# Requirements Engineering for Self-Adaptive Systems: Core Ontology and Problem Statement

Nauman A. Qureshi[1], Ivan J. Jureta[2], and Anna Perini[1]

[1] Fondazione Bruno Kessler - IRST, Software Engineering Research Group
Via Sommarive, 18, 38050 Trento, Italy
{qureshi,perini}@fbk.eu
[2] FNRS & Louvain School of Management, University of Namur, Belgium
ivan.jureta@fundp.ac.be

**Abstract.** The vision for self-adaptive systems (SAS) is that they should continuously adapt their behavior at runtime in response to changing user's requirements, operating contexts, and resource availability. Realizing this vision requires that we understand precisely how the various steps in the engineering of SAS depart from the established body of knowledge in information systems engineering. We focus in this paper on the requirements engineering for SAS. We argue that SAS need to have an internal representation of the requirements problem that they are solving for their users. We formally define a minimal set of concepts and relations needed to formulate the requirements problem, its solutions, the changes in its formulation that arise from changes in the operating context, requirements, and resource availability. We thereby precisely define the *runtime requirements adaptation problem* that a SAS should be engineered to solve.

**Keywords:** Requirements Engineering, Runtime, Adaptation Problem, Self-Adaptive Systems.

## 1 Introduction

Contemporary software systems, such as service-based mobile applications that are increasingly immersed in users' everyday life, must continuously adapt their behavior to changes in users' requirements, operating conditions, and resource availability [5]. For instance, a music download application may need to behave differently when the user's device is connected through the mobile phone to the Internet, than when it is connected via Wi-Fi, when the device is plugged to a docking station rather than on battery power, when the user's preferred music delivery service is not available, and another needs to be selected, and so on. Such software has to cope with such problems as incomplete specifications of its operating conditions, unanticipated events, variable quality of service from third-party services.

The vision for self-adaptive systems (SAS) is that they should continuously adapt their behavior at runtime in response to changing user's requirements, operating contexts, and resource availability. Realizing this vision requires that we understand precisely how the various steps in the engineering of SAS depart from the established body of knowledge in information systems engineering. Research agendas for SAS

H. Mouratidis and C. Rolland (Eds.): CAiSE 2011, LNCS 6741, pp. 33–47, 2011.

have been proposed in various communities. For instance, the Software Engineering for Self-Adaptive Systems (SEAMS) community focuses on issues pertaining to system architectures[1], while the Requirements Engineering community has proposed methods for analyzing requirements for self-adaptivity and suggested that requirements should become artifacts used, processed, and changed at runtime [2,11,18,21]. This led to proposals for various methods to engineer requirements for SAS. However, there has been limited consensus among the research communities on two issues: (i) what main concepts and relations are needed to define the requirements for SAS, and (ii) how does the requirements problem (i.e., the problem to solve during requirements engineering) differ for SAS, compared to systems that are not self-adaptive.

Taking the perspective of requirements engineering (RE) for SAS, we envision SAS to have an internal representation of their user's requirements, and of their operational environment, by being equipped with automated reasoning capabilities for monitoring, analyzing changes that occur dynamically at runtime and finding solutions (i.e. set of tasks that can satisfy the requirements using an available services or otherwise) to meet them, thus ensuring the consistency with the intended system's requirements.

The aim of this paper is to identify concepts and relations, which are necessary to deal with while eliciting and analyzing requirements for SAS and are important to take adaptation decision at runtime by the system itself. This leads us to formulate the *runtime requirements adaptation problem* that a SAS should be engineered to solve.

Section 2 presents the conceptual tools used in the rest of the paper. We introduce the runtime requirements adaptation problem in Section 3. In Section 4, we present how adaptation problem is connected to RE by extending the core ontology for RE and proposing concepts and relationships needed to determine the runtime requirements adaptation problem using examples related to travel planning software. Related work and discussion are presented in Section 5 and 6. The paper ends with conclusions and a summary of directions for future work.

## 2 Preliminaries

### 2.1 General Requirements Problem

The overall aim of RE is to identify the purpose of the system-to-be and to describe as precisely and completely as possible the properties and behaviors that the system-to-be should exhibit in order to satisfy that purpose. This is also a rough statement of the requirements problem that should be solved when engineering requirements.

Zave & Jackson [24] formalized the requirements problem as finding a specification (S) in order to satisfy requirements (R) and not violate domain assumptions (K), to ensure that $K, S \vdash R$. This formulation highlights the importance of the specification to be consistent with domain assumptions, and that requirements should be derivable from $K$ and $S$. It was subsequently argued that there is more to the requirements problem than this formulation states [7]. Namely, a new core ontology for requirements (CORE) was suggested along with a new formulation of the requirements problem to recognize that in addition to goals and tasks, different stakeholders have different preferences

[1] http://www.hpi.uni-potsdam.de/giese/public/selfadapt/front-page

over requirements, that they are interested in choosing among candidate solutions to the requirements problem, that potentially many candidate solutions exist (as in the case of service-/agent-oriented systems, where different services/agents may compete in offering the same functions), and that requirements are not fixed, but change with new information from the stakeholders or the operational environment. In absence of preferences, it is (i) not clear how candidate solutions to the requirements problem can be compared, (ii) what criteria (should) serve for comparison, and (iii) how these criteria are represented in requirements models.

Techne [8], an abstract requirements modeling language was recently introduced as a starting point for the development of new requirements modeling languages that can be used to represent information and perform reasoning needed to solve the requirements problem. Techne is abstract in that it assumes no particular visual syntax (e.g., diagrammatic notation such as present in Tropos [14]), and it includes only the minimum concepts and relations needed to formalize the requirements problem and the properties of its solutions. Techne is a convenient formalism for the formulation of the runtime requirements adaptation problem, as it is adapted to the concepts, such as goal, task, domain assumption, and relations (e.g. Consequence, Preference, Is-mandatory, Is-optional) that remain relevant for the RE of SAS. To keep the discussion simple in this paper, we assume that requirements and other information is available in propositional form, so that every proposition is nothing but a natural language sentence. We overview below the requirements problem formulation using parts of Techne that we need in the rest of the paper. In [15], we provide definitions of the consequence relation $\mid\!\sim_{\vec{r}}$ used in the definition of the candidate solution below.

**Definition 1.** ***Requirements problem: Given*** *the elicited or otherwise acquired: domain assumptions (in the set* $\boldsymbol{K}$*),tasks in* $\boldsymbol{T}$*,goals in* $\boldsymbol{G}$*, quality constraints in* $\boldsymbol{Q}$*, softgoals in* $\boldsymbol{S}$*, and preference, is-mandatory and is-optional relations in* $\boldsymbol{A}$*,* ***find*** *all* ***candidate solutions*** *to the requirements problem and compare them using preference and is-optional relations from* $\boldsymbol{A}$ *to identify the most desirable candidate solution.*

**Definition 2.** ***Candidate solution:*** *A set of tasks* $\boldsymbol{T}^*$ *and a set of domain assumptions* $\boldsymbol{K}^*$ *are a* ***candidate solution*** *to the requirements problem if and only if:*

1. $\boldsymbol{K}^*$ *and* $\boldsymbol{T}^*$ *are not inconsistent,*
2. $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\vec{r}} \boldsymbol{G}^*, \boldsymbol{Q}^*$*, where* $\boldsymbol{G}^* \subseteq \boldsymbol{G}$ *and* $\boldsymbol{Q}^* \subseteq \boldsymbol{Q}$*,*
3. $\boldsymbol{G}^*$ *and* $\boldsymbol{Q}^*$ *include, respectively, all mandatory goals and quality constraints, and*
4. *all mandatory softgoals are approximated by the consequences of* $\boldsymbol{K}^* \cup \boldsymbol{T}^*$*, so that* $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\vec{r}} \boldsymbol{S}^{\boldsymbol{M}}$*, where* $\boldsymbol{S}^{\boldsymbol{M}}$ *is the set of mandatory softgoals.*

We start below from the CORE ontology and problem formulation in Techne, and add concepts specific to the RE for SAS, which leads us to an ontology for requirements in SAS and the formulation of the *requirements problem in context* for SAS. We subsequently show how to formulate the *runtime requirements adaptation problem* as a dynamic RE problem of changing (e.g. switching, re-configuring, optimizing) the SAS from one requirements problem to another requirements problem, whereby the changing is due to change in requirements, context conditions, and/or resource availability.

### 2.2 Adaptive Requirements and Continuous Adaptive RE (CARE) Framework

To support the analysis at design-time, we proposed *adaptive (functional or non-functional) requirements*. They have some degree of flexibility in their achievement conditions, which in turn requires the monitoring of the specification, while taking into account the changes in the operating context, evaluation criteria and alternative software behaviors [16].

More recently, we have proposed a Continuous Adaptive Requirements Engineering (CARE) framework [17,18,19] that views adaptive requirements as runtime artifact that reflect the system's monitoring and adaptation capabilities. RE is performed at runtime by updating initial requirements with new ones, or by removing requirements.

We proposed a classification of adaptation types in the CARE framework, that can be performed by the system itself or by involving the user (both the end-user or the designer) [19]. Mainly, type 1 and 2 adaptations are performed by the system itself, type 1 corresponds to system exploiting existing available solutions when needed, where type 2 is related to monitored information i.e. exploiting it to evaluate changes and select alternative solution. Type 3 and 4 adaptations involves users. In type 3 end-users may express new requirements or change existing ones at any given time, by giving input information correspondingly system analyzes it by finding solutions (adapts) for new/refined needs of the end-users. In type 4, requirements for which there are no possible solutions available analyst/designers are involved for offline evolution of the system.

## 3 Runtime Requirements Adaptation Problem

Various definitions of SAS have been offered in the literature. We remain aligned with the usual conception, namely, that a SAS is a software system that can alter its behavior in response to the changes that occur dynamically in its operating environment. The operating environment can include anything that is observable by the software itself including operational setting in a context, end-user's input, profile and resources.

SAS must be "aware" at runtime of the changes in requirements, its operating context, and in the availability of resources. SAS at runtime need the ability to sense changes. We interpret this as a sort of RE that we call "RE@runtime", where SAS plays – to the feasible extent – the role of an analyst. It has a representation of requirements, of the conditions in its operating contexts (acquired through sensors, for instance), and of the resources it uses. It can add, remove, or otherwise change these, depending on the changes detected through interfaces with the users, environment, and resources. New information thereby acquired can affect the "requirements problem" leading the SAS to query the user for new requirements, or otherwise adapt following the adaptation types [19]. E.g., (a) select a predefined available behavior or look for an alternative behavior the SAS has been designed and implemented for, by exercising its internal monitor-eval-adapt loop (i.e., adaptation types 1 and 2); (b) compose a new solution by exploiting knowledge on available services or explicitly acquired through user's input or change in context (i.e., adaptation type 3); (c) if no solution can be found for the new information, SAS must inform the user for further instructions (adaptation type 4).

We see runtime adaptation as a *dynamic RE problem*, where changes in requirements, context, and resources lead to a new requirements problem, and this in turn requires the resolution of that new problem. The dynamic aspect is that move from one requirements problem formulation to another due to the changes that the SAS can detect. The problem of reformulating the requirements problem when changes occur, and then solving the changed problem is what we call the *Runtime requirements adaptation problem*. To resolve this problem, we argue that it is necessary to know what kinds of information and which relations are crucial for the system to capture. We therefore make explicit the dynamic parts in the requirements problem formulation based on the CORE ontology.

## 4 RE for SAS and Its Core Ontology

Building upon the above considerations, we argue that concepts such as user's context and resources must be considered as first class citizens on top of the existing CORE ontology to engineer requirements for SAS. We add two new concepts, Context and Resource on top of the CORE ontology to accommodate the changes that might occur at runtime, which not only demands adaptation (i.e. dynamically changing from one requirements problem to another) but also requires an update to the specification (i.e. refinement of requirements).

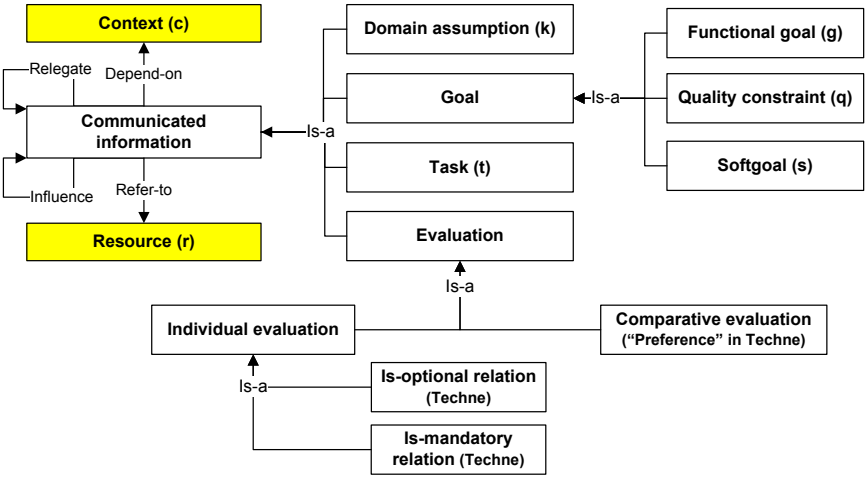


**Fig. 1.** Revised ontology with new concepts (Context and Resource) and relations (Relegation and Influence) related to the concepts of the CORE ontology for RE (for details, see [7])

The revised ontology of the concepts is proposed taking into account the concepts proposed in Techne [8] to support the definition of *runtime requirements adaptation problem*. In Fig. 1, concepts (Context and Resource) and relations (Relegation and Influence) are presented, relating them to the concepts of the CORE ontology. Below, we start by introducing the concept of a requirements database.

**Definition 3.** ***Requirements database:*** *A requirements database, denoted $\Delta$ is the set of all information elicited or otherwise acquired during the* RE *of a system-to-be.*

*Remark 1.* Since $\Delta$ should include all information elicited or otherwise acquired in RE, it should include all instances of domain assumptions (i.e. invariants in the application domain), goals, softgoals, quality constraints, and tasks that we elicited, found through refinement or otherwise identified during RE. One can view $\Delta$ as a repository of information that is usually found in what is informally referred to as a "requirements model". The notation used in the definition of the requirements problem, note that $\Delta = \mathbf{K} \cup \mathbf{T} \cup \mathbf{G} \cup \mathbf{Q} \cup \mathbf{S}$. ■

*Remark 2.* Below, we will use the term *requirement* to abbreviate "member of the requirements database $\Delta$". I.e., we will call every member of $\Delta$ a requirement. ■

To get to the definition of the *runtime requirements adaptation problem*, we start introducing the Context concept. In [15], we discussed how this definition relates to existing conceptions and use of Context in the AI and RE literatures (e.g., [12,6,20,1]).

**Definition 4.** ***Context:*** *An instance $C$ of the Context concept is a set of information that is presupposed by the stakeholders to hold when they communicate particular requirements. We say that every requirement depends on one or more contexts to say that the requirement would not be retracted by the stakeholders in every one of these contexts.*

Firstly, we need a language to write this information that is presupposed, and is thereby in the set of information that we call a particular context. We develop that language below. Secondly, the dependence of a requirement on a context means that every requirement is specific to one or more contexts, and thus, requirements need to be annotated by contexts, which begs additional questions on how the engineer comes to determine contexts. At this point, we revise the Techne language to allow information that is included in contexts. This results in adding one more sort.

**Definition 5.** ***Techne for SAS:*** *The language $\mathcal{L}_{SAS}$ is a finite set of expressions, in which every expression $\phi \in \mathcal{L}_{SAS}$ satisfies the following* BNF *specification*[2]*:*

$$x ::= \boldsymbol{k}(p) \mid \boldsymbol{g}(p) \mid \boldsymbol{q}(p) \mid \boldsymbol{s}(p) \mid \boldsymbol{t}(p) \tag{4.1}$$

$$q ::= \boldsymbol{c}(p) \tag{4.2}$$

$$w ::= x \mid q \tag{4.3}$$

$$y ::= \bigwedge_{i=1}^{n} w_i \rightarrow w \mid \bigwedge_{i=1}^{n} w_i \rightarrow \bot \tag{4.4}$$

$$\phi ::= w \mid \boldsymbol{k}(y) \mid \boldsymbol{c}(y) \tag{4.5}$$

*Remark 3.* We used (indexed/primed p, q, r) as an arbitrary atomic statement, every $\phi$ an arbitrary complex statement, and every x an arbitrary label to represent Techne labeled propositions i.e. domain assumption (**k**(p)), a goal (**g**(p)), etc., to distinguish from these basic labeled propositions the context propositions (i.e., propositions about

[2] In BNF: "::=" reads "defines"; "|" reads "or".

context), $\mathbf{c}$(p) is added separately in the BNF specification, via q, and every w can either be x or q. Every y represents a complex statement as a formula with conjunction and implication such that y can be either w or $\bot$, where w is some requirement in a context propositions and $\bot$ refers to logical inconsistency. We can then rewrite $\phi$ as a complex statement consists of either w or $\mathbf{k}(y)$ or $\mathbf{c}(y)$. ■

**Definition 6.** ***Consequence relation of Techne in context:*** *Let $\Pi \subseteq \mathcal{L}_{SAS}$, $\phi \in \mathcal{L}_{SAS}$, and $z \in \{\phi, \bot\}$, then:*

1. *$\Pi \mid\!\sim_{\mathcal{C}\tau} \phi$ if $\phi \in \Pi$, or*
2. *$\Pi \mid\!\sim_{\mathcal{C}\tau} z$ if $\forall 1 \leq i \leq n$, $\Pi \mid\!\sim_{\mathcal{C}\tau} \phi_i$ and $\boldsymbol{k}(\bigwedge_{i=1}^{n} \phi_i \rightarrow z) \in \Pi$.*

The consequence relation $\mid\!\sim_{\mathcal{C}\tau}$ is sound w.r.t. standard entailment in propositional logic. It deduces only positive statement by being paraconsistent, thus all admissible candidate solutions are found via paraconsistent and non-monotonic reasoning. Reasoning is paraconsistent because an inconsistent $\Delta$ or $C$ should not allow us to conclude the satisfaction of all requirements therein; it is non-monotonic in that prior conclusions drawn from a $\Delta$ or a $C$ may be retracted after new requirements are introduced.

We also need a function that tells us which contexts a requirement applies to.

**Definition 7.** ***Contextualization function:*** *Let $\boldsymbol{C}$ be the set of all contexts. $\mathcal{C} : \wp(\mathcal{L}_{SAS}) \longrightarrow \wp(\boldsymbol{C})$ (where $\wp$ returns the powerset) is called the contextualization function that for a given set of formulas returns the set of contexts to which these formulas apply to. By "apply to", we mean that $C \in \mathcal{C}(\phi)$ iff the following conditions are satisfied:*

1. *$C, \phi \not\mid\!\sim_{\mathcal{T}} \bot$, i.e., $\phi$ is not inconsistent with context $C$,*
2. *$C$ is such that $\exists X \subseteq \Delta$ such that $C, X \mid\!\sim_{\mathcal{T}} \phi$, i.e., the context $C$ together with some requirements $X$ from $\Delta$ lets us deduce $\phi$.*

Several remarks are in order. Firstly, with $\mathcal{L}_{\mathbf{SAS}}$, we now have a new sort for expressions that are members of a set that defines a context. Recall that we defined an instance $C$ of Context as a set of information, so that now $\mathcal{L}_{\mathbf{SAS}}$ tells us that one member of that set can either be a proposition $p$, denoted $\mathbf{c}(p)$, or can be a formula with implication, denoted $\mathbf{c}(y)$ in the BNF specification. E.g., if the engineer assumes that the stakeholders wants that her goal $\mathbf{g}(p)$ for "arrive at destination" be satisfied both in the context $C_1$ in which the context proposition "$\mathbf{c}(q)$: flight is on time" holds (i.e., $\mathbf{c}(q) \in C_1$), and in the context $C_2$ in which the context proposition "$\mathbf{c}(r)$: flight is delayed but not more than 5 hours" holds (i.e., $\mathbf{c}(r) \in C_2$), then $C_1 \in \mathcal{C}(\mathbf{g}(p))$ and $C_1 \in \mathcal{C}(\mathbf{g}(p))$.

Secondly, observe that the BNF specification lets us write formulas in which we combine context propositions and requirements, e.g.: $\mathbf{k}(p) \wedge \mathbf{c}(q) \rightarrow \bot$, which the requirements engineer can use to state that the domain assumption $\mathbf{k}(p)$ that was communicated by the stakeholder does not hold in contexts in which the context proposition $\mathbf{c}(q)$ holds.

Since we can combine context formulas and requirements, we can state very useful relations, such as that some requirements conflict with some contexts, by saying that these requirements are inconsistent with some of the context formulas in these contexts. As an aside, rules that connect requirements and context formulas need not be specified in a definite way by the requirements engineer. It is also possible to learn them by asking feedback to the user i.e. a SAS at runtime can perform this task. For example,

if the system asks the user a question of the form: *Your flight is delayed by 5 hours or more. Do you wish to rebook a flight for the next day?* This question can be reformulated as a question on which of these two formulas to add to the current context of the user (i.e. the context in which we asked the user that question):

$$\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_1) \rightarrow \bot) \tag{4.6}$$

$$\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_2) \rightarrow \bot) \tag{4.7}$$

where $\mathbf{c}(p)$ is for "flight delayed by more than 5 hours", $\mathbf{g}(q_1)$ is for the goal "keep the booked flight", and $\mathbf{g}(q_2)$ is for the goal "rebook the same flight for the next day". If the user answers "yes", then add formula $\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_1)) \rightarrow \bot$ to the context in which we asked the user that question; if the user answers "no", then we add $\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_2)) \rightarrow \bot$ to the current context.

We now add the Resource concept. Since the formal language that we use in this paper is propositional, we will keep the resource concept out of it.

**Definition 8.** ***Resource:*** *An instance $R$ of the Resource concept is an entity referred to by one or more instances of Communicated information.*

In order to introduce resources in the definition of the runtime requirements adaptation problem, we need a function that tells us which resources are referred to by a task, domain assumption, or a context proposition, as these resources will need to be available and used in some way in order to ensure that the relevant domain assumptions and context propositions hold, and that the tasks can be executed.

**Definition 9.** ***Resource selector function:*** *Let **C** be the set of all contexts. Given a set of tasks, domain assumptions, and/or context propositions, the resource selector function returns the identifiers of resources necessary for the domain assumptions and/or context propositions to hold, and/or tasks to be executed:*

$$\mathcal{R} : \wp(\boldsymbol{T} \cup \boldsymbol{K} \cup \bigcup \boldsymbol{C}) \longrightarrow \wp(\boldsymbol{R}) \tag{4.8}$$

The domain of $\mathcal{R}$ are domain assumptions, context propositions, and tasks. The reason that goals, softgoals, and quality constraints are absent is that the resources will be mobilized to realize a candidate solution to the requirements problem, and the candidate solution includes only domain assumptions and tasks. Since these domain assumptions and tasks are contextualized, we need to ensure the availability of resources that are needed in the context on which these domain assumptions and tasks depend on. Note also that we have $\bigcup \mathbf{C}$ because $\mathbf{C}$ is a set of sets, so that we need to get the union of all of the sets in $\mathbf{C}$.

We can now formulate the runtime requirements adaptation problem for SAS.

**Definition 10.** ***Runtime requirements adaptation problem: Given*** *a candidate solution $CS(C_1)$ in the context $C_1 \in \boldsymbol{C}$ to the requirements problem $RP(C_1)$ in context $C_1 \in \boldsymbol{C}$, and a change from context $C_1$ to $C_2 \neq C_1$, **find** the requirements problem $RP(C_2)$ in context $C_2 \in \boldsymbol{C}$ and choose among candidate solutions to $RP(C_2)$ a solution $CS(C_2)$ in the context $C_2$ to the requirements problem $RP(C_2)$ in the context $C_2 \in \boldsymbol{C}$.*

The definition of the runtime requirements adaptation problem reflects the intuition that by changing the context, the requirements problem may change – as requirements can change – and from there, a new solution needs to be found to the requirements problem in the new context.

We now reformulate the requirements problem so as to highlight the role of context in it, as well as of the resources.

**Definition 11.** ***Requirements problem RP***$(C)$ ***in context*** $C$***: Given*** *the elicited or otherwise acquired: domain assumptions in the set* $\boldsymbol{K}$*, tasks in* $\boldsymbol{T}$*, goals in* $\boldsymbol{G}$*, quality constraints in* $\boldsymbol{Q}$*, softgoals in* $\boldsymbol{S}$*, preference, is-mandatory and is-optional relations in* $\boldsymbol{A}$*, a context* $C$ *on which* $\boldsymbol{K} \cup \boldsymbol{T} \cup \boldsymbol{G} \cup \boldsymbol{Q} \cup \boldsymbol{S}$ *and* $\boldsymbol{A}$ *depend on,* ***find*** *all* ***candidate solutions in context*** $C$ *to RP*$(C)$ *and compare them using preference and is-optional relations from* $\boldsymbol{A}$ *to identify the most desirable candidate solution.*

**Definition 12.** ***Candidate solution*** $CS$ ***in the context*** $C$***:*** *A set of tasks* $\boldsymbol{T}^*$ *and a set of domain assumptions* $\boldsymbol{K}^*$ *are a* ***candidate solution in the context*** $C$ *to the requirements problem RP*$(C)$ *in context* $C$ *if and only if:*

1. $\boldsymbol{K}^*$ *and* $\boldsymbol{T}^*$ *are not inconsistent,*
2. $C, \boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{CT}} \boldsymbol{G}^*, \boldsymbol{Q}^*$*, where* $\boldsymbol{G}^* \subseteq \boldsymbol{G}$ *and* $\boldsymbol{Q}^* \subseteq \boldsymbol{Q}$*,*
3. $\boldsymbol{G}^*$ *and* $\boldsymbol{Q}^*$ *include, respectively, all mandatory goals and quality constraints,*
4. *all mandatory softgoals are approximated by the consequences of* $C, \boldsymbol{K}^* \cup \boldsymbol{T}^*$*, so that* $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{CT}} \boldsymbol{S^M}$*, where* $\boldsymbol{S^M}$ *is the set of mandatory softgoals, and*
5. *resources* $\mathcal{R}(C \cup \boldsymbol{K}^* \cup \boldsymbol{T}^*)$ *needed to realize this candidate solution are available.*

As discussed earlier, we view runtime requirements adaptation problem as a dynamic RE problem. To support the analysis, we add two relations in the CORE ontology. We now define the *relegation relation* via the inference and preference relations in Techne.

**Definition 13.** ***Relegation relation:*** *A relegation relation is an* $(n+1)$*-ary relation that stands between a requirement* $\phi \in \Delta$ *and* $n$ *other sets of requirements* $\Pi_1, \Pi_2, \ldots, \Pi_n \subseteq \Delta$ *if and only if there is an inference relation from every* $\Pi_i$ *to* $\phi$ *and there is a binary relation:* $\succ_\phi \subseteq \{\Pi_i \mid 1 \leq i \leq n\} \times \{\Pi_i \mid 1 \leq i \leq n\}$ *whereby* $\Pi_i \succ_\phi \Pi_j$ *if it is strictly more desirable to satisfy* $\phi$ *by ensuring that* $\Pi_i$ *holds, than to satisfy* $\phi$ *by ensuring that* $\Pi_j$ *holds.*

The inference relations required by a relegation relations indicate that a relegation relation can only be defined for requirements that we know how to satisfy in different ways. For example, if we have a goal $\mathbf{g}(p)$, and we have two ways to satisfy that goal, e.g.:

$$\Pi_1 = \{\mathbf{t}(q_1), \mathbf{b}(\mathbf{t}(q_1) \rightarrow \mathbf{g}(p))\} \tag{4.9}$$

$$\Pi_2 = \{\mathbf{t}(q_2), \mathbf{b}(\mathbf{t}(q_2) \rightarrow \mathbf{g}(p))\} \tag{4.10}$$

then we have satisfied the first condition from the definition of the relegation relation, since $\Pi_1 \mid\!\sim_{\mathcal{CT}} \mathbf{g}(p)$ and $\Pi_2 \mid\!\sim_{\mathcal{CT}} \mathbf{g}(p)$.

The second condition in the definition of the relegation relation says that we need to define a preference relation $\succ_{\mathbf{g}(p)}$ between different ways of satisfying $\mathbf{g}(p)$. Observe that we define $\succ_{\mathbf{g}(p)}$ between *sets* of information, not pieces of information. The Techne

preference relation defines preference between individual pieces of information, so we can use preference relations between members of $\Pi_1$ and $\Pi_2$ to define $\succ_{\mathbf{g}(p)}$.

Suppose that $\mathbf{t}(q_1) \succ \mathbf{t}(q_2)$, i.e., that we prefer to execute task $\mathbf{t}(q_1)$ to executing the task $\mathbf{t}(q_2)$. We can define $\succ_{\mathbf{g}(p)}$ as a function of the Techne preference relation, i.e., $\succ_{\mathbf{g}(p)} \stackrel{def}{=} f(\succ)$, that is, from the information that the preference relation already includes. Namely, in this example it is appropriate to say that, if $\mathbf{t}(q_1) \succ \mathbf{t}(q_2)$, then $\Pi_1 \succ_{\mathbf{g}(p)} \Pi_2$. Since we have only $\Pi_1$ and $\Pi_2$, it is enough to know that $\Pi_1 \succ_{\mathbf{g}(p)} \Pi_2$ to know everything we need to define the relegation relation.

Namely, the relegation relation $(\mathbf{g}(p), \Pi_1, \Pi_2, \succ_{\mathbf{g}(p)})$ tells us that, if we cannot satisfy $\mathbf{g}(p)$ through $\Pi_1$ then we will relegate to $\Pi_2$, i.e., satisfy $\mathbf{g}(p)$ through $\Pi_2$.

Finally, we define the *influence relation*. Note that it is simple here, since we have no numerical values, so we cannot speak about influence as correlation. We can only say that some information influences some other information if the absence of the former makes it impossible for us to satisfy the latter.

**Definition 14.** ***Influence relation:*** *An influence relation is a binary relation from $\psi \in \mathcal{L}_{SAS}$ to $\phi \in \mathcal{L}_{SAS}$, iff either:*

1. $\exists \Pi \subseteq \Delta \cup C$ *s.t.* $\Pi \vdash_{\mathcal{CT}} \phi$ *and* $\Pi \setminus \psi \nvdash_{\mathcal{CT}} \phi$, *or*
2. $\forall \Pi \subseteq \Delta \cup C$ *s.t.* $\Pi \vdash_{\mathcal{CT}} \phi$ *and* $\Pi \setminus \psi \nvdash_{\mathcal{CT}} \phi$.

*In the first case above, we say that $\psi$ weakly influences $\phi$, denoted $\psi \xrightarrow{wi} \phi$. In the second case above, we say that $\psi$ strongly influences $\phi$, denoted $\psi \xrightarrow{si} \phi$.*

*Remark 4.* If $\psi \xrightarrow{si} \phi$, then we have no way to satisfy $\phi$ if $\psi$ is not satisfied. If $\psi \xrightarrow{wi} \phi$, then some ways of satisfying $\phi$ cannot be used to do so if $\psi$ is not satisfied. ■

### 4.1 Runtime Requirements Adaptation Problem Illustration

We now revisit the above definitions and use scenarios from travel exemplar case study to illustrate how SAS, instantiating CARE and running on user's mobile phone, resolves the "runtime requirements adaptation problem" at runtime.

For example, user arrives at the airport to avail her flight from Italy to Canada via Paris for a business meeting. While at the airport after the boarding, user want to connect to the Internet using her mobile phone to check emails and flight details before checking in for the plane. Moreover, user wants to be informed about any flight delay.

Taking the above example, we now present SAS adaptation sequence at runtime in case of change in context $C$ along the time $T = t^1, ....t^n$ as shown in the Fig.2. Let $CS$ be a set of candidate solution, thereby determining the runtime requirements adaptation problem as a combination of instances of the tasks $\mathbf{T}^*$) and domain assumptions $\mathbf{K}^*$ such that $\mathbf{G}^*, \mathbf{Q}^*$ and $\mathbf{S}^{\mathbf{M}}$ are satisfied. In case of changes in the context $C = C_1, ....C_n$ overtime for which $CS$ needs to be re-evaluated by the system and $R$ is required to be used or identified in a given context $C$ to realize $CS$. By re-evaluation we mean that system at runtime exploits its monitored information, evaluate all the possible alternative $CS$ or search for new ones (i.e. exploiting available services) that can satisfy the runtime adaptation problem in response to changes in the $C$ therefore adapting to the

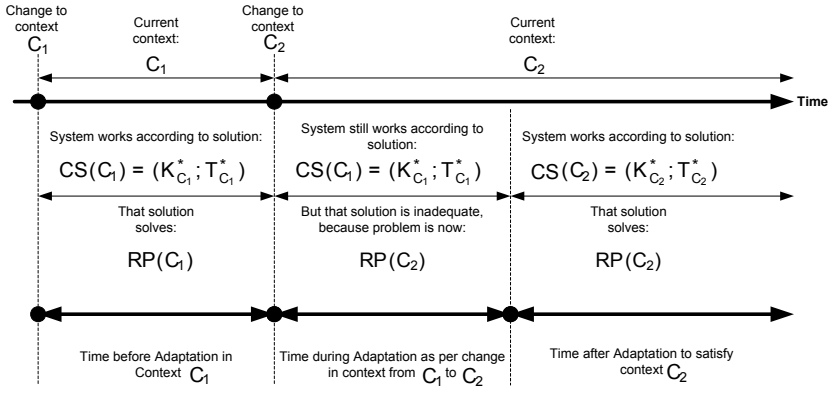


**Fig. 2.** System Adaptation Sequence in Time

candidate solution $CS$. At this SAS may perform at sub-optimal level and can exploit automated reasoning techniques such as planning or decision making techniques such as analytic hierarchy process (AHP)[3].

**Before Adaptation:** Assume that at time $t^1$, the user's goals $\mathbf{G}^*$ are *to connect to the Internet for checking details of itinerary* and *inform about the flight delays* with the quality constraint $\mathbf{Q}$ *to have the Internet connectivity not less than 256Kbps*. To satisfy these requirements, SAS is running according to its candidate solution $CS$ i.e., using the set of tasks $\mathbf{T}^*$, e.g. *search for available connection*, *enable Wi-Fi*, *get itinerary details* and *show flight itinerary*, in the current context i.e. $C_1$ is *at the airport*, and the domain assumption $K^*_{t1}$ *Internet must be available at the airport*, is not violated. This implies that, $CS(C_1) = (K^*_{C1}, T^*_{C1})$ and $CS(C_1)$ satisfies the runtime requirements adaptation problem i.e. $RP(C_1)$. We can rewrite this as:

$$C_1, \mathbf{K}^*_{C_1}, \mathbf{T}^*_{C_1} \models_{\mathcal{CT}} \mathbf{G}^*_{C_1}, \mathbf{Q}^*_{C_1}$$

where $\mathcal{R}(C_1 \cup \mathbf{K}^*_{C_1} \cup \mathbf{T}^*_{C_1})$ identifies the set of resources $\mathbf{R}$ available, e.g. (Airport Wi-Fi hotspot, Mobile Phone of the user) to perform $\mathbf{T}^*_{C_1}$.

**During Adaptation:** SAS while executing $CS(C_1)$, observes a change in context i.e. *the airport Wi-Fi connection becomes unavailable* at time $t^2$. Due to this change in context from $C_1$ to $C_2$, the existing candidate solution $CS(C_1)$ might be valid but is not adequate to satisfy the current context $C_2$. As a consequence, the SAS needs to re-evaluate its candidate solutions $CS$ by searching in its solution base i.e. $\Delta$ or looking for solutions that can be realized through available services. For instance, a new candidate solution $CS(C_2)$ could be, e.g. *connect to the Internet using data services either 3G or Edge on mobile phone R*; or *recommending user to move to the area where the signal strength is stronger*; or *avail the Internet on the free booths*. At this stage, SAS

[3] Discussing such techniques are out the scope of this paper. We present three scenarios to illustrate how the SAS can adapt at runtime by resolving the runtime requirements adaptation problem.

may use relegation relation to infer, if the $\mathbf{G}^*$ with a $\mathbf{Q}$ is *to have the Internet connectivity not less than 256Kbps* can be relegated. After re-evaluating the possibilities, SAS finds $CS(C_2)$ i.e. set of tasks $\mathbf{T}^*$, e.g. *enable 3G or Edge service* and *connect to the Internet* with a refined $\mathbf{Q}$ i.e. *Internet connectivity greater than 256Kbps* for the user. At this stage the influence relation is also used to ascertain the influence of $CS(C_2)$ on user's goals and preference, e.g. *Hi-speed Internet is preferred than no Internet connection*. SAS can derive conclusions that adapting to $CS(C_2)$ will not affect $K^*_{C2}$ i.e. *Any flight information must be communicated to the customer* and goal $\mathbf{G}^*$ i.e. *to connect to the Internet to view itinerary* and *inform about the flight delays* will be satisfied. Therefore, $CS(C_2) = (K^*_{C2}, T^*_{C2})$; satisfying the runtime requirements adaptation problem i.e. $RP(C_2)$. We can rewrite this as:

$$C_2, \mathbf{K}^*_{C_2}, \mathbf{T}^*_{C_2} \mid\!\sim_{\mathcal{C}\tau} \mathbf{G}^*_{C_2}, \mathbf{Q}^*_{C_2}$$

where $\mathcal{R}(C_2 \cup \mathbf{K}^*_{C_2} \cup \mathbf{T}^*_{C_2})$ identifies the set of resources $\mathbf{R}$ available, e.g. (Access 3G or Edge data services, Mobile Phone of the user) to perform $\mathbf{T}^*_{C_2}$.

**After Adaptation:** At time $t^3$, SAS adapts to the candidate solution $CS(C_2)$ taking into account the context $C_2$ and available resources $R$ i.e. Access 3G or Edge data services, Mobile Phone of the user, thus not violating the $K^*_{C2}$. Adaptation is performed dynamically at runtime by changing (e.g. switching, re-configuring, optimizing) SAS from one requirements problem to another i.e. $RP(C_1)$ to $RP(C_2)$, in response to changes in the context, user's needs or resource variability.

## 5 Related Work

Requirements engineering is carried out at the outset of the whole development process, but in the context of SAS, RE activities are also needed at runtime thus enabling a seamless adaptation. Research on SAS has recently received attention from variety of research communities mainly targeting the software engineering of SAS from requirements, design and implementation perspectives. Focusing on requirements engineering for SAS, research agenda from SEAMS [5] and RE community has identified key challenges that must be addressed while developing such systems.

For instance, in [23], a declarative language (RELAX) is proposed to capture uncertainty, using temporal operators (such as eventually, until) by formalizing the semantics in Fuzzy Branching Temporal logic, to specify requirements for SAS. Similarly, adopting goal-oriented analysis for adaptive systems, mitigation strategies are proposed to accommodate uncertainty and failures by using obstacle analysis in [4]. Requirements reflection is another aspect, where ideas from computational reflection has been borrowed to provide SAS the capability to be aware of its own requirements [21]. Similarly, online goal refinement [9] is of prime importance considering the underline architecture of the intended SAS. Taking the engineering perspective, making the role of feedback loops more explicit in the design of SAS has been recognized as a key requirement for developing SAS in [3].

In our previous work, we proposed similar ideas to engineer adaptive requirements using goal models and ontologies by making explicit the requirements for feedback loop (i.e. monitoring, evaluation criteria, and adaptation alternative) more explicit in [16]. We

extend this work in [17,18,19] by proposing a Continuous Adaptive RE (CARE) framework for continuous online refinement of requirements at runtime by the system itself involving the end-user. We proposed an architecture of an application that instantiate CARE. We proposed a classification of adaptation at runtime by exploiting incremental reasoning over adaptive requirements represented as runtime artifact. Similar ideas has been proposed treating goals as fuzzy goals formalized using fuzzy logic representing strategies for adaptation and operationalizing them as BPEL processes in [2]. Another variation of this idea has been advocated in [22] as "Awareness Requirements", as a way to express constraints on requirements as meta requirements to deal with uncertainty while developing SAS.

In goal-oriented modeling, *Tropos* has been extended to capture the contextual variability (mainly location) [1] by leveraging the concept of variation points [10] exploiting the decomposition rules in a goal tree. Mainly, it helps in linking the alternative in the goal model to the corresponding context (location) that helps in monitoring facts and reasoning for adaptation in case of change in the context (location). Extended design abstractions, including environment models, explicit goal types, and conditions for building adaptive agents have been proposed as an extension of *Tropos*, in Tropos4AS [13].

## 6 Discussion

It is worth to further discuss assumptions underlying the suggested problem formulation, its generality as well as its practical impact. The problem formulation suggested in this paper makes no assumptions and imposes no constraints on how the information that is used and acquired. We thereby recognize that not all information can be collected during requirements engineering, or at design time, but that this will depend on the technologies used to implement the system. For example, the information about the context, the formulas in $C$ may – if the implementation technology allows – be obtained by recognizing patterns in the data that arrives through sensors, then matching patterns of data to templates of proposition or implications. We stayed in the propositional case, since this was enough to define the main concepts and relations, and subsequently use them to formulate the runtime requirements adaptation problem. The actual system will operate using perhaps more elaborate, first-order formalisms to represent information, so as to make that information useful for planning algorithms applied to identify candidate solutions. However, regardless of the formalism used, the system still needs to be designed to ensure the general conditions and relations that the problem formulation states: e.g., that the system needs an internal representation of information pertaining to contexts, domain assumptions, tasks, goals, and so on, that goals and quality constraints are satisfied through consistent combinations of $C$, $\mathbf{K}$ and $\mathbf{T}$, among others.

Concerning generality of the proposed problem formulation and practical implication, our aim in this work was first to understand the general problem, and then focus on developing particular requirements modeling languages to handle it. In this regards we believe that recently proposed frameworks for engineering requirements for SAS can be reconnected to our problem formulation. Consider, for example RELAX [23], which proposes a formalism for the specification of requirements and a particular way to relax them at runtime: if a requirement cannot be satisfied to the desired extent, then

alternative requirements can be specified in RELAX, stating thereby how achievement conditions for the original requirement can be relaxed. This mechanism can be considered a particular way to implement the Relegate relation, that is the RELAX mechanism obtains a straightforward interpretation in the language we used here. There can be other ways to handle uncertainty and relaxation of requirements, and our aim in this paper was to remain independent of particular approaches.

## 7 Conclusions and Future Work

We argued in this paper for a general formulation of the runtime requirements adaptation problem, using recent work on the revised general requirements problem and its core ontology. Taking into account our work on continuous adaptive RE in CARE, and the types of adaptation defined in CARE, in this paper, we proposed to make explicit the dynamic parts in requirements representation and formulated the runtime requirements adaptation problem. In particular, two key concepts help managing runtime requirements changes, namely the concept of *context* and *resource*, while the *relegation* and *influence* relations capture changes at runtime. The proposed *runtime requirements adaptation problem* is envisioned as dynamic RE problem for adaptive systems i.e. reformulating the requirements problem when changes occur that a SAS can detect, and then solving the changed problem at runtime.

Ongoing work aims at exploiting these formal definitions of concepts and relations into a more concrete modeling and analysis language. The concept of requirements database $\Delta$ introduced in this paper provides premise to define operations (e.g. adding, removing, substituting requirements) that a SAS may perform over $\Delta$ to update its own specification at runtime thus help realizing continuous adaptive RE (see CARE [17,18]). Moreover, the application at runtime of automated reasoning (such as in AI planning) and decision-making techniques (e.g., Analytic Hierarchy Process) may be relevant for the engineering and running of SAS. They require further investigation.

## References

1. Ali, R., Dalpiaz, F., Giorgini, P.: Location-based software modeling and analysis: Tropos-based approach. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 169–182. Springer, Heidelberg (2008)
2. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 18th IEEE Int. Requirements Eng. Conf., Sydney, Australia, pp. 125–134 (2010)
3. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H.M., Kienle, H., Litoiu, M., Müller, H.A., Pezzè, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. Software Engineering for Self-Adaptive Systems 5525, 48–70 (2009)
4. Cheng, B.H.C., Sawyer, P., Bencomo, N., Whittle, J.: A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 468–483. Springer, Heidelberg (2009)
5. Cheng, B.H., de Lemos, R., Giese, H., Inverardi, P., Magee, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)

6. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5(1), 4–7 (2001)
7. Jureta, I.J., Mylopoulos, J., Faulkner, S.: Revisiting the core ontology and problem in requirements engineering. In: 16th IEEE Int. Requirements Eng. Conf., pp. 71–80 (2008)
8. Jureta, I.J., Borgida, A., Ernst, N.A., Mylopoulos, J.: Techne: Towards a new generation of requirements modeling languages with goals, preferences, and inconsistency handling. In: 18th IEEE Int. Requirements Eng. Conf., Sydney, Australia, pp. 115–124 (2010)
9. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. In: Future of Software Engineering, 2007. FOSE 2007, pp. 259–268 (May 2007)
10. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E., Mylopoulos, J.: On goal-based variability acquisition and analysis. In: 14th IEEE Int. Requirements Eng. Conf., pp. 79–88 (2006)
11. Liaskos, S., McIlraith, S.A., Mylopoulos, J.: Integrating preferences into goal models for requirements engineering. In: 18th IEEE Int. Requirements Eng. Conf., Sydney, Australia, pp. 135–144 (2010)
12. McCarthy, J.: Notes on formalizing context. In: Proceedings of the 13th International Joint Conference on Artifical Intelligence, vol. 1, pp. 555–560. Morgan Kaufmann Publishers Inc., San Francisco (1993)
13. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. In: ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2008), pp. 9–16. ACM, New York (2008)
14. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: High variability design for software agents: Extending Tropos. TAAS 2(4) (2007)
15. Qureshi, N.A., Jureta, I., Perini, A.: On runtime requirements adaptation problem for self-adaptive systems, SE Research Group Technical Report (TR-FBK-SE-2010-1), FBK, Trento, Italy (2010), `http://se.fbk.eu/files/TR-FBK-SE-2010-1.pdf`
16. Qureshi, N.A., Perini, A.: Engineering adaptive requirements. In: ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2009), pp. 126–131. IEEE Computer Society, Washington, DC, USA (2009)
17. Qureshi, N.A., Perini, A.: Continuous adaptive requirements engineering: An architecture for self-adaptive service-based applications. In: First Int. Workshop on Requirements@Run.Time (RE@RunTime), Sydney, Australia, pp. 17–24 (2010)
18. Qureshi, N.A., Perini, A.: Requirements engineering for adaptive service based applications. In: 18th IEEE Int. Requirements Eng. Conf., Sydney, Australia, pp. 108–111 (2010)
19. Qureshi, N.A., Perini, A., Ernst, N.A., Mylopoulos, J.: Towards a continuous requirements engineering framework for self-adaptive systems. In: First Int. Workshop on Requirements@Run.Time (RE@RunTime), held at (RE 2010), Sydney, Australia, pp. 9–16 (2010)
20. M., Salifu, Yu, Y., Nuseibeh, B.: Specifying monitoring and switching problems in context. In: 15th IEEE Int. Requirements Eng. Conf., pp. 211–220 (2007)
21. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-aware systems a research agenda for re for self-adaptive systems. In: 18th IEEE Int. Requirements Eng. Conf., Sydney, Australia, pp. 95–103 (2010)
22. Souza, V.E.S., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness requirements for adaptive systems, technical Report DISI-10-049, DISI, Universit‘a di Trento, Italy (2010)
23. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H., Bruel, J.-M.: RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In: 17th IEEE Int. Requirements Eng. Conf., Atlanta, pp. 79–88 (2009)
24. Zave, P., Jackson, M.: Four dark corners of requirements engineering. ACM Trans. Softw. Eng. Methodol. 6(1), 1–30 (1997)