The Requirements Problem for Adaptive Systems

IVAN J. JURETA, Fonds de la Recherche Scientifique – FNRS and University of Namur
ALEXANDER BORGIDA, Rutgers University
NEIL A. ERNST, University of British Columbia
JOHN MYLOPOULOS, University of Trento

Requirements Engineering (RE) focuses on eliciting, modeling, and analyzing the requirements and environment of a system-to-be in order to design its specification. The design of the specification, known as the Requirements Problem (RP), is a complex problem-solving task because it involves, for each new system, the discovery and exploration of, and decision making in a new problem space. A system is adaptive if it can detect deviations between its runtime behavior and its requirements, specifically situations where its behavior violates one or more of its requirements. Given such a deviation, an Adaptive System uses feedback mechanisms to analyze these changes and decide, with or without human intervention, how to adjust its behavior as a result. We are interested in defining the Requirements Problem for Adaptive Systems (RPAS). In our case, we are looking for a *configurable* specification such that whenever requirements fail to be fulfilled, the system can go through a series of adaptations that change its configuration and eventually restore fulfilment of the requirements. From a theoretical perspective, this article formally shows the fundamental differences between standard RE (notably Zave and Jackson [1997]) and RE for Adaptive Systems (see the seminal work by Fickas and Feather [1995], to Letier and van Lamsweerde [2004], and up to Whittle et al. [2010]). The main contribution of this article is to introduce the RPAS as a new RP class that is specific to Adaptive Systems. We relate the RPAS to RE research on the relaxation of requirements, the evaluation of their partial satisfaction, and the monitoring and control of requirements, all topics of particular interest in research on adaptive systems [de Lemos et al. 2013]. From an engineering perspective, we define a proto-framework for solving RPAS, which illustrates features needed in future frameworks for adaptive software systems.

Categories and Subject Descriptors: D.2.1 [**Requirements**]: Languages

General Terms: Languages, Theory

Additional Key Words and Phrases: Adaptive systems, requirements engineering, requirements problem, requirements modelling language, requirements problem for adaptive systems

ACM Reference Format:
Ivan J. Jureta, Alexander Borgida, Neil A. Ernst, and John Mylopoulos. 2014. The requirements problem for adaptive systems. ACM Trans. Manag. Inform. Syst. 5, 3, Article 17 (September 2014), 33 pages.
DOI: http://dx.doi.org/10.1145/2629376

This work was supported in part by ERC advanced grant 267856, titled "Lucretius: Foundations for Software Evolution" http://www.lucretius.eu.
Authors' addresses: I. J. Jureta, Department of Business Administration, University of Namur, 8 rempart de la Vierge, B-5000 Namur, Belgium; email: ivan.jureta@unamur.be; A. Borgida, Department of Computer Science, Rutgers University, Piscataway, NJ 08855; email: borgida@cs.rutgers.edu; N. A. Ernst, Department of Computer Science, University of British Columbia, ICICS/CS Building 201-2366 Main Mall, Vancouver, B. C. V6T 1Z4, Canada; email: neil.ernst@gmail.com; J. Mylopoulos, Information Engineering and Computer Science, University of Trento, 5 Via Sommarive, I-38123 Povo, Italy; email: jm@disi.unitn.it.

DOI: http://dx.doi.org/10.1145/2629376

1. INTRODUCTION

1.1. Uncertainty and Risk in Requirements

Requirements Engineering (RE) focuses on eliciting, modeling, and analyzing the requirements and environment of a system-to-be in order to design a specification. The design of the specification, usually called the *Requirements Problem* (RP), is a complex problem-solving tas because it involves the discovery and exploration of, and decision making in, a large problem space. Unsurprisingly, it has been repeatedly found through experience that it is hard to get the requirements right because "it is really impossible for a client, even working with a software engineer, to specify completely, precisely, and correctly the exact requirements of a modern software product before trying some versions of the product" [Brooks 1986, p. 1075].

A fundamental difficulty in solving the RP is that requirements often reflect engineers' (be they software engineers, product designers, or business analysts) interpretations of future expectations and needs of stakeholders.

Uncertainty is therefore an essential property of requirements. So is risk: uncertainty is usually quantified by probability and risk as the sum of the products of the probability of each undesirable outcome and its corresponding cost.

It is impossible to eliminate all requirements risks at design time, when the specification of the first or any subsequent release is produced. It is impossible because the system-to-be is situated in a complex environment of societal rules, commitments, incentives, laws, contracts, organizations, and, to borrow again from Brooks, "[t]hese all change continually, and their changes inexorably force change upon the software product" [Brooks 1986, p. 1070].

1.2. Adaptive Systems as a Response to Requirements Risk

Although there is no silver bullet for requirements uncertainty, there are strategies to mitigate risk. Design time, for example, may involve prototypes and frequent releases, collecting feedback, then taking that feedback into account for the design of each next release.

A risk management strategy that has been steadily gaining attention is the engineering of *Adaptive Systems* [Cheng et al. 2009; de Lemos et al. 2013]. A system is adaptive if it can detect differences between its requirements and runtime performance and can adjust its behavior to cope with such deviations.

Adaptive Systems engineering, when seen as a risk management strategy, is one that spans design time and runtime: Design time, because design decisions influence the range of monitored inputs for the system, and the feedback mechanisms it will have; and runtime, because these mechanisms enable the system to react to at least some changes rather than ignore them.

1.3. Exactly Why and How Is Requirements Engineering for Adaptive Systems Nonstandard?

It has been argued that RE for adaptive software systems is nonstandard [Cheng et al. 2009; Whittle et al. 2010; de Lemos et al. 2013] because it is impossible to know at design time all possible changes in requirements and environment conditions and therefore build into the specification all the best ways for it to respond at runtime. Although some changes can be anticipated at design time, many will only be identified and understood when or after they happen.

The observation that we are not very good at predicting the future, can, however, be made independently of Adaptive Systems. It applies for RE for all kinds of systems. The interesting question for research for RE on Adaptive Systems is therefore this:

Exactly why and how is Requirements Engineering for Adaptive Systems different from standard Requirements Engineering?

It is interesting, because an answer will suggest what can be retained in, discarded from, or added to standard RE modeling languages, frameworks, and methodologies to address Adaptive Systems RE; that is, to help design Adaptive Systems.

The aim of this article is to give a formal answer to this question and thereby help connect existing research, inform future work, and move closer to a deeper classification of research in Adaptive Systems RE. It might also help skeptics decide if/how novel is the problem posed by Adaptive Systems RE.

1.4. Contributions: A New Requirements Problem

To answer the central question in this article, we start from the *de facto* standard RP from Zave and Jackson [1997], then illustrate and discuss its differences from the *Requirements Problem for Adaptive Systems* (RPAS).

We then introduce a formalism that is just expressive enough to be useful in solving RPAS, includes concepts and relations already appearing in mainstream RE, and includes features central in RE for Adaptive Systems. These features are:

—Monitoring and control of requirements [Fickas and Feather 1995; Feather et al. 1998; Robinson 2006]); that is, the idea that satisfaction of requirements should be monitored and that failure to reach required satisfaction levels should trigger changes in system behavior;
—Probabilistic relaxation of requirements [Letier and van Lamsweerde 2004]; that is, instead of requiring that some requirement be satisfied to the same extent all of the time, probabilistic relaxation weakens such requirements so that they can be satisfied some of the time to some desired frequency;
—Fuzzy relaxation of requirements [Whittle et al. 2010; Baresi et al. 2010]), where black-and-white satisfaction of a requirement is replaced with satisfaction to some level on a continuous scale defined by a fuzzy membership function;
—Evolution requirements [Souza et al. 2012] that concern how to change the system when some other requirements succeed or fail.

It turns out that extending the simple but abstract modeling language Techne [Jureta et al. 2010] to allow constraints over quantitative (numerical) variables is enough to capture key ideas in monitoring, relaxation, and adaptation, and to show how they contribute to the RPAS.

This article presents two main contributions:

—The RPAS and its solution concept, called *Configurable Specification*, are formally defined. A Configurable Specification amounts to a set of requirements configurations and evolution requirements for switching between configurations. Each configuration is shown to satisfy all properties required by Zave and Jackson [1997] and in our prior work [Jureta et al. 2008]. These parallels show exactly how the requirements problem and Configurable Specification concepts are different from more traditional RE.
—To introduce the RPAS and Configurable Specification concepts, we define a simple formalism in which we can model (i) probabilistic and fuzzy relaxation, (ii) monitored and controlled variables, (iii) evolution requirements. It is a *proto-framework* because (a) it is impractical in its current form, but is enough to allow us to present and discuss the salient properties of the RPAS and Configurable Specification concepts, and (b) it can serve as a prototype for early formal modeling languages for RE of Adaptive Systems.

1.5. Organization

This article is organized as follows. Section 2 gives an informal introduction to and the definition of the RPAS. Section 3 contrasts RPAS to the the standard RP in RE; there, we use a simple example to discuss how the RPAS departs from the standard RP. Section 4 introduces informally a modeling framework that we use in the rest of the article to model instances of the RPAS. Section 5 recalls key ideas in the existing RE research on Adaptive Systemand thereby identifies the notions that have to be accommodated in the RPAS, its solution concept, and the modeling framework. Section 6 focuses on the solution concept, the Configurable Specification, and defines its components. We then revisit in Section 7 the question of how exactly RE for Adaptive System differs from standard RE. We summarize conclusions and open issues in Section 8. This article has electronic appendices in which we formally define the modeling framework, and explain how to use it to capture the key notions from related work on RE for Adaptive System.

2. THE REQUIREMENTS PROBLEM FOR ADAPTIVE SYSTEMS

2.1. Premises for the Definition of the Problem

The RPAS is based on the premise that *the overall aim when designing an Adaptive System is to make sure that the system satisfies its stakeholders' requirements as much as feasible over time, as its environment conditions change and/or its components fail*.

To adapt, the Adaptive System has to gather data about events in its operating environment and about the functioning of its own components. At all times, and on the basis of these observations, the Adaptive System has to estimate the level to which it satisfies stakeholders' requirements. If the levels of satisfaction are inadequate, the Adaptive System has to make changes to its operational configuration in order to satisfy requirements.

This leads to key observations about the *runtime* of Adaptive System. (i) The level at which requirements are satisfied will vary due to failures in the system and changes in its environment. (ii) It is necessary to monitor the level of satisfaction in order to know when the system needs to adapt. (iii) When the system adapts, it may have different ways of adapting, and each of these ways may have a different impact on requirements satisfaction levels. (iv) Whenever it needs to adapt, the system should adapt in a way that optimizes levels of requirements satisfaction, relative to the newly observed failure of a component or of a change in the environment.

The observations about runtime have important implications for the *design time* of Adaptive Systems. Due to observation (i), it may be too idealistic and impractical to think of requirements as being either satisfied or not because this may lead to too many failed requirements, too often. It can be more practical, therefore, to define multivalued scales of requirements satisfaction, in which failure equates to only some of many values. This is done through the **relaxation of requirements**, where the idea is to replace binary levels of satisfaction with, for example, continuous scales of satisfaction or by letting the requirement be binary, but tracking the frequency at which they are being satisfied or failing and then using that frequency as the measure of the degree to which these requirements are satisfied. Observation (ii) suggests that it is necessary, at design time, to define the levels of requirements satisfaction that trigger adaptation. If the requirement has a binary satisfaction scale, being either satisfied or not, it may be relevant to define the minimal probability of observing its satisfaction; for example, in an ambulance dispatch system, asking for the probability of at least 0.95 that an ambulance arrives to an incident location within 14 minutes of being dispatched to it. This would translate, at runtime, into looking at the frequency of incidents in which the ambulance arrived 14 minutes or more from its dispatch and triggering adaptation

if that frequency is 5% or more of all incidents to which an ambulance was dispatched. If the requirement has a scale with many levels of satisfaction, then a threshold value has to be defined on that scale such that, when the satisfaction is below threshold, the system needs to adapt. This has led to research on **awareness requirements**, which are used to define these thresholds for triggering adaptation.

At runtime, when awareness requirements are satisfied, feedback loops become active, and the system adapts. Because of observations (iii) and (iv), it is necessary to define at design time the requirements that the system should satisfy when adapting. These are the so-called **evolution requirements** [Souza et al. 2012], and they place constraints on how the system adapts. In the terminology of research on the RE for Adaptive Systems, evolution requirements place constraints on the range of **reconciliation tactics** that the system may choose to apply, when adapting. For example, if the requirement to compute ambulance location fails, one evolution requirement could be to try to satisfy instead the requirement that assistants should compute ambulance location manually. Another evolution requirement could be to try to compute ambulance location from ambulance's onboard record.

2.2. Adaptation as Switching between Configurations

Given the observations and implications discussed in the preceding section, the runtime of an Adaptive System looks like a sequence of time periods of two kinds: stability, when no awareness requirement is violated, and adaptation, which lasts while awareness requirements are violated and the system is moving through unstable states toward another stable period. Evolution requirements define the transitions that take place during adaptation periods.

This leads to the following terminology in this paper. A **configuration** describes the tasks that the system should execute and the environmental conditions within which these tasks can be executed. If these conditions change, the same tasks may no longer be feasible. A configuration describes system responsibilities and its environmental assumptions.

The difference between the notion of configuration and the usual notion of specification is that, when we talk about configurations, we assume that there is a so-called Configurable Specificationand that one configuration amounts to one set of values of all the parameters of that specification. By assigning a value to each configurable parameter of the specification, we are choosing one of its possible configurations.

During adaptation, the system **switches** from one configuration to another and does so because awareness requirements became violated during the last period of stability. When switching from one configuration to another—that is, when changing the parameters of the specification—the system must satisfy the evolution requirements; it has to choose the new set of tasks by taking into account the environmental conditions, the requirements to satisfy, and the evolution requirements.

An individual configuration has to satisfy a number of properties; we mention them here and define them formally in Section 6. It has to be consistent, so that executing tasks does not make it impossible to execute others in the same configuration or possible to violate the environment conditions. It must satisfy all mandatory requirements to their threshold levels because it would otherwise not be acceptable to stakeholders. It has to include only those tasks that are necessary and sufficient to reach thresholds for the satisfaction of the requirements and no unused tasks. Finally, it has to be Pareto efficient with regards to optional requirements; these requirements are optional in the sense of being nice-to-have, but, if they fail, the system still remains acceptable to the stakeholders. If there were two configurations that the system could choose from to exit an adaptation period, and if their only difference is that one satisfies more desirable

optional requirements, then the system choosing the other would not be Pareto efficient with regards to optional requirements.

2.3. Configurable Specifications

An individual configuration or, in other words a nonparameterizable specification, cannot be the solution to the RPAS. This is because a single configuration is only applicable in the environmental conditions that it describes and when the system is capable of executing those tasks that it describes. The single configuration says nothing about the system when these environmental conditions change or when some of its capabilities fail, thus making it unable to execute the tasks that the configuration prescribes.

At runtime, the Adaptive System will be switching between configurations and should do so in such a way that it satisfies the evolution requirements. The Adaptive System will, at runtime, be realizing a specific sequence of configurations.

A specific sequence of configurations is not the solution to the RPASbecause it is not possible to predict how the environmental conditions will vary over time, when exactly the system will fail, and which tasks it will fail to execute. In other words, we cannot at design time predict the best sequence of configurations to follow at runtime.

Moreover, a motivation for enabling adaptivity is to delegate to the system at least part of the decision making involved in choosing which configuration to switch to. The solution should recognize that there can be varying degrees of autonomy with which the system will choose the next configuration when the current one fails.

This discussion led us to introduce the concept of **Configurable Specification** as the solution concept for the RPAS.

Definition 2.1. **Configurable Specification:** A Configurable Specification is a pair $\mathcal{R} = (\mathcal{S}, \mathcal{E})$, where $\mathcal{S} = \{S_1, \ldots, S_n\}$ is a set of n configurations, and $\mathcal{E}$ is a set of evolution requirements.

The solution to the RPAS includes no concept, which is specifically useful in describing how exactly adaptation should be done. Evolution requirements impose constraints on this, and, if needed, feedback loops can be defined in tasks, in configurations.

2.4. The Requirements Problem for Adaptive Systems

The RPAS is the problem of designing a **Configurable Specification** that can satisfy the given requirements to the highest feasible level for the predicted failures and environmental changes.

Definition 2.2. **The RPAS: Given** (i) stakeholders' requirements, (ii) stakeholders' preferences over different levels of requirements satisfaction, and (iii) domain assumptions about the operating environment and its expected changes at runtime, **find** the Configurable Specification such that, if the system-to-be is implemented so as to be capable of realizing the configurations and satisfying the evolution requirements in the Configurable Specification, then it is expected at runtime to satisfy the requirements to their most desirable feasible levels.

The RPAS does not commit to what it means exactly that requirements are satisfied to their most desirable feasible levels; this will depend on the preferences over requirements and the specifics of scales of satisfaction used to quantify the degree of satisfaction. Also note that configurations might share the same tasks, yet these tasks could be parameterized differently in different configurations.

3. FROM THE STANDARD REQUIREMENTS PROBLEM TO THE REQUIREMENTS PROBLEM FOR ADAPTIVE SYSTEMS

3.1. The Standard Requirements Problem

The standard view in RE is that the system specification is produced incrementally, starting from requirements and domain knowledge. The latter captures properties of the environment where the system will run. The goal is to produce a specification which, together with the given domain knowledge, satisfies requirements and is consistent with both requirements and domain knowledge.

This important and general conceptualization of the aim in RE is most clearly formulated in Zave and Jackson's seminal paper, "Four dark corners of requirements engineering" [Zave and Jackson 1997]. This view, called ZJ hereafter, is echoed in some of the most influential research in the field, which both preceded and followed the said paper, including, for example, contributions from Boehm [1988], Boehm et al. [1995], Dardenne et al. [1993], Darimont and van Lamsweerde [1996], van Lamsweerde et al. [1998], van Lamsweerde and Letier [2000], Letier and van Lamsweerde [2004], Mylopoulos et al. [1992], Greenspan et al. [1994], Castro et al. [2002], Robinson et al. [2003], Nuseibeh et al. [1994], and Hunter and Nuseibeh [1998].

More specifically, Zave and Jackson suggested that, in any concrete systems engineering project, RE is successfully completed when the following five conditions are satisfied [Zave and Jackson 1997]:

(1) *"There is a set R of requirements. Each member of R has been validated (checked informally) as acceptable to the customer, and R as a whole has been validated as expressing all the customer's desires with respect to the software development project.*
(2) *There is a set K of statements of domain knowledge. Each member of K has been validated (checked informally) as true of the environment.*
(3) *There is a set S of specifications. The members of S do not constrain the environment; they are not stated in terms of any unshared actions or state components; and they do not refer to the future.*
(4) *A proof shows that* $K, S \vdash R$. *This proof ensures that an implementation of S will satisfy the requirements.*
(5) *There is a proof that S and K are consistent. This ensures that the specification is internally consistent and consistent with the environment."*

According to Zave and Jackson, the aforementioned conditions "establish minimum standards for what information should be represented in a requirements language."

If the satisfaction of these conditions marks the end of RE in any systems engineering project, then we can give the following compact formulation of the standard RP, which we call the ZJ RP hereafter.

Definition 3.1. **Zave and Jackson Requirements Problem (ZJ RP):** Given a set R of requirements and a set K of domain knowledge, find a specification S such that (i) there is a proof of R from K and S, written $K, S \vdash R$, and (ii) K and S are consistent.

It is useful to observe the following about the ZJ RP. First, it suggests that the basic categories of information to distinguish in the RP are requirements, domain knowledge, and specification. Second, it imposes two conditions on the members of these categories. The first condition is called the *Satisfaction Condition* and the second the *Consistency Condition* in the rest of this article.

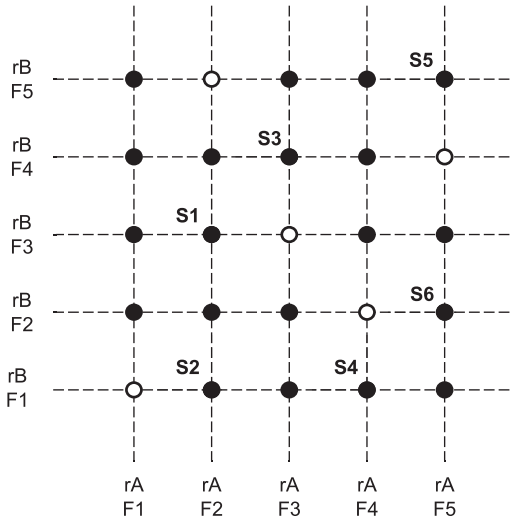

Fig. 1. rA and rB are two functional requirements that both need to be satisfied by a system. rAF1 to rAF5 are alternative functionalities that satisfy rA, whereas rBF1 to rBF5 are alternative functionalities that satisfy rB. Filled circles are combinations of functionalities that satisfy both rA and rB and are thereby alternative configurations of a system; empty circles denote incompatible combinations of functionalities and are not configurations.

3.2. Adaptation to Change

Having recalled the ZJ RP as the standard RP for RE, the goal now is to clarify what is meant by adaptation when talking about RE for Adaptive Systems. This will be used in subsequent sections to discuss why and how the RPAS differs from the ZJ RP.

To clarify what is meant by adaptation, consider a trivial example. In this example, by *functional requirement* we mean a requirement that is either satisfied or not; it is if a system can deliver a specific functionality, which is, roughly speaking, that it can do something we can observe. By *quantitative variable requirement* we mean a requirement that assigns a desirable value or range of values to a variable that is not binary.

We have only two functional requirements, rA and rB, to satisfy. We know that we can satisfy rA by implementing one of five different functionalities, denoted rAF1 to rAF5, and rB by implementing another five different functionalities, denoted rBF1 to rBF5. For simplicity, let all 10 functionalities be different and not related in terms of refinement or parthood; that is, they are neither more detailed variants of one another, nor parts of one another.

With two functional requirements and five functionalities satisfying each, there are 25 combinations of the 10 functionalities. But some functionalities are not compatible. This means that we cannot make a system that includes them both. Some combinations of functionalities therefore do not give a configuration that satisfies both rA and rB.

The part of the example introduced so far can be drawn as in Figure 1. There, filled circles are configurations; that is, combinations of functionalities that satisfy both rA and rB. Empty circles are incompatible combinations of functionalities, and because we cannot make a system that has those functionalities, these empty circles are not configurations.

If our problem was to find one combination in which functionalities are compatible and that satisfies both rA and rB, then this can be any one of the 20 configurations in Figure 1.

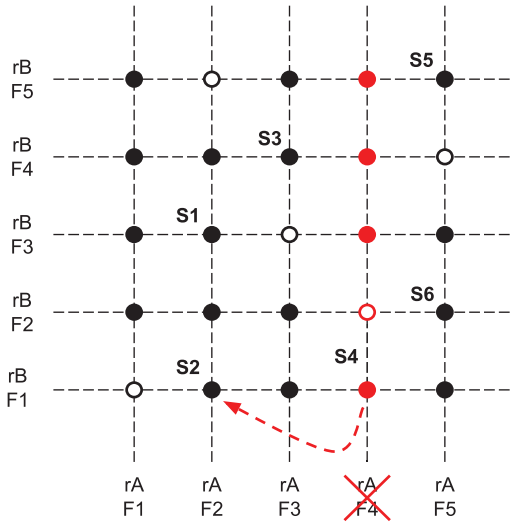

Fig. 2. System runs according to configuration S4. Then, functionality rAF4 fails, and the system activates functionality rAF2 instead, switching thereby from S4 to S2.

Consider adaptation to the failure of a functionality. If we were to design a system according to one configuration among those in Figure 1, suppose that we chose, for example, S4, so that the system has functionalities rAF4 and rBF1. If rAF4 fails, the system fails. If it were an Adaptive System, then it would be designed to switch to another functionality at runtime, for example, from rAF4 to rAF2. Such a change is, in the terminology introduced so far, a switch from one configuration to another by changing the parameters of the Configurable Specification. This is illustrated in Figure 2.

So far, and including adaptation shown in Figure 2, there is nothing that the ZJ RP fails to capture. Namely, if S in the ZJ RP is not one filled circle but a set that includes all filled circles together with conditions for activating each subset S1, S2, and so on, switching between these subsets requires no changes to the ZJ RP: The problem is still to find S, given R and K, such that S satisfies the Satisfaction Condition and Consistency Condition.

Now, assume that we have two quantitative variable requirements to satisfy in addition to rA and rB. The first relates to scalability and the second to how the system compares to its competitors. Let Var1 be the variable in the first and Var2 in the second quantitative variable requirement. Var1 can be "number of users that can simultaneously use the system" and Var2 "number of products available for purchase."

We prefer large to small values for both Var1 and Var2, and we cannot accept values that are below some threshold. This is drawn in Figure 3, where T1 is the threshold for Var1 and T2 for Var2, so that the shaded area shows all acceptable combinations of Var1 and Var2 values.

If the system were running according to S4, then it satisfies all four requirements because its values over Var1 and Var2 are above their respective thresholds. If rAF4 fails, the system would need to switch from S4 to either S3 or S5 to still satisfy all four requirements; if it switched to S2, it would satisfy rA, rB, and the requirement quantified by Var1, but not the requirement quantified by Var2.

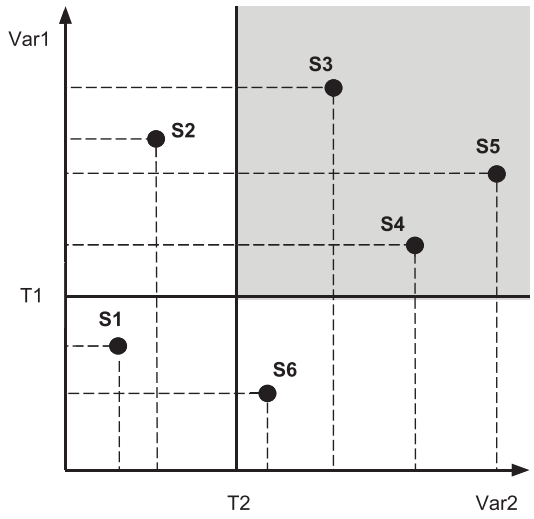

Fig. 3. Variables Var1 and Var2 quantify the level of satisfaction of two quantitative variable requirements. Hypothetical simulations of configurations S1 to S6 yield values show in the figure. T1 is the threshold value for Var1 and T2 for Var2.

As long as the system can switch from one configuration to *any* other configuration, provided that the latter satisfies all requirements and domain knowledge, then we can capture with the ZJ RP the problem of designing that system's specification.

But this fails to capture two important insights. First, the ZJ RP can be read as the problem of finding any single configuration that satisfies the Satisfaction Condition and the Consistency Condition. There is no reason why such a specification should result in a system capable of coping with requirements failure or environmental changes. In the RPAS, we are looking for a collection of such specifications, each of which applies in different conditions. We need a Configurable Specification that can be configured into each of these different nonadaptive specifications. Second, once there are different configurations that the system can switch to, a mechanism is needed to decide which of the configurations it is best to switch to.

3.3. Why Is RE for Adaptive Systems Nonstandard?

To see what optimality means here and why it makes RPAS nonstandard, we continue the example from the previous section.

We said that we prefer higher values of Var1 and Var2. To make this more precise, we need to say which combinations of values we prefer over others. Since the region above the thresholds T1 and T2 is large, it is interesting to indicate (i) the shape of indifference curves in that region and (ii) the direction where these indifference curves are over more desirable combinations of Var1 and Var2 values.

Figure 4 shows hypothetical indifference curves in the region above thresholds T1 and T2. Each indifference curve *is the set of Var1 and Var2 value combinations that are equally preferred*. So, every specification that has Var1 and Var2 values on the same indifference curve as S3 is equally preferred to S3. The arrow indicates the direction in which Var1 and Var2 value combinations are preferred, so that any specification on the indifference curve with S5 is strictly preferred to any specification on the indifference curve with S3.

Having clarified with indifference curves what we mean by preference for higher Var1 and Var2 values, we now go back to Figure 1. There, we had 20 alternative

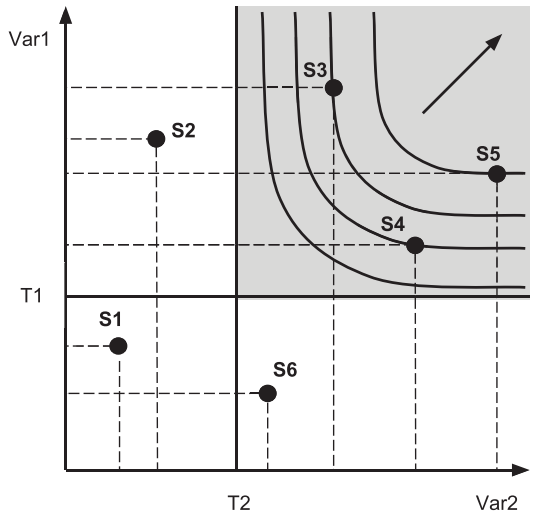

Fig. 4. Hypothetical indifference curves over value combinations of Var1 and Var2. One indifference curve includes all Var1 and Var2 value combinations that are equally desirable. For example, any configuration on the same indifference curve as S3 is equally desirable as S3. The arrow indicates the direction in which value combinations are more desirable. Therefore, S5 is preferred to S3, and S3 is preferred to S4.

configurations. Moreover, we said that adaptation amounts to moving from one to another of these configurations, based on monitored input of the system and feedback mechanisms that indicate what configuration to switch to. As discussed earlier, only some of this can be captured in terms of a ZJ RP.

By adding the two quantitative variable requirements, with their Var1 and Var2, we restricted the set of acceptable configurations to some subset of the 20 shown in Figure 1.

Given several configurations, all above T1 and T2 thresholds and all satisfying rA and rB, which one should we choose?

The answer is simple: Given the indifference curves, we should choose any configuration that is on the most desirable indifference curve. More generally, we want the system to switch, every time it needs to adapt, to the configuration that is the most desirable among those that are feasible.

This is not to say that we cannot also capture this notion of optimality in ZJ RP. For example, we can have as an additional member of R in ZJ RPthe proposition that there should be no feasible configuration that is on a more desirable indifference curve than the chosen configuration. We can therefore see RPAS as a subclass of the ZJ RP, although doing so seems rather odd because there were no considerations of configurations, adaptation, preference, uncertainty, or optimality in defining the ZJ RP.

RE for Adaptive Systems is nonstandard precisely because it is not clear how exactly notions key to Adaptive Systems, such as configurations, adaptation, preference, uncertainty, optimality, and so on, fit in the ZJ RP. Clarifying this requires that we define the RPASand thereby answer exactly *how* RE for Adaptive Systems differs from standard RE.

4. BASIC FRAMEWORK

This section introduces a framework needed to model the kinds of examples used in the previous section. We need it to make our discussion of RPAS more precise. We introduce

the framework informally in this section. It is formalized in the electronic appendices to this article.

The framework distinguishes different types of requirements and allows relations between requirements. We discuss how these types and relations are used to compare alternative requirements configurations. To illustrate the framework, we use the London Ambulance Service (LAS) case study [Anonymous 1993].

4.1. Classification of Requirements

The term *quantitative variable* refers to any variable that takes a number as its value. The term *propositional variable* is used as in classical propositional logic—it is a variable that refers to a proposition and gets either True or False as its value.

A requirement is either one of the following three types:

(1) the assignment of True to a propositional variable; for example, "q_1 should be true," where q_1 refers to the proposition "Ambulance shall arrive at an incident location within 14 minutes,"
(2) the assignment of a value to a quantitative variable; for example, "v should be exactly 50," where v is the number of ambulances per 1.000.000 inhabitants,
(3) the restriction of the range of values of a quantitative variable; for example, "average value of $t_{1,c}$ should be at most 15 seconds," where $t_{1,c}$ is the time the caller waits for a control assistant.

In item 1, we talk about a **functional requirement**; in items 2 and 3, of a **quantitative variable requirement**.

Figures 5–6 show some of the functional and quantitative variable requirements for LAS. Annotations of functional requirements, visible in the two figures, indicate their types according to the two additional classification dimensions, discussed next.

The functional and quantitative distinction gives one classification dimension for requirements. The second classification dimension distinguishes requirements according to the role they play in the RP. For that second dimension, we use the types introduced in our prior work on a core ontology for RE [Jureta et al. 2008]; this means that a requirement is:

k: a **domain assumption** if it is assumed to hold for the domain. Consider two examples:

—In Figure 5, $\mathbf{k}(r_3)^{\mathbf{M}}$ is "Callers report imprecise incident location."
—Let μ be a function over a quantitative variable t_x, where t_x is the time to establish if two or more calls refer to the same incident location, and $\mu(t_x)$ the degree of satisfaction of LAS stakeholders for a given value of t_x. (Another way to see μ is that it is a utility function.) The domain assumption $\mathbf{k}(\mu(t_x = 30 \text{ sec}) = 0.6)$ says that the degree of satisfaction with value $t_x = 30$ sec is 0.6 on some scale.

g: a **goal** is a propositional variable that describes a desired state (by the stakeholders); for example, in Figure 5, the goal $\mathbf{g}(p_4)$ indicates that it is desired that "Fill incident report" is True. Goals are always functional requirements. Quality constraint, defined next, is analogous to a goal, but for quantitative variable requirements.

q: a **quality constraint** exists if it is desired that a quantitative variable gets the value (in the range of values that) the requirement assigns to it. For example, the quality constraint $\mathbf{q}(t_x = 60 \text{ sec})$ says that t_x should have the value 60 seconds, whereas $\mathbf{q}(t_x \leq 60 \text{ sec})$ says that t_x should have the value of at most 60 seconds.

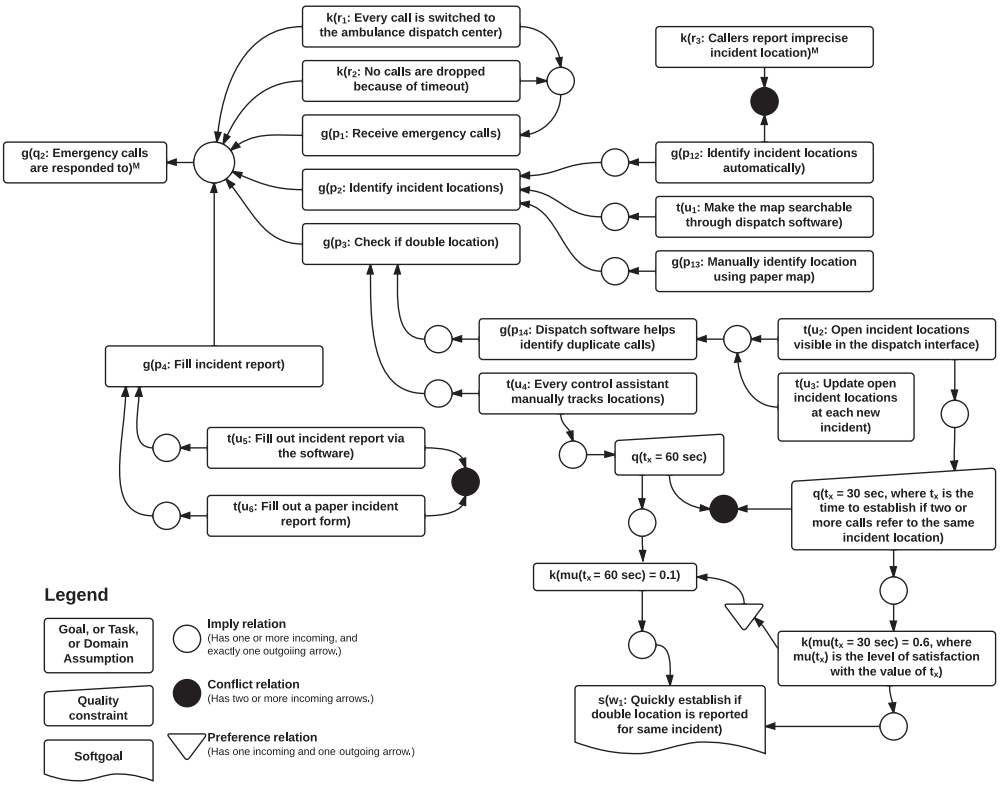

Fig. 5. One part of requirements for LAS.

s: a **softgoal** refers to a desirable range of values for a variable in which the range is only vaguely specified. Consider "Quickly establish if double location is reported for same incident," where "quickly" makes the statement vague, leading us to consider this a softgoal, denoted $\mathbf{s}(\tilde{w}_1)$, where $\tilde{w}_1$ refers to the given vague statement.

t: a **task** is an action/function that should be executed. In Figure 5, for example, a task is $\mathbf{t}(u_1)$, where u_1 refers to the proposition "Make the map searchable through dispatch software." When a task is over a quantitative variable, it says that it is necessary to ensure that the given variable gets the assigned value and that it is known how to do so (otherwise, it would not have been a task but a quality constraint). For example, if v is the number of new control assistants hired every year, there can be a task $\mathbf{t}(v = 3)$ to indicate simply that three control assistants should be hired every year, and it is not necessary to model the details of how it will be ensured that three control assistants are hired every year.

The third and final classification dimension for requirements concerns if they *must* be satisfied (i.e., obtain the value True if functional or some specified value if quantitative) or can be allowed to fail. A requirement can be **mandatory**, **optional**, or **neither**. $\mathbf{g}(q_2)^{\mathbf{M}}$ in Figure 5 is a mandatory goal, which means that it must be satisfied. $\mathbf{g}(p_{15})^{\mathbf{O}}$ in Figure 6 is an optional goal, meaning that it is more desirable that it is satisfied than not, but we will still accept a system that fails to satisfy $\mathbf{g}(p_{15})^{\mathbf{O}}$.

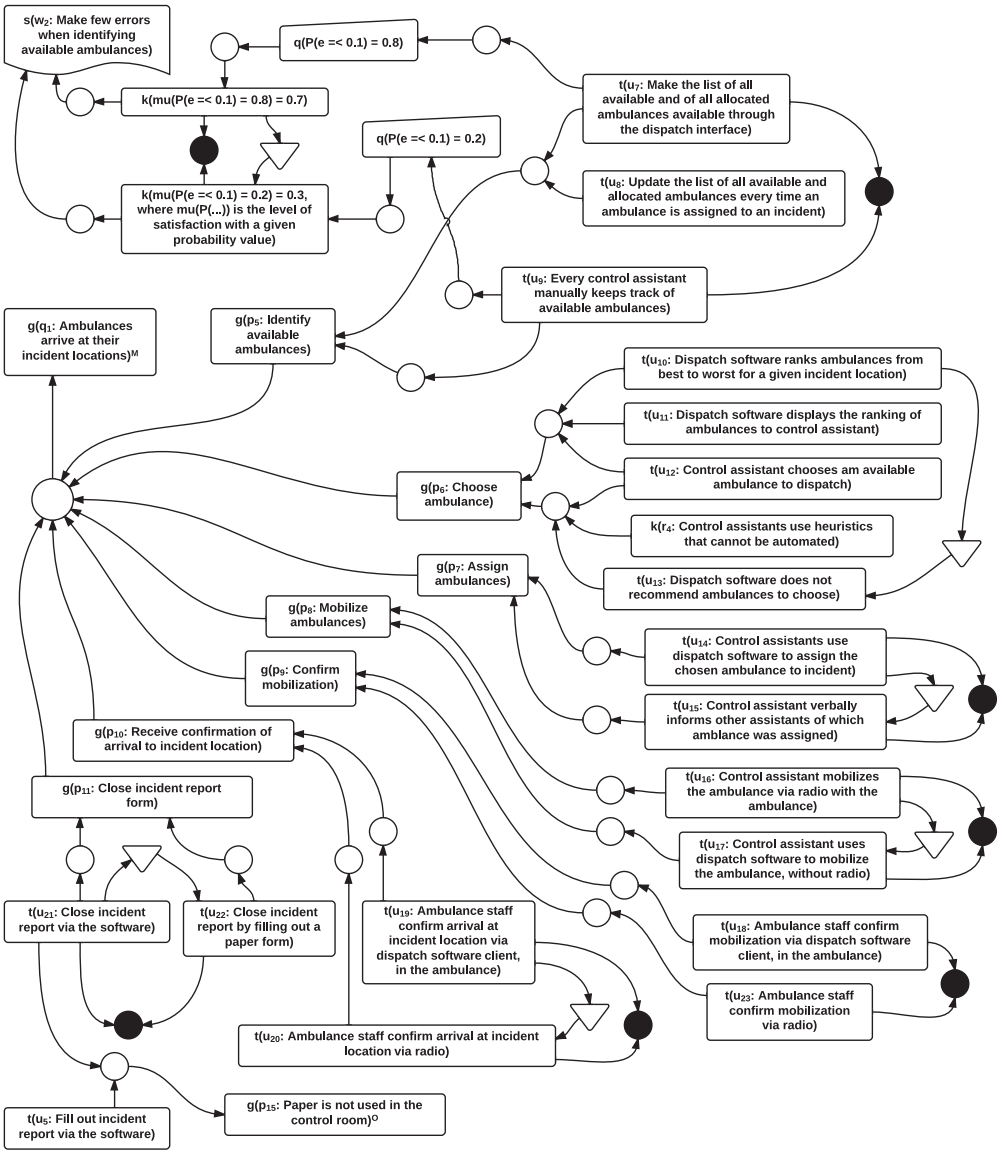

Fig. 6. Another part of requirements for LAS.

4.2. Relations between Requirements

We use three kinds of relations between requirements:

—The imply relation drawn with empty circles in Figures 5 and 6 represents nonmaterial **implication** [Jureta et al. 2010], understood as a conditional, if-then relation, so that the line from $\mathbf{t}(u_1)$ to $\mathbf{g}(p_2)$ abbreviates the formula $\mathbf{i}(\mathbf{t}(u_1) \rightarrow \mathbf{g}(p_2))$; that is, a relation according to which, if we can deduce $\mathbf{t}(u_1)$, then we can also deduce $\mathbf{g}(p_2)$. When the left-hand side of the implication includes only tasks and domain assumptions, we say that it *operationalizes* the requirement on the right-hand side. In this example, $\mathbf{t}(u_1)$ operationalizes $\mathbf{g}(p_2)$. The implication relation is understood as an *operationalization* relation if and only if it relates tasks and domain

assumptions to goals or quality constraints; otherwise, it is read as a *refinement* relation: In Figure 5, $\mathbf{i}(\mathbf{k}(r_1) \wedge \mathbf{k}(r_2) \wedge \mathbf{g}(p_1) \wedge \mathbf{g}(p_2) \wedge \mathbf{g}(p_3) \wedge \mathbf{g}(p_4) \rightarrow \mathbf{g}(q_2)^{\mathbf{M}})$ says that the domain assumptions $\mathbf{k}(r_1)$ and $\mathbf{k}(r_2)$ and the goals $\mathbf{g}(p_1)$ to $\mathbf{g}(p_4)$ refine $\mathbf{g}(q_2)^{\mathbf{M}}$.

—Logical inconsistency is represented by the **conflict** relation; for example, we assume in Figure 5 that $\mathbf{t}(u_5)$ cannot be carried out together with $\mathbf{t}(u_6)$. The conflict relation between these tasks, drawn in Figure 5, abbreviates the assumption $\mathbf{i}(\mathbf{t}(u_2) \wedge \mathbf{t}(u_4) \rightarrow \bot)$: If we can deduce both, then we will also conclude logical inconsistency.

—Finally, we can have a **preference** relation between requirements to indicate their relative desirability. For example, we write $\mathbf{t}(u_{10}) \succ \mathbf{t}(u_{13})$ to say that making sure $\mathbf{t}(u_{10})$ gets the value True is strictly more desirable than making sure of the same for $\mathbf{t}(u_{13})$. We draw this as in Figure 6.

Preferences are not written in $\mathbf{i}()$ because they differ from implication and conflict: The latter two are used when doing deduction from requirements, whereas preferences are not and are instead used only for the comparison of requirements.

The idea of using only the implication and conflict relations in deductions fits the observation from Robinson et al. [2003] that there are essentially two kinds of relations used in RE: positive and negative. Negative relations exist between requirements that either cannot be satisfied together at all, or can, but not all to the desired extent. Typical kinds of negative relations are conflict [van Lamsweerde et al. 1998] and obstruction [van Lamsweerde and Letier 2000]. Positive relations exist between requirements that can be satisfied together, and these are relations found to be useful in designing or comparing solutions to requirements problems. Typical examples are goal refinement [Darimont and van Lamsweerde 1996], means-ends [Yu and Mylopoulos 1994], and operationalization [Dardenne et al. 1993] relations. Appendix C in the electronic appendices discusses how implication and conflict can be used to define oft-cited positive and negative relations in RE research and thereby illustrates the versatility of the relatively simple requirements ontology we outlined earlier using our classification dimensions on requirements and the relations.

4.3. Comparison of Configurations

Figures 5–6 gave alternative operationalizations of the two mandatory top-level goals $\mathbf{g}(q_1)^{\mathbf{M}}$ and $\mathbf{g}(q_2)^{\mathbf{M}}$.

Each combination of operationalizations that results in the satisfaction of all mandatory goals in Figures 5 and 6 is also called a **configuration**.

There are many alternative configurations that can be identified in Figures 5 and 6. Of these, three are shown in Figures 7–9. They are referred to, respectively, by S_1, S_2, and S_3; Table I shows the domain assumptions and tasks included in each.

Given several configurations, we want to be able to compare them in order to choose the most desirable one. Figures 7, 8, and 9 each show one configuration. Each configuration satisfies the two mandatory top-level goals, $\mathbf{g}(q_1)^{\mathbf{M}}$ and $\mathbf{g}(q_2)^{\mathbf{M}}$. One difference between S_2 and both S_1 and S_3, is that, in S_2, the dispatch software does not keep track of open incident locations automatically whether because of malfunction or otherwise, thus requiring the control assistants to manually keep track of incident locations. Another difference of S_2 from both S_1 and S_3 is that we could switch to S_2 if it was no longer possible to use dispatch software to keep track of which ambulances are available; that is, if $\mathbf{t}(u_7)$ and/or $\mathbf{t}(u_8)$ fail, we would need to switch to $\mathbf{t}(u_9)$. Notice that switching to $\mathbf{t}(u_9)$ does not necessarily mean always switching to S_2 because there can be other configurations, not shown in this article, that include $\mathbf{t}(u_9)$ and are different from S_2.

Both Figures 5 and 6 include information for the comparison of configurations:

—There are softgoals, such as $\mathbf{s}(\tilde{w}_1)$, which come with a satisfaction function μ over the quantitative variable t_x. Such softgoals are used to compare configurations as follows: Different configurations will result in different values of t_x and different values $\mu(t_x)$.

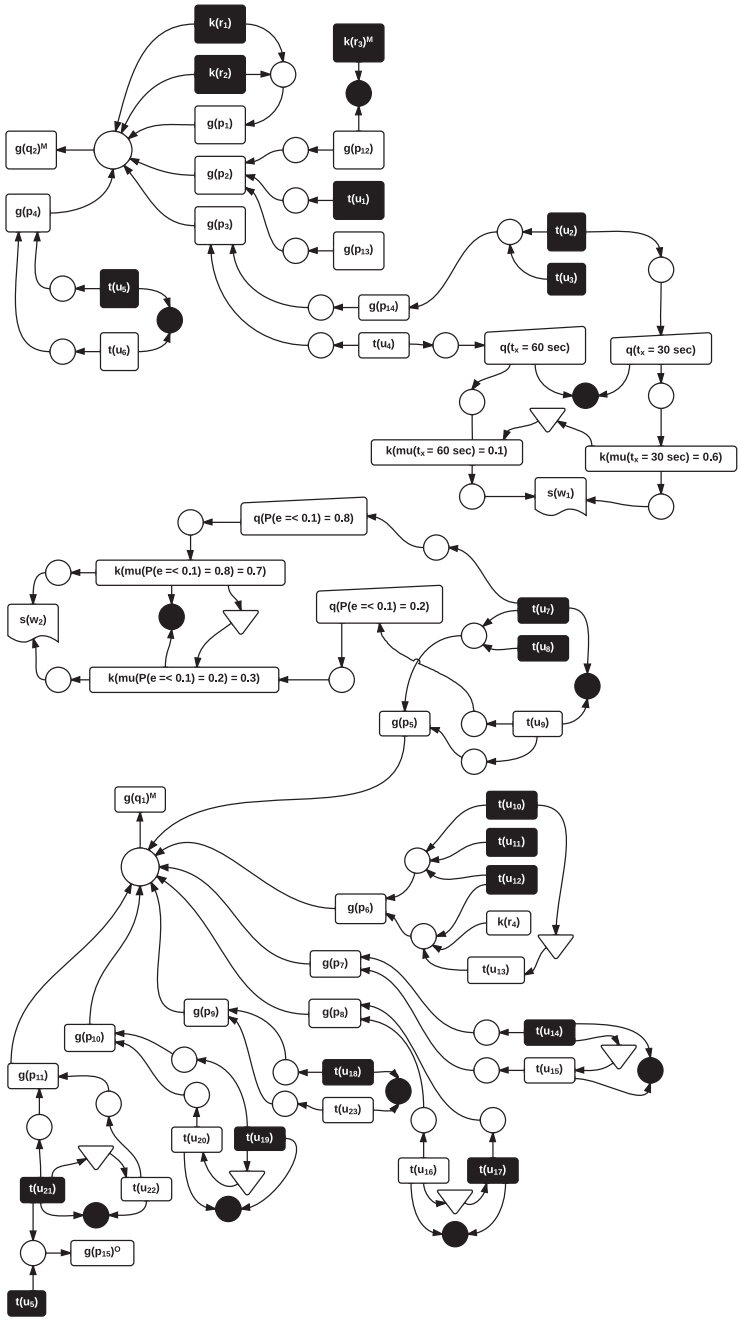

Fig. 7. One configuration, denoted S_1 for LAS requirements from Figures 5 and 6. Highlighted domain assumptions and tasks operationalize the two mandatory top-level goals, $\mathbf{g}(q_1)^{\mathbf{M}}$ and $\mathbf{g}(q_2)^{\mathbf{M}}$.

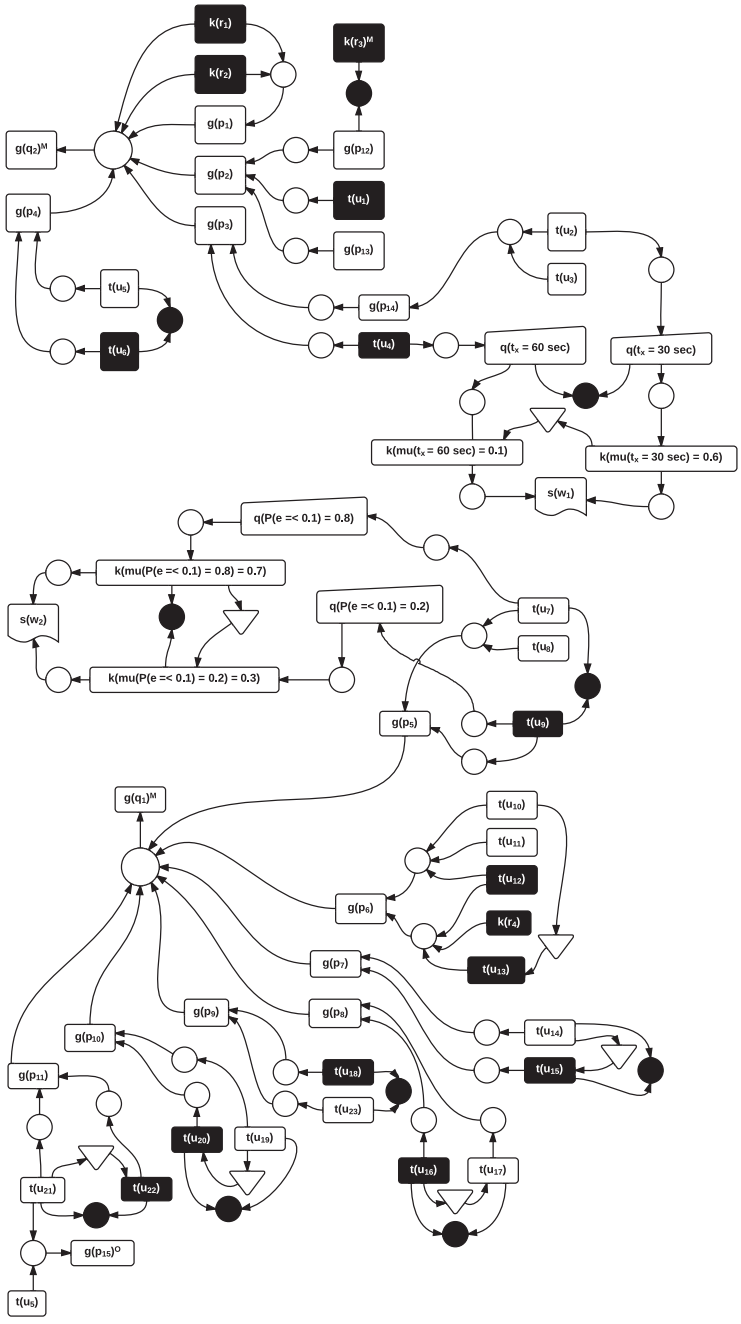

Fig. 8. Another configuration, denoted S_2, for LAS requirements from Figures 5 and 6. Highlighted domain assumptions and tasks operationalize the two mandatory top-level goals, $\mathbf{g}(q_1)^{\mathbf{M}}$ and $\mathbf{g}(q_2)^{\mathbf{M}}$.

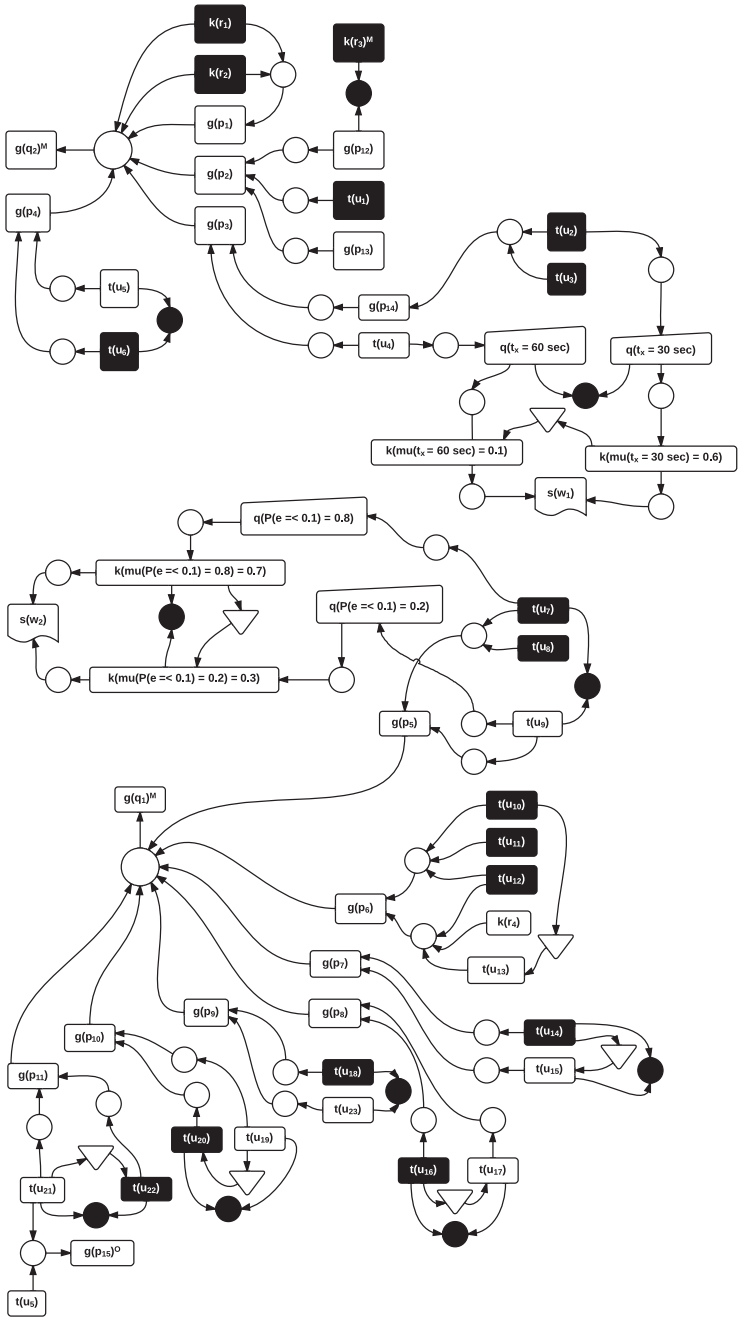

Fig. 9. A third configuration, denoted S_3, for LAS requirements from Figures 5 and 6. Highlighted domain assumptions and tasks operationalize the two mandatory top-level goals, $\mathbf{g}(q_1)^{\mathbf{M}}$ and $\mathbf{g}(q_2)^{\mathbf{M}}$.

Table I. Content of Each Configuration Shown in Figures 7–9

		S_1	S_2	S_3
Every call is switched to the ambulance dispatch center	$\mathbf{k}(r_1)$	x	x	x
No calls are dropped because of timeout	$\mathbf{k}(r_2)$	x	x	x
Callers report imprecise incident location	$\mathbf{k}(r_3)^{\mathbf{M}}$	x	x	x
Control assistant uses heuristics that cannot be automated	$\mathbf{k}(r_4)$		x	
Make the map searchable through dispatch software	$\mathbf{t}(u_1)$	x	x	x
Open incident locations visible in the dispatch interface	$\mathbf{t}(u_2)$	x		x
Update open incident locations at each new incident	$\mathbf{t}(u_3)$	x		x
Every control assistant manually tracks locations	$\mathbf{t}(u_4)$		x	
Fill out incident report via the software	$\mathbf{t}(u_5)$	x		
Fill out a paper incident report form	$\mathbf{t}(u_6)$		x	x
Make the list of all available and of all allocated ambulances available through the dispatch interface	$\mathbf{t}(u_7)$	x		x
Update the list of all available and allocated ambulances every time an ambulance is assigned to an incident	$\mathbf{t}(u_8)$	x		x
Every control assistant manually keeps track of available ambulances	$\mathbf{t}(u_9)$		x	
Dispatch software ranks ambulances from best to worst for a given incident location	$\mathbf{t}(u_{10})$	x		x
Dispatch software displays the ranking of ambulances to control assistant	$\mathbf{t}(u_{11})$	x		x
Control assistant chooses available ambulances to dispatch	$\mathbf{t}(u_{12})$	x	x	x
Dispatch software does not recommend ambulances to choose	$\mathbf{t}(u_{13})$		x	
Control assistants use dispatch software to assign the chosen ambulance to incident	$\mathbf{t}(u_{14})$	x		x
Control assistant verbally informs other assistants of which ambulances are assigned	$\mathbf{t}(u_{15})$		x	
Control assistant mobilizes the ambulance via radio with the ambulance	$\mathbf{t}(u_{16})$		x	x
Control assistant uses dispatch software to mobilize the ambulance, without radio	$\mathbf{t}(u_{17})$	x		
Ambulance staff confirm mobilization via dispatch software client in the ambulance	$\mathbf{t}(u_{18})$	x	x	x
Ambulance staff confirm arrival at incident location via dispatch software client, in the ambulance	$\mathbf{t}(u_{19})$	x		
Ambulance staff confirm arrival at incident location via radio	$\mathbf{t}(u_{20})$		x	x
Close incident report via the software	$\mathbf{t}(u_{21})$	x		
Close incident report by filling out a paper form	$\mathbf{t}(u_{22})$		x	x
Ambulance staff confirm mobilization via radio	$\mathbf{t}(u_{23})$			

According to Figures 7–9, task $\mathbf{t}(u_2)$ in configurations S_1 and S_3 satisfies the quality constraint $\mathbf{q}(t_x = 30 \text{ sec})$, whereas the task $\mathbf{t}(u_4)$ in configuration S_2 satisfies the quality constraint $\mathbf{q}(t_x = 60 \text{ sec})$. Using μ, we then know that satisfaction is higher when $\mathbf{q}(t_x = 30 \text{ sec})$ is satisfied than when $\mathbf{q}(t_x = 60 \text{ sec})$ is satisfied. Finally, because the softgoal says that we prefer to quickly establish double location, the softgoal gives the preference $\mathbf{k}(\mu(t_x) = 0.6)^{\mathbf{M}} \succ \mathbf{k}(\mu(t_x) = 0.1)^{\mathbf{M}}$, indicating, for this softgoal only, that there are better configurations than S_2. We later revisit how the softgoals, in general, give us such preferences.

—Preference relations are not only generated via softgals, as in the preceding case. The preference $\mathbf{t}(u_{14}) \succ \mathbf{t}(u_{15})$ is independent of softgoals and will be satisfied if we choose a configuration that includes $\mathbf{t}(u_{14})$.

—Different configurations may satisfy different optional goals. The optional goal $\mathbf{g}(p_{15})^{\mathbf{O}}$ is only satisfied by configuration S_1 in Figure 7.

5. KEY IDEAS IN ADAPTIVE SYSTEMS RE

This section reviews key ideas in RE for Adaptive System. The language introduced in the previous section, together with the key ideas presented in this section, form the basis for defining the RPAS in Section 6.

5.1. Monitoring and Reconciliation Tactics

Adaptive behavior entails monitoring of requirements and control over a collection of behaviors, each of which can satisfy requirements. Fickas and Feather [1995] suggested that for this we must start with a specification of all alternative behaviors to be considered in a requirements model; that is, alternative refinements and operationalizations of all requirements. The model is then a source of *monitored* variables, the observed values of which will trigger behaviors, called "reconciliation tactics," that are intended to change the values of *control* variables that can affect how one operationalizes a requirement. This forms a *control loop* through which the system observes and reacts by switching from one configuration to another at runtime.

For illustration, consider the goal $\mathbf{g}(p_3$: Check if double location) for LAS in Figure 5, which has two operationalizations: One way to satisfy $\mathbf{g}(p_3)$ is by executing both tasks $\mathbf{t}(u_2)$ and $\mathbf{t}(u_3)$; the other is via $\mathbf{t}(u_4)$. A reconciliation tactic could be, for example, "if $\mathbf{t}(u_2)$ fails, then operationalize $\mathbf{g}(p_3)$ by $\mathbf{t}(u_4)$."

Feather et al. [1998] combined the requirements monitoring mechanisms from Fickas and Feather with KAOS [Dardenne et al. 1993] to make an RE framework in which requirements can be converted into constraints on events, and events are monitored at runtime. Robinson [2006] suggested a requirements monitoring platform that can be combined with a requirements modeling language via translation rules between the expressions of the latter and those accepted by the former, then suggested how to use Object Constraint Language as the modeling language for the platform [Robinson 2008].

5.2. Computed Reconciliation Tactics

An important characteristic of a reconciliation tactic (and the same applies to adaptation goals [Baresi et al. 2010]) is that it is defined in relation to *local* requirements; that is, it says what to do in relation to a single requirement that is not being satisfied. There are two problems with defining a reconciliation tactic by looking mainly at local requirements:

—It is not clear that following any or every reconciliation tactic will make sure that the new system configuration is consistent with the requirements. It is not clear what happens if the reconciliation tactic works locally, but activates tasks that are inconsistent with some other already active part of the system.
—One local change may seem fine by itself and may result in a new and consistent configuration, but this new configuration may be a dominated one: that is, maybe instead of satisfying one reconciliation tactic and obtaining some quality level, we could have used several adaptations, which result in a configuration that ensures higher quality. The local approach to the definition of reconciliation tactics comes with the risk of adapting to a suboptimal or inconsistent configuration.

We therefore contend that instead of defining reconciliation tactics by considering their impact only on a small set of requirements, *reconciliation tactics should be defined so as to ensure that the configuration obtained after adaptation via reconciliation tactics is desirable in its entirety in terms of various criteria used to compare alternative requirements configurations* (including, e.g., preferences over requirements, probabilities

of requirements satisfaction, and so on). This has two consequences on the formulation of the problem and solution concepts in Adaptive Systems RE:

(1) We do not define reconciliation tactics *while writing the requirements model* (or, in other words at design time), but we **compute** reconciliation tactics after we have identified at least some configurations from a set of alternative configurations, all of which satisfy some set of properties (e.g., are consistent) and satisfy some preferences. Instead, we define evolution requirements that the computed reconciliation tactics have to satisfy. Consider Figures 7–9, where each configuration satisfies the mandatory goals from in Figures 5–6 for the LAS system: For example, if the system is configured according to S_1, and $\mathbf{t}(u_5)$ fails, then it may be more desirable to adapt to S_3 than to S_2, so that reconciliation tactics can be identified directly by comparing S_1 to S_3 and defining what tasks to add and delete in order to switch from $\mathbf{t}(u_{21})$ to $\mathbf{t}(u_{22})$, from $\mathbf{t}(u_{19})$ to $\mathbf{t}(u_{20})$, from $\mathbf{t}(u_{17})$ to $\mathbf{t}(u_{16})$, and from $\mathbf{t}(u_5)$ to $\mathbf{t}(u_6)$.
(2) The local/global distinction has an impact on how we *select* configurations: It is likely that the system *can* adapt from one to *one among several alternative configurations*; for example, if $\mathbf{t}(u_{17})$ fails in configuration S_1, then we can consider both S_2 and S_3, which use the alternative $\mathbf{t}(u_{16})$. Commitment to one of these configurations should take into account the evolution requirements and the consequences of adapting to the configuration: For example, perhaps configuration S_2 is more desirable than S_3 if only these two are compared, but we may know that if we select S_2, then, in the long term, it is not sustainableif we are to keep satisfying some quality constraint; or, we may have an evolution requirement that says that, if $\mathbf{t}(u_{17})$ fails, then the configuration to adapt to must exclude $\mathbf{t}(u_{23})$.

5.3. Probabilistic and Fuzzy Relaxation of Requirements

Adaptive behavior, by definition, means that the system cannot satisfy all the specific requirements and meet the desired quality levels all of the time—it may do worse, but it may also do better.

Relaxation of requirements has been suggested to cover both cases: A requirement can be relaxed to allow the system to fail it, but we restrict how often it can do so (via probabilistic relaxation), or to avoid overly constraining the system when it may deliver higher quality than expected (via fuzzy relaxation). Relaxation of requirements is closely related to the evaluation of their satifaction because one aim of relaxation is to quantify the degree of satisfaction of a requirement. Two broad approaches to relaxation and satisfaction evaluation have been suggested based on *objective criteria* (where measurement is in terms of quantifiable phenomena in the environment or the system) or on *subjective criteria* (where a number is given to reflect personal impression of satisfaction, regardless of measurable phenomena in the domain). Souza et al. [2012] called awareness requirements those requirements that one obtains after relaxing some original requirements.

An important approach to relaxation based on objective criteria was suggested by Letier and van Lamsweerde [2004], who extended the goal models of KAOS to allow the specification of measures on system behaviors and probability functions over the values of these measures. Using such a formalism, they can state, for example, that in the LAS system, 95% of ambulances should reach the designated place of incident in at most 14 minutes after an incident is reported. Values of measures and probabilities are computed for every requirements configuration, all of which are inputs to a decision-making process that aims at selecting a single configuration.

Approaches based on subjective criteria were revived recently for RE of Adaptive Systems. Whittle et al. [2010] associate fuzzy membership functions with functional requirements in order to allow and quantify departures from the ideal satisfaction of

such requirements. The fuzzy membership function can be interpreted as a satisfaction function, and they use modalities to relax requirements: For example, to relax the task $\mathbf{t}(u_{18})$ in Figure 6, we would write [As early as possible]$\mathbf{t}(u_{18})$. This means that we would add a function μ , which returns a level of satisfaction when given a duration between time "now" and the time when $\mathbf{t}(u_{18})$ is executed, whereby μ "has its maximum value at 0 (i.e., at the current point in time) [and] tails off gradually ad infinitum (i.e., it has a triangular membership graph that is asymptotic)" [Whittle et al. 2010].

Baresi et al. [2010] also use fuzzy membership functions in relaxing requirements and so are able to treat functional requirements as quantitative ones. They introduce degrees of satisfaction between the binary 0 (for violated) and 1 (satisfied); thus, adjusted requirements are associated to "adaptive" requirements that relate ranges of values of the fuzzy membership function to trigger switching between configurations. For example, if there is a quality constraint $\mathbf{q}(v < 6 \text{ hrs})$—which would be written as a goal $\mathcal{G}(v < 6\text{hrs})$ in their notation because they allow quantitative variables in goals—then relaxing it would amount to replacing it with $\mathbf{q}(v <_f 6\text{hrs})$ (in their notation, $\mathcal{G}(v <_f 6\text{hrs})$), where $\leq_f$ is a fuzzy operator. Their interpretation of $v <_f 6\text{hrs}$ is that there is a fuzzy membership function μ that returns the level of satisfaction as a function of v, and the shape of μ is predefined (for $<_f$ in $\mathcal{G}(v <_f 6\text{hrs})$, it is positive and constant until $v = 6$, then decreases up to the satisfaction value 0 for some $v > 6$).

6. CONFIGURABLE SPECIFICATIONS, CONFIGURATIONS, AND EVOLUTION REQUIREMENTS

The RPAS and the corresponding Configurable Specification concept are needed because monitoring, control, evolution requirements, and probabilistic and fuzzy relaxation are not formulated in the standard RP. These features affect the properties that solutions need to satisfy and influence the criteria used to compare alternative solutions. This section presents and discusses the formal properties of the Configurable Specification concept as the solution concept for RPAS. In Section 7, we compare RPAS and Configurable Specification concepts to Zave and Jackson [1997] and Techne [Jureta et al. 2010].

The requirements problem is a *design* and *decision-making* problem. By design, we mean that alternative solutions to the problem are not available but need to be created: For example, in Figures 5–6, we start from two broad goals, $\mathbf{g}(q_1)^{\mathbf{M}}$ and $\mathbf{g}(q_2)^{\mathbf{M}}$, so that we need to elicit further requirements and information, refine goals, identify conflicts, and define operationalizations, all of which fall under design.[1] Decision making requires the identification of criteria for the comparison of alternative solutions and the application of a decision rule to rank alternative solutions from the most desirable to the least desirable.

To provide definitions, we start with the notion of a requirements database Δ—a set of requirements and the relations between them. For example, the requirements database for LAS in this article includes all requirements and relations in Figures 5 and 6.

The novelty of the RPAS and Configurable Specification concepts lies in the properties of requirements configurations, evolution requirements, and in how Configurable Specifications can be ranked. We discuss these in turn next.

6.1. Properties of Requirements Configurations

Let us first look at the configurations appearing in a Configurable Specification.

[1]Note that "design" is typically used for steps of systems development that come after RE, such as architectural design. The term "design" is used here in a broader sense to encompass all activities (e.g., elicitation, validation) necessary to make alternative options on which we then need to decide.

Definition 6.1. A **configuration** in a Configurable Specification is a set S of domain assumptions and tasks in Δ such that S satisfies the Consistency, Functional threshold achievement, Quantitative threshold achievement, Conformity, Dominance, and Minimality conditions.

6.1.1. Consistency. Every requirements configuration S must be consistent; that is, $S \not\vdash_{\tau} \bot$, where $\vdash_{\tau}$ is the consequence relation defined in Section A.2. The Adaptive System should work to satisfy a consistent set of requirements. In Figures 7–9, every configuration is such that there are no conflicts between its members. Remark that in a Configurable Specification, $\bigcup S$ can and will often be inconsistent because alternative configurations need not be consistent together. If we adapt from S_1 to S_2, note that $S_1 \cup S_2 \vdash_{\tau} \bot$, since there are conflicts (e.g., between $\mathbf{t}(u_{16})$ and $\mathbf{t}(u_{17})$).

6.1.2. Functional Threshold Achievement. Every configuration S should operationalize every mandatory goal in Δ—a set written as $\Delta^{\mathbf{M}}_{\mathbf{g}}$. An operationlization defines tasks and domain assumptions with which we can fulfill a goal. For example, in Figure 5, if the task $\mathbf{t}(u_1)$ is satisfied, then $\mathbf{g}(p_2)$ is satisfied, so that the former operationalizes the latter. Since a requirement can be operationalized by alternative sets of tasks and domain assumptions, we have a function $\mathsf{Op}(\varphi)$ (defined in Section A.3), which, given a mandatory goal $\varphi \in \Delta^{\mathbf{M}}_{\mathbf{g}}$, returns the set of all operationalizations of φ. For example, $\mathsf{Op}(\mathbf{g}(p_7))$ includes two operationalizations of $\mathbf{g}(p_7)$ in Figure 6. This property is restricted to *functional* requirements, for which satisfaction is binary, and thus to goals, since a goal is either satisfied or not. The property requires *threshold* satisfaction because it requires only that all *mandatory* goals be satisfied by a configuration.

A consistent set of requirements must meet both the functional and quantitative thresholds, for it otherwise fails to satisfy the requirements that must be satisfied.

6.1.3. Quantitative Threshold Achievement. Every configuration S should operationalize every mandatory quality constraint in Δ—a set written as $\Delta^{\mathbf{M}}_{\mathbf{q}}$. Once again, the operationalization function $\mathsf{Op}(\varphi)$ is used to return all sets of operationalizations of a quality constraint $\varphi = \mathbf{q}(\alpha)$, whereby some set Π is such an operationalization iff it includes domain assumptions and tasks in which there are assignments of values to all variables in α, and these assignments are such that the constraint in α is satisfied.

Quality constraints identify desirable values of quantitative variables. For example, quality constraints may thus place constraints on values of measures of some behavior of the system-to-be. In LAS, and in relation to the response to emergency calls and the mobilization of ambulances, the time that these activities take is a crucial part of quality of service. We can use the following variables, where c identifies a call and e a unique incident:

$t_{1,c}$: Time the caller waits for a control assistant;
$t_{2,c}$: Time to identify incident location;
$t_{3,c}$: Time to fill out incident report;
$t_{4,e}$: Time to mobilize an ambulance;
$t_{5,e}$: Time for the mobilized ambulance to arrive and confirm arrival at incident location.

Instead of defining bounds on all of these variables, a government standard may not be as specific and may impose a maximal time over a subset of activities that need to be executed between the placement of the emergency call and the confirmation of arrival of an ambulance to location; for example:

$\mathbf{q}(t_{6,c} \leq 3\text{min}: t_{6,c}$ is the duration between the switching of the call c to the dispatch center to the mobilization of the ambulance to the incident location)

where it is clear that the value of $t_{6,c}$ depends on $t_{1,c}$, $t_{2,c}$, $t_{3,c}$, and $t_{4,e}$. If we assume that $t_{6,c} = t_{1,c} + t_{2,c} + t_{3,c} + t_{4,e}$, then we add this to the requirements database as a domain assumption over quantitative variables, $\mathbf{k}(t_{6,c} = t_{1,c} + t_{2,c} + t_{3,c} + t_{4,e})$. The domain assumption says that there is a *quantitative refinement* relation between variables, and it specifies the functional relation between the refined $t_{6,c}$ and the variables refining it. In contrast to the refinement relation, quantitative refinement is not over requirements but over variables in quantitative variable requirements.

Although it may be useful to set a precise bound on the value of an aggregate variable, as $t_{6,c}$ in the preceding paragraph, it may be more interesting to set bounds relative to the values of other variables. For example, if we want $t_{2,c}$ to be at most 110% of its average value over the past three months, then we write

$\mathbf{q}(t_{2,c} \leq (1.1/3)(v_{1,m-3} + v_{1,m-2} + v_{1,m-1}))$;

$\mathbf{k}(v_{1,m} = n(m)^{-1} \sum_{i=1}^{n(m)} \sum_{c=1}^{c(i)} t_{2,c})$, where m is the month identifier, $n(m)$ the number of days in m, c is the call identifier on a given day, $c(i)$ is the total number of calls received on day i of month m;

where the domain assumption is a quantitative refinement of $v_{1,m}$, the average time, over a month, that it takes to identify the location.

The quality constraints and quantitative refinements need to be related to goals, tasks, and domain assumptions in order to determine if the former are satisfied. If there is a running system, the values of $t_{2,c}$ are recorded, and it is straightforward to check if the constraint $t_{2,c} \leq (1.1/3)(v_{1,m-3} + v_{1,m-2} + v_{1,m-1})$ is satisfied.

Before the system is in operation, we can simulate values of $t_{2,c}$ by assuming that $t_{2,c}$ is a random variable that has some probability distribution, so that we would have $\mathbf{k}(t_{2,c} \backsim \mathcal{N}(60sec, 45sec^2))$ if we assume that $t_{2,c}$ follows a normal distribution with mean 60sec and variance 45sec^2 when the task $\mathbf{t}(u_1)$ is satisfied, which we model by a refinement $\mathbf{k}(\mathbf{t}(u_1) \rightarrow \mathbf{k}(t_{2,c} \backsim \mathcal{N}(60sec, 45sec^2)))$. This assumption may be based on data from a pilot study, from expert opinion, or from data on systems that also satisfy $\mathbf{t}(u_1)$ and are already in operation.

6.1.4. Conformity. Every configuration S must include all strict domain assumptions and all mandatory tasks, that is, $\Delta_{\mathbf{k}}^{\mathbf{M}} \cup \Delta_{\mathbf{t}}^{\mathbf{M}} \subseteq S$. This property asks that all strict domain assumptions are not violated, and all mandatory tasks are executed in every configuration.

The Functional and Quantitative threshold achievement and Conformity properties ensure that a configuration satisfies all that *must* be satisfied.

6.1.5. Dominance. Every configuration S must be maximal with regard to optional requirements; that is, $\nexists S' \subseteq \Delta$ such that (i) both S and S' satisfy the Consistency, Functional threshold achievement, Quantitative threshold achievement, and Conformity conditions, (ii) $S \subset S'$, and (iii) $S' \backslash S$ contains optional **k** or **t** requirements.

This condition formalizes the idea of optional requirements, which are desirable to satisfy but can be violated. A configuration should include as many such defeasible domain assumptions and as many optional tasks up to the point at which adding any further defeasible domain assumptions and/or optional tasks violates the Consistency, Functional, and Quantitative threshold achievements and Conformity properties. The Dominance property ensures that every configuration is Pareto efficient with regards to optional requirements because this condition makes it impossible to add optional domain assumptions and tasks to any S and still ensure that S is a configuration.

6.1.6. Minimality. A set S satisfying all the properties just described must be minimal in order to qualify as a configuration. Minimality requires that a configuration

includes only the domain assumptions and tasks that are needed to satisfy exactly the Consistency, Functional threshold achievement, Quantitative threshold achievement, Conformity, and Dominance properties.

All three configurations S_1, S_2, and S_3 in Figures 7–9 satisfy all six properties.

In terms of our discussion of RE for Adaptive System in Section 3, we have defined here the properties that are satisfied by every individual configuration shown in Figure 1. We now consider switching between configurations via evolution requirements.

6.2. Evolution Requirements

An evolution requirement specifies the differences that are allowed between the previous configuration and the next configuration to switch to. It does so by describing the presence/absence of certain requirements in pairs of related configurations, and it is triggered by the failure of monitored assumptions, tasks, or quality constraints. Any requirement monitored for failure is an **awareness requirement**.

In Figure 7, if task $\mathbf{t}(u_{21})$ fails, we might want to replace $\{\mathbf{t}(u_{19}), \mathbf{t}(u_{17}), \mathbf{t}(u_5)\}$ with $\{\mathbf{t}(u_{22}), \mathbf{t}(u_{20}), \mathbf{t}(u_{16}), \mathbf{t}(u_6)\}$. In general, we will therefore want to specify the conditions monitored for failure and the changes (additions, removals) that must be seen in the successor configuration when failure occurs. This evolution requirement can then tell us that if we are running the configuration S_1 in Figure 7 and the system fails to satisfy $\mathbf{t}(u_{21})$, then it could adapt from S_1 to S_3. Note that an evolution requirement $\langle T, A, D\rangle$ might then be thought of as a simple operator in AI planning, where T is trigger condition, whereas A and D are add/delete lists.

Definition 6.2. An **evolution requirement** is an operator of the form $\langle T, A, D\rangle$, where T is a set of requirements present in the initial configuration (which are monitored to fail) whereas A and D are, respectively, sets of requirements that must be present and absent in the final configuration if the operator applies.

Note therefore that if operator $\langle T, A, D\rangle$ applied in going from configuration S_i to S_{i+1}, then $T \cup D \subseteq S_i$ and $A \cap S_i = $ must hold and analogously with S_{i+1}. Note also that it is quite acceptable that multiple operators apply at the same time when connecting S_i to S_{i+1} and that (as with the so-called ramification problem in AI) there may be additional changes needed in S_{i+1} in order to make it a valid configuration.

In this sense, requirements are monitored, whereas evolution requirements act as a control mechanism that guides adaptation by changing the target requirements configuration that the Adaptive System needs to satisfy.

6.3. Comparison of Configurations in a Configurable Specification

Given several configurations, the configuration that ranks best is the optimal configuration. Ranking of configurations requires the identification of comparison criteria and the application of a decision rule that establishes a ranking of configurations in a Configurable Specificationon the basis of criteria.

Comparison criteria are either optional requirements or preferences.

Preferences can be individually added to Δ or can be obtained through the relaxation of requirements or from softgoals. Preferences are illustrated in Figures 5 and 6, where, for example, $\mathbf{t}(u_{16})$ is strictly preferred to $\mathbf{t}(u_{17})$, which indicates that, for this criterion only, a configuration having the former task is strictly more desirable than the configuration having the latter task.

6.3.1. Preferences through Probabilistic Relaxation. Continuing the example from Section 6.1.3, the upper bound on $t_{2,c}$ in $\mathbf{q}(t_{2,c} \leq (1.1/3)(v_{1,m-3} + v_{1,m-2} + v_{1,m-1}))$ may

still be too idealistic because callers may provide information of very different quality about the incident location.

Probabilistic relaxation of a quality constraint is done in two steps:

(1) The variable constrained in **q** is redefined as a random variable, and an assumption is made on the probability distribution of that random variable. Adding $\mathbf{k}(t_{2,c} \backsim \mathcal{N}(60sec, 45sec^2))$ to Δ makes $t_{2,c}$ into a random variable that follows a normal distribution.
(2) The quality constraint to be relaxed is removed from Δ, and a new quality constraint is added. The new constraint specifies a bound not on the value of the now random variable, but on the probability that its value is in some range. Since we made $t_{2,c}$ into a random variable in the first step, we now replace $\mathbf{q}(t_{2,c} \leq (1.1/3)(v_{1,m-3} + v_{1,m-2} + v_{1,m-1}))$ with $\mathbf{q}(P(t_{2,c} \leq (1.1/3)(v_{1,m-3} + v_{1,m-2} + v_{1,m-1})) \geq 0.90)$; that is, we now require that the minimal probability should be 0.90 for $t_{2,c}$ to be at most 110% of its three-month average.

Random variables and the quality constraints thereon are important for decision making because we use them to set desired probability levels over random variables. In more informal terms, this means that we can write quality constraints and domain assumptions that reflect, respectively, the confidence that we desire to reach in relation to system behaviors and the confidence that we actually have.

Figure 6 indicates that $\mathbf{t}(u_7)$ operationalizes the quality constraint $\mathbf{q}(P(e \leq 0.10) = 0.8)$, whereas $\mathbf{t}(u_8)$ operationalizes $\mathbf{q}(P(e \leq 0.10) = 0.2)$. The two quality constraints suggest that the two tasks result in different probabilities of having less than 10% of erroneous identifications of available ambulances. Figure 6 further indicates the level of satisfaction with each of the two probability levels.

If we needed to do probabilistic relaxation of a goal, rather than of a quality constraint, this process is changed as follows. Given a goal, say $\mathbf{g}(p_9)$ in Figure 5, to perform probabilistic relaxation, we remove it from the requirements and add a quality constraint $\mathbf{q}(p'_9 \geq 90\%)$, where p'_9 is the percentage of confirmed mobilizations over all mobilizations, and 90% is a threshold value. We then proceed to relax this quality constraint in the same way as earlier.

This same approach applies when doing probabilistic relaxation of a domain assumption that is over a propositional variable, such as $\mathbf{k}(r_3)^{\mathbf{M}}$.

When we do probabilistic relaxation of an optional requirement, the new requirement obtained by relaxation is also optional; if it was mandatory, the new requirement would also remain mandatory.

6.3.2. Preferences through Fuzzy Relaxation. Fuzzy relaxation of a quality constraint involves two steps:

(1) The quality constraint is removed, and a fuzzy membership function is defined over the variable from the removed quality constraint. In the second step, a softgoal is defined over the variable from the relaxed requirement. Consider again $\mathbf{q}(t_{2,c} \leq (1.1/3)(v_{1,m-3} + v_{1,m-2} + v_{1,m-1}))$, and we now apply fuzzy relaxation. We start by removing this quality constraint from Δ. A fuzzy membership function over $t_{2,c}$, denote it $\mu(t_{2,c})$, depends on how stakeholders evaluate, in terms of desirability, the various values of $t_{2,c}$. For example, we could use $\mu(t_{2,c}) = e^{-t_{2,c}}$, so that the higher the value of $t_{2,c}$, the lower $\mu(t_{2,c})$ is, which reflects the idea that the more time it takes to identify an incident location, the more the stakeholders are dissatisfied, whereby their satisfaction increases as $t_{2,c}$ approaches 0. If we adopt μ as defined, we have removed the quality constraint on $t_{2,c}$, and we quantify satisfaction as a function of $t_{2,c}$.

(2) To finish with the fuzzy relaxation of the quality constraint on $t_{2,c}$, a softgoal needs to be added to the requirements database over values of $t_{2,c}$. Softgoals have had various definitions in RE, but there seems to be agreement on their two properties: (i) they are used for the comparison of alternatives, and (ii) they are vague because they refer to some desirable values of variables, even though it may not be clear which exact values or of which variables. For example, *Response time should be low* is a typical softgoal in which the variable is suggested, but it is not clear which specific range of its values qualify as *low*; in the softgoal *High safety*, not only is it not clear how to measure *safety*, it is also not apparent when *safety* is *high*. When used in fuzzy relaxation, a softgoal is defined over a known variable: In our example, it is reasonable to prefer lower over higher values of $t_{2,c}$, and we consequently have the softgoal

$\mathbf{s}($ LET $x_1 \stackrel{def}{=} \text{VAL}(S_1, t_{2,c})$ AND $x_2 \stackrel{def}{=} \text{VAL}(S_2, t_{2,c})$;
IF $\mu(x_1) > \mu(x_2)$ THEN ADD
$\{\mathbf{k}(t_{2,c} = x_1), \mathbf{k}(t_{2,c} = x_2), \mathbf{k}(t_{2,c} = x_1) \succ \mathbf{k}(t_{2,c} = x_2)\}$
TO Δ;)

where $\text{VAL}(S_1, t_{2,c})$ returns the value of $t_{2,c}$ in configuration S_1. Although this formulation seems very different from saying "Low $t_{2,c}$," this is precisely what it does, as long as $\mu(t_{2,c})$ decreases when $t_{2,c}$ increases. $\succ$ denotes the strict preference relation, and $\mathbf{k}(t_{2,c} = x_1) \succ \mathbf{k}(t_{2,c} = x_2)$ says that $\mathbf{k}(t_{2,c} = x_1)$ is strictly more desirable than $\mathbf{k}(t_{2,c} = x_2)$.

The important point is that the softgoal specifies a macro, and the macro generates domain assumptions and preference relations. For any two configurations S_1 and S_2, the macro compares the satisfaction (returned by satisfaction function μ) with values that $t_{2,c}$ has in each of those two configurations. Depending on the comparison between these values, the macro adds two domain assumptions and a preference relation to Δ. The reason we add them to Δ and not individual configurations is because we can add one of these domain assumptions only if doing so does not violate the conditions that a configuration must satisfy (i.e., adding one of these domain assumptions may make a configuration inconsistent and thus no longer a configuration at all).

This macro ensures that if we compare two configurations, S_1 and S_2, and $t_{2,c}$ obtains the value x_1 in S_1 and the value x_2 in S_2, then we will prefer *over this criterion (independently of other criteria)* the configuration in which $t_{2,s}$ obtains the value that results in higher satisfaction. The macro thus conveys the idea that, whenever we are given two values of $t_{2,c}$, we prefer the one that we are more satisfied with.

Fuzzy relaxation of a quality constraint over a variable v thus works by (i) removing the quality constraint on v, (ii) adding a fuzzy membership function $\mu(v)$ on v, (iii) interpreting $\mu(v)$ as the level of satisfaction with the value v, and (iv) adding a softgoal macro that generates preference relations that reflect the shape of μ.

6.3.3. Preferences Based on Softgoals. Not all softgoals are macros used in fuzzy relaxation. A softgoal can be introduced in Δ even if we do not know the exact variable it refers to or the exact range of values that it makes desirable. Very broad softgoals are allowed, such as

$\mathbf{s}(\tilde{p}_{17}$: Ambulances should arrive quickly at incidents)

where we write the tilde over p because the content of the softgoal is not a propositional variable, as in a goal or task, because it is not clear from the statement referred to by $\tilde{p}_{17}$ how to measure the time of arriving at incident scenes (e.g., does it begin at call reception or at ambulance mobilization?), and it is not clear when a time to arrive at an incident counts as *quickly*. There are two ways to **approximate** such softgoals: by refinement or by satisfaction functions, and both can result in preferences being added to Δ.

When a softgoal is refined, it is treated just as any other requirement. For example, we may define the following refinement of $\mathbf{s}(\tilde{p}_{17})$

$$\mathbf{q}(t_{7,e} \leq 15\text{min})$$
$$\mathbf{k}(\mathbf{q}(t_{7,e} \leq 15\text{min}) \rightarrow \mathbf{s}(\tilde{p}_{17}))$$
$$\mathbf{k}(t_{7,e} = t_{1,c} + t_{2,c} + t_{3,c} + t_{4,e} + t_{5,e})$$

where we refined the softgoal with the quality constraint over the quantitative variable $t_{7,e}$ (i.e., if we satisfy $\mathbf{q}(t_{7,e} \leq 15\text{min})$; then the if-then domain assumption just presented, by implication, tells us to deduce $\mathbf{s}(\tilde{p}_{17})$), which is itself a function of times introduced before. We could find another refinement of $\mathbf{s}(\tilde{p}_{17})$ and have a preference between the two.

A softgoal can be approximated using a satisfaction function and by proceeding in a similar way to fuzzy relaxation. Given $\mathbf{s}(\tilde{p}_{17})$, we decide which variable it refers to, and we assume it is $t_{7,e}$, such that $\mathbf{k}(\mathbf{q}(t_{7,e} \leq 15\text{min}) \rightarrow \mathbf{s}(\tilde{p}_{17}))$. Second, we remove the softgoal $\mathbf{s}(\tilde{p}_{17})$. Third, we define a function $\mu_1(t_{7,e})$, and we interpret the value given by $\mu_1(t_{7,e})$ as the level of satisfaction with the value $t_{7,e}$. Fourth, we add a softgoal macro over $t_{7,e}$:

$\mathbf{s}($ LET $x_1 \stackrel{def}{=} \text{VAL}(S_1, t_{7,e})$ AND $x_2 \stackrel{def}{=} \text{VAL}(S_2, t_{7,e})$;
IF $\mu_1(x_1) > \mu_1(x_2)$ THEN ADD
$\{\mathbf{k}(t_{7,e} = x_1), \mathbf{k}(t_{7,e} = x_2), \mathbf{k}(t_{7,e} = x_1) \succ \mathbf{k}(t_{7,e} = x_2)\}$
TO Δ;)

This case is also illustrated with the softgoal $\mathbf{s}(\tilde{w}_1)$ and function $\mu(t_x)$ in Figure 5.

6.3.4. Use of Criteria for Ranking. Given a set of criteria and Configurable Specifications, two kinds of *decision rules* are needed: (i) to establish a ranking of configurations from the most desirable to the least desirable in a Configurable Specification, and (ii) to establish a ranking of configuration sequences that can be generated with a Configurable Specification. A configuration sequence is generated from the configurations and evolution requirements in a Configurable Specification.

RPAS, Configurable Specification, and configuration concepts are too general to prescribe a particular decision rule. This also reflects the idea that no universal decision rule can be given: Every decision rule gives one or more criteria priority over others, so that domain- and/or project-independent decision rules should be much more interesting. Note that the decision rule to apply will depend on how tradeoffs are resolved through, among other methods, stakeholder negotiations; these concerns remain outside the scope of RPAS definition.

Various decision rules can be used to rank configurations. Examples are:

—**R1:** Rank highest the configuration S if S maximizes some quantitative variable v, and rank other configurations in a descending order by value of v that each achieves: that is, S so that $\text{MAX}(v)$ iff $\forall S'$, S' is a configuration and $max(\text{VAL}(S, v)) \geq max(\text{VAL}(S', v))$, where max returns the largest constant in a given set. The variable v may be quantifying utility, for example, and its value may be obtained through the aggregation of values of quantitative variables appearing in each configuration; we do not address how exactly the aggregation is done, but contributions on multi-criteria decision making may be relevant.
—**R2:** Same as R1, but rank highest the configuration that minimizes v; that is, rank from $\text{MIN}(v)$ to the configuration that results in the highest value of v.
—**R3:** Same as R1, augmented for the following condition: The chosen configuration should satisfy the maximal number of optional and preferred requirements, where in a preference relation $\phi \succ \psi$ or $\phi \succeq \psi$, ϕ is the preferred requirement.

R1 and R2 ignore preferences and optional requirements that are independent of v, and both give highest priority to the value of a chosen variable. In that case, the decision rule requires us to solve a single-objective optimization problem. The third rule is significantly different because it has two high priorities; namely, the value of the chosen quantitative variable and the number of optional and preferred requirements. The application of the third rule requires the resolution of a multiobjective optimization problem.

The decision rules that rank configuration sequences can concern the maximization or minimization of a quantitative variable over a sequence of configurations from the Configurable Specification. We may ask, with such a decision rule, that optimization satisfies constraints on efficiency of evolution. For example, the decision rule may be as follows:

—**R4:** Choose configuration sequence X if (i) it maximizes the sum of values of the quantitative variable v over all configurations in X, under the constraints that (ii) v must never fall below value q in X, and (ii) no configuration in X differs from the configuration that immediately precedes it by more than w requirements.

R4 is a decision rule pattern in which a concrete rule is obtained by setting q and w to constants. R4 illustrates that constraints need not be limited only on individual configurations, but also on the sequence of configurations because it limits the number of changes that can be performed, which may reflect the necessity to conserve resources during evolution.

The choice of a Configurable Specification does not impose one sequence of configurations that the system should follow: Adaptive behavior means that, for example, if we adopt rule R3, the system will start from the configuration in which the domain assumptions are consistent with the conditions in the environment of the system. It will then adapt according to the satisfaction/failure of monitored requirements. It may be relevant to dynamically (at runtime) compute new configurations, the discussion of which we leave as an open issue.

7. HOW EXACTLY IS RE FOR ADAPTIVE SYSTEM NONSTANDARD?

Monitoring and control of requirements are not ideas new to RE for Adaptive Systems. They have been around at least since the mid-1990s [Fickas and Feather 1995]. In a sense, any change of requirements that is intended to broaden the range of stimuli (desirable or undesirable) that a system can observe and react to is a matter of requirements adaptation. And this is hardly a new issue in RE. Relaxation, whether probabilistic or fuzzy, is also not a new topic; the former has been discussed at least since 2004 [Letier and van Lamsweerde 2004], the latter at least since 1996 [Liu and Yen 1996] in relation to imprecise requirements (i.e., softgoals in this article).

A novelty in Adaptive Systems RE is that, in order to provide methodological support to solve RPAS, it is necessary to design concepts, ontologies, formalisms, and other tools for dealing with both anticipated change in the system environment and unanticipated change.

Design of configurations and Configurable Specifications is a response to anticipated changes. Fuzzy and probabilistic relaxation are responses only to some forms of unanticipated change and are insufficient: They still assume that all changes will produce stimuli that are observable using sensors engineered into the system and that the values of measurements through these sensors and on those stimuli will fall within ranges covered by relaxed requirements.

We have identified no mechanism in this article that deals with change that fails these properties; that is, which involves stimuli for which there are no sensors or

where measurements result in values outside the scales defined through requirements relaxation.

Novelty, then, makes it necessary to have RE frameworks that are capable of both the specification of and reasoning about monitoring and control considerations and of probabilistic and fuzzy relaxation. And if the aim is to also address a broader range of unanticipated changes, then there is a need for conceptual tools that are unclear to us at the moment. This also answers the question of what to retain within, discard from, or add to existing RE modeling languages, frameworks, methodologies to address Adaptive Systems RE: Such artifacts and methodologies are likely to need means for requirements relaxation and the specification of evolution requirements to be relevant for Adaptive Systems RE.

If we take the framework introduced in this article as one that has means for doing RE while taking monitoring and control aspects and fuzzy and probabilistic relaxation into account, then we can give more precise observations on how RPAS and Configurable Specification differ from the ZJ RP and standard RE.

There is in the ZJ RP no information to define comparison criteria. We offered a revised definition of the requirements problem [Jureta et al. 2008] (JMF hereafter) that has a richer ontology (to account for concepts that have established themselves in RE; e.g., goals and softgoals), allows inconsistencies between alternative configurations (each of which satisfies ZJ conditions), and introduces preferences to allow the comparison of configurations in terms of desirability, and we argued that logical consequence in early RE is more adequately described by a nonmonotonic and paraconsistent consequence relation $\sim_?$ [Jureta et al. 2010] than $\vdash$ from classical logic. These revisions make it hard to synthesize the requirements problem as $K, S \vdash R$, since consistency and achievement become only *a subset* of the conditions that a solution should satisfy. Note that alternative solutions are indistinguishable in ZJ RP because of the absence of comparison criteria.

Both the ZJ RP and JMF formulations of the requirements problem place considerable emphasis on properties that can be established from propositional variables alone. With ZJ RP, the aim was to design one solution that is consistent and achieves requirements. With JMF, the aim was to design several configurations, all of which are consistent and achieve a threshold (i.e., all mandatory requirements), then proceed to decision making; that is, compare configurations in terms of which preferred and optional requirements they satisfy, then choose one of them.

In RPAS, the aim is to choose Configurable Specifications, whereby the configurations in a Configurable Specification all are consistent and achieve requirements (as in ZJ RP), but also to distinguish mandatory from optional requirements and those that can be compared in terms of preference (as in JMF). By satisfying Consistency and Functional threshold achievements, each configuration in a Configurable Specification satisfies the ZJ RP conditions. By additionally satisfying Conformity, every configuration satisfies all properties that a candidate solution should satisfy in JMF. From there on, the departures of the configuration, evolution requirement, Configurable Specification, and RPAS concepts from ZJ RP and JMF are:

(1) A configuration satisfies Quantitative threshold achievement, Dominance, and Minimality in addition to Consistency, Functional threshold achievement, and Conformity.
(2) Decision rules for the ranking of configurations and of Configurable Specifications can be formulated as an optimization problem, in which the aim is to optimize one or more quantitative variables. This was not of interest in ZJ RPand was very limited in JMF.

(3) There are new kinds of preferences and tradeoffs to consider. Although a configuration may give an optimal value of a variable in one Configurable Specification, we may prefer another Configurable Specification that fails to include that configuration but ensures some suboptimal but acceptable value of the variable over several configurations in the Configurable Specification. Instead of focusing on the desirability of individual solutions, it is necessary to look at the desirability of Configurable Specifications.

Because of the problem of designing, finding, and ranking Configurable Specifications, RPAS reflects the consequences of adaptation behavior: A Configurable Specification is a set of configurations that need not have a predefined sequence, but must have evolution requirements, whereas the system needs to be designed so that it is able to monitor requirements and satisfy evolution requirements by controlling its behavior by switching between the highest ranking yet feasible configurations.

8. CONCLUSIONS AND OPEN ISSUES

We have presented two results. First, by defining the configuration, evolution requirement, and Configurable Specification concepts, we suggested a precise definition of the RPAS. We related these concepts to the key notions from RE of Adaptive Systems; namely, monitoring, control, evolution requirements, and probabilistic and fuzzy relaxation. This led us to argue that there are fundamental differences between Zave and Jackson's and our conceptions of the requirements problem and the RPAS, its solution concepts, and the decision rules used to rank solutions. Second, we used a simple modeling framework throughout the article, one based on Techne [Jureta et al. 2010], which serves as a proto-framework, being an illustration of features needed in future requirements modeling languages relevant to RE for Adaptive Systems.

There are a number of interesting open issues. The shift to configurations and Configurable Specifications suggests that research into extended planning and single- and multiobjective optimization may be a source for further advances in RE frameworks and tool support. It is necessary to look into how the RPAS relates to mixed-integer and mixed-variable programming, how to make adaptation rules on the basis of Configurable Specifications, and what decision rules may be relevant for decision making in RE. New frameworks for solving RPAS need to be connected to frameworks for controlling the evolution of services, such as those that are capable of addressing structural, behavioral, and quality of service changes [Andrikopoulos et al. 2012]. Work is needed on the integration of probabilistic and fuzzy relaxation, monitoring, and control within *practical* modeling languages for early and late requirements phases.

Our focus in this article was on modeling and reasoning aids for the design of the technological side of evolving sociotechnical systems. Our conceptualization of the RPAS did not pay particular attention to potentially complicated interactions between creating Adaptive Systems and the organizational, economic, social, or political aspects of the environments in which they are created and/or will run. Arriving at the current RPAS and its solution concept while relating it to a considerable body of existing work has been itself a challenge. This obliged us to leave outside the scope of our work the interplay between the possibilities or affordances [Zammuto et al. 2007] that RPAS, Configurable Specification, and the resulting Adaptive Systems do and will offer to organizations, the interdependencies between solving RPAS and influencing or designing the human organization that will use it, and the effect of understanding these interdependencies on the very formulation of the RPAS, its solution concept, and conceptual tools for finding solutions. This article will hopefully inform these future efforts.

ELECTRONIC APPENDIX

The electronic appendix for this article can be accessed in the ACM Digital Library.

REFERENCES

V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou. 2012. On the evolution of services. *IEEE Transactions on Software Engineering* 38, 3 (2012), 609–628.

Anonymous. 1993. *Report of the Inquiry Into The London Ambulance Service*. Technical Report. The Communications Directorate, South West Thames Regional Authority.

L. Baresi, L. Pasquale, and P. Spoletini. 2010. Fuzzy goals for requirements-driven adaptation. In *Proceedings of the IEEE International Requirements Engineering Conference.*

B. W. Boehm. 1988. A spiral model of software development and enhancement. *IEEE Computer* 21, 5 (1988), 61–72.

B. W. Boehm, P. K. Bose, E. Horowitz, and M. J. Lee. 1995. Software requirements negotiation and renegotiation aids: A theory-w based spiral approach. In *ICSE*, Dewayne E. Perry, Ross Jeffrey, and David Notkin (Eds.). ACM, 243–253.

F. P. Brooks. 1986. No silver bullet—essence and accidents of software engineering (invited paper). In *IFIP Congress*. 1069–1076.

J. Castro, M. Kolp, and J. Mylopoulos. 2002. Towards requirements-driven information systems engineering: the Tropos project. *Information Systems* 27, 6 (2002), 365–389.

B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee (Eds.). 2009. *Software Engineering for Self-Adaptive Systems [Outcome of a Dagstuhl Seminar]*. Lecture Notes in Computer Science, Vol. 5525. Springer.

A. Dardenne, A. van Lamsweerde, and S. Fickas. 1993. Goal-directed requirements acquisition. *Science of Computer Programming* 20, 1–2 (1993), 3–50.

R. Darimont and A. van Lamsweerde. 1996. Formal refinement patterns for goal-driven requirements elaboration. In *SIGSOFT FSE*.

R. de Lemos, H. Giese, H. A. Müller, and M. Shaw (Eds.). 2013. *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24–29, 2010 Revised Selected and Invited Papers*. Lecture Notes in Computer Science, Vol. 7475. Springer.

N. A. Ernst, A. Borgida, I. J. Jureta, and J. Mylopoulos. 2013. Agile requirements engineering via paraconsistent reasoning. *Information Systems* 43, (2014), 100–116.

M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. 1998. Reconciling system requirements and runtime behavior. In *IWSSD*. IEEE Computer Society, Washington, DC, 50.

S. Fickas and M. S. Feather. 1995. Requirements monitoring in dynamic environments. In *Proceedings of the IEEE International Requirements Engineering Conference*. IEEE Computer Society, 140–147.

S. Greenspan, J. Mylopoulos, and A. Borgida. 1994. On formal requirements modeling languages: RML revisited. In *Proceedings of the 16th International Conference on Software Engineering*. 135–147.

A. Hunter and B. Nuseibeh. 1998. Managing inconsistent specifications: Reasoning, analysis, and action. *ACM Transactions on Software Engineering Methodologies* 7, 4 (1998), 335–367.

I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos. 2010. Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In *Proceedings of the IEEE International Requirements Engineering Conference.*

I. J. Jureta, J. Mylopoulos, and S. Faulkner. 2008. Revisiting the core ontology and problem in requirements engineering. In *IEEE International Requirements Engineering Conference.* IEEE Computer Society.

E. Letier and A. van Lamsweerde. 2004. Reasoning about partial goal satisfaction for requirements and design engineering. In *SIGSOFT FSE*, Richard N. Taylor and Matthew B. Dwyer (Eds.). ACM, 53–62.

X. F. Liu and J. Yen. 1996. An analytic framework for specifying and analyzing imprecise requirements. In *Proceedings of the 18th International Conference on Software Engineering*. IEEE Computer Society, 60–69.

J. Mylopoulos, L. Chung, and B. Nixon. 1992. Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering* 18, 6 (1992), 483–497. DOI: http://dx.doi.org/10.1109/32.142871

B. Nuseibeh, J. Kramer, and A. Finkelstein. 1994. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on Software Engineering* 20, 10 (1994), 760–773.

N. A. Qureshi, I. J. Jureta, and A. Perini. 2011. Requirements engineering for self-adaptive systems: Core ontology and problem statement. In *Proceedings of the Conference on Advanced Information Systems Engineering*.

W. N. Robinson. 2006. A requirements monitoring framework for enterprise systems. *Requirements Engineering* 11, 1 (2006), 17–41.

W. N. Robinson. 2008. Extended OCL for goal monitoring. *ECEASST* 9 (2008).

W. N. Robinson, S. D. Pawlowski, and V. Volkov. 2003. Requirements interaction management. *ACM Computing Surveys* 35, 2 (2003), 132–190.

V. E. S. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. 2012. Requirements-driven software evolution. *Computer Science - Research and Development* (2012), 1–19. DOI:http://dx.doi.org/10.1007/s00450-012-0232-2

V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. 2013. Awareness requirements. In *Software Engineering for Self-Adaptive Systems II*, Rogério Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). Lecture Notes in Computer Science, Vol. 7475. Springer, 133–161.

A. van Lamsweerde, R. Darimont, and E. Letier. 1998. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering* 24, 11 (1998), 908–926.

A. van Lamsweerde and E. Letier. 2000. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering* 26, 10 (2000), 978–1005.

J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J.-M. Bruel. 2010. RELAX: A language to address uncertainty in self-adaptive systems requirements. *Requirements Engineering* 15, 2 (2010), 177–196.

E. S. K. Yu and J. Mylopoulos. 1994. Understanding "Why" in software process modelling, analysis, and design. In *Proceedings of the 16th International Conference on Software Engineering* 159–168.

Raymond F. Zammuto, Terri L. Griffith, Ann Majchrzak, Deborah J. Dougherty, and Samer Faraj. 2007. Information technology and the changing fabric of organization. *Organization Science* 18, 5 (2007), 749–762.

P. Zave and M. Jackson. 1997. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering Methodologies* 6, 1 (1997), 1–30.

Received April 2013; revised December 2013; accepted April 2014

Online Appendix to: The Requirements Problem for Adaptive Systems

IVAN J. JURETA, Fonds de la Recherche Scientifique – FNRS and University of Namur
ALEXANDER BORGIDA, Rutgers University
NEIL A. ERNST, University of British Columbia
JOHN MYLOPOULOS, University of Trento

A. FORMALIZATION

This section defines the modeling language used throughout the article. The modeling language is called Techne 2 (T2) because it extends Techne 1 [Ernst et al. 2013] by adding quantitative variable requirements, quality constraints, softgoals, and preferences.

A.1. Language

The set of requirements $\mathcal{L}$ is the union of four disjoint sets of formulas: simple requirements on propositional variables $\mathcal{L}^P$, simple requirements on quantitative variables $\mathcal{L}^N$, simple softgoals $\mathcal{L}^S$, and complex requirements $\mathcal{L}^C$.

Every **simple functional requirement** (i.e., every $a \in \mathcal{L}^P$) satisfies the following BNF specification:

$$a ::= \mathbf{k}(p) \mid \mathbf{g}(p) \mid \mathbf{t}(p), \tag{A.1}$$

where p is a symbol for a propositional variable. A requirement over a propositional variable can be a domain assumption (**k**), a goal (**g**), or a task (**t**). The informal reading of the members of $\mathcal{L}^P$ is

k(p): it is believed that p is satisfied;
g(p): it is desired that p be satisfied;
t(p): the excution of the task makes p satisfied.

Every **simple quantitative variable requirement** (i.e., every $b \in \mathcal{L}^N$) satisfies the following BNF specification:

$$x ::= n \mid v \mid x + x \mid x - x \mid x \cdot x \mid x/x \mid x^x \tag{A.2}$$

$$\begin{aligned} y ::= {} & x > x \mid x < x \mid x = x \mid x \geq x \\ & \mid x \leq x \mid x \neq x \mid x \backsim pdf \end{aligned} \tag{A.3}$$

$$b ::= \mathbf{k}(y) \mid \mathbf{q}(y) \mid \mathbf{t}(y), \tag{A.4}$$

where n is a real number, v is a quantitative variable, and *pdf* is a probability density function. The informal reading is:

k(y): it is believed that the condition y is satisfied;
q(y): it is desired that the condition y be satisfied;
t(y): the excution of the task makes y satisfied.

Simple softgoals are kept outside $\mathcal{L}^P$ and $\mathcal{L}^N$ because the natural language statement in a softgoal (e.g., *High usability*, *High performance*), denoted $\tilde{p}$, refers to some desirable values of variables in such a way that it is not clear what the exact values

DOI: http://dx.doi.org/10.1145/2629376

are or how the values of these variables can be observed or measured. Every $c \in \mathcal{L}^S$ satisfies:

$$c ::= \mathbf{s}(\tilde{p}). \tag{A.5}$$

A $\tilde{p}$ is called the content of a softgoal and is a vague proposition. By being vague, $\tilde{p}$ is neither a propositional variable, nor a quantitative variable, nor a condition on quantitative variables. The symbol used is intentionally different from those inside domain assumptions, goals, quality constraints, and tasks.

A **complex requirement** relates simple requirements and/or softgoals or is a mandatory or optional requirement. Every $h \in \mathcal{L}^C$ satisfies the BNF specification

$$d ::= a \mid b \mid c \tag{A.6}$$

$$e ::= \bigwedge_{i=1}^{m \geq 1} d_i \rightarrow d \mid \bigwedge_{i=1}^{m \geq 2} d_i \rightarrow \bot \tag{A.7}$$

$$f ::= \mathbf{i}(e) \tag{A.8}$$

$$g ::= d^{\mathbf{M}} \mid d^{\mathbf{O}} \mid f. \tag{A.9}$$

As a convention, we consider every complex requirement with an implication connective as information that defines a configuration. This is because we consider such statements as not being given in an RPAS, but part of what we add during the design of the solution.

We use lowercase letters of the Greek alphabet to denote generic requirements, members of $\mathcal{L} = \mathcal{L}^P \cup \mathcal{L}^N \cup \mathcal{L}^S \cup \mathcal{L}^C$. Uppercase Greek letters denote sets of requirements. Preferences over simple requirements and softgoals are kept in the separate set.

The set of **preferences** $\mathcal{P}$ includes all preference relations over simple requirements and/or softgoals; every $w \in \mathcal{P}$ conforms to the specification

$$w ::= d \succeq d \mid d \succ d \mid d \approx d. \tag{A.10}$$

Note that this disallows preferences between complex requirements; preferences themselves cannot be marked mandatory or optional nor can they participate in a refinement or in a conflict.

Definition A.1. The language is the tuple $(P, \mathbb{R}, V, \tilde{P}, \mathcal{L}, \mathcal{P})$, where $\mathcal{L}$ is the set of requirements that satisfy the BNF specification in Equations A.1–A.9, and $\mathcal{P}$ is the set of preferences that satisfy the BNF specification in Equation A.10, whereby the propositional variables in requirements and preferences come from the set P, numbers from the set $\mathbb{R}$ of real numbers, quantitative variables from V, and vague propositions from $\tilde{P}$.

Definition A.2. Any $\Delta \subseteq \mathcal{L} \cup \mathcal{P}$ is a requirements database.

A.2. Consequence Relation

We use the following consequence relation, defined by extending the Techne consequence relation to allow deduction over statements with quantitative variables.

Definition A.3. The **consequence relation** $\mid\!\sim_?$ is such that, for $\Pi \subseteq \mathcal{L}$, $\phi \in \mathcal{L}$, and $x \in \{\phi, \bot\}$:

—$\Pi \mid\!\sim_? \phi$ if $\phi \in \Pi$, or
—$\Pi \mid\!\sim_? x$ if for every i such that $1 \leq i \leq n$, $\Pi \mid\!\sim_? \phi_i$ and $\mathbf{i}(\bigwedge_{i=1}^{n} \phi_i \rightarrow x) \in \Pi$, or
—If every $y_1, \ldots, y_n$ is a statement about real values and variables that conforms to Eq. A.3, and if $\{y_1, \ldots, y_{n-1}\} \vdash y_n$ in real arithmetic, then $\mathbf{x}_1(y_1), \ldots, \mathbf{x}_{n-1}(y_{n-1}) \mid\!\sim_? \mathbf{x}_n(y_n)$, where $\mathbf{x}_1, \ldots \mathbf{x}_{n-1} \in \{\mathbf{k}, \mathbf{t}\}$ and $\mathbf{x}_n \in \{\mathbf{k}, \mathbf{t}, \mathbf{q}\}$.

The consequence relation $\vdash_?$ is sound with regards to standard entailment $\vdash$ in classical propositional logic, but it is incomplete in two ways: It only considers deducing positive atomic facts, and no ordinary proofs based on arguing by contradiction go through; thus it is paraconsistent.

A.3. Operationalization Function

The purpose of the operationalization function is, given a requirement ϕ and a set of requirements X, to return, if it exists, every set $Y \subseteq X$ that satisfies the following conditions. If ϕ is a requirement over a propositional variable, then Y includes only those domain assumptions and tasks necessary to deduce ϕ. If ϕ is a requirement over a quantitative variable, then Y includes only those domain assumptions and tasks that ensure that the condition in ϕ is satisfied when all of the variables in that condition are replaced by the values that they are assigned by the requirements in Y.

To define the operationalization function, we define here the Select function and one useful set. We assume that $\mathcal{O} = \{\mathbf{k}, \mathbf{g}, \mathbf{q}, \mathbf{s}, \mathbf{t}, \mathbf{i}\}$, and we denote by $\wp(X)$ the powerset of X.

Definition A.4. The **select function**

$$Select : \mathcal{O} \times \{\text{“}empty\text{”}, \mathtt{o}, \mathtt{M}\} \times \wp(\mathcal{L}) \longrightarrow \wp(\mathcal{L}) \tag{A.11}$$

is defined as follows, for $\mathbf{x} \in \mathcal{O}$, $\mathbf{y} \in \{\text{“}empty\text{”}, \mathtt{o}, \mathtt{M}\}$, and $\Pi \subseteq \mathcal{L}$:

$$Select(\mathbf{x}, \mathbf{y}, \Pi) \stackrel{def}{=} \{\phi \mid \mathbf{x}(\phi)^{\mathbf{y}} \in \Pi\}. \tag{A.12}$$

To simplify notation, we use the following abbreviation:

$$Select(\mathbf{x}, \mathbf{y}, \Pi) \equiv \Pi^{\mathbf{y}}_{\mathbf{x}}. \tag{A.13}$$

Given a set of expressions, the Select function returns its subset in which all members have the labels $\mathbf{x} \in \mathcal{O}$ and $\mathbf{y} \in \{\text{“}empty\text{”}, \mathtt{o}, \mathtt{M}\}$.

Definition A.5. **Useful set:** Let $\Pi \subseteq \mathcal{L}$. Then

$$\mathsf{CON}(\Pi) \stackrel{def}{=} \left\{ \Phi \subseteq \Pi \mid \Phi \not\vdash_? \bot \text{ and } \Delta_{\mathbf{i}} \cup \bigcup_{\mathbf{x} \in \mathcal{O}} \Delta^{\mathtt{M}}_{\mathbf{x}} \subseteq \Phi \right\}. \tag{A.14}$$

$\mathsf{CON}(\Pi)$ is the set of all consistent subsets of Π, whereby every one of these subsets must include all mandatory requirements from Δ.

The reason why every $\phi \in \mathsf{CON}(\Pi)$ must include all mandatory requirements is that we require the set of mandatory requirements to be consistent and, as we will see later, to be included in all operationalizations because an operationalization must not be inconsistent with mandatory requirements.

Definition A.6. The **operationalization function**

$$\mathsf{Op} : \bigcup_{\forall \mathbf{x}, \mathbf{y}} \Delta^{\mathbf{y}}_{\mathbf{x}} \longrightarrow \wp\left(\wp\left(\bigcup_{\forall \mathbf{z}, \mathbf{w}} \Delta^{\mathbf{w}}_{\mathbf{z}}\right)\right) \tag{A.15}$$

for $\mathbf{y}, \mathbf{w} \in \{\text{“}empty\text{”}, \mathsf{o}, \mathsf{M}\}$, $\mathbf{x} \in \{\mathbf{g}, \mathbf{q}, \mathbf{s}\}$, and $\mathbf{z} \in \{\mathbf{k}, \mathbf{t}, \mathbf{i}\}$, is defined as follows:

$$\mathsf{Op}(\phi) \stackrel{def}{=} \Big\{ \Pi \in \mathsf{CON}(\Delta) \mid \Pi \vdash_{\mathcal{T}} \phi \text{ and } \Pi \setminus \{\phi\} \subseteq \bigcup_{\forall \mathbf{z}, \mathbf{w}} \Delta_{\mathbf{z}}^{\mathbf{w}}, \text{ and } \nexists \Phi \subset \Pi,\ \Phi \vdash_{\mathcal{T}} \phi \Big\}. \tag{A.16}$$

Every member of the set $\mathsf{Op}(\phi)$ is a minimal consistent set of tasks and domain assumptions that is sufficient to operationalize the goal, quality constraint, or softgoal ϕ. Informally, $\mathsf{Op}(\phi)$ tells us all the ways in (i.e., subsets of) Δ of satisfying a goal, quality constraint, or softgoal that are consistent with all mandatory requirements.

A.4. Softgoal Macros

Softgoal macros automatically change the requirements database when it already contains some specific requirements.

Macros rely on the special *monitoring function* VAL. Given a set of requirements $X \subseteq \Delta$ and a quantitative variable v, if there are tasks and domain assumptions in X that assign a value $n \in \mathbb{R}$ to v (e.g., a task such that $\mathbf{t}(v = n)$ where n is a constant), then $n \in \text{VAL}(X, v)$.

Definition A.7. The **monitoring function**

$$\text{VAL} : \wp(\Delta) \times V \longrightarrow \wp(\mathbb{R}) \tag{A.17}$$

is defined as follows, for $X \subseteq \Delta$, $v \in V$, $\mathbf{x} \in \{\mathbf{k}, \mathbf{q}, \mathbf{t}\}$, and $\mathbf{y} \in \{\text{“}empty\text{”}, \mathsf{o}, \mathsf{M}\}$:

$$\text{VAL}(X, v) = \{n \in \mathbb{R} \mid \exists Y \subseteq X \text{ so that } Y \in \mathsf{Op}(\mathbf{x}(v = n)^{\mathbf{y}})\}, \tag{A.18}$$

$\text{VAL}(X, v)$ is a set of all constants that are assigned through operationalizations in X to v.

The softgoal macro applies to any quantitative variable v for which $\text{VAL}(\Delta, v) \neq \emptyset$ and for which there is a satisfaction function $\mu(v)$. This is the case when there is in Δ an operationalization of a requirement that assigns a value to v and a domain assumption that defines a variable v' as the output of the satisfaction function on v (i.e., there exists $\mathbf{k}(v' = \mu(v)) \in \Delta$). The softgoal macro automatically adds preference relations between sets of requirements operationalize requirements that assign constants to v, whereby the preferences are defined to reflect the values returned by μ.

Definition A.8. For every quantitative variable $v \in V$ such that $\text{VAL}(\Delta, v) \neq \emptyset$ and there exists $\mathbf{k}(v' = \mu(v)) \in \Delta$, where μ is informally interpreted as a satisfaction function, there is the following **softgoal macro**:

$\mathbf{s}($ $\forall x_1, x_2 \in \text{VAL}(\Delta, v), x_1 \neq x_2,$
IF $\mu(x_1) = \mu(x_2)$ THEN ADD
$\{\mathbf{k}(v = x_1), \mathbf{k}(v = x_2),$
$\mathbf{k}(v = x_1) \approx \mathbf{k}(v = x_2)\}$
TO Δ;
IF $\mu(x_1) > \mu(x_2)$ THEN ADD
$\{\mathbf{k}(v = x_1), \mathbf{k}(v = x_2),$
$\mathbf{k}(v = x_1) \succ \mathbf{k}(v = x_2)\}$
TO Δ;).

Two remarks are in order. First, note that the preferences are added to reflect the values not of v but of the satisfaction level $\mu(v)$ on v. Second, note the difference between

the definition of the softgoal macro here and the macro we used in Section 6.3.2: There, the macro was simpler because we assumed that we already knew the configurations.

A.5. Configuration Concept

Definition A.9. A **configuration** S, defined from the requirements database Δ, is a set

$$S \subseteq \bigcup_{\forall \mathbf{x},\mathbf{y}} \Delta_{\mathbf{x}}^{\mathbf{y}}, \text{ for } \mathbf{x} \in \{\mathbf{k},\ \mathbf{t},\ \mathbf{i}\},\quad \mathbf{y} \in \{\text{“}\textit{empty}\text{”}, \mathtt{o}, \mathtt{M}\} \tag{A.19}$$

of domain assumptions, tasks, and imply and conflict relations that satisfies the following properties:

(1) Consistency: $S \not\vdash_{\mathcal{T}} \bot$;
(2) Functional threshold achievement: $\forall \phi \in \Delta_{\mathbf{z}}^{\mathtt{M}}$, $\mathbf{z} \in \{\mathbf{g}, \mathbf{s}\}$, there exists $\Pi \in \mathsf{Op}(\phi)$ such that $\Pi \subseteq S$;
(3) Quantitative threshold achievement: $\forall \phi \in \Delta_{\mathbf{q}}^{\mathtt{M}}$, $\exists \Gamma \in \mathsf{Op}(\phi)$ such that $\Gamma \subseteq S$;
(4) Conformity: $\Delta_{\mathbf{k}}^{\mathtt{M}} \cup \Delta_{\mathbf{t}}^{\mathtt{M}} \subseteq S$;
(5) Dominance: $\nexists S'$ such that both S and S' satisfy conditions 1–4, $S \subset S'$, and $\exists \Delta_{\mathbf{x}}^{\mathtt{o}} = S' \setminus S$, such that $\Delta_{\mathbf{x}}^{\mathtt{o}} \neq \emptyset$ and $\mathbf{x} \in \{\mathbf{k}, \mathbf{t}\}$;
(6) Minimality: $\nexists S'$ such that S' satisfies the conditions 1–5 and $S' \subset S$.

The Threshold achievement property requires that a configuration satisfies all mandatory goals, quality constraints, and softgoals. Satisfaction depends on the presence of operationalizations in S for each of the mandatory goals, quality constraints, and softgoals.

The Conformity property asks that all strict domain assumptions are not violated, and all mandatory tasks are executed. The Achievement and Conformity properties ensure that the configuration satisfies all that *must* be satisfied.

According to the Dominance property, every configuration will be maximal with regards to optional requirements. This property formalizes the idea of the optional relation as holding on requirements that are desirable to satisfy but can be violated. A configuration should include as many defeasible domain assumptions and as many optional tasks up to the point at which adding any further defeasible domain assumptions and/or optional tasks violates the Consistency, Functional and Quantitative threshold achievements, Conformity, or Minimality properties. The Dominance condition ensures that every configuration is Pareto efficient with regards to optional requirements because this condition makes it impossible to add optional domain assumptions and tasks to any S and still ensure that S is a configuration.

The Minimality property requires that a configuration includes only the domain assumptions and tasks that are needed to satisfy exactly the Consistency, Threshold achievement, Conformity, and Dominance properties.

B. RELATING THE FORMALIZATION TO REQUIREMENTS ENGINEERING FOR ADAPTIVE SYSTEMS

We illustrated throughout the main article how the contributions here depart from prior efforts in the understanding of the requirements problem and solution concepts in mainstream RE and in Adaptive Systems RE.

The Configurable Specification concept is inspired by our discussion of the role of contexts, time, and resources in the requirements problem for Adaptive Systems [Qureshi et al. 2011]. We have used Techne [Jureta et al. 2010] here as a starting point and extended its syntax to allow requirements that place constraints on quantitative variables. This has resulted in a more general treatment of operationalization

because we have the operationalization of both functional and quantitative variable requirements. As in the case of Techne, the formalism here does not provide a visual syntax and is thus not itself a model language for early requirements in the same sense that, for example, KAOS goal models [Dardenne et al. 1993] and i^* actor models [Yu and Mylopoulos 1994] are. Two limitations of Techne are overcome here. First, we can make explicit the constraints on quantitative variables. Second, Techne now distinguishes between stable facts, in the form of mandatory domain assumptions, and defeasible information, in the form of optional domain assumptions.

In the remainder of this section, we look at how the language presented in this article can be used to model information that was recognized as crucial in the research into the relaxation of requirements [Letier and van Lamsweerde 2004; Whittle et al. 2010; Baresi et al. 2010], the evaluation of their partial satisfaction [Letier and van Lamsweerde 2004], and the monitoring and control of requirements [Feather et al. 1998; Robinson 2008]. The aim is not to suggest the language here as a general modeling language, but rather to show that it covers the main ideas presented in that related research.

B.1. Fuzzy Relaxation

Baresi et al. [2010] associate a fuzzy operator with a predefined membership function.

For example, if in our framework there is a quality constraint $\mathbf{q}(v < 6\text{hrs})$ (a goal $G(v < 6\text{hrs})$ for Baresi et al. because they allow quantitative variables in goals), then relaxing it would amount to replacing it with $\mathbf{q}(v <_f 6\text{hrs})$ (in their notation, $\mathcal{G}(v <_f 6\text{hrs})$), where $<_f$ is a fuzzy operator. Their interpretation of $v <_f 6\text{hrs}$ is that there is a fuzzy membership function μ that returns the level of satisfaction as a function of v, and the shape of μ is predefined (for $<_f$, it is positive and constant until $v = 6$, then it decreases up to the satisfaction value 0 for some $v > 6$). The operator $<_f$ can be defined in our language by reproducing, in a domain assumption, a function that has the form of the fuzzy membership function for $<_f$, as defined by Baresi et al. We can define as follows a macro that takes a fuzzy goal of the form $\mathcal{G}(v <_f n)$, with $n \in \mathbb{R}$, and transforms it into requirements that can be added to our requirements database Δ:

$\forall \mathcal{G}(v <_f n)$ `WHERE` $v \in V$ `AND` $n \in \mathbb{R}$ `ADD` $\{\mathbf{k}(v' = \mu(v))\}$ `TO` Δ, `AND APPLY THE SOFTGOAL MACRO ON` Δ and v

where $v' \in V$, the set of quantitative variables, and μ is a function defined according to the function pattern defined by Baresi et al. for the fuzzy operator $<_f$. It is straightforward to define similar macros for all other fuzzy operators defined by Baresi et al.

Baresi et al. also define binary connectives, such as fuzzy conjunction. These can be defined here as well as functions of those variables that are defined using fuzzy membership functions. Each binary fuzzy operator gives one function specified in a domain assumption and using an approximation relation. In the formalism from Baresi et al., one way to define fuzzy conjunction $\wedge_f$ between two variables v_1 and v_2, each of which has an accompanying fuzzy membership function $\mu(v_1)$ and $\mu(v_2)$, is as follows: $\mu(v_1 \wedge_f v_2) = \mu(v_1) \cdot \mu(v_2)$. In our language, $\mu(v_1)$ and $\mu(v_2)$ give satisfaction levels. The fuzzy conjunction connective between two quality constraints, respectively over variables v_1 and v_2, is introduced in Δ as the domain assumption $\mathbf{k}(v_3 = \mu(v_1) \cdot \mu(v_2))$, where v_3 is the joint level of satisfaction over variables v_1 and v_2.

B.2. Probabilistic Relaxation

To handle idealistic requirements, Letier and van Lamsweerde [2004] suggest the association of probability estimates to constraints on quantitative variables. This is

what we allow in the T2 language, and this has been illustrated earlier (see Section 6.3.1 in the main article).

B.3. Monitoring and Control Variables

The importance of monitoring and control of requirements is highlighted in Fickas and Feather [1995], Feather et al. [1998], and Robinson [2006]. It is on the basis of the ontology for requirements that we identify controlled variables: If a variable appears in a task, it is a controlled variable. Any variable can be monitored, but all variables in goals, quality constraints, and domain assumptions must monitored.

B.4. Adaptation Rules

Adaptation rules have been called reconciliation tactics [Feather et al. 1998], adaptive goals [Baresi et al. 2010], and adaptivity mechanisms [Souza et al. 2013]. We have not discussed how they ought to be specified or actually implemented, but we have suggested how they can be identified. The Configurable Specification adopted for a system gives a set of configurations and evolution requirements. It consequently indicates what changes between two configurations; that is, which requirements are dropped, which become optional, which others become mandatory, and so on. Differences between every two consecutive configurations in the Configurable Specification specify the effect that adaptation rules should have, regardless of how they are specified or implemented, whereas evolution requirements specify constraints that the adaptation rules must not violate.

B.5. Limitations

Although it may be possible to specify time and temporal constraints by having a quantitative variable, the values of which are defined by a clock, and to map every configuration to a temporal interval, this would clearly not be a convenient way to model temporal constraints on Configurable Specifications. More appropriate in this respect may be requirements modeling languages built on top of linear temporal logic [Dardenne et al. 1993] or branching temporal logic [Whittle et al. 2010]. It is, however, not clear how requirements models built with such languages relate to Configurable Specifications; that is, are these requirements models describing constraints on a single configuration or on parts of Configurable Specifications?

Techne has previously been critiqued for ignoring the notion of agent and the fact that requirements belong to individuals that may thus be modeled as agents. We continue to ignore them here, mainly because they can be introduced in a straightforward manner, yet would complicate notation and presentation without adding much to the main purpose of this article.

To add agents, consider first why they need to be added. If the aim is to help figure out who needs to negotiate requirements conflicts, then assume there is a set of identifiers for agents and add a function that returns, for every requirement, one or more agents who agree on it. If the aim is to assign responsibilities of tasks to agents, then assume a set of identifiers for these agents and define a function that maps every task to one or more agents responsible for its execution. Either of the two uses of agents has no influence on the structure of the RPAS, other than suggesting that RE does involve negotiation and the assignment of responsibilities.

C. RELATIONS DEFINED USING IMPLICATION AND CONFLICT

We discuss in this section how oft-cited relations between requirements can be defined using our requirements types and the implication and conflict relations. Our aim is to illustrate the versatility of the language we used in the article despite its simplicity. We start by introducing the *Minimal Consistent Inference* (MCI) relation.

Definition C.1. Let $\Delta \subseteq \mathcal{L}$ and suppose $Impl(\Delta)$ returns all implications in Δ. A proposition q in a T2 requirements database Δ will be said *to be in the MCI relation* to propositions $\{p_1, \ldots, p_n\} \subseteq \Delta$, $n \geq 1$ if and only if:

(1) $(p_1 \wedge \ldots \wedge p_n \rightarrow q) \in Impl(\Delta)$;
(2) $\nexists \Pi.\Pi \subset \{p_1, \ldots, p_n\}$ and $\Pi \cup Impl(\Delta) \vdash_? q$;
(3) $\nexists \gamma.\gamma \in Impl(\Delta), \{\gamma\} \cup \{\bigwedge_{i=1}^{n} p_i\} \vdash \bot$.

The first condition requires that there be an implication between the propositions. The second, a minimality condition, requires that there be no subset of the premises from which the consequence can be deduced via $\vdash_?$. The third condition requires that the premises be consistent in one step.

We use MCI to define the concept of *argument* in T2. An argument puts together the premises and the conclusion that stand in an MCI relation.

Definition C.2. The pair (Π, q) is an argument in a requirements theory Δ if and only if:

(1) $\Pi \subseteq \Delta$,
(2) q is in the MCI relation to all propositions in $\Pi \backslash Impl(\Delta)$.

By restricting the types of requirements in premises and the conclusion of arguments, we can define a taxonomy of relations and thereby illustrate that T2 can capture oft-cited relations in requirements modeling languages.

MCI is the root of the relations taxonomy and is specialized onto the Refinement, Realization, and Conflict relations. Each of these is defined by restricting the types allowed in the premises and the conclusion, as follows:

—Refinement is a relation that concludes a requirement, and not all of its premises are domain assumptions and/or tasks;
—Realization, or operationalization, is a relation that concludes a requirement, and premises are only domain assumptions and/or tasks;
—Conflict is a relation that concludes $\bot$.

It is then straightforward to observe the following:

—The goal refinement relation from Darimont and van Lamsweerde [1996] can be defined in T2 as a specialization of the Refinement relation. Namely, Goal Refinement in T2 is the Refinement relation, in which all premises and the concusion are of type Goal. Recall that Darimont and van Lamsweerde defined goal refinement as the relation between a goal being refined and subgoals that refine it, the latter having to satisfy three conditions: (i) be sufficient to deduce the refined goal, (ii) be minimal, and (iii) be consistent. All three conditions are satisfied here because Goal Refinement is a specialization of Refinement, which is in turn a specialization of MCI.
—Yu and Mylopoulos [1994] introduced task decomposition in i* (i-star). It is similar to goal refinement, with two differences: (i) The requirement being refined/decomposed must be a task, and (ii) it can be refined by any combination of goals and tasks.[2] Task Decomposition can be defined in T2 as a specialization of the Realization relation in which all premises and the conclusion are all of type Task.
—The goal operationalization relation in KAOS [Dardenne et al. 1993] is similar to the means-ends relation in i*. The idea of both is that tasks should be executed in order to satisfy goals. Operationalization in KAOS stands between goals and constraints,

[2]There is no concept in i* [Yu and Mylopoulos 1994] that corresponds to Domain assumption, so it is not allowed here to have Domain assumptions in the decomposition of a task.

Table II. Common Conflict Relations Translated to T2

Relation in Robinson and colleagues' survey [Robinson et al. 2003]	*Corresponding relation in* T2
Process-level deviation: Deviation of the actual process of developing the system from the predefined process.	Either a Type B conflict, in which the blocker is a domain assumption stating the deviation between the planned and actual development process, or a Type C conflict, in which one of the domain assumptions states that deviation.
Instance-level deviation: An instance of an implemented class violates a requirement.	A Type B conflict, which has one blocking domain assumption. That domain assumption names the instance responsible for the violation, and the alternative in the Type B conflict is the requirement violated by that instance.
Terminology clash (also *Structure clash*): A member of the semantic domain is being referred to using more than one symbol/expression.	None of the conflict relations captures terminology clashes. The terminology clash is an error in the use of a requirements modeling language.
Designation clash: A symbol/expression refers to two or more different members of the semantic domain.	As for the terminology clash, a designation clash is an error in the use of the formalism and cannot be captured in the formalism.
Conflict: A set of requirements is logically inconsistent.	Conflict relation.
Divergence (also *Obstruction*): A set of requirements is logically inconsistent when a certain sequence of events can occur.	A Type B conflict, in which the blocked requirements are alternatives and the blocker is a domain assumption describing the problematic sequence of events.
Competition: A kind of divergence where particular instances of a requirement can cause a divergence.	A Type B conflict, which has one blocking domain assumption. That domain assumption names the instance responsible for the violation, and the alternative in the Type B conflict is the requirement violated by that instance.

whereby a constraint is operational in the sense that it is formulated in terms of objects and actions available to the agents in/of the system. Means-ends instead emphasizes the role of goals as reasons why tasks are executed (i.e., a task exists in a requirements database because it is a means to a goal). The Goal Operationalization relation, which captures the idea of both Goal operationalization and means-ends, is a specialization of the Realization relation, in which all premises are tasks or domain assumptions, and the conclusion is a goal.

To see the kinds of conflict we can capture using our Conflict relation, we first need the concept of Alternative.

Definition C.3. Given $\Pi \subseteq \Delta$, $\Phi \subset \Pi$ is an **alternative** in Π if and only if:

(1) There is an argument $(\Pi, \bot)$;
(2) $\Phi \nvdash \bot$;
(3) $\forall \Psi.\Psi \subseteq \Pi$ if $\Psi \nvdash \bot$ then $\Phi \not\subset \Psi$; that is, Φ is a maximally consistent subset of Π;
(4) Φ does not include only implications and/or Domain assumptions. In other words, Φ includes at least one goal and/or task.

The set of all alternatives in Π is denoted $\mathsf{Alt}(\Pi)$.

Using the Alternative concept, we specialize the Conflict relation as follows:

—Type A Conflict relation is the Conflict relation between premises in the argument $(\Pi, \bot)$ if and only if there are at least two alternatives in Π; that is, $|\mathsf{Alt}(\Pi)| \geq 2$.

—Type B Conflict relation is the Conflict relation between premises in the argument $(\Pi, \bot)$ if and only if $|\mathsf{Alt}(\Pi)| = 1$. Informally, a Type B conflict involves domain assumptions that are blocking the satisfaction of goals or the execution of tasks. If instances in Π are in Type B Conflict, then we have $\mathsf{Block}(\Pi) \stackrel{def}{=} \Pi - \mathsf{Alt}(\Pi)$, and we call $\mathsf{Block}(\Pi)$ the set of blockers.
—Type C Conflict relation is the Conflict relation between premises in the argument $(\Pi, \bot)$ if and only if $|\mathsf{Alt}(\Pi)| = 0$. Type C Conflict involves a minimally inconsistent set that includes only Domain assumptions.

We can also relate Conflict in T2 with notions of conflict in other requirements modeling languages.

In KAOS, the Conflict relation is also a minimally inconsistent set of requirements. Obstruction and Divergence are two relations, also in KAOS, that involve domain assumptions that block, in the sense just discussed, the satisfaction of a goal or the execution of a task.

Table II summarizes the translation of conflict relations identified in Robinson, Pawlowski, and Volkov's survey [Robinson et al. 2003] into T2. Conflicts listed in that table cannot obtain in T2 definitions that are as convenient as, for example, Type A, Type B, or Type C Conflict relations. In a propositional formalism such as T2, there is no elegant way to formally talk about instances of classes and their deviations: A proposition stating the deviation of an instance will be different from a proposition stating normal behavior of other instances, but there is no relation that would say that the two propositions talk about instances of the same class.