On Runtime Requirements Adaptation Problem for Self-Adaptive Systems

Nauman A. Qureshi[1], Ivan 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 service-based mobile applications are increasingly immersed in user's daily life to help achieving their needs. Such applications, which operate in open and distributed environment as the Internet, must respond continuously by adapting their behavior to changing user's needs and manage unanticipated changes in the environment conditions at runtime [7], taking into account the availability of resources that is also changing dynamically. For instance, jukebox application enabling the user to download music as per her preferences (e.g. classic rock) while detecting the availability of the network (e.g. Wifi, Edge) and available services (e.g. youtube). Such applications adapt their behavior, having to cope with partial knowledge, uncertainty about unanticipated events (e.g. loss of connectivity) and time constraints. Realizing such self-adaptive systems (SAS - hereafter) challenges the current methods and techniques to engineer them i.e. specifying requirements for adaptivity, design for adaptivity and implementing adaptive behaviors.

Currently, different research communities are proposing a research agenda to address these open issues in engineering such systems. Worth mentioning is the Software Engineering for Self-Adaptive Systems (SEAMS) community which focuses on system architecture aspects[3], and more recently the Requirements Engineering community that has proposed methods for analyzing requirements for self-adaptivity and the idea of making requirements as runtime artifact to be exploited (reasoned on and refined) at runtime[26, 3, 29, 19].

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 task that can satisfy the requirements using e.g. an available service) to meet them, thus ensuring the consistency with the original system's requirements. This internal control mechanism in SAS to monitor, evaluate and adapt (M-E-A loop) makes them particularly difficult to engineer.

Various methods have been proposed so far to engineer requirements for SAS. However, there has been limited consensus among the research communities on what concepts and abstractions are needed to define the requirements for SAS such that RE for SAS to be successfully completed and what is the role of requirements in the adaptation processes at runtime.

The aim of this paper is to identify the kinds of information that pertain to requirements, as well as relations between these information, of which a SAS needs to have an internal representation in order to ensure that it satisfies as best as feasible its users requirements even when there are changes in its operating context and/or in the resources available to it. To this aim, 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 answer the following research questions in the RE for SAS:

1. What concepts and relations capture the requirements-related information that a SAS needs to have an internal representation of in order to satisfy users' requirements, and their change across changing operating contexts and/or resource availability?
2. How do these concepts and relations come together in the formulation of the requirements problem that SAS need to solve?
3. What relationships should be maintained in case of changes that occur dynamically at runtime (unanticipated, or initiated by users)?

This leads us to precisely define the *runtime requirements adaptation problem* that a SAS should be engineered to solve. We thereby suggest concepts and relations that 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.

The rest of the paper is organized by presenting background of our work in Section 2. We define the runtime adaptation problem in Section 3. In Section 4, we present how adaptation problem is connected to RE by revisiting the core ontology for RE and proposing concepts and relationships needed to determine the runtime requirements adaptation problem using examples from travel exemplar case study. Related work and Discussion are presented in Section 5 and 6. Conclusions and future directions are presented in Section 7.

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

2 Background

Our work builds upon problem definitions and ideas developed in studies pertaining to specific, although related areas, namely engineering of SAS, as well as in works on RE methods and specifically on foundations of RE. Basic concepts are recalled below.

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 for any system.

Zave & Jackson formalized the requirements problem as finding a specification (S) in order to satisfy requirements (R) and not violate domain assumptions (K), and thereby ensuring 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 [12]. 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 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.

New concepts suggested in CORE led to the revised formulation of the requirements problem: **Given** the elicited domain assumptions, goals, quality constraints, softgoals, tasks, some of which are optional, mandatory, and/or preferred over others, **find** tasks and domain assumptions which satisfy all mandatory goals, quality constraints, and ideally also satisfy at least some of the preferred and/or optional goals and quality constraints. A candidate solution to this problem will be a consistent set of tasks and domain assumptions which satisfy all mandatory goals and quality constraints. Candidate solutions are compared on the basis of which preferred and/or optional goals and quality constraints they satisfy, so as to select one solution among the candidates, and engineer the system-to-be according to the requirements and other information in that solution.

Techne [13], an abstract requirements modeling language was recently introduced as 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 those present in i-star [33] and Tropos [23]), and it includes only the minimum concepts and relations needed to formalize the requirements problem and the properties of its solutions.

Techne is consequently a convenient formalism for the formulation of the runtime requirements adaptation problem, as it is simple and adapted to the concepts, such as goal, task, domain assumption, and relations (e.g. Techne 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. Further details on the formalism of Techne concepts and relations is available in [13]. We overview below the requirements problem formulation using parts of Techne that we need in the rest of the paper.

Definition 1. ***Techne language:*** *The language $\mathcal{L}$ is a finite set of expressions, in which every expression $\phi \in \mathcal{L}$ satisfies the following* BNF *specification:*

$$x ::= \boldsymbol{k}(p) \mid \boldsymbol{g}(p) \mid \boldsymbol{q}(p) \mid \boldsymbol{s}(p) \mid \boldsymbol{t}(p) \tag{2.1}$$

$$y ::= \bigwedge_{i=1}^{n} x_i \rightarrow x \mid \bigwedge_{i=1}^{n} x_i \rightarrow \bot \tag{2.2}$$

$$\phi ::= x \mid \boldsymbol{k}(y) \tag{2.3}$$

Remark 1. Uppercase Greek letters Δ, Π, Φ, Ψ denote sets of expressions, i.e., $\Delta \subseteq \mathcal{L}$. Lowercase Greek letters ϕ, ψ, α, β, γ denote individual expressions, i.e., members of $\mathcal{L}$. We index or prime symbols as needed. $\rightarrow$ reads "if–then" and $\wedge$ reads "and". Lowercase Latin alphabet letters p, q, r, s denote propositions. ■

Labels on propositions and expressions refer to concepts in the core ontology for requirements, and are assigned as follows:

- **Domain assumptions:** $\mathbf{k}(p)$ if p is an instance of the Domain assumption concept in the CORE ontology, refers to conditions that are believed to hold (e.g., legal norms, assumptions about how some part of the environment behaves, what properties it has, and so on).
- **Goals:** $\mathbf{g}(p)$ if p is an instance of the Goal concept, i.e., p refers to conditions, the satisfaction of which is desired, binary, and verifiable (e.g., desired functionalities of the system-to-be).
- **Quality constraints:** $\mathbf{q}(p)$ if p is an instance of the Quality constraint concept, that is, refers to conditions that constrain the desired values of non-binary measurable properties or behaviors (e.g., the length of the encryption key, the response time of a software module).
- **Softgoals:** $\mathbf{s}(p)$ if p is an instance of the Softgoal concept, i.e., refers to a vague condition that constrains desired values of (potentially) not directly measurable properties or behaviors (e.g., software should respond quickly, interface should be usable).
- **Tasks:** $\mathbf{t}(p)$ if p is an instance of the Task concept, and thereby refers to behaviors of the system-to-be and/or within its operating environment (e.g., find, transform, produce information, manipulate objects).

The language we have is simply a sorted propositional language in which only conjunction and implication connectives are allowed, and are used in a restricted way, as the BNF specification makes clear. The ontology lets us define a sorting function as follows.

Definition 2. ***Sorting function:*** *Sort* $: \mathcal{L} \longrightarrow \mathcal{O}$*, where* $\mathcal{O}$ *is the set of sort labels, one for each of the following concepts: Strict domain assumption, Defeasible domain assumption, Goal, Quality constraint, Softgoal and Task.*

Remark 2. To simplify notation, we will abbreviate $Sort(\phi) = \mathbf{g}$ by $\mathbf{g}\phi$, to say that ϕ is a goal. This abuses the notation somewhat, since the BNF specification tells us that $\phi = \mathbf{g}(p)$. ■

We define the Techne consequence relation as follows.

Definition 3. ***Techne consequence relation:*** *Let* $\Pi \subseteq \mathcal{L}$, $\phi \in \mathcal{L}$, *and* $z \in \{\phi, \perp\}$, *then:*

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

Remark 3. The consequence relation $\mid\!\sim_{\tau}$ is sound w.r.t. standard entailment in propositional logic, but is incomplete in two ways: it only considers deducing positive atoms, and no ordinary proofs based on arguing by contradiction go through, thus being paraconsistent. ■

Techne includes three relations that remain outside the formal Techne language, as they do not participate in the definition of the consequence relation in Techne. These relations are as follows.

Definition 4. ***Preference relation:*** *The preference relation* $\succ \subseteq \mathcal{L} \times \mathcal{L}$ *is an irreflexive binary relation read as follows: if ensuring that* $\phi \in \mathcal{L}$ *holds is strictly more desirable than ensuring that* $\psi \in \mathcal{L}$ *holds, then we say that* ϕ *is strictly preferred to* ψ *and denote this* $\phi \succ \psi$.

Remark 4. We do not require the preference relation to be complete, transitive, or have other specific properties, other than that it is irreflexive. The appropriate choice of these additional properties will depend on, e.g., the automated reasoning framework chosen to compute the most preferred requirements given a set of preference relations. ■

Definition 5. ***Is-mandatory relation:*** *The is-mandatory relation is a unary relation on a requirement to indicate that the requirement must be satisfied, or equivalently, that every candidate solution must include the mandatory requirement.*

Definition 6. ***Is-optional relation:*** *The is-optional relation is a unary relation on a requirement to indicate that it would be desirable for a candidate solution to include that requirement, but that doing so is not mandatory, or equivalently, that if two candidate solutions differ only in one optional requirement, then the candidate solution which includes that optional requirement is strictly preferred to the candidate solution which does not include that optional requirement.*

The consequence relation leads us to the following conception of the *candidate solution* concept.

Definition 7. ***Requirements problem: Given*** *the elicited or otherwise acquired: domain assumptions (in the set* **K***),tasks in* **T***,goals in* **G***, quality constraints in* **Q***, softgoals in* **S***, and preference, is-mandatory and is-optional relations in* **A***,* ***find*** *all* ***candidate solutions*** *to the requirements problem and compare them using preference and is-optional relations from* **A** *to identify the most desirable candidate solution.*

Remark 5. **A** is a set that includes all preference relations, all is-optional, and all is-mandatory relations. ■

Definition 8. ***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. $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{T}} \boldsymbol{G}^*, \boldsymbol{Q}^*$*, 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* $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{T}} \boldsymbol{S}^{\boldsymbol{M}}$*, 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 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 a characterization of requirements as "adaptive requirements" that are concerned with a degree of flexibility in their achievement conditions (either functional or non-functional requirement) that leads to monitoring specification taking into account the variability in the operational context, evaluation criteria and alternative software behaviors [24].

More recently, we have proposed a Continuous Adaptive Requirements Engineering (CARE) framework [26, 27, 25] i.e. a framework for continuous refinement of requirements (i.e. treating them as runtime artifact) and solution provisioning (i.e. leveraging available services) by the system at runtime involving the user. In particular, in the CARE framework, "adaptive requirements" are analyzed and modeled as runtime artifact that reflects system's monitoring and adaptation behaviors, and RE is performed at runtime by the system treating user's requests, making the system to update its initial requirements itself by adding new solutions or deleting the existing ones. Note that we take into account user's requests as new requirement or as a refinement of existing ones.

We have proposed 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) [27]. 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 Defining 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 context, end-user's input and resource variabilities. Such continuously running systems requires flexibility in managing themselves by exercising autonomic properties called as Self-*[15] in response to the changing requirements at runtime.

By definition, SAS at runtime, must be "aware" of the changes and of how they can affect meeting its's existing and current requirements. This calls for designing SAS that are aware of its operational context and its own requirements at runtime [29, 26]. Following this premise, SAS at runtime exercise its sensing capabilities to monitor the changes in the operational environment and evaluate the need to adapt its behavior per-se. We interpret this as a sort of "RE" that we call RE@runtime, where the SAS plays the role of an analysts. Being aware of its requirements we mean SAS can exploit its specification including solutions (itself). It can elicit new elements (e.g. user's needs, context settings and resource variabilities) that affect the initial "requirements problem" leading SAS to analyze new requirements (i.e. refining) by performing adaptation according to the adaptation types defined in [27]. For instance, 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 (ref. to type 1, 2 adaptations); b) compose a new solution by exploiting knowledge on available services or explicitly acquired through user's input or change in context settings (ref. to type 3 adaptation); c) adaptation for which there is no possible solution, SAS must provide a feedback to the user (e.g. design, analyst) for an offline evolution (ref. to type 4 adaptation). With this, SAS can perform elicitation and analysis of new requirements although both processes are more strictly interleaved at runtime.

Recent work on the CARE framework and the types of adaptation proposed in it has led us view runtime requirements adaptation problem by looking at the requirements problem as a "dynamic problem" i.e. solving the requirements problem with continuous refinement of requirements by acquiring new ones. To fulfill this aim, we propose to make explicit the dynamic parts in the requirements problem formulation based on the CORE ontology. Besides that we look at the requirements problem as an optimization problem, as recently argued in [13].

4 RE for SAS and its Core Ontology

Building upon the above considerations, we argue that concepts such as user's context e.g. profile, location, operational) and resource (e.g. tangible, intangible including money, social-relations) must be considered as first class citizens in 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). Moreover, these concepts can enhance the tool set for eliciting and analyzing requirements at runtime.

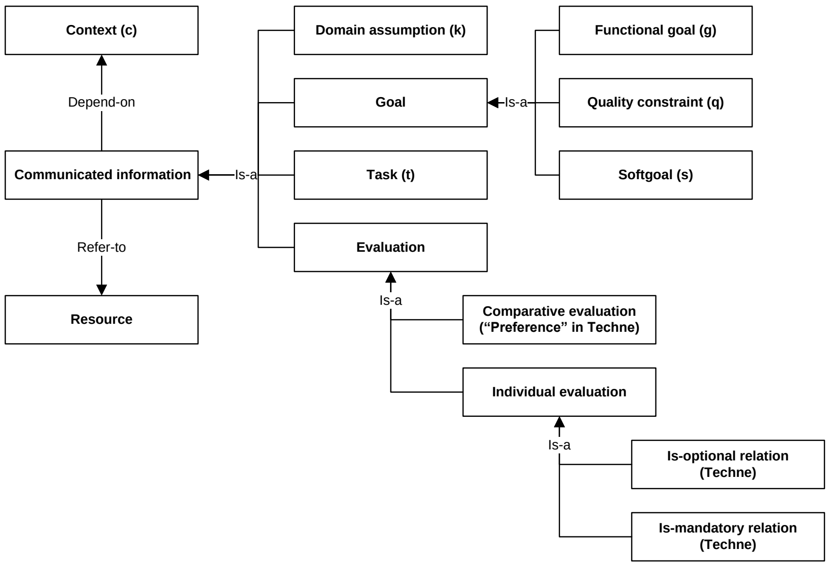

Fig. 1. Context and Resource concepts related to the concepts of the CORE ontology for requirements adapted from [12].

The revised taxonomy of the concepts is proposed taking into account the necessary concepts proposed in Techne [14] to support the definition of runtime "**runtime requirements adaptation problem**". In Fig. 1, concepts Context and Resource are presented relating to the concepts of the CORE ontology. Below, we use the concept of requirements database.

Definition 9. ***Requirements database:*** *A requirements database, denoted Δ is the set of all information elicited or otherwise acquired during the* RE *of a system-to-be.*

Remark 6. Firstly, $\Delta \subseteq \mathcal{L}$, i.e., every member of Δ is an expression in $\mathcal{L}$. Secondly, since Δ should include all information elicited or otherwise acquired in RE, it should include all instances of domain assumptions, goals, softgoals, quality constraints, and tasks that we elicited, found through refinement or otherwise identified during RE. Thirdly, one can view Δ as a repository of information that is usually found in what is informally referred to as a "requirements model". Finally, in 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 7. Below, we will use the term *requirement* to abbreviate "member of the requirements database Δ". I.e., we will call every member of Δ a requirement. ■

To get to the definition of the *runtime requirements adaptation problem*, we start introducing the Context concept. We are aware of the existing definitions and use of Context in the existing AI and RE literature, for instance [21, 9, 28, 2].

Definition 10. ***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.*

Remark 8. 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. ■

The term context can be interpreted in seminal ways, for example, an object, an entity, a surrounding, an environment (operational, external to the system), a set of conditions, a location, a situation. Generally context is a very imprecisely defined concept, it can be defined as *surrounding, circumstances, environment, background, or settings which determine, specify, or clarify the meaning of an event.*[4]

The concept of context has been used in AI and RE literature to characterize information internal or external to the system-to-be. For instance, in AI literature, the term *context has been characterized and formalized to make systems reason on either the circumstances where particular propositions holds or at a certain time instance where the particular proposition might hold* by McCarthy in [21]. In RE literature *context and context-aware systems are defined as an abstraction of location, an event, environment or as a set of conditions that may change overtime* in [28, 11, 10, 2, 1, 30]. Another most common and well accepted definition of context to date is by Dey in [9] i.e. *Context is any information that can be used to characterize the situation of an entity*. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.

Alternatively, *a context **C** refers to any information that helps in characterizing a perceived state of the world*. A context can be characterized as an event or a state of the entity (involving the user) and the operational environment in which the system operates.

[4] http://en.wikipedia.org/wiki/Context

Example: The user is traveling, during transit, the context of the user has changed from *traveling* to *stay at airport*.
Example: Studying *indoor* or *outdoor*, with a profile setting *silent* or *airport* or *normal*, all refers to the perceived state of the world.

At this point, we revise the Techne language to allow information that is included in contexts. This results in adding one more sort.

Definition 11. ***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:*

$$x ::= \boldsymbol{k}(p) \mid \boldsymbol{g}(p) \mid \boldsymbol{q}(p) \mid \boldsymbol{s}(p) \mid \boldsymbol{t}(p) \tag{4.4}$$

$$q ::= \boldsymbol{c}(p) \tag{4.5}$$

$$w ::= x \mid q \tag{4.6}$$

$$y ::= \bigwedge_{i=1}^{n} w_i \to w \mid \bigwedge_{i=1}^{n} w_i \to \bot \tag{4.7}$$

$$\phi ::= w \mid \boldsymbol{k}(y) \mid \boldsymbol{c}(y) \tag{4.8}$$

Remark 9. We used (indexed/primed p, q, r) as an arbitrary atomic statement, every ϕ 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 context), 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 ϕ as a complex statement consists of either w or $\mathbf{k}(y)$ or $\mathbf{c}(y)$. ■

Definition 12. ***Consequence relation of Techne in context:*** *Let $\Pi \subseteq \mathcal{L}_{SAS}$, $\phi \in \mathcal{L}_{SAS}$, and $z \in \{\phi, \bot\}$, then:*

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

Remark 10. The consequence relation $\mid\!\sim_{\mathcal{CT}}$ 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 Δ 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 Δ 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 13. ***Contextualization function:*** *Let **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., ϕ 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 Δ lets us deduce ϕ.*

Remark 11. 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. The formula $\mathbf{k}(p) \wedge \mathbf{c}(q) \rightarrow \bot$ itself can be a context formula – we state this by writing $\mathbf{c}(\mathbf{k}(p) \wedge \mathbf{c}(q) \rightarrow \bot)$ – in order to capture the idea that it is presupposed that $\mathbf{c}(q)$ and $\mathbf{k}(p)$ cannot hold together. Observe that if we write $\mathbf{c}(\mathbf{k}(p) \wedge \mathbf{c}(q) \rightarrow \bot)$, then by the definition of the contextualization function $\mathcal{C}$, we can see that $\mathbf{c}(q) \notin \mathcal{C}(\mathbf{k}(p))$, which informally tells us that we cannot ensure $\mathbf{k}(p)$ in contexts in which we have $\mathbf{c}(q)$.

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.

Thirdly, we could have been more demanding in the definition of the contextualization function $\mathcal{C}$. In the above definition, $\mathcal{C}(\phi)$ will be the set of all information consistent with ϕ, which is usually too much: that set will have information which is not used to derive ϕ, even though it is consistent with ϕ. We leave this definition as it is, however, mainly because it covers the said and other more specific cases and we do not need to focus on minimizing the set $\mathcal{C}(\phi)$ in this paper, given that problems of automated reasoning are outside of the scope in the present discussion.

As an aside, rules that connect requirements and context formulas need not be specified in a definite way by the requirements engineer. It is possible to learn them by asking feedback to the user. 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 (the context in which we asked the user that question):

$$\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_1) \rightarrow \bot) \tag{4.9}$$

$$\mathbf{c}(\mathbf{c}(p) \wedge \mathbf{g}(q_2) \rightarrow \bot) \tag{4.10}$$

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 14. ***Resource:*** *An instance R of the* Resource *concept is an entity either tangible or intangible referred to by one or more instances of* Communicated information.

The concept of resource **R** has been well supported in RE methods such as in goal-oriented approaches [17, 33, 4]. Generally resource can be defined as: *"A resource is any physical or virtual entity of limited availability that needs to be consumed to obtain a benefit from it"*.[5] It can be either natural resource, human resource or tangible/intangible resource.

We refer to tangible/intangible resources in our work e.g. resources that can be consumed or used by the system for the sake of information processing. To be precise, in our case we refer them as physical/tangible entities e.g. mobile phone, ticket itinerary. Whereas, they can also be intangible i.e. user assets i.e. social relations or contacts.

Alternatively, *a resource **R** can be characterized as tangible entity in the world or an intangible entity*.

Example: The user is boarding for the flight. At the check-in counter the itinerary represents an informational resource. It can be either printed or kept in a digital format.

Example: User can view Itinerary on their mobile devices. In this case, itinerary represents a digital information resource, where as mobile device refers to a physical resource. Another case is that user can view their agenda, personal contacts or information about their Money balance. Here, we refer to three assets (intangible resources) i.e. Agenda, Personal Contacts/Social relations and Money. All these can be modified as per user's discretion.

In order to introduce resources in the definition of the 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 15. ***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*

[5] http://en.wikipedia.org/wiki/Resource

returns the identifiers of resources necessary for the domain assumptions and/or context propositions to hold, and/or tasks to be executed:

$$\mathcal{R} : \wp(\boldsymbol{T} \cup \boldsymbol{K} \cup \bigcup \boldsymbol{C}) \longrightarrow \wp(\boldsymbol{R}) \tag{4.11}$$

Remark 12. 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 **C** is a set of sets, so that we need to get the union of all of the sets in **C**. ■

We can now formulate the requirements adaptation problem for SAS.

Definition 16. ***Runtime requirements adaptation problem: Given*** *a candidate solution $S(C_1)$ in the context $C_1 \in$ **C** to the requirements problem RP(C_1) in context $C_1 \in$ **C**, and a change from context C_1 to $C_2 \neq C_1$,* ***find***

1. *the requirements problem RP(C_2) in context $C_2 \in$ **C** and*
2. *choose among candidate solutions to RP(C_2) a solution $S(C_2)$ in the context C_2 to the requirements problem $RP(C_2)$ in the context $C_2 \in$ **C**.*

Remark 13. The definition of the 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 17. ***Requirements problem RP(C) in context C: Given*** *the elicited or otherwise acquired:*

- *domain assumptions in the set **K**,*
- *tasks in **T**,*
- *goals in **G**,*
- *quality constraints in **Q**,*
- *softgoals in **S**,*
- *preference, is-mandatory and is-optional relations in **A**,*
- *a context C on which $\boldsymbol{K} \cup \boldsymbol{T} \cup \boldsymbol{G} \cup \boldsymbol{Q} \cup \boldsymbol{S}$ and **A** depend on,*

*the requirements problem in C" by "**find** all candidate solutions $S_1(C), \ldots, S_n(C)$".*

Definition 18. ***Candidate solution $S(C_1)$ 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. *$C, \boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{T}} \boldsymbol{G}^*, \boldsymbol{Q}^*$, 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 $\boldsymbol{K}^*, \boldsymbol{T}^* \mid\!\sim_{\mathcal{T}} \boldsymbol{S}^{\boldsymbol{M}}$, where $\boldsymbol{S}^{\boldsymbol{M}}$ 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.*

We can define the relegation relation via the inference and preference relations in Techne.

Definition 19. ***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 the following conditions are satisfied:*

1. *$\forall 1 \leq i \leq n$, $\Pi_i \mid\!\sim_{\mathcal{T}} \phi$, i.e., there is an inference relation from every Π_i to ϕ;*
2. *there is a binary relation: $\succ_\phi \subseteq \{\Pi_i \mid 1 \leq i \leq n\} \times \{\Pi_i \mid 1 \leq i \leq n\}$ such that, for any three different $\Pi_i, \Pi_j, \Pi_k \in \{\Pi_i \mid 1 \leq i \leq n\}$:*
   (a) $\succ_\phi$ is irreflexive, i.e., it is not the case that $\Pi_i \succ_\phi \Pi_i$;
   (b) $\succ_\phi$ is transitive, i.e., if $\Pi_i \succ_\phi \Pi_j$ and $\Pi_j \succ_\phi \Pi_k$, then $\Pi_i \succ_\phi \Pi_k$; and
   (c) $\succ_\phi$ is connected, i.e., for any two Π_i and Π_j, either $\Pi_i \succ_\phi \Pi_j$ or $\Pi_j \succ_\phi \Pi_i$.
   whereby $\Pi_i \succ_\phi \Pi_j$ if it is strictly more desirable to satisfy ϕ by ensuring that Π_i holds, than to satisfy ϕ by ensuring that Π_j holds.

Remark 14. 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.:

$$\Pi_1 = \{\mathbf{t}(q_1), \mathbf{b}(\mathbf{t}(q_1) \rightarrow \mathbf{g}(p))\} \quad (4.12)$$

$$\Pi_2 = \{\mathbf{t}(q_2), \mathbf{b}(\mathbf{t}(q_2) \rightarrow \mathbf{g}(p))\} \quad (4.13)$$

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

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 Π_1 and Π_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 $\succ_{\mathbf{g}(p)} = f(\succ)$, 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 Π_1 and Π_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 Π_1 then we will relegate to Π_2, i.e., satisfy $\mathbf{g}(p)$ through Π_2. ■

The purpose of Relegation relation **Rel** is being primarily twofold. First, it facilitates engineer at design-time to analyze requirements (including Goals, quality constraints, preferences) and relegate their associated conditions (e.g. pre/post, achievement, trigger conditions) by anticipating runtime failures scenarios. Second, it enables SAS at runtime to analyze requirements (including Goals, quality constraints, preferences) in case of changes that can occur dynamically e.g. change in user's context, violation of domain assumption, resource usage or change in user's need or preference, either through sensing the operational environment or explicitly given by the end-user. A **Rel** is applied to counter uncertainties (e.g. unanticipated event) by flexibly relegating some of the requirements, if needed, to achieve critical ones.

We take an example of SAS running on user's mobile phone and monitors user's itinerary during her travel. It observes a prolonged delay in the flight due to Icelandic volcanic eruptions. Satisfying user's high level goal at runtime e.g. "Travel for Business Meeting" equates to the execution of a plans e.g. "Book Flight", "Meeting Scheduled", along with their specific parameters and conditions e.g. "flight origin and destination", "flight schedules", "meeting location", "time of meeting". In this case, SAS must re-evaluate the goal conditions and plans parameters and decide to relegate some conditions or preference criteria to formulate a solution that helps achieving the high level goal of the user. SAS may decide to relegate the goal achievement condition by adding another auxiliary goal "Travel for Business 2 days after" taking into account the available information, i.e. "flight schedules delayed for 2 days", user's current context is "at airport". The instance of the goal i.e. "Travel for Business" is linked with the new goal "Travel for Business 2 days after" using **Rel** by operationalizing it with a similar solution with changed parameters i.e. "Book flight after 2 days". This solution is recommended to the end-user. Upon acceptance, the resource i.e. Travel itinerary will be changed subsequently. Alternatively, end-user could change her itinerary by explicitly providing the input to the SAS. **Rel** does not manifest the original goal of the user to be compromised rather relegate it as per the changes in the operational context.

Qualitative Example of Relegation Relation: Goal "Meeting scheduled" with a subsequent quality constraint as "Meeting scheduling will be done in 1 day". For instance, at runtime while the user is traveling encounters a flight delay. It does not mean that the goal becomes invalid, rather the conditions to achieve the same goal requires to be relaxed. This means, evaluating the new available information i.e. context conditions i.e. "At airport" and "Flight Delayed", the goal of the user e.g. "Meeting scheduled" cannot be achieved with the existing solution i.e."Flight to Destination", so it must be relaxed.

In this case by applying **Rel**, either the solution needs to be replaced that operationalizes the goal or an instance of the same goal with revised conditions is linked using **Rel** with the original goal e.g. "Meeting scheduled in the Morning" relaxing the quality constraint it has i.e. "Meeting scheduling will be done in < 1 but > 2 days" such that the preference of the end-user is not disturbed i.e. "prefer morning meetings". In this example, the instance of the original goal is not compromised rather relegated to a new goal with different conditions to achieve it. An alternative plan is generated i.e. "Book flight in morning" to operationalize this new auxiliary goal that stands justified to the previous solution i.e. "Book flight", thus changing the parameters in the solution.

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 that some information influences some other information if the absence of the former makes it impossible for us to satisfy the latter.

Definition 20. ***Influence relation:*** *An influence relation is a binary relation from $\psi \in \mathcal{L}_{\boldsymbol{SAS}}$ to $\phi \in \mathcal{L}_{\boldsymbol{SAS}}$, iff either*

1. $\exists \Pi \subseteq \Delta \cup C,\ \Pi \mid\!\sim_{\mathcal{CT}} \phi$ *and* $\Pi \setminus \psi \not\mid\!\sim_{\mathcal{CT}} \phi$*, or*
2. $\forall \Pi \subseteq \Delta \cup C,\ \Pi \mid\!\sim_{\mathcal{CT}} \phi$ *and* $\Pi \setminus \psi \not\mid\!\sim_{\mathcal{CT}} \phi$*.*

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

Remark 15. If $\psi \xrightarrow{si} \phi$, then we have no way to satisfy ϕ if ψ is not satisfied. If $\psi \xrightarrow{wi} \phi$, then some ways of satisfying ϕ cannot be used to do so if ψ is not satisfied. ■

The purpose for Influence relation is to assess the impact of change in existing concepts and their relationships. An influence relation (**Inf** hereafter) is needed to analyze the impact of change on the instance of concepts with in CORE. This means, if change in the operational environment or in end-user need at runtime causes a change in the instances of the CORE, SAS at runtime needs to be able to ascertain the impact of this change on the existing specification before adapting to an alternative solution. This idea of introducing this relation has been initially argued in our recent work on design methodology for real services [20], where the dynamic semantics of process activities is specified as OCL constraints to analyze the impact of their execution in modifying assets (a type of intangible resource).

In this work, we make an explicit relationship type to provide analysis capability to ascertain the impact of change either motivated by the operational environment, system itself, or by the end-user. At design-time, an engineer performs this analysis as per their expertises by anticipating the dynamic changes that could lead a change in runtime behavior. At runtime, enabling SAS with this analysis capability could result in justifying the adaptation and providing meaningful feedback to the end-user by deriving conclusions on user's goal satisfaction.

Qualitative Example of Influence Relation: Influence relation (**Inf**) determines the process of internal assessment over the dependencies among the concepts in CORE. For example, if the user books the business flight to attend the meeting and confirms his travel itinerary, any subsequent change has consequences over the existing instances (e.g. goals). The change e.g. change in leg of the flight must not invalidate the satisfaction of goal "Book Itinerary" and on a higher level goal "Travel for business". The SAS needs to ascertain the impact of user's request i.e. "Change flight leg" on the existing goal "Book Itinerary" and on the resource "Travel Itinerary" and asset "Money".

In this case, the new goal "Change Itinerary" must be analyzed with respect to new information i.e. change in operational context, existing goals, resources and their operationalizations. In this scenario, it can be ascertain by the SAS taking into account the dependencies exist in the specification e.g. goal tree. Identifying the potential dependencies, it can associate consequences in achieving this new goal e.g. "flight change is eligible" or not by taking into account the domain assumptions associated with the goal "Book Itinerary" i.e. ticket change implies to a charge of amount 100 Euros. Similar dependencies can be collected and subsequent consequences are ascertain by the SAS to analyze the impact of changed solution. Moreover, in doing this what impact of the new solution that operationalize the new goal will have on the resources i.e. "Travel Itinerary" and asset "Money" e.g. how much compensation has to be paid to change the flight. Similarly, in case of removal of an instance the impact of this change on the specification can be derived. **Inf** is, however, critical to ascertain the quality of the adaptation process and evaluating the satisfaction of the user's needs.

4.1 Runtime Requirements Adaptation Problem Illustration

We now revisit the above definitions and using 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. Given the requirements problem in [12, 13]

$$\mathbf{K}^*, \mathbf{T}^* \mid\!\!\approx_{\tau} \mathbf{G}^*, \mathbf{Q}, \mathbf{S^M}$$

Where $\mathbf{K}^*$ is the *believed content* of the specification, $\mathbf{G}^*, \mathbf{Q}, \mathbf{S^M}$ are *desired contents* in the specification as communicated by the stakeholders, such that to arrive to a specification $\mathbf{T}^*$, which brings about the state of the world in which $\mathbf{G}^*$ are satisfied without violating $\mathbf{K}^*$, and by assuring $\mathbf{Q}$. The defeasible consequence relation ($\mid\!\!\approx_{\tau}$) enables non-monotonic satisfaction of goals $\mathbf{G}^*$ if $\mathbf{K}^*$ holds, that is, it can accommodate change in domain assumptions or users goals, which may emerge during requirements refinement process. The reason is that the acquisition of new knowledge at runtime requires revision of solution space, so that inference is defeasible and existing specification $\mathbf{T}^*$ (solutions) must be revised as per new information.

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 $\mathbf{S^M}$ 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 candidate solution CS. At this SAS may perform at sub-optimal level and can exploit automated reasoning techniques such as planning, 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.

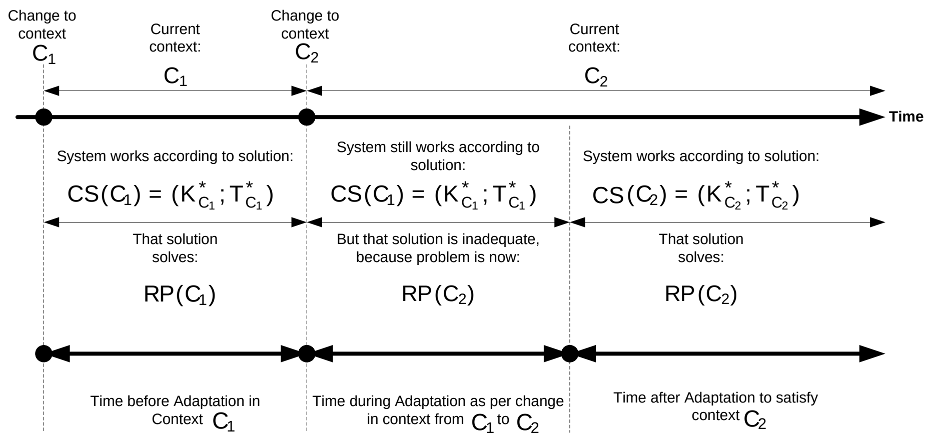
Change to context C_1
Current context: C_1
Change to context C_2
Current context: C_2
Time
System works according to solution:
$CS(C_1) = (K^*_{C_1}; T^*_{C_1})$
System still works according to solution:
$CS(C_1) = (K^*_{C_1}; T^*_{C_1})$
System works according to solution:
$CS(C_2) = (K^*_{C_2}; T^*_{C_2})$
That solution solves:
$RP(C_1)$
But that solution is inadequate, because problem is now:
$RP(C_2)$
That solution solves:
$RP(C_2)$
Time before Adaptation in Context C_1
Time during Adaptation as per change in context from C_1 to C_2
Time after Adaptation to satisfy context C_2

Fig. 2. System Adaptation Sequence in Time

Before Adaptation: Lets consider that at time t^1, to satisfy 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 **Q** is *to have the Internet connectivity not less than 256Kbps*, SAS is exercising its candidate solution CS i.e. set of tasks $\mathbf{T}^*$, e.g. *search for available connection*, *enable Wifi*, *get itinerary details* and *show flight itinerary*, in the current context i.e. C_1 is *at the airport*, and the domain assumption i.e. K^*_{t1} e.g. *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:

$$C_1, \mathbf{K}^*_{C_1}, \mathbf{T}^*_{C_1} \mid\!\!\sim_{\mathcal{T}} \mathbf{G}^*_{C_1}, \mathbf{Q}^*_{C_1}$$

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

During Adaptation: SAS while executing $CS(C_1)$, monitors a change in context i.e. *the airport WiFi 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. Δ 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 may use relegation relation to infer, if the $\mathbf{G}^*$ with a **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 **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:

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

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 $\mathbf{T}^*_{C_2}$.

In Fig.2, the initial solution $CS(C_1)$ becomes invalid due to change in the user's context (e.g. Home to Airport) or change in the environment (e.g. Swear rain caused all the flights to be delayed), i.e. $\neg\ CS(C_1)$ (e.g. flight service becomes unavailable), and the system requires to re-evaluate other instances of the specification. At this stage, relegation relation can be exploited to analyze the alternative ways to satisfy the user's goals, thus the new information is observed to relegate the requirement to another with an alternative solution ($CS(C_1)$ with different set of $\mathbf{T}^*$) to achieve the same goal.
During Adaptation (Alternative Scenario): At time t^2, during the adaptation phase the SAS re-evaluates its CS solution based (or requirements database i.e. Δ), to find a solution set that satisfies the user's goals. A candidate solution could be $CS(C_2)$: *to provide a directed map to the user for shopping at airport;* or $CS(C_3)$: *a possibility to stay at courtesy lounge of the airline*. In other case, $\mathbf{S}^4$: *user flight from Paris to Amsterdam can be changed/canceled and a new itinerary is created using Thaly's train online reservation*. The system re-evaluates the most candidate solution by ranking them taking into account the user's preferences i.e. **A**. For example, $CS(C_4) \succ CS(C_3)$ $\succ CS(C_2)$. At this stage the influence relation is used to ascertain the influence on changing the itinerary (R) on goals and/or on the set of quality constraints or preferences. For instance, SAS can derive conclusions that adapting to $CS(C_4)$ leads to change in R (i.e. Itinerary) in case of C_2(i.e. user at Airport), where (K^*_{C2} (i.e. Any Euro-flight ticket can be transferable to any Euro-rail economy or first class ticket) does not violates and affects of $CS(C_4)$ to be the candidate solution for the achievement of $\mathbf{G}^*$ (i.e. Travel for Business) and $\mathbf{Q}^*$ quality constraint (i.e. reach destination not later than 3 hours) and $\mathbf{S}^*$ (preference i.e. Hi-speed train is preferred than waiting at airport).
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 variabilities.

5 Related Work

Requirements engineering (RE) 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 [7] and RE community has identified key challenges that must be addressed while developing such systems.

For instance, in [32], a declarative language is proposed to capturing 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 [6]. 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 [29]. Similarly, online goal refinement [16] 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 [5].

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 [24]. We extend this work in [26, 27, 25] 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 [3]. Another variation of this idea has been advocated by Mylopulous[6] as Awareness Requirements as way to express requirements as meta requirements to deal with uncertainty while developing SAS.

In context of goal-oriented modeling, *Tropos* has been extended to capture the contextual variability (mainly location) [2] by leveraging the concept of variation points [18] 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). Subsequently, extended design abstractions i.e. environment model, explicit goal types and conditions for building adaptive agents have been proposed by extending *Tropos* as Tropos4AS [22]. A requirements driven reconfiguration is proposed leveraging the concept of context and monitor-diagnosis-compensate loop in [8].

6 Discussion

The problem formulation suggested in this paper makes no assumptions and imposes no constraints on how the information used in the problem formulation is 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

[6] Invited talk at SEAMS 2010 and Dagstuhl Seminar 2010, titled: *Awareness Requirements*

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, K and T, among others. In other words, we may change the formalism, but the principles on what information we will be dealing with, and what relations it should stand in, when engineering SAS does not change: we will still have goals to satisfy, preferences to take into account, domain assumptions, and context conditions that we should not violate.

RELAX [31] was introduced as a formalism for the specification of requirements in a way that allows their relaxation (deidealization) at runtime: if a requirement cannot be satisfied to the desired extent, then alternative requirements can be specified in RELAX, stating thereby how that requirement can be relaxed at runtime. It is possible to specify that a requirement be relaxed by satisfying another requirement (i.e., instead of requirement A, satisfy requirement B), by changing the time of its satisfaction, by minimizing the difference between the current time and the time it is satisfied, by changing the frequency at which it needs to be satisfied, by changing the value of a measure that should be achieved according to the requirement, and so on. What is interesting about RELAX is that it recognizes the need to actually move away from very precise goals and quality constraints when engineering the requirements for SAS, and upwards (in terms of abstraction and imprecision) towards vague statements. RELAX operators, such as ”eventually”, ”before”, ”after”, ”as close as possible to”, ”as many, few as possible” are vague – as in linguistics: namely, a term is vague if we have no criterion to say when exactly that term applies. E.g., can we be satisfied by anytime (and thereby unknowingly far off) satisfaction when using the operator ”eventually”? How close do we need to be to a value in order to say that we are ”as close as possible to” it? While it is certainly valid to want to defer decisions by using vague operators, and thereby let the system decide by itself, one must specify how the system should decide – i.e., we can defer the decision, but we cannot avoid making it, and this is in RELAX done by saying that the system should decide through membership functions, or more generally, vagueness of the operators is resolved through fuzzy logic. What we are interested in here, is if RELAX can specify something that we cannot capture in the problem formulation. Since all that RELAX does is delegates the choice between explicit alternatives at design time by the engineers and stakeholders, to the same choice at runtime by the system and on the basis of membership functions, it still operates in the general same way as our formalism: it says that we need to have alternatives, to compare them, and to choose one of them. It is straightforward how to rewrite every RELAX operator via the preference and inference relations in the language we used in this paper. As an example, consider this, borrowed from Whittle et. al. [31]:

> "The fridge SHALL detect and communicate information with AS MANY food packages AS POSSIBLE."

Above, the capitalized words are operators in the RELAX formalism. The system will understand this through a membership function which has a value zero when the fridge detects and communicates information with no food packages at all, and increases continuously with the increase in the number of packages that the fridge detects and communicates with. This can be very well expressed without going to fuzzy logic. Namely, we start by saying that the fridge has the softgoal "$\mathbf{s}(p)$: detect and communicate with as many food packages as possible", and we can approximate that softgoal by a range of quality constraints: "$\mathbf{q}(q_1)$: detect and communicate with 1 food package", "$\mathbf{q}(q_2)$: detect and communicate with 2 food packages", and so on, with the number of packages that increases. We then specify the preference relations, saying that for every $i > 0$ $\mathbf{q}(q_{i+1})$ is strictly to $\mathbf{q}(q_i)$. This is obviously more cumbersome than to specify a membership function, but our point in this paper is not to show that we do something more efficiently than others, but to give a *general* framework in which we can represent and understand the key concepts, relations, and problems to be solved in the RE of SAS. Our Relegate relation is a more general relation than the RELAX operators, since we do not commit to fuzzy logic: we only ask for a way to represent alternatives and to compare them. RELAX is a particular way to relax requirements (a particular way to implement the Relegate relation), that obtains a straightforward interpretation in the language we used here, and thereby, the RELAX operators cannot capture information other than that which we can capture in the formalism in this paper. There are other ways to handle uncertainty and relaxation of requirements, and our aim in this paper was to remain independent of particular approaches. RELAX adopts a particular approach, and our argument is that we should first understand the general problem, and then focus on developing particular requirements modeling languages to handle it.

Engineering of SAS aims to resolve runtime requirements adaptation problem by finding a candidate solution in response to changing context, resource variabilities and emerging end-user's needs. Intuitively, resolving such problem amounts to an optimization problem, where admissible set of candidate solutions (e.g. realized through available services) need to be exploited at runtime such that SAS is able to reason on them taking into account the contextual changes and existing requirements. SAS requires practical methods to perform non-monotonic reasoning at runtime to find candidate solutions. There are various ways in which to search for candidate solutions e.g. search based RE and AI planning techniques, which are also promising in this regard. Discussing these is outside of the scope of this paper.

7 Conclusions and Future Work

In this paper we presented runtime requirements adaptation problem taking into account our work on continuous adaptive RE (CARE) and they types of adaptation proposed in CARE, we proposed to make explicit the dynamic parts in the requirements problem formulation based on the CORE ontology and Techne. We proposed two concepts and relations to formulate the runtime requirement adaptation problem for SAS. Moreover, we extended Techne by adding these concepts and relation and suggested formalism to give a *general* framework in which we can represent and understand the key concepts, relations, and problems to be solved in the RE of SAS.

We are further investigating on exploiting these formal definitions of concepts and relations into a more concrete modeling and analysis language to engineer requirements for SAS that can not only support the engineer at design-time but also provide reasoning capabilities to the SAS to perform RE at runtime by itself involving the user. To fulfill this aim, we are considering Techne as an abstract RML to derive a sub language to represent adaptive requirements [24] for SAS. Apart from design-time support, at runtime SAS can exploit such requirements to continuously refine and to reason on its own requirements (e.g. leveraging CARE [26]), thus resolving the runtime requirements adaptation problem taking into account the changing context, resource variabilities and end-user's needs.

References

1. Ali, R., Dalpiaz, F., Giorgini, P.: A goal modeling framework for self-contextualizable software. In: 14th International Conference on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD09). Springer, Springer, Amsterdam, The Netherlands (08/06/2009 2009)
2. Ali, R., Dalpiaz, F., Giorgini, P.: Location-based software modeling and analysis: Tropos-based approach. In: Li, Q., Spaccapietra, S., Yu, E.S.K., Olivé, A. (eds.) ER. Lecture Notes in Computer Science, vol. 5231, pp. 169–182. Springer (2008)
3. Baresi, L., Pasquale, L., Spoletini, P.: Fuzzy goals for requirements-driven adaptation. In: 18th IEEE Int. Requirements Eng. Conf. pp. 125–134. Sydney, Australia (2010)
4. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
5. Brun, Y., Serugendo, G.D.M., Gacek, C., Giese, H.M., Kienle, H., Litoiu, M., Mller, H.A., Pezz, M., Shaw, M.: Engineering self-adaptive systems through feedback loops. Software Engineering for Self-Adaptive Systems 5525, 48–70 (2009)
6. 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: MoDELS. Lecture Notes in Computer Science, vol. 5795, pp. 468–483. Springer (2009)
7. Cheng, B.H., de Lemos, R., Giese, H., Inverardi, P., Magee, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26. Springer-Verlag, lncs 5525 edn. (2009)
8. Dalpiaz, F., Giorgini, P., Mylopoulos, J.: An architecture for requirements-driven self-reconfiguration. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE. Lecture Notes in Computer Science, vol. 5565, pp. 246–260. Springer (2009)
9. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. 5(1), 4–7 (2001)
10. Feather, M.S., Fickas, S., Lamsweerde, A.V., Ponsard, C.: Reconciling system requirements and runtime behavior. In: IWSSD '98: Proceedings of the 9th international workshop on Software specification and design. p. 50. IEEE Computer Society, Washington, DC, USA (1998)
11. Finkelstein, A., Savigni, A.: A framework for requirements engineering for context-aware services. In: In Proc. of 1st International Workshop From Software Requirements to Architectures (STRAW 01). pp. 200–1 (2001)

12. 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)
13. 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. pp. 115–124. Sydney, Australia (2010)
14. 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: International Conference on Requirements Engineering (RE). Sydney, Australia (Sep 2010)
15. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer 36(1), 41–50 (2003)
16. Kramer, J., Magee, J.: Self-managed systems: an architectural challenge. Future of Software Engineering, 2007. FOSE '07 pp. 259–268 (May 2007)
17. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: 5th IEEE International Symposium on Requirements Engineering. p. 249 (2001)
18. 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)
19. Liaskos, S., McIlraith, S.A., Mylopoulos, J.: Integrating preferences into goal models for requirements engineering. In: 18th IEEE Int. Requirements Eng. Conf. pp. 135–144. Sydney, Australia (2010)
20. Marchetto, A., Nguyen, C.D., Francescomarino, C.D., Qureshi, N.A., Perini, A., Tonella, P.: A design methodology for real services. In: PESOS '10: Proceedings of the 2010 ICSE Workshop on Principles of Engineering Service Oriented Systems (2010)
21. 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, CA, USA (1993)
22. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. In: SEAMS '08: Workshop on Software engineering for adaptive and self-managing systems, colocated with ICSE 2008. pp. 9–16. ACM, New York, NY, USA (2008)
23. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: High variability design for software agents: Extending Tropos. TAAS 2(4) (2007)
24. Qureshi, N.A., Perini, A.: Engineering adaptive requirements. In: Workshop on Software Engineering for Adaptive and Self-Managing Systems. pp. 126–131. Vancouver, BC (May 2009)
25. Qureshi, N.A., Perini, A.: Continuous adaptive requirements engineering: An architecture for self-adaptive service-based applications. In: Proceedings of the First International Workshop on Requirements@run-time, held at (RE'10). Sydney, Australia (2010)
26. Qureshi, N.A., Perini, A.: Requirements engineering for adaptive service based applications. In: 18th IEEE Int. Requirements Eng. Conf. pp. 108–111. Sydney, Australia (2010)
27. Qureshi, N.A., Perini, A., Ernst, N.A., Mylopoulos, J.: Towards a continuous requirements engineering framework for self-adaptive systems. In: Proceedings of the First International Workshop on Requirements@run-time, held at (RE'10). Sydney, Australia (2010)
28. Salifu, M., Yu, Y., Nuseibeh, B.: Specifying monitoring and switching problems in context. In: 15th IEEE Int. Requirements Eng. Conf. pp. 211–220 (2007)
29. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-Aware Systems. In: International Conference on Requirements Engineering (RE). Sydney (2010)
30. Sitou, W., Spanfelner, B.: Towards requirements engineering for context adaptive systems. In: COMPSAC '07: Proceedings of the 31st Annual International Computer Software and Applications Conference. pp. 593–600. IEEE Computer Society, Washington, DC, USA (2007)

31. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B., Bruel, J.M.: Relax: a language to address uncertainty in self-adaptive systems requirement. Requirements Engineering 15, 177–196 (2010)
32. 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. pp. 79–88. Atlanta (2009)
33. Yu, E.: Modelling Strategic Relationships for Process Reengineering. Ph.D. thesis, University of Toronto, Department of Computer Science, University of Toronto (1995)