solutions and output the resulting RELAXed goal models. Guidelines for defining these parameters are problem dependant and should be explored as a part of future work. Furthermore, the requirements engineer must also configure a set of weights to be used within the fitness calculation. These weights are used to define the relative importance of each fitness sub-function.

(3) Evaluate RELAXed Models. RELAXed goal models are evaluated based on their performance during simulation, with a strong emphasis towards minimizing the number of adaptations and number of RELAXed goals. 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 runs 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 favor RELAXed goal models that minimize the number of RELAXed goals and minimize the number of adaptations.

The first fitness sub-function, FF_{nrg}, rewards candidate solutions that minimize the number of RELAXed goals in order to limit introducing an unnecessary amount of flexibility into a goal model:

$$FF_{nrg} = 1.0 - \left(\frac{|relaxed|}{|Goals_{non-invariant}|} \right),$$

where $|relaxed|$ and $|Goals_{non-invariant}|$ are the number of RELAXed and non-invariant goals in the DAS goal model, respectively. This fitness sub-function explicitly discourages AutoRELAX from unnecessarily introducing RELAX operators needlessly.

The second fitness sub-function, FF_{na}, rewards candidate solutions that minimize the number of adaptations performed by the DAS in response to minor and transient environmental conditions, in order to reduce overhead incurred on performance and cost:

$$FF_{na} = 1.0 - \left(\frac{|adaptations|}{|faults|} \right),$$

where $|adaptations|$ represents the total number of adaptations performed by the DAS, and $|faults|$ measures the total number of adverse environmental conditions introduced throughout a simulation, thereby reducing the number of passive and quiescent components at run time (Kramer and Magee 1990).[1] 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. The operational cost associated with each adaptation can vary due to the number of nodes that have to be put into quiescent or passive states in order to ensure system consistency during a particular adaptation process. Reducing the number of passive and quiescent components minimizes the impact of an adaptation by enabling the DAS to continue providing important functionality to its stakeholders even during the reconfiguration process. Using RELAX operators increases a system's flexibility and ability to tolerate uncertainty without adaptations, and therefore avoiding the need to move system components to passive or quiescent states.

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

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

where the α_{nrg} and α_{na} coefficients reflect the relative importance of each fitness sub-function, the sum of which must equal 1.0.[2] 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 that AutoRELAX only outputs *viable* RELAXed goal models that satisfy all invariant goals. While a linear weighted sum can provide a simpler interface for balancing competing concerns in a large search space, multi-objective optimization (Deb et al. 2002) provides an advanced technique for finding a set of optimal solutions while also balancing differing objectives. We intend to explore multi-objective optimization as applied to AutoRELAX as part of a future work.

(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 (Holland 1992), a technique that randomly selects groups of 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 Fig. 5a 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 that lie between the boundaries of two randomly chosen indices. In contrast, Fig. 5b 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

[1] Component refers to an identifiable executable element, such as a software component or network node.

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

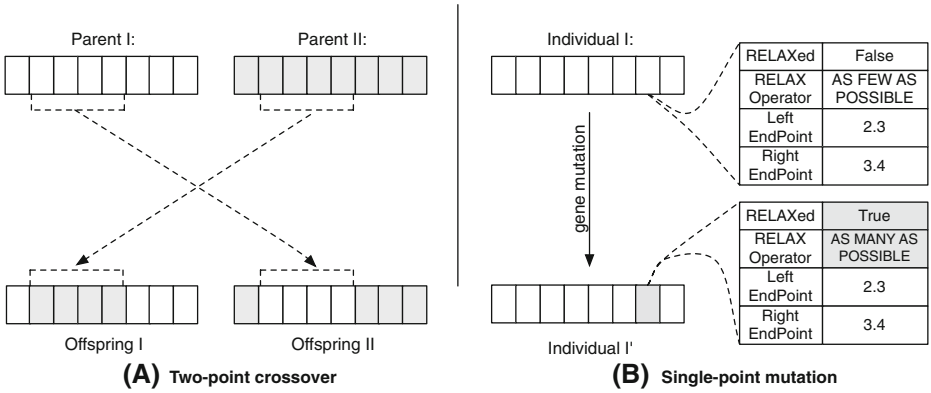

Fig. 5 Generating new RELAXed goal models with crossover and mutation operators

with the "AS MANY AS POSSIBLE" RELAX operator. Mutation does not affect the defined fuzzy logic boundary values as it would alter the meaning of a fully satisficed goal. In this manner, while crossover attempts to construct better solutions by combining good elements from existing RELAXed goal models, mutation introduces diverse goal RELAXations that might not be obtainable via the crossover operator alone.

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.

3.3 SAW Integration

This paper extends the original AutoRELAX (Ramirez et al. 2012) with the SAW optimization (van der Hauw 1996; Eiben et al. 1998) in order to explore the space of possible fitness sub-function weighting schemes for finding an optimal combination in a given environment. Specifically, in determining the overall fitness of an individual within a population, it is often necessary to consider several factors when distilling an individual's performance into a single value. This process is typically accomplished with *weighted fitness sub-functions* that each measure specific aspects of an individual and sum together the results that determine overall fitness. Weighting may be introduced in order to give preference to specific sub-functions, where the sum of all weights should equal one.

The weight assigned to each sub-function is typically determined by combinations of domain knowledge, trial and error, and, if available, empirical evidence. When little empirical evidence is available or execution conditions are unstable, it is useful to have an algorithmic approach to determine optimal values for these weights. SAW gradually updates the weighting scheme over the course of an evolutionary algorithm to explore how various combinations of weights will affect overall fitness.

We have implemented the *online* method of SAW optimization as described in the original SAW paper (van der Hauw 1996) as it provides a fast, efficient algorithm for determining optimal weight combinations and can easily be implemented within existing frameworks. In this approach, the weights can be seeded with values identified by a requirements engineer. Feedback from the application is examined periodically throughout the simulation.

The fitness sub-functions whose value is the least fit when compared to the other fitness sub-functions have their associated weight increased to provide more importance to that specific fitness sub-function, and the remaining weights are then all normalized to ensure that all sum to 1.0.

We augment Step (3) of the AutoRELAX process shown in Fig. 3 to use the online implementation of the SAW technique. The results of the fitness sub-function calculations are examined periodically.[3] The weight associated with a fitness sub-function that is performing the poorest is incremented, thus emphasizing that specific sub-function in the overall fitness calculation. In this regard, AutoRELAXed goal models are guided towards mitigating the uncertainties calculated by the offending fitness sub-function.

4 Experimental Results

This section presents experimental results obtained by applying AutoRELAX to the RDM and SVS applications. First, we introduce each case study's simulation environment and its configuration, including sources of uncertainty and possible reconfigurations that address adverse system and environmental conditions. Next, we present experiments that compare and evaluate unRELAXed, manually RELAXed, and AutoRELAXed goal models. Following, we used SAW to determine the weighting scheme of the fitness sub-functions that further improve the AutoRELAX results. For each experiment, we state and evaluate experimental hypotheses, as well as discuss the resulting goal models.

4.1 RDM Study

This section describes the experimental setup and results from the RDM case study.

4.1.1 RDM Implementation

We implemented the RDM network as a completely connected graph. For this work, the network comprises 25 data mirrors and 300 network links that can be activated to distribute data messages. The performance attributes for each data mirror and network link were randomly generated based on previous experimental specifications by Keeton et al. (2004). Moreover, each RDM network simulation executes for 150 time steps during which 20 data messages are randomly inserted at different data mirrors and time steps. Those RDMs are then responsible for distributing new data messages to all other data mirrors within the network.

The RDM simulation implementation can be configured to introduce various forms of system and environmental uncertainty. Within this simulation, system uncertainty includes noisy and unreliable monitoring data produced by potentially unreliable, imprecise, and inaccurate sensors within the monitoring infrastructure of the RDM network. Likewise, environmental uncertainty includes unpredictable network link failures, as well as dropped and delayed data messages.

The RDM can self-adapt to continue delivering critical functionality to its stakeholders if these sources of uncertainty manifest themselves. To this end, the RDM network monitors the utility values of each goal that reflect their satisfaction at each simulation time step.

[3]For this article, we examined the fitness sub-function results every 5^{th} generation.

Unsatisfied goals serve as adaptation triggers that cause the RDM to re-evaluate its current configuration and goal realization strategy. If an adaptation is warranted, then the RDM network invokes the Plato reconfiguration engine (Ramirez et al. 2009, 2010) to select a more suitable network topology and propagation parameters. Plato explicitly searches for safe adaptation paths for a DAS to ensure that configuration changes do not negatively affect overall performance. Moreover, each data mirror implements the dynamic change management (DCM) protocol previously introduced by Kramer and Magee (1990) to safely reach the selected target configuration while preserving consistency. DCM provides an approach for incorporating run-time changes into a DAS safely by separating structural from application concerns.

For the following experiments, we reuse the fitness functions previously presented in Section 3. Reusing these fitness functions for comparing goal models serves two key purposes. First, this objective evaluation criteria enables us to assess the benefits of RELAXing a goal model to address different sources of environmental uncertainty. Second, these fitness functions enable us to demonstrate whether AutoRELAX is capable of generating viable RELAXed models that are as good, if not better, than those manually created by a requirements engineer. Since there is a randomized component in both AutoRELAX and the RDM network simulation, we conducted 50 trials of each experiment and, where applicable, plot mean values with corresponding error bars.

4.1.2 RDM Uncertainty

This experiment evaluates and compares the resulting RELAXed goal models produced by AutoRELAX with two different goal models of the same RDM application: the unRELAXed goal model previously shown in Fig. 1 and a goal model that was manually RELAXed. To create the manually RELAXed goal model, a small group of engineers independently created first drafts of the goal model. They then collaborated to combine their respective goal models to create a single "common goal model" that they iteratively refined to reflect their cumulative expertise and domain knowledge. The following four different goal RELAXations were introduced into the manually RELAX goal model: Goal (C) was RELAXed to allow larger exposures to data loss, Goal (D) was RELAXed to add temporal flexibility when diffusing data, Goal (F) was RELAXed to allow up to three simultaneous network partitions, and Goal (I) was RELAXed to tolerate dropped data messages. The RELAX operators and fuzzy logic boundary functions are summarized in Table 2.

Table 2 Summary of manually RELAXed goals for the RDM application

Goal	Description	RELAX operator	Boundaries
C	Achieve [$DataAtRisk$] <= [$RiskThreshold$]	AS FEW AS POSSIBLE	±5.0 % risk
D	Achieve [$DiffusionTime$] <= [$MaxTime$]	AS FEW AS POSSIBLE	±5.0s
F	Achieve [$NetworkPartitions$] == 0	AS CLOSE AS POSSIBLE TO [quantity]	±1.0 partitions
I	Achieve [$DataSent$] == [$NumberofDataCopies$]	AS CLOSE AS POSSIBLE TO [quantity]	±5.0 copies

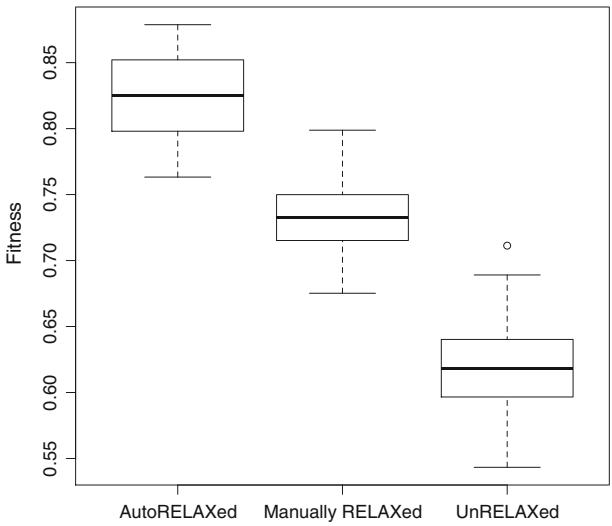

Fig. 6 Fitness values comparison between RELAXed and unRELAXed goal models for the RDM

For this experiment, we define the first null hypothesis, $H1_0$, as follows: "there is no difference in fitnesses achieved by a RELAXed and an unRELAXed goal model." In addition, we also define a second null hypothesis, $H2_0$, as follows: "there is no difference in fitness values between RELAXed goal models generated by AutoRELAX and those manually created by a requirements engineer." Also note that for this experiment we used a weighting scheme for fitness sub-functions based on previous empirical results. Specifically, we set α_{nrg} to 0.3 and α_{na} to 0.7, thus emphasizing the minimization of the number of needed adaptations.

Not all adaptations achieve the same fitness. As such, Fig. 6 presents three sets of box plots that capture the breakdown of fitness by 1) AutoRELAX-generated models, 2) a goal model manually RELAXed by a requirements engineer, and 3) an unRELAXed goal model, respectively (the small circle above the unRELAXed box plot indicates an outlier). Each of these experiments were performed with 50 replicates. As these box plots illustrate, despite the fitness advantage unRELAXed goal models obtain as they have no goal RELAXations (see Section 3, FF_{nrg}), RELAXed goal models achieved statistically significant higher overall fitness values than unRELAXed goal models ($p < 0.001$, Welch Two Sample t-test).[4] These results enable us to reject our first null hypothesis, $H1_0$, as well as conclude that RELAX does reduce the number of adaptations when addressing system and environmental uncertainty.

The box plots in Fig. 6 also demonstrate that AutoRELAX-generated RELAXed goal models achieved statistically significant higher fitness values than those manually RELAXed 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 7 presents three sets of box plots that capture the adaptation costs incurred by 1) AutoRELAX-generated models, 2) a manually created RELAXed goal model, and 3)

[4]Because the datasets used in this paper are drawn from a normal distribution, are of unequal sample size, and have unequal variance, we use the Welch t-test to test for statistical significance.

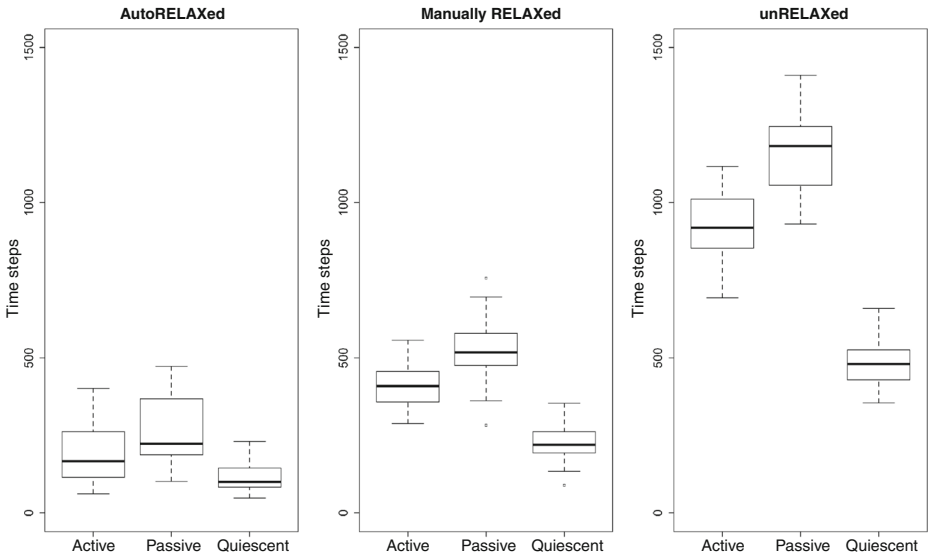

Fig. 7 Adaptation costs comparison between RELAXed and unRELAXed goal models for the RDM

an unRELAXed goal model, respectively. Specifically, each set of box plots measures the amount of time that components in the RDM network spent in active, passive, and quiescent modes *during* reconfigurations (these plots do not include time outside of a reconfiguration). As Fig. 7 shows, option (1) (AutoRELAX) is preferable because it has less negative impact on overall system functionality. Specifically, by carefully weakening the satisfaction criteria of non-invariant goals (i.e. making the satisfaction criteria more flexible), the number of adaptations decrease and so does the cumulative amount of time components spend in passive and quiescent modes during a reconfiguration.

Within the RDM application, adaptations are particularly disruptive because data mirrors may be unable to continue the data replication and distribution process required to ultimately satisfy Goal (A). In particular, self-reconfigurations in the RDM network attempt to improve the satisfaction of Goals (C), (D), and (F) by reconnecting a partitioned network and balancing performance and reliability data propagation concerns in response to adverse system and environmental conditions. Nevertheless, adaptations can directly hinder the data diffusion task by placing data mirrors in passive and quiescent states. While active data mirrors may initiate and service transaction requests, passive data mirrors are unable to distribute new data, and quiescent data mirrors are unable to receive, replicate, and distribute new data across the network. In this manner, an adaptation can *temporarily* restrict the process of message diffusion such that Goals (C), (D), (G), (H), (I), and (J) are unsatisfied until the reconfiguration completes. For this reason, limiting the number of passive and quiescent data mirrors can improve the amount of functionality delivered to stakeholders and therefore better satisfy these goals.

Both Figs. 6 and 7 show that AutoRELAX is able to generate RELAXed goal models that perform better than manually RELAXed goal models. Examining the AutoRELAX-generated 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), and (I), 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.

Figure 8 provides additional information about the types of goal models generated by AutoRELAX for varying degrees of uncertainty. Specifically, environments with a low degree of uncertainty specify a very small probability for failures that may occur, including failed network links and dropped messages. Conversely, environments with high uncertainty can experience a very high amount of possible failures within the system and environment. In particular, Fig. 8 plots the *sorted* number of RELAXed goals per trial where the first environment has a low degree of uncertainty and the second environment has a high degree of uncertainty. Each point represents the mean number of RELAXed goals applied to goal models generated throughout each evolutionary execution. As this figure illustrates, in 49 out of 50 trials, AutoRELAX introduced a greater than or equal number of RELAX operators to the goal model subjected to a higher degree of uncertainty than to the goal model subjected to a lower degree of uncertainty. In contrast, in 29 out of 50 trials, AutoRELAX introduced either zero or one RELAX operator to the goal model subjected to a lower degree of uncertainty. Moreover, the positive correlation between the two curves suggests that AutoRELAX gradually introduces goal RELAXations in response to increasing degrees of system and environmental uncertainty. This plot suggests that AutoRELAX introduces flexibility in how a goal can be satisficed only if necessary.

Lastly, Fig. 9 presents a complementary view of the plot shown in Fig. 8. This plot shows the effects of goal RELAXation upon the number of adaptations triggered during each

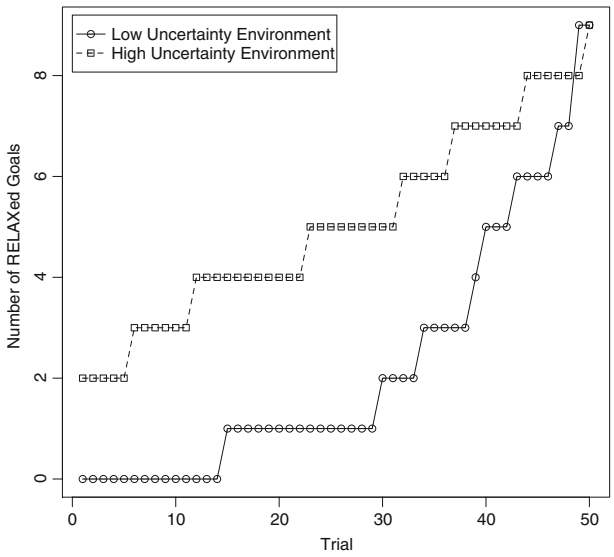

Fig. 8 Mean number of RELAXed goals for varying degrees of system and environmental uncertainty for the RDM

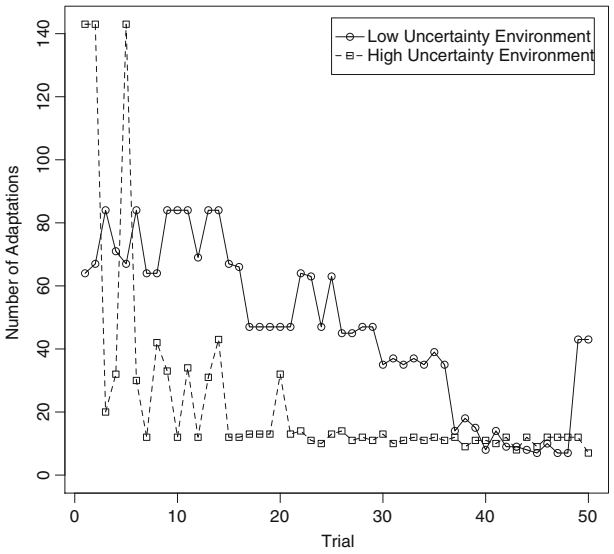

Fig. 9 Mean number of adaptations triggered, sorted by number of RELAXed goals for the RDM

experiment trial, sorted again in the x-axis by the number of goal RELAXations introduced in the goal model for each trial. Figure 9 illustrates that as the number of goal RELAXations increased, the number of adaptations triggered due to system and environmental uncertainty decreased. These observations confirm that goal RELAXation can prevent possibly unnecessary run-time self-reconfigurations in response to minor and transient environmental conditions. In particular, AutoRELAX introduced RELAX operators that enabled the RDM network to tolerate minor variance in the set of encountered operational contexts. Conversely, as the variance in operational contexts increased, more adaptations were required for the RDM network to continue satisfying its requirements.

4.1.3 Dynamic Weight Adjustment

In this experiment, we compare and evaluate the effectiveness of AutoRELAX at optimizing the fitness sub-function weighting scheme with and without the SAW technique. We first executed AutoRELAX in 50 separate environmental configurations without using SAW, where we set default weights for the fitness function elements. Next, we ran SAW-AutoRELAX (i.e., AutoRELAX with the SAW optimization enabled) to explore how varying combinations of weights affected the overall fitness performance. Subsequently, we also examined how SAW- AutoRELAX can optimize a weighting scheme and thereby impact overall performance within a single environmental configuration.

As with the AutoRELAX experiment, we defined two null hypotheses. The first null hypothesis, $H1_0$, states: "there is no difference between an automatically RELAXed goal model with static weights and an automatically RELAXed goal model that uses a SAW-optimized weighting scheme across different environments." The second null hypothesis, $H2_0$, states: "there is no difference between RELAXed goal models with optimized and unoptimized weights in a static environment."

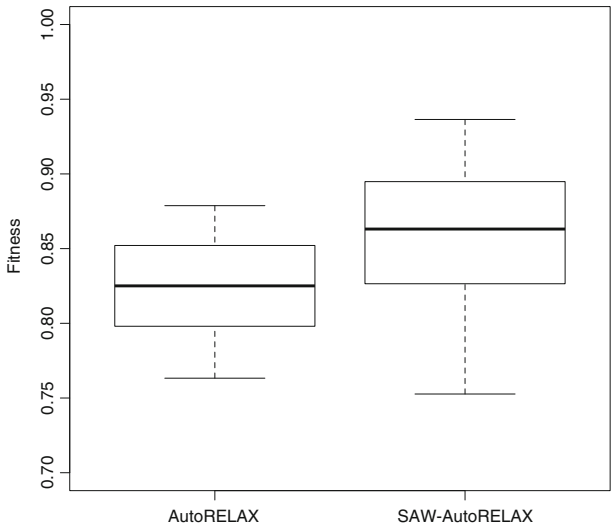

Fig. 10 Fitness values comparison between AutoRELAXed and SAW-optimized AutoRELAXed goal models for the RDM

In order to emphasize the importance of reducing the number of performed adaptations, the RDM weights were set as follows:

$$\alpha_{nrg} = 0.3,$$
$$\alpha_{na} = 0.7$$

Figure 10 presents two box plots with fitness values obtained from AutoRELAXed models and from SAW-optimized AutoRELAXed models, respectively. As these plots show,

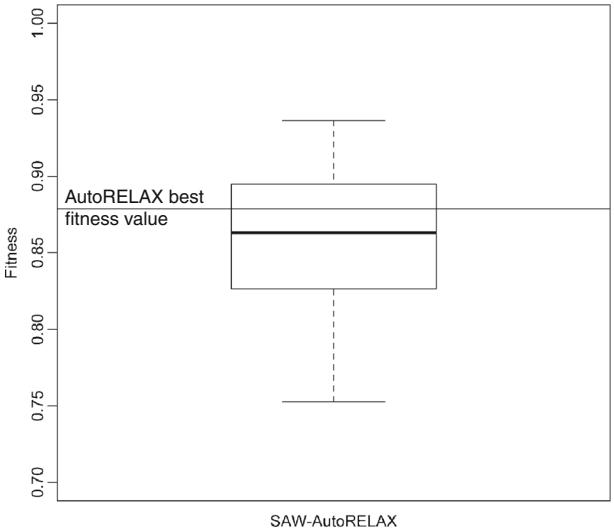

Fig. 11 Fitness values of SAW-optimized AutoRELAXed goal models in a single environment

AutoRELAXed goal models that have a weighting scheme optimized for their given environment tend to statistically perform better than those with a static weighting scheme ($p < 0.001$, Welch Two Sample t-test). These results enable us to reject $H1_0$, concluding that the weighting scheme has a direct impact on the overall fitness of an AutoRELAXed goal model.

SAW-AutoRELAX produced an interesting result for a small subset of environments. In these, the weighting scheme converged to $\alpha_{nrg} = 1.0$ and $\alpha_{na} = 0.0$, which ensures that the number of RELAXations applied to the goal model is minimized. This result indicates that while RELAX is a useful method for mitigating uncertainty, it is not always necessary to introduce RELAXations when the existing goal model can sufficiently handle the current system and environmental configurations. In comparing the two populations, we can also reject $H2_0$ ($p < 0.001$, Welch Two Sample t-test), concluding that SAW-optimized goal models tend to perform better than non-optimized goal models in the same environment.

Figure 11 exhibits a box plot of the fitness values from SAW- AutoRELAXed goal models in a single environment, with a randomly selected amount of uncertainty, over 50 separate trials. The solid horizontal line represents the best fitness that AutoRELAX was able to achieve with no optimization applied in the tested environment. The results show that, while AutoRELAX by itself was able to generate a higher fitness than the median SAW- AutoRELAX fitness, optimizing with SAW enabled the achievement of 100 % fitness (as shown in Fig. 12). SAW also generated fitness values less than AutoRELAX, implying that not all weighting schemes are appropriate for a given environment. Figure 12 also illustrates the range of SAW fitnesses in a single environment. Each box plot represents the fitnesses of the first 25 trials in the single environment experiment, again with the best AutoRELAX fitness represented by a horizontal line.

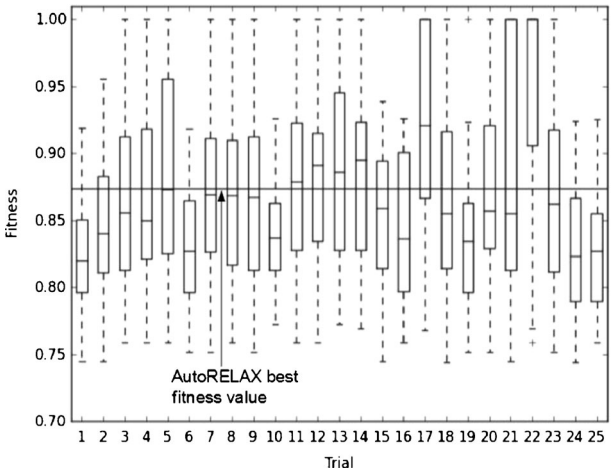

Fig. 12 Comparison of SAW-optimized fitness values from different trials in a single environment

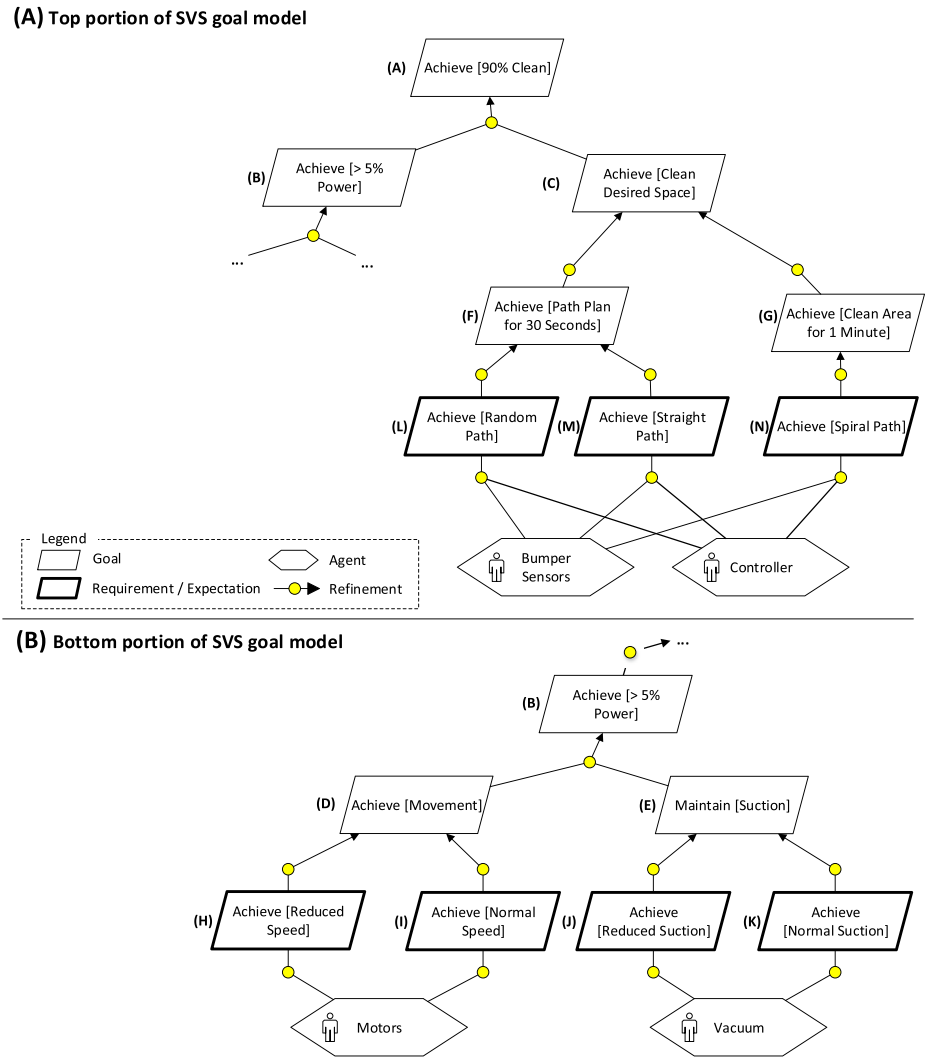

Fig. 13 KAOS goal model for the smart vacuum system application

4.2 SVS Study

SVSs have been available in the consumer market for quite some time, with the most notable example being iRobot's Roomba.[5] An SVS is capable of cleaning a desired space by utilizing sensor inputs to balance path planning and power conservation. Available sensors may include bumper sensors and motor sensors. Bumper sensors provide feedback when the robot has collided with an object, cliff sensors prevent the robot from falling down stairs, and motor sensors provide information on the individual velocities and power modes for

[5] See http://www.irobot.com/

the wheels and suction units. A controller processes the sensor data and then determines an optimal path, as well as conserving battery power as necessary.

Given its relative level of sophistication, an SVS can also be modeled as an adaptive system (Bencomo et al. 2010; Bencomo and Belaggoun 2013). The SVS can self-reconfigure at run time by performing mode changes (Neema et al. 1999) that are dynamically configured at run time. The possible modes allow the SVS to select an optimal configuration that can properly mitigate uncertainties within the system and environment, such as noisy sensor data or the amount and location of dirt spread throughout the room. Possible modes include various pathfinding algorithms, reduced power consumption modes, and obstacle avoidance measures, each of which have parameters with a range of possible values. Therefore, there are numerous possible combinations of values for each of the different models. Figure 13 presents a KAOS goal model for the SVS application, split on Goal (B) for presentation purposes. The remainder of this section describes the SVS implementation, experimental setup, and results from the case study.

4.2.1 SVS Implementation

We implemented the SVS as a fully autonomous, differential wheeled robot within the Open Dynamics Engine physics framework[6] using the specification from the iRobot Roomba vacuum systems as the basis for the real-world functionality. The SVS encompasses 7 independent touch sensors that provide feedback to its controller regarding contact with solid objects, 2 wheel velocity sensors, and a vacuum sensor that provides feedback on suction power. For this experiment, the SVS was placed in a square room surrounded by four walls with no obstacles to impede its path. Furthermore, dirt objects were randomly placed to cover 90 % of the room's available floor space. Figure 14 gives a snapshot of the SVS simulation environment, including the robotic vacuum, bounding walls, and randomly placed dirt objects.

The SVS simulation can also be configured to provide system and environmental uncertainty. In this case, system uncertainty comprises noisy data produced by the various sensors on the SVS, including touch and wheel velocity sensors. Environmental uncertainty comprises the amount of dirt spread throughout the room and the amount of power consumed by the SVS while attempting to navigate the room.

The SVS self-adapts as necessary to continually provide satisfactory performance. Similar to the RDM, the SVS monitors the utility values of each goal at each time step and performs adaptations as necessary (as with the RDM study, we manually developed the utility functions, following the Athena approach (Ramirez and Cheng 2011). To accomplish this task, the SVS can select a new mode of operation that is more suitable to the current system and environmental conditions, such as reduced power modes or various path plans.

4.2.2 SVS Uncertainty

The SVS case study has the same experimental setup as that used for the RDM in that the experiment evaluates and compares unRELAXed, manually RELAXed, and AutoRELAXed goal models. The unRELAXed goal model also had 4 RELAXations introduced: Goal (A) was RELAXed to allow the amount of remaining dirt to be greater than specified, Goal (B) was RELAXed to allow flexibility in remaining battery power, and Goals (F) and (G) were

[6]See http://www.ode.org.

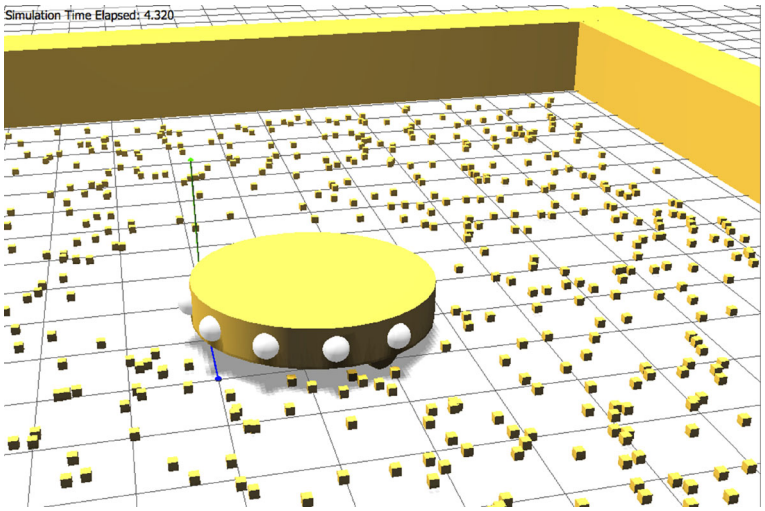

Fig. 14 SVS simulation

RELAXed in order to provide the SVS an adjustable amount of time for the execution of each path plan. Table 3 summarizes the RELAX operators and fuzzy logic boundaries for the manually RELAXed goal model.

The AutoRELAX fitness function required modification to support the SVS application. Two additional sub-functions were required, maximization of room cleanliness (FF_{clean}) and minimization of faults (FF_{faults}). The first fitness sub-function rewards candidate solutions that maximize the amount of dirt that has been removed by the SVS:

$$FF_{clean} = \left(\frac{|dirt\ removed|}{|amount\ of\ dirt|} \right),$$

and where $|dirt\ \ removed|$ represents the amount of dirt cleaned by the SVS and $|amount\ of\ dirt|$ represents the total amount of dirt introduced at the start of the simulation. The second additional fitness sub-function rewards candidates that have minimized the amount of faults encountered, including running out of battery power or having no forward velocity, indicating the SVS may be trapped in a corner:

$$FF_{faults} = \left(\frac{1.0}{|faults|} \right),$$

Table 3 Summary of manually RELAXed goals for the SVS application

Goal	Description	RELAX Operator	Boundaries
A	Achieve [90 % Clean]	AS MANY AS POSSIBLE	±5.0 %
B	Achieve [> 5 % Power]	AS MANY AS POSSIBLE	±5.0 %
F	Achieve [path plan for 30s]	AS CLOSE AS POSSIBLE TO [quantity]	±15.0 seconds
G	Achieve [clean area for 1m]	AS CLOSE AS POSSIBLE TO [quantity]	±15.0 seconds

where $|faults|$ represents the total amount of faults encountered by the SVS. These additional fitness sub-functions can be combined with the original AutoRELAX sub-functions in a linear weighted sum:

$$Fitness\ Value = \begin{cases} [\alpha_{nrg} * FF_{nrg}] & + \quad \% \text{ number of RELAXed goals} \\ [\alpha_{na} * FF_{na}] & + \quad \% \text{ number of adaptations} \\ [\alpha_{clean} * FF_{clean}] & + \quad \% \text{ percentage of dirt removed} \\ [\alpha_{faults} * FF_{faults}] & + \quad \% \text{ number of faults} \end{cases}$$

where the α_{clean} and α_{faults} coefficients reflect the relative importance of each additional fitness sub-function, and the sum of all coefficients must equal 1.0. Specifically, we set α_{nrg} to 0.1, α_{na} to 0.4, α_{clean} to 0.2, and α_{faults} to 0.3. As with the RDM study, hypothesis $H1_0$ states that "there is no difference in fitnesses achieved by a RELAXed and an unRELAXed goal model", and hypothesis $H2_0$ states that "there is no difference in fitness values between RELAXed goal models generated by AutoRELAX and those manually created by a requirements engineer."

Figure 15 presents three sets of box plots that demonstrate the overall fitness calculated for 1) AutoRELAX-generated models, 2) a goal model manually RELAXed by a requirements engineer, and 3) an unRELAXed goal model, respectively, with each performed with 50 replicates. As the results demonstrate, the RELAXed goal models achieved statistically significant higher fitness values than the unRELAXed goal models ($p < 0.001$, Welch Two Sample t-test). These results enable us to reject the first null hypothesis, $H1_0$, as well as conclude that RELAX can reduce the number of faults encountered when mitigating system and environmental uncertainty. The plots in Fig. 15 also show that AutoRELAX-generated goal models achieved statistically significant higher fitness values than those that were manually

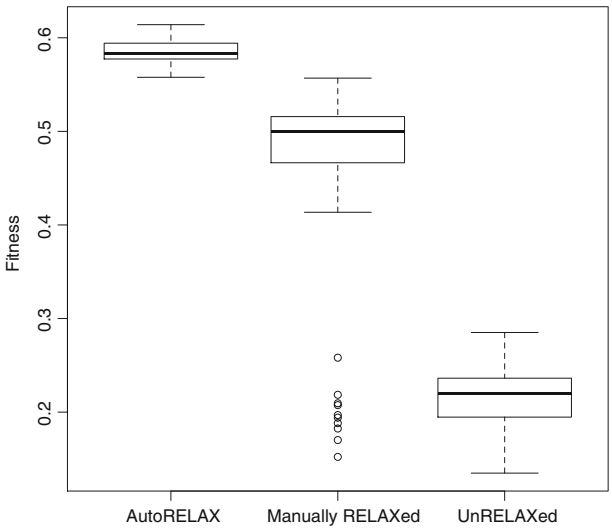

Fig. 15 Fitness values comparison between RELAXed and unRELAXed goal models for the SVS

RELAXed ($p < 0.001$, Welch Two Sample t-test), thus allowing us to reject our second null hypothesis, $H2_0$. We conclude that AutoRELAX can generate RELAXed goal models that address uncertainty better than manually RELAXed goal models.

The SVS application is sensitive to faults encountered throughout simulation, specifically those that cause the robot to cease proper function. For example, Goals (I) and (K) describe the normal cleaning and movement capabilities, respectively, of the SVS. There is, however, an associated amount of battery power drained by satisfying each of the aforementioned goals, and as a result the SVS can run out of power before the simulation completes, effectively causing a fault that restricts the satisfaction of Goal (B). By changing configuration modes to operate in a reduced power mode, such as Goal (H) or Goal (J), the SVS can effectively extend the amount of time that it is operational. Furthermore, applying RELAX operators to Goals (D) and (E) can further extend the amount of time that the SVS operates in low power mode, extending its battery life even further.

Figure 16 provides additional information about the types of generated AutoRELAXed goal models. Specifically, this figure plots the *sorted* number of RELAXed goals per trial in two separate environments, with each data point representing the mean number of RELAXed goals generated throughout the evolutionary run for each individual goal model. The first environment contained a low degree of uncertainty, and conversely the second contained a high degree of uncertainty. Environments specified to have a low uncertainty have a reduced probability for failure occurrence, and environments with high uncertainty have a higher probability of failure occurrence. In all trials, AutoRELAX generated more RELAXations in the environment with higher uncertainty than in the environment with low uncertainty. In both environments, AutoRELAX started by introducing a relatively high number of RELAXations into the goal model, suggesting that the SVS simulation requires a large amount of flexibility to complete successfully. Moreover, the positive correlation between

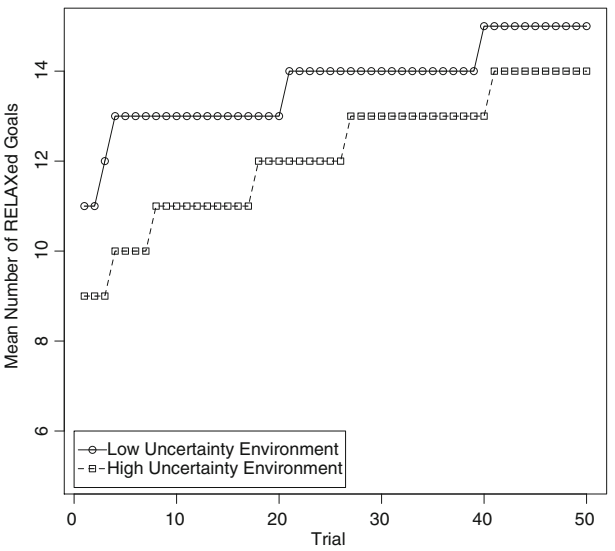

Fig. 16 Mean number of RELAXed goals for varying degrees of system and environmental uncertainty for the SVS

each curve suggests that the gradual increase of goal RELAXations is in response to an increase of uncertainties within the system and environment.

Finally, Fig. 17 presents the complementary view to Fig. 16, showing the effects of goal RELAXation upon the number of faults triggered during each trial. This figure shows that, by introducing goal RELAXations, the amount of system faults remains relatively low. In low uncertainty environments, the probability of faults occurring is small. As a result, optimizing with SAW tends to minimize fault occurrence further, resulting in a mean of zero faults occurring across all trials. The resulting weighting schemes reflect this behavior as well. The weight associated with minimizing faults tended to be larger than the weights associated with maximizing cleanliness.

4.2.3 Dynamic Weight Adjustment

This experiment reuses the experimental setup as defined for the RDM in Section 4.1.3, including the use of 50 replicate trials and null hypotheses ($H1_0$ and $H2_0$). $H1_0$ states that "there is no difference between an automatically RELAXed goal model with static weights and an automatically RELAXed goal model that uses a SAW-optimized weighting scheme across different environments," and $H2_0$ states that "there is no difference between RELAXed goal models with optimized and unoptimized weights in a static environment." The SVS fitness sub-function weights were set as follows, in order to emphasize the importance of reducing the number of faults encountered while still minimizing the number of performed adaptations:

$$\begin{aligned} \alpha_{nrg} &= 0.1, \\ \alpha_{na} &= 0.4, \\ \alpha_{faults} &= 0.3, \\ \alpha_{clean} &= 0.2 \end{aligned}$$

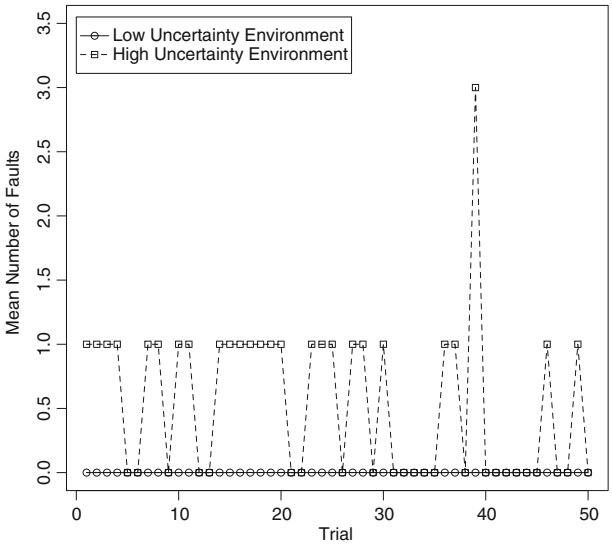

Fig. 17 Mean number of faults for varying degrees of system and environmental uncertainty for the SVS

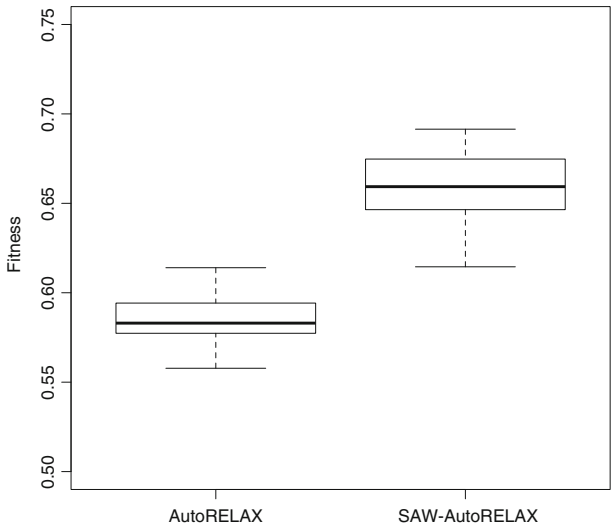

Fig. 18 Fitness values comparison between AutoRELAXed and SAW-optimized AutoRELAXed goal models for the SVS

Figure 18 presents two box plots that depict fitness values from AutoRELAXed and SAW-AutoRELAXed models, respectively. As was shown with the RDM case study, AutoRELAXed goal models with an optimized set of weights tend to statistically perform better than their unoptimized counterparts ($p < 0.001$, Welch Two Sample t-test). We are thus able to reject $H1_0$, and conclude that the weighting scheme directly impacts an AutoRELAXed model's fitness.

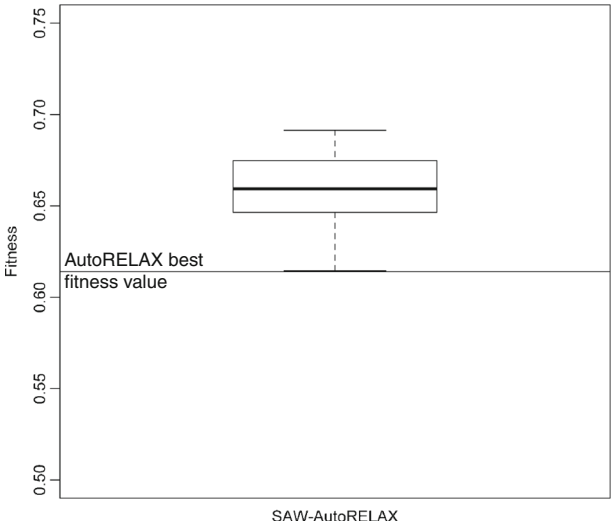

Fig. 19 Fitness values of SAW-optimized AutoRELAXed goal models in a single environment

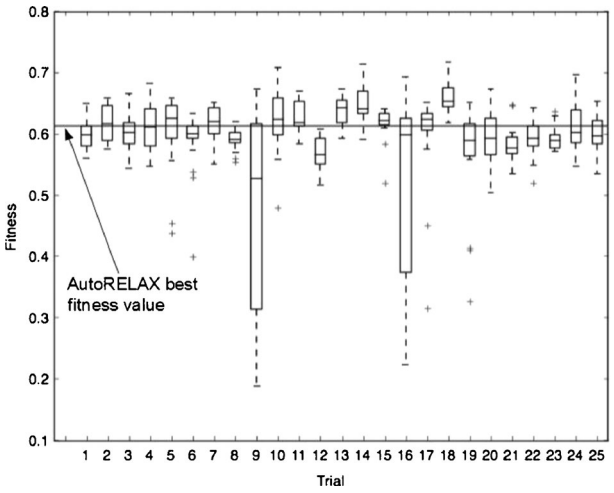

Fig. 20 Comparison of SAW-optimized fitness values from different trials in a single environment

Figure 19 shows a box plot of fitness values obtained from goal models that were generated by SAW- AutoRELAX and optimized for a single environment over 50 separate trials. The solid horizontal line represents the best fitness that AutoRELAX achieved without the SAW optimization in the given environment. In this case, optimizing the weighting scheme produced goal models that were as good if not better than those generated with AutoRELAX, indicating that the initial weighting scheme may be unoptimized for the tested environmental configuration. Figure 20 presents the range of fitness values observed from AutoRELAXed goal models optimized with SAW in a single environment over 50 trials, with the solid horizontal line representing the best fitness generated by AutoRELAX without the SAW optimization.

5 Discussion

To the best of our knowledge, this paper presented one of the first empirical results about how goal RELAXation can affect the abilities of a DAS to satisfy its requirements in the presence of system and environmental uncertainty. Specifically, AutoRELAX automatically introduces RELAX operators to non-invariant goals to make the satisfaction criteria more flexible without adversely affecting the satisfaction of invariant goals. The introduction of the SAW optimization to AutoRELAX enabled the weighting scheme of the fitness subfunctions to be tailored to a particular environment. As experimental results suggest, these additional flexibilities enable adaptive systems to reduce the number of adaptations performed in response to minor and transient sources of uncertainty. Moreover, by reducing the number of adaptations performed, the DAS is able to deliver more overall functionality to its stakeholders at run time.

Experimental results demonstrate that flexibility in how and when a DAS satisfies its requirements can be desirable. Too much flexibility, however, can also be detrimental to how a DAS provides its functionality to stakeholders. To this end, AutoRELAX explicitly rewards candidate goal models for balancing competing concerns between minimizing goal RELAXations and minimizing the number of adaptations triggered due to uncertainty. As

such, AutoRELAX attempts to preserve the original goal model by RELAXing goal satisfaction criteria only when it provides a benefit to the DAS. This observation implies that AutoRELAX can and will produce goal models without any RELAX operators if the DAS can handle such system and environmental uncertainty on its own, as was the case in several experiment trials. The experimental results from SAW- AutoRELAX further corroborate this observation by further emphasizing the minimization of the number of RELAXed goals in environments with low uncertainty, resulting in AutoRELAXed goal models with a minimal number of RELAXations.

Furthermore, experimental results established that different weights can be more suitable than static weights in order to address each of the specific operational contexts. The static weights were fixed for the AutoRELAX fitness sub-functions as derived manually by a requirements engineer, and then compared to algorithmically determined weights generated by SAW- AutoRELAX in varying environmental contexts. Given that overall fitness tended to increase with weights determined by SAW- AutoRELAX, we conclude that it can be beneficial to have weighted fitness sub-functions tailored to their environmental context.

Lastly, experimental results for another case study from a different application domain reaffirm that AutoRELAX is a domain-independent technique. Both case studies demonstrate that overall fitness can be increased by automatically defining RELAX operators for goal models. Moreover, introducing the SAW optimization can further enhance fitness by appropriately weighting the fitness sub-functions for different environments.

We have identified the following threats to validity for this research. This study was intended as a proof of concept to determine the feasibility of automatically producing viable RELAXed goal models. We applied AutoRELAX to problems provided by industrial collaborators and by industrial applications, and as a point of reference compared the AutoRELAXed goal models to those manually developed by requirements engineers. One threat to validity is that the derived utility functions may not accurately capture the desired behavior of the goal model. Another threat is that unknown dependencies between goals may manifest as feature interactions once one or more of the dependent goals have been RELAXed. Detecting and mitigating unanticipated feature interactions based on goal model analysis is part of our ongoing work.

SAW- AutoRELAX was implemented as a further proof of concept to determine if automatically adapting the weighting scheme of an AutoRELAXed goal model will increase overall fitness in a given environment. Threats to validity with respect to SAW- AutoRELAX include achieving similar results with different metrics for determining requirements for fitness sub-function weight updates at run time.

6 Related Work

This section presents related work on specifying, detecting, and mitigating the occurrence of obstacles in a DAS. Specifically, we overview techniques for expressing uncertainty in requirements, monitoring requirements satisfaction in uncertain environments, and mitigating adverse system and environmental conditions.

Expressing Uncertainty in Requirements Researchers are implicitly, and now more recently, explicitly acknowledging that uncertainty permeates the design, implementation, and execution of a DAS (Letier and van Lamsweerde 2004; Cheng et al. 2009; Baresi et al. 2010; Bencomo et al. 2010; Sawyer et al. 2010; Welsh and Sawyer 2010; Esfahani 2011; Esfahani et al. 2011; Ramirez et al. 2011; Welsh et al. 2011). Much recent work has focused

on explicitly identifying and documenting sources of uncertainty and evaluating their possible effects upon the DAS. These approaches and others attempt to raise awareness about uncertainty all the way from requirements elicitation to run-time decision-making in DASs. For instance, Welsh et al. (2010, 2011) introduced the concept of a *Claim* as a marker of uncertainty in requirements and design decisions based on possibly incorrect assumptions. Likewise, in both Esfahani (2011) and Esfahani et al. (2011) an analytical framework was proposed that combines mathematical approaches for modeling and assessing uncertainty in adaptation decisions from a risk-management perspective.

Recent research into the formal analysis of system design models has also yielded methods for identifying uncertainty. Wei et al. (2011) used model checking as a means to verify fixed portions of system models, with model variations being designated as points of uncertainty to be managed. Additionally, Filieri et al. (2011) have explored probabilistic model checking as a means for quantifying unpredictable changes in order to proactively mitigate a reduction in requirements satisfaction.

In addition to designing frameworks for documenting and managing sources of uncertainty, researchers have also focused on leveraging fuzzy set theory to represent and analyze the effects of uncertainty in requirements. For instance, Whittle et al. (2009) introduced the RELAX requirements specification language to facilitate the identification and analysis of sources of environmental uncertainty in a DAS. Cheng et al. (2009) extended the RELAX language to support the modeling of RELAXed goals in a KAOS goal model (Dardenne et al. 1993; van Lamsweerde 2009). In a similar approach, Baresi et al. (2010) as well as Pasquale and Spoletini (2011) introduced FLAGS, a KAOS goal modeling framework that introduces the concept of a fuzzy goal whose satisfaction can also be evaluated through fuzzy logic functions.

These approaches for documenting, representing, analyzing, and managing uncertainty ultimately depend on a requirements engineer who evaluates the likelihood of various sources of uncertainty arising at run time, as well as their effects upon a DAS. In particular, a requirements engineer using either RELAX, FLAGS, or Claims must currently manually determine which goals may become unsatisfied at run time without disrupting invariant requirements, as well as how much flexibility can be introduced into each non-invariant goal's satisfaction criteria. AutoRELAX automates this entire identification and assessment process by generating goal RELAXations that specify the extent to which goals may become temporarily unsatisfied at run time without violating invariant requirements. Although AutoRELAX focuses on RELAXing functional non-invariant KAOS goals, it can also be extended to automatically identify fuzzy goals in FLAGS, as well as automatically RELAX the satisficement criteria of soft goals (Chung et al. 2000).

Requirements Monitoring and Reflection Feather et al. (1998) and Fickas and Feather (1995) developed requirements monitoring frameworks that can detect the occurrence of obstacles and, in certain circumstances, suggest system reconciliations in response if necessary. More recently, the promotion of requirements to live run-time entities whose satisfaction can be evaluated in support of adaptation decisions has been suggested (Bencomo et al. 2010; Sawyer et al. 2010). A feedback loop-based Awareness Requirements construct (Souza and Mylopoulos 2011) has also been recently introduced where meta-level requirements monitor and manage the satisfaction of other system requirements.

While these approaches incorporate requirements into the decision-making process of an adaptive system, they do not directly 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.

Obstacle Mitigation Strategies have been proposed to systematically identify, analyze, and resolve obstacles, or specific conditions that prevent requirements from being fully satisfied (van Lamsweerde and Letier 2000; van Lamsweerde 2009). Most of their mitigation strategies focus on revising goals such that obstacles are either prevented or their effects negated. However, if an obstacle cannot be directly prevented or addressed, then one of their proposed mitigation strategies consists of tolerating a goal violation. This obstacle toleration strategy is akin to the RELAX and FLAGS approaches. Note, however, that neither RELAX, FLAGS, nor the strategies presented by van Lamsweerde and Letier (2000) specify the extent to which a goal can become unsatisfied without adversely affecting other goals, in particular invariant goals. From this perspective, AutoRELAX complements their proposed mitigation strategies by automatically determining if, and to what extent, a non-invariant goal may become unsatisfied at run time.

Although the aforementioned heuristics (van Lamsweerde and Letier 2000; van Lamsweerde 2009) facilitate the systematic identification and analysis of obstacles, the execution environment of some application domains may contain unpredictable or poorly understood phenomena that may still prevent a DAS from satisfying its requirements. As such, a recent framework (Letier and van Lamsweerde 2004) was introduced for specifying the probability of a goal being partially satisfied at run time. These goal satisfaction probabilities, which can be obtained from a domain expert or derived from actual system usage data, can help identify previously unknown obstacles. In contrast to AutoRELAX, however, their framework treats requirements as being either strictly satisfied or not. This difference notwithstanding, AutoRELAX could leverage goal satisfaction probabilities to identify which goals might benefit from RELAXation.

7 Conclusion

This paper described AutoRELAX, an approach for automatically generating RELAXed goal models. AutoRELAX uses a genetic algorithm to explore possible goal RELAXations to be used in order to mitigate system and environmental uncertainties, relieving requirements engineers from having to consider the large amount of possible strategies for handling uncertainty during system design. As an extension to further improve AutoRELAX, we also used SAW, a technique for optimizing the weighting scheme of the fitness sub-functions used by AutoRELAX. SAW is integrated within AutoRELAX to search for optimal combinations of weights that AutoRELAX can implement in its fitness sub-functions in order to customize the generated goal model for a given environment. AutoRELAX and SAW were applied to an RDM application that must distribute data to all network nodes while self-reconfiguring as network failures or dropped messages occur, and to an SVS application that must successfully clean the majority of a room.

Experimental results from AutoRELAX show that goal RELAXations tend to concur with specific types of uncertainty. For the RDM case study, RELAX operators were introduced to Goals (F) and (J) depending on whether network links had a higher probability of failure than messages dropped, and vice versa. Specifically, time-based RELAXations applied to Goal (J) provided extra time necessary for the goal to be satisfied. Goal (I) satisfaction was directly affected by both network link failures and dropped messages, causing (I) to be violated most often and requiring network self-reconfiguration. Implementing SAW

to balance the weighting scheme of AutoRELAX's fitness sub-functions tended to improve overall fitness with respect to the given environment. Results demonstrated that while minor variance in operational contexts can be mitigated by introducing RELAX operations, increasing variance in operational contexts more often required self-reconfiguration in order for the RDM to continue to satisfy requirements. For the SVS case study, AutoRELAX tended to apply RELAXations to Goals (D) and (E), as RELAXing those goals effectively extended the operating time available to the SVS, thus satisfying Goals (A) and (B). Furthermore, implementing SAW for the SVS case study also demonstrated that optimizing the weighting scheme of the fitness sub-functions can have a positive impact on overall performance, based on the system and environmental conditions.

Future directions include exploring different fitness function aggregates, including the optimization of their particular weighting schemes, as well as exploring how different genetic operators can guide the search process. Alternative methods for determining an optimal set of goal RELAXations, such as multidimensional optimization and probabilistic methods, may also be considered. Finally, incorporating other available types of fuzzy logic membership functions for the RELAX operators is also of interest.

Acknowledgments We gratefully acknowledge conceptual and implementation contributions from Jared M. Moore. Much of the original work with AutoRELAX was done by Andres J. Ramirez.

This work has been supported in part by NSF grants CCF-0854931, CCF-0750787, CCF-0820220, DBI-0939454, Army Research Office grant W911NF-08-1-0495, and 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. The authors declare that they have no conflict of interest.

References

Baresi L, Pasquale L, Spoletini P (2010) Fuzzy goals for requirements-driven adaptation. In: Proceedings of the 18th IEEE international requirements engineering conference. IEEE, Sydney, pp 125–134

Bencomo N, Whittle J, Sawyer P, Finkelstein A, Letier E (2010) Requirements reflection: requirements as runtime entities. In: Proceedings of the 32nd ACM/IEEE international conference on software engineering. ACM, Cape Town, pp 199–202

Bencomo N, Belaggoun A (2013) Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks. In: Requirements engineering: Foundation for Software Quality. Springer, pp 221–236

Cheng BHC, Sawyer P, Bencomo N, Whittle J (2009) A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: ACM/IEEE international conference on model driven engineering languages and systems (MODELS'09). Lecture notes in computer science. Springer-Verlag, Denver, pp 468–483

Chung L, Nixon B, Yu E, Mylopoulos J (2000) Non-functional requirements in software engineering. Kluwer

Craenen B, Eiben A (2001) Stepwise adaption of weights with refinement and decay on constraint satisfaction problems. In: Proceedings of the genetic and evolutionary computation conference (GECCO 2001), pp 291–298

Dardenne A, van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 20(1-2):3–50

Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. In: IEEE transactions on evolutionary computation, vol 6. IEEE, pp 182–197

de Grandis P, Valetto G (2009) Elicitation and utilization of application-level utility functions. In: Proceedings of the sixth international conference on autonomic computing (ICAC'09). ACM, Barcelona, pp 107–116

Eiben Á, van der Hauw J (1997) Solving 3-sat by gas adapting constraint weights. In: IEEE international conference on evolutionary computation. IEEE, pp 81–86

Eiben A, van der Hauw J (1998) Adaptive penalties for evolutionary graph coloring. In: Artificial evolution. Springer, pp 95–106

Eiben A, van der Hauw J, van Hemert J (1998) Graph coloring with adaptive evolutionary algorithms. J Heuristics 4:25–46

Esfahani N (2011) A framework for managing uncertainty in self-adaptive software systems. In: Proceedings of the 26th IEEE/ACM international conference on automated software engineering. Lawrence, Kansas

Esfahani N, Kouroshfar E, Malek S (2011) 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, pp 234–244

Feather MS, Fickas S, van Lamsweerde A, Ponsard C (1998) Reconciling system requirements and runtime behavior. In: Proceedings of the 8th international workshop on software specification and design. IEEE Computer Society, Washington, DC, pp 50–59

Fickas S, Feather MS (1995) Requirements monitoring in dynamic environments. In: Proceedings of the second IEEE international symposium on requirements engineering. IEEE Computer Society, Washington, DC, pp 140–147

Filieri A, Ghezzi C, Tamburrelli G (2011) Run-time efficient probabilistic model checking. In: Proceedings of the 33rd international conference on software engineering. ACM, Waikiki, pp 341–350

Holland JH (1992) Adaptation in natural and artificial systems. MIT Press, Cambridge

Jackson M, Zave P (1995) Deriving specifications from requirements: an example. In: Proceedings of the 17th international conference on software engineering. ACM, Seattle, pp 15–24

Ji M, Veitch A, Wilkes J (2003) Seneca: remote mirroring done write. In: USENIX 2003 annual technical conference. USENIX Association, Berkeley, pp 253–268

Keeton K, Santos C, Beyer D, Chase J, Wilkes J (2004) Designing for disasters. In: Proceedings of the 3rd USENIX conference on file and storage technologies. USENIX Association, Berkeley, pp 59–62

Kramer J, Magee J (1990) The evolving philosophers problem: dynamic change management. IEEE Trans Soft Eng 16(11):1293–1306

Letier E, van Lamsweerde A (2004) Reasoning about partial goal satisfaction for requirements and design engineering. In: Proceedings of the 12th ACM SIGSOFT international symposium on foundations of software engineering. ACM, Newport Beach, pp 53–62

Neema S, Bapty T, Scott J (1999) Development environment for dynamically reconfigurable embedded systems. In: Proceedings of the international conference on signal processing applications and technology. Orlando

Pasquale L, Spoletini P (2011) Monitoring fuzzy temporal requirements for service compositions: motivations, challenges, and experimental results. In: Proceedings of the 2011 international workshop on requirements engineering for systems, services and systems of systems. IEEE, Trento, pp 63–69

Ramirez AJ, Knoester DB, Cheng BHC, McKinley PK (2009) Applying genetic algorithms to decision making in autonomic computing systems. In: Proceedings of the sixth international conference on autonomic computing. Barcelona, pp 97–106

Ramirez AJ, Knoester DB, Cheng BHC, McKinley PK (2010) Plato: a genetic algorithm approach to run-time reconfiguration of autonomic computing systems. Clust Comput to appear

Ramirez AJ, Cheng BHC (2011) 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

Ramirez AJ, Jensen AC, Cheng BH, Knoester DB (2011) Automatically exploring how uncertainty impacts behavior of dynamically adaptive systems. In: Proceedings of the 2011 international conference on automatic software engineering, ASE'11. Lawrence, Kansas

Ramirez AJ, Fredericks EM, Jensen AC, Cheng BHC (2012) Automatically relaxing a goal model to cope with uncertainty. In: Fraser G, Teixeira de Souza J (eds) Search based software engineering, lecture notes in computer science, vol 7515. Springer, Berlin, pp 198–212

Sawyer P, Bencomo N, Letier E, Finkelstein A (2010) Requirements-aware systems: a research agenda for re self-adaptive systems. In: Proceedings of the 18th IEEE international requirements engineering conference. Sydney, pp 95–103

Souza VES, Mylopoulos J (2011) From awareness requirements to adaptive systems: a control-theoretic approach. In: Proceedings of the second international workshop on requirements at run time. IEEE Computer Society, Trento, pp 9–15

van der Hauw K (1996) Evaluating and improving steady state evolutionary algorithms on constraint satisfaction problems. Master's thesis, Leiden University

van Lamsweerde A (2009) Requirements engineering: from system goals to UML models to software specifications. Wiley
van Lamsweerde A, Letier E (2000) Handling obstacles in goal-oriented requirements engineering. IEEE Trans Softw Eng 26(10):978–1005
Walsh WE, Tesauro G, Kephart JO, Das R (2004) Utility functions in autonomic systems. In: Proceedings of the first IEEE international conference on autonomic computing. IEEE Computer Society, New York, pp 70–77
Wei O, Gurfinkel A, Chechik M (2011) On the consistency, expressiveness, and precision of partial models. J Inf Comput 209(1):20–47
Welsh K, Sawyer P (2010) Understanding the scope of uncertainty in dynamically adaptive systems. In: Proceedings of the sixth international working conference on requirements engineering: foundation for software quality, vol 6182. Springer, Essen, pp 2–16
Welsh K, Sawyer P, Bencomo N (2011) Towards requirements aware systems: run-time resolution of design-time assumptions. In: Proceedings of the 26th IEEE/ACM international conference on automated software engineering. IEEE Computer Society, Lawrence, pp 560–563
Whittle J, Sawyer P, Bencomo N, Cheng BHC, Bruel JM (2009) RELAX: Incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 17th international requirements engineering conference (RE '09). IEEE Computer Society, Atlanta, pp 79–88
Witty R, Scott D (2001) Disaster recovery plans and systems are essential. Technical Report FT-14-5021, Gartner research

Erik M. Fredericks is a Ph.D. student in the Department of Computer Science and Engineering at Michigan State University. His research interests include evolutionary computation, software engineering, requirements engineering, run-time software testing, and cyber-physical systems. The confluence of these research areas forms the basis of his doctoral research, as software assurance for real-world systems must be continually provided as system and environmental conditions change over time. Prior to starting his doctoral studies, Erik worked as a software engineer and software project manager in the automotive industry for several years. He received his BS from Lake Superior State University in 2007 and MS from Oakland University in 2011. He can be reached at the Department of Computer Science and Engineering, Michigan State University, 3115 Engineering Building, East Lansing, MI 48824; freder99@cse.msu.edu; www.efredericks.net.

Byron DeVries received his BCS from Calvin College in 2005 and his MS from Grand Valley State University in 2013 in Computer Science and Computer Information Systems, respectively. Currently, he is working towards a Ph.D. in computer science in the Department of Computer Science and Engineering at Michigan State University. His research interests include model-driven engineering and analysis of dynamically adaptive systems.

Betty H.C. Cheng is a professor in the Department of Computer Science and Engineering at Michigan State University. Her research interests include dynamically adaptive systems, requirements engineering, model-driven engineering, automated software engineering, and harnessing evolutionary computation to address software engineering problems. These research areas are used to support the development of high-assurance adaptive systems that must continuously deliver acceptable behavior, even in the face of environmental and system uncertainty. Example applications include intelligent transportation and vehicle systems. She collaborates extensively with industrial partners in her research projects in order to ensure real-world relevance of her research and to facilitate technology exchange between academia and industry. Previously, she was awarded a NASA/JPL Faculty Fellowship to investigate the use of new software engineering techniques for a portion of the shuttle software. She spent one sabbatical working with the Motorola Software Labs investigating automated analysis techniques of specifications of telecommunication systems. During her most recent sabbatical, she was awarded an international faculty scholarship to explore research techniques for specifying and managing uncertainty in high-assurance systems. Her research has been funded by several federal funding agencies, including NSF, ONR, DARPA, NASA, AFRL, ARO, and numerous industrial organizations. She serves on the editorial boards for Requirements Engineering Journal, and Software and Systems Modeling, and IEEE Transactions on Software Engineering. She was the Technical Program Co-Chair for IEEE International Conference on Software Engineering (ICSE-2013), the premier and flagship conference for software engineering. She received her BS from Northwestern University in 1985 and her MS and PhD from the University of Illinois-Urbana Champaign in 1987 and 1990, respectively, all in computer science. She may be reached at the Department of Computer Science and Engineering, Michigan State Univ., 3115 Engineering Building, East Lansing, MI 48824; chengb@cse.msu.edu; www.cse.msu.edu/~chengb.