

# Towards a Continuous Requirements Engineering Framework for Self-Adaptive Systems

Nauman A. Qureshi, Anna Perini  
Fondazione Bruno Kessler - IRST  
Via Sommarive, 18, 38050 Trento, Italy  
{qureshi,perini}@fbk.eu

Neil A. Ernst  
Department of Computer Science  
University of Toronto, Canada  
nernst@cs.toronto.edu

John Mylopoulos  
Dept. Inf. Eng. and Comp. Sci.  
University of Trento, Italy  
jm@disi.unitn.it

**Abstract**—Requirements engineering (RE) for self-adaptive systems (SAS) is an emerging research area. The key features of such systems are to be aware of the changes in both their operating and external environments, while simultaneously remaining aware of their users’ goals and preferences. This is accomplished by evaluating such changes and adapting to a suitable alternative that can satisfy those changes in the context of the user goals. Most current RE languages do not consider this ‘reflective’ and online component of requirements models.

In this paper, we propose a new framework for building SAS that is goal- and user-oriented. We sketch a framework to enable continuous adaptive requirements engineering (CARE) for SAS that leverage requirements-aware systems and exploits the *Techne* modeling language and reasoning system. We illustrate our framework by showing how it can handle an adaptive scenario in the travel domain.

## I. INTRODUCTION

Many software-intensive systems are now expected to operate continuously, such as service based applications (SBA), that rely on third party services and operate in an open environment such as the Internet. And while they must be always-available, these systems must also adapt to any and all changes in their operating environment. The only way to understand what changes are acceptable in a system is with respect to its requirements, and more specifically, its intentions. Consequently, requirements researchers have begun to consider the implications of involving requirements throughout the software lifecycle [1], or ‘requirements at run-time’. Work in this area has targeted dynamic requirements specification [2], requirements monitoring [3], [4], managing uncertainty at requirements time [5] and treating requirements models as live artifacts [6], [7].

More recently, the concepts of ‘reflective requirements’<sup>1</sup>, that is, requirements which can be examined by the system itself, has been further elaborated by Sawyer et al. in [8] and motivated the proposal of a research agenda for *requirements-aware systems*.

In our view, the role of a reflective requirements is to provide guidance to the system when considering changes or new requirements at run-time. Beside the core issues

mentioned above, we believe that the following research questions must be addressed to realize requirements-aware systems: which framework (including concepts and methods) can support RE performed at run-time? And in particular, can we talk about RE activities at run-time, and how can we do them in practice?

In this position paper, we discuss these research questions, elaborating on some of our recent work [9], [10], [11]. First, we revisit the existing RE ontology [12] and consider the potential of a novel RE language (*Techne*) [9], [10] for modeling requirements for SAS. Second, we elaborate on the need for continuous RE and consider the recently proposed **CARE** method [11] in combination with **Techne**, as a candidate approach to enable requirements-aware systems.

In the rest of the paper, we describe an extended example (in Section II), that will be used to support our discussion throughout the paper. Our understanding of adaptation in online systems is presented in Section III; recently proposed concepts for RE of SAS are revisited in Section IV; the difference between RE performed at design- and at run-time is discussed along the motivation for continuous RE in Section V. We then introduce CARE, our framework for Continuous Adaptive RE in Section VI; and conclude with some thoughts on future research in Section VII.

## II. EXTENDED EXAMPLE

To illustrate our approach we have chosen a travel example, called *Travel Companion* (a service based application).

A *Travel Companion* helps its users by monitoring their travel itineraries. Travel itineraries are generated by accessing available services such as airlines. It is based on available services, which are partially known at design-time, and also can be added to manage uncertainty/variability at run-time. It provides an interface to the user to browse and search available travel and allied services to prepare her travels. It also allows for plan modification, either from user requests, or initiated by the *Travel Companion* on behalf of the user, based on her current context and preferences.

The system helps the user to better manage and organize her travels by being aware of run-time changes. The key adaptation in the *Travel Companion* is to identify new

<sup>1</sup>attributed to Anthony Finkelstein, quoted in [1]

competing services, and to translate service requests as a candidate requirement update. Furthermore, the system is able to manage the overall activities of the user and provide better services to her by reasoning over the existing and new requirements. Travel Companion is flexible to accommodate the changes in user requirements, quality preferences and contextual changes that might cause it to adapt dynamically by composing an optimal solution (set of services) for the user on the fly.

To support discussing the difference between RE at design and run-time, and the need for continuous RE, we describe a scenario we expect a requirements-aware SAS to handle.

**Scenario:** *The Icelandic volcano recently caused lengthy delays in flights airport closures. The severity of this event was not anticipated at design-time; nearly all flights were cancelled and trains and buses overwhelmed. The Travel Companion, while monitoring the user's itinerary, helps to find the most suitable alternative for the traveler to reach her original destination after her business meeting. Trains and car rentals are considered but are too expensive per the user's preferences. The user and system work in conjunction to find the best alternatives which satisfy preferences and mandatory goals.*

### III. ADAPTATION TYPES

The classification discussed in [13] proposes four level of RE for SAS (called dynamically adaptive systems in that paper). Consider (after [13]) a system realized in (alternative) programs  $S \in \mathcal{S}$  for a given set of (anticipated) domains  $D \in \mathcal{D}$ . Level 1 RE is performed by the designers to produce a set of programs  $\mathcal{S}$ , anticipating a set of possible domains  $\mathcal{D}$ . Level 2 RE is performed by the system itself, at runtime, to select between programs as the operating environment changes. Level 3 RE is done to update the requirements themselves in response to unanticipated changes. Finally, Level 4 RE refers to research on RE in DAS (such as this paper).<sup>2</sup> In the context of this paper we consider self-adaptive service based applications as a type of DAS.

We believe there are three different tasks involved in requirements-aware systems: one, how to change the current  $S$  to a new  $S'$ , with respect to the system requirements – i.e., the mechanics behind switching programs; two, how to select which new  $S'$  to choose, with respect to the requirements – i.e., the decision problem; and three, corresponding to Level 3, how to update the requirements themselves when there is no available  $S \in \mathcal{S}$  which responds to the new  $D$  – the requirements problem.

We borrow from ideas in [13], [11] and [8] to illustrate our classification of adaptation types a SAS must accommodate.

<sup>2</sup>Aside: the difference between autonomic or SAS and requirements-aware systems is that the former does not consider requirements explicitly, while the latter always makes reference to a consistent and contemporaneous requirements model.

**Type 1** consists of changes which are anticipated at design-time and for which alternatives exist in the specification, and the best alternative is obvious. This type conforms to Level 1 in [13] and Type 1 in [11].

Example: *During the World Cup, Twitter's servers become unresponsive. The manager service automatically offloads jobs to an Amazon content distribution network.* Type 1 events are familiar and anticipated. This is the classical autonomic computing scenario.

**Type 2** consists of changes in the environment e.g. context, preferences etc. The system must monitor such changes; after evaluating the change it adapts to the most feasible solution that satisfies the change on its own. Here, the system mainly exploits its alternative space of solutions by being aware of its user's goals and preference. This type relates to Level 2 in [13] and Type 2 in [11].

Example: *A mobile system knows the user wants a four-star hotel near the beach (monitor e.g. preferences), but cannot optimize both variables simultaneously. It tries to book the cheaper three-star hotel near the beach, which also satisfies the user's cost preference, or it proposes to her a four-star near the city center.*

**Type 3** consists of changes which are unanticipated by the designer, due to partial or uncertain knowledge about the environment. The events are also unfamiliar to the user. However, the system can query the user for new requirements and attempt to replan dynamically to accommodate them. This point is elaborated further below, but relies on a shared ontology to infer similarity and techniques to find the appropriate service satisfying that requirement. This type relates to the Level 3 in [13] and Type 3 in [11].

The main difference with respect to Level 3 in [13] is that our framework involves the user in specifying new requirements (where *user* is seen as distinct from the *designer*). The system itself analyzes the user's request as a refinement of existing requirements or an entirely new requirement. Once an appropriate solution is found, this request is updated in the original specification at run-time. When there is no solution found, the system opts for Type 4. In Cheng et. al.'s classification [13], Level-3 RE is performed before Level-2, but the order can be changed. In our case, we perform both Level-2 and Level-3 concurrently unless we have to do offline maintenance or evolution (our Type-4).

Example: *Julia reaches Oslo, but all hotels are full, and the system specification does not include the alternative of searching for home-stays. At this point, the system prompts Julia to say that there is no suitable itinerary, and asks for possible work-arounds. Julia informs the system that a home-stay (a sub-concept of Accommodation) is possible, and the system attempts to find a service which can handle this (e.g. using swoogle.umbc.edu).*

In the above example, user specifying an option of "Home-Stay" is considered as a new requirement for the system to operationalize. Such requirement is an alternative

to users' high level goal "Find Accommodation". In service-oriented RE (SORE) [14], user requirements are used to discover services either at design-time or at run-time to operationalize them. The requirements analysis is conducted with a pre design-time knowledge about the existing available services. Requirements model requires a continuous revision as new services are discovered to operationalize them, differently as in case of object oriented analysis and design.

**Type 4** consists of Type 3 events for which there is *no* possible addition or simple refinement. A new specification and the relative new system are generated. In our view of requirements-aware systems, this is the worst-case scenario.

We term Type 4 changes *system evolution*, as distinct from *system adaptation* (Types 1-3). This is roughly analogous to biological systems, where adaptation reflects the ability of an individual to cope with environmental variance, while evolution is a widespread genetic change to prolonged stimuli. One should be careful not to over-extend this metaphor, however, as biological systems are not designed nor does evolution progress towards 'better' designs.

#### IV. CONCEPTS

RE aims at identifying the purpose for the system-to-be by analyzing knowledge about domain assumptions, goals and preferences of the stakeholders. This knowledge is typically collected at the outset of the system development process, but in the case of systems that will operate in an open and dynamic environment, this knowledge refers also to aspects that undergo changes in an unpredictable way. This has called for revisiting traditional concepts and methods of RE. We will first revisit recent proposals to express requirements for SAS and requirements-aware systems, which motivates the proposal of *continuous RE*. We investigate how **Techne**, an abstract modeling language which exploits the core ontology for RE, can provide solutions to represent those type of requirements and supports *continuous RE*.

##### A. Adaptive Requirements

The concept of *Adaptive Requirements* has been proposed in [15] to describe a system's properties, that include not only functional or non-functional requirements but also monitoring specification, evaluation criteria and adaptive actions specified as plans, which are needed to ensure that the system's behaviour meets them when operating in a dynamic environment. Such plan-oriented specification, defined at design-time, can be used by the systems at run-time as a *Live Artifact* to reason for adaptation and to re-plan, if necessary. Adaptive requirements capture a dynamic view of system's requirements. They encompasses all the possible alternatives that are considered at design-time, i.e. it defines Type 1 adaptation.

Concepts similar to that of adaptive requirements have been recently proposed by Baresi et. al. [6] and Mylopou-

los<sup>3</sup>. The former extends the KAOS method [16] to model functional requirements, incorporating "Adaptive goals" to represent adaptation strategies, and countermeasures to address goal violations and goal conflicts. Subsequently, they map these models to executable business processes to derive possible service compositions at run-time. The latter proposes "Awareness Requirements" as a candidate solution to manage uncertainty. In goal models requirements are expressed as meta specifications of other requirements to provide awareness capabilities to the system.

##### B. Service Requests and Continuous RE

At run-time, besides changes in the operational context that may be captured by the system through monitoring, new requests may be raised by the users. In [2], the concept of *Service Requests* has been introduced to manage "online RE". We leverage this concept in [11] and provide a machine-readable format to express such requests in XML. Fig 1 shows such a Run-time Requirements Artifact (RRA). RRA can be expressed by the user (switching to Type 3 adaptation) or it can be generated by the system taking into account the dynamic changes in the environment through monitoring (switching to Type 2 adaptation).

```
<?xml version="1.0" encoding="UTF-8"?>
<Request xmlns="http://www.example.org/Request">
  <RequestDescription>
    <Source>User </Source>
    <Func_Goal> Find Accommodation</Func_Goal>
    <Alternative Func_Plan> Friend's House</Alternative Func_Plan>
    <Context> Location: Ams_Airport;</Context>
    <Quality_Attribute> Convenience</Quality_Attribute>
    <Preference> ~Taxi & ~Metro </Preference>
    <Stimulus>Search Map & Search Transport</Stimulus>
    <Date> Current</Date>
    <Time> Evening</Time>
    <Response> Map: Google Map; Amsterdam Transport Service</Response>
    <Resource>Mobile, Contact List </Resource>
  </RequestDescription>
</Request>
```

Figure 1. Users' Service Request as RRA, Adapted from [11].

The key benefit of translating and acquiring a user's service requests (RRA) as requirement is two-fold. First, the user is involved in the RE process enabling the Travel Companion to (1) get new requirements (new services to make use of, existing services to drop); (2) refining existing requirements, which just requires an alternative solution to be added with the existing one to satisfy them; (3) to accommodate user's preferences over time by refining the selection of services for quality reasons. Secondly, the SAS (e.g. Travel Companion) does RE on its own for the user. It provides an optimal solution to the user based on her preference e.g., minimum cost and convenience, context (profile, location, operational) and resource variability e.g. mobile device; or changes that might occur due to unanticipated events e.g. flight delays.

<sup>3</sup>Invited talk at SEAMS 2010, titled: *Awareness Requirements*

### C. CORE ontology for RE

According to [12] the requirements problem definition is stated as:

$$K, P \vdash G, Q, A$$

where  $G$  is a set of goals,  $Q$  a set of quality constraints on soft goals, and  $A$  the user’s attitudes to elements of the requirements (such as preferences or desired elements).  $K$  represents domain assumptions and  $P$  a set of tasks to perform (the specification). These elements extend the existing RE ontology as  $W, S \vdash R$  by attaching the concepts of goals and softgoals to the  $R$  component, and incorporating research about stakeholders’ attitudes to requirements and specifications (e.g., prioritization).

Given that (K) is the *believed content* of the specification, (G,Q,A) are *desired content* in the specification as communicated by the stakeholders. The requirements problem is to arrive at a specification (P), which brings about the state of the world in which (G,Q) are satisfied without violating (K), and by accommodating (A). The defeasible consequence relation ( $\vdash$ ) enables non-monotonic satisfaction of goals (G) if (K) holds, that is it can accommodate change in domain assumptions or users goals, which may emerge during requirements refining. This set of concepts (K, P, G, Q, A) defines the core ontology for the requirements problem.

Considering the above definition of the requirements problem in the case of SAS, the changes in the environment are either perceived as a change in user’s context (profile, location or device), operational context (execution environment) or are influenced or constrained by the resources (tangible or in-tangible). For example, when a user arrives at the airport, the assumption (K) is that the Internet must be available at the airport. The goal (G) is to connect to the Internet for checking details of her itinerary, (Q) is to have good connectivity. In this case the plan (P) must be to look for available connection, which in turn satisfies the goal (G) and (Q).

### V. TECHNE LANGUAGE AND INCREMENTAL REASONING

Techne is an abstract modeling language that adopts the core ontology for RE to express system requirements. Goals and quality constraints may be mandatory or optional, and preference relations can be defined over them (A). A requirement problem is represented as a net with labeled nodes and edges, which captures instances of the concepts and relationships of the ontology. A solution consists of a collection of tasks and quality constraints, such that, when the tasks are executed they entail the satisfaction of all mandatory goals and maximize satisfying the optional ones. Different solutions can be associated to a requirements net. They may be classified in terms of preference and optionality.

Here we investigate how using **Techne** models with support for expressing stakeholder preferences, combined with a solution repository with incremental revisions, enables managing Type 3 adaptation situations.

#### A. RE at Design-time

At design-time for a specific system, elicitation is performed for domain-specific goals (e.g., *Travel on Business*). This domain RE model is used to create a domain module under the control of the system. From the resulting goal model, the **Techne** evaluation procedure [10] finds Pareto-optimal sets of solutions, and the specification is created as a single optimal solution  $P'$ . The remaining optimal solutions are stored in a solution repository.

Fig. 2 shows our **Techne** model for the travel domain (bottom) and a possible system architecture<sup>4</sup> (top). **Techne** models extend goal models with the notion of user attitudes, which are captured as either preferences between model elements, or optional (nice-to-have) requirements.

**Design-time Reasoning.** For the example, a specification  $P$  (also known as a ‘solution’ to the requirements problem) is found as follows. We first find sets of tasks and domain assumptions which can make our *mandatory* requirements hold (mandatory elements are distinguished with double-lined edges). In the figure, the mandatory goals are **Travel on Business** and **Business Successful** for the domain module, and **Provide Assistance Service** for the system model. Typically, but not necessarily, such elements are high-level goals for the system.

A combination of tasks and domain assumptions is a *candidate solution* for the problem of satisfying the mandatory goals. There are possibly many candidate solutions. For example, the set  $S_1$ : {**Receive (Tickets) via Email, Book Plane, Reserve Online**} is a candidate solution. An alternative is to replace **Receive (Tickets) via Email** with **Receive (Tickets) by Mail** to produce  $S_2$ .

To sort candidates, we use a decision procedure for ranking them, taking into account the number of preferences satisfied and the number of options included. A candidate solution with strictly more of either satisfied preferences or included options is a preferred solution. Returning to the solution  $S_1$ , we see that since **Receive via Email** is preferred to **Receive via Mail**,  $S_1$  dominates  $S_2$ .  $S_1$  has one satisfied preference and one optional goal achieved (**Convenience**) while  $S_2$  has none.

Preferences and options are incomparable, so this is a multi-criteria decision problem, and there may be a Pareto-front of equally acceptable solutions (e.g., solutions which satisfy one preference and two options, or two preferences and one option, respectively, are incomparable). One such solution is chosen by the system, and used as our initial specification  $P$ .

<sup>4</sup>There is no canonical graphical syntax for Techne: shown is a possible syntax.

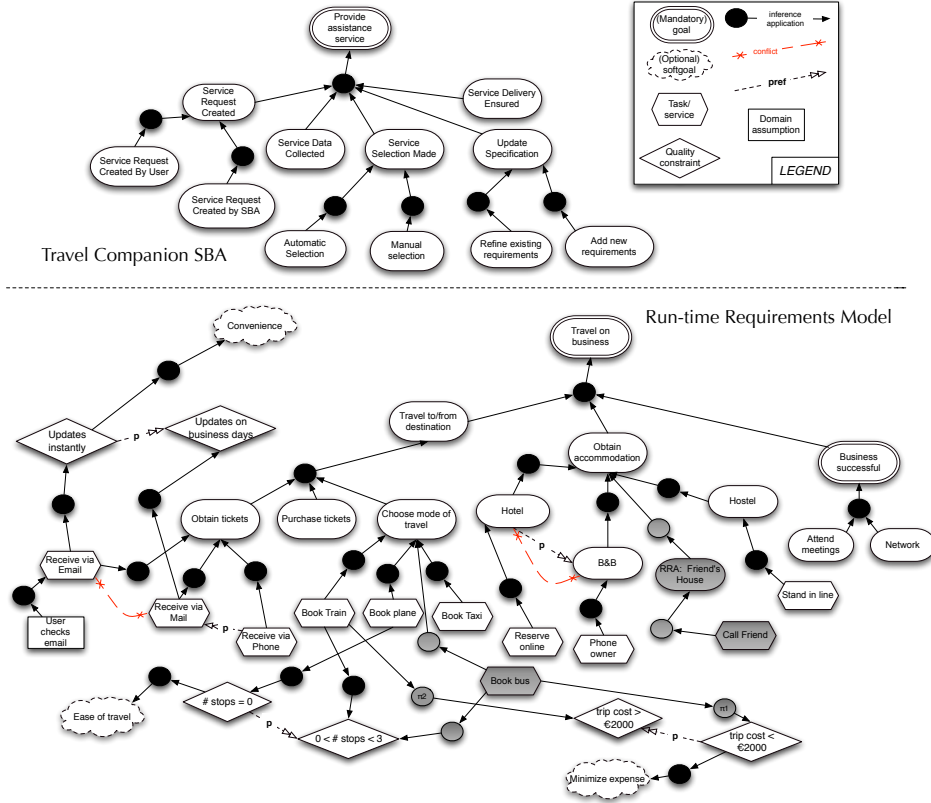


Figure 2. **Techne** model for the Travel Companion example. Grey nodes represent added requirements. Portions of the model are missing where necessary for space reasons.

The remaining candidate solutions are stored in a requirements repository. The idea is that as our system detects changes in the situation, other solutions (alternatives) become applicable, and another might be more suitable. Consider the case where a user is offline. In this case, the task *Receive via Email* is no longer possible, and the system will have to switch to another task to achieve the goal *Obtain Tickets*.

### B. RE at Run-time

At run-time, the solution repository is searched by the system (e.g. *Travel Companion*), and a solution which matches the current situation is selected ( $P$ ). The system monitors the environment for changes (such as a delayed flight), and then makes system adaptations according to which Type the change impacts. In Types 1 and 2, the existing solutions in the repository suffice for adaptation. In Type 3, the user can add requirements, including preferences, and **Techne** re-evaluates the set of solutions incrementally. In Type 4, the system is taken offline and we begin at the ‘design-time’ phase, so to speak.

**Run-time reasoning.** There are two components to the run-time reasoning. In one, we leverage **Techne** to find and retrieve requirements solutions from the repository. In

the other, the system works with the user to identify new requirements or modifications to the model, which are then fed back into the model reasoner.

Consider the Iceland volcano scenario described above. While the system is in operation, it detects that the user’s flight was cancelled. The system attempts to update the itinerary to accommodate flight cancellations. It cannot automatically shift plans (Type 1), since there is no obvious choice. It proposes to the user a train trip via Austria to Norway. Since a (Type 2) event occurred, it attempts to replan using an existing alternative. Since the optional goal *Ease of Travel* is not satisfied, and yet the train will increase the price of the trip ( $\pi_2$ ), the user reject this. As there are no remaining alternatives, the system shifts to (Type 3), and prompts the user for a run-time requirement. The user suggests booking a bus (grey node  $\pi_1$  in Fig. 2). Choosing to go by bus satisfies the *Minimize Cost* goal, although *Ease of travel* is not satisfied.

It should be noted that this example is similar to examples in the AI planning domain. Planning is a lower-level operation using goal operationalizations, devising a plan to satisfy them. Our RE reasoning is more high level, and involves design decisions about the system to be, maintaining a separation between the problem space and the solution space

(this division is increasingly fuzzy, however, see e.g. [17]). In a low-level implementation, of course, both AI planning and RE reasoning are both search problems.

## VI. CONTINUOUS ADAPTIVE REQUIREMENTS ENGINEERING (CARE) FRAMEWORK

In [11], we proposed **CARE**, a framework for Continuous Adaptive Requirements Engineering. We extend this framework by leveraging **Techne** and its reasoning support.

We adopt a goal-oriented framework based on the core ontology for RE [12]. In this, a requirements engineering activity finds combinations of tasks and domain assumptions which make high-level system goals hold. We leverage **Techne** ([9]), its approach for expressing stakeholder priorities in the form of preferences and options, and its mechanisms for dynamic incremental reasoning to support adaptive requirements.

Our framework is shown in Fig. 3. We separate the elicitation phase into design-time RE (done by the system analyst in conjunction with stakeholders) shown in the upper part of the Fig. 3. Subsequently, variants of all optimal solutions expressed using **Techne** language are stored in the **Techne** solution base i.e.  $P = P_1, \dots, P_n$  as shown on right side of the Fig 3. At this stage the meta level goals for the system (e.g. Travel Companion) are also elicited and elaborated, which later on guides the system's actions to reflect the users' goal at run-time. We represent such goals in the upper part of the Fig. 2.

For a common understanding, we use shared ontology that provides concepts to support **CARE** activities as shown on the left side of the Fig 3. The run-time RE (done by the system in conjunction with a user) is shown in the lower part of Fig. 3. Each phase consists of creating or updating the requirements model, and a reasoning phase using that model.

We now discuss **CARE**'s key activities i.e.: Service Request Acquisition, Service Lookup, Service Selection and Update Specification along with their respective input/output operations, i.e.  $S_{request}$ ,  $S_{availability}$ ,  $S_{confirmation}$  and  $S_{pec_{update}}$  as shown in lower part of the Fig. 3. We instantiate **Techne** specification instance as  $P'$ , which is an input to the CARE activities.

**Service Request Acquisition** ( $S_{request}$ ): is performed to acquire service requests either as a result of monitoring users' context and/or preferences or expressed as a search query in natural language (using keywords or a whole phrase) by her. Once the service request (RRA) is acquired from the user, the existing specification  $P'$  is compared with the given request to identify the operation to be performed (switch to Type 3). These operations include: Add a new goal ( $Add_{Goal}$ ); Add new alternative ( $Add_{Goal/Plan}$ ); Suspend ( $Suspend_{Goal/Plan}$ ), Resume ( $Resume_{Goal/Plan}$ ) or (Relax  $Relax_{Goal/Plan}$ ) an existing goal/means alternative that satisfies the existing goal.

Note, the information given by the user or monitored by the system is an input to the reasoning process which ensures correct identification of operations to be performed with respect to the existing specification  $P'$ . These operations and reasoning rules are beyond the scope of this paper. Requirements monitoring is essential to determine requirements satisfaction in changing environment conditions leading the system to re-configure to itself [18]. In our case, the system can itself generate a service request through monitoring, i.e. (switch to Type 2) when any change in context conditions or violation of a goal achievement condition is detected. The system may opt to take corrective actions or propose a set of actions to the user in case of error interpreting her input (switch to Type 3). A run-time requirement artifact (RRA) is generated to lookup relevant services that operationalize the given request.

**Example:** *In the scenario, the main goal of the user is FindAccommodation. Due to a flight delay in reaching Amsterdam, the hotel booking has been canceled. The system proposes some alternative accommodations which are not satisfactory. The user then generates a RRA Stay at Friend's Place by providing her friend's address. This option was not perceived during design-time and is treated as new requirement, which operationalizes the original goal.*

**Techne** looks for more than just a single optimal solution to a requirements problem. It instead presents a set of solutions, which helps manage uncertainty. Uncertainty is managed by suggesting possible alternatives for the system, trying to anticipate possible changes; at run-time, the uncertainty is resolved by monitoring and switching plans. In a way we described a case where the user changed her accommodation goals to stay with a friend. This new requirement is added to the model as a RRA at run-time, and the corresponding implication for the specification is either explicitly given ("I will call her") or obtained from a service registry ("Amsterdam White Pages"). The solution with the task Reserve Online is removed and returned to the repository, and the new task added to the specification  $P'$ .

**Service Lookup** ( $S_{availability}$ ): This activity is initiated after analyzing the RRA (switch to Type 3) or as a result of monitoring the performance of an existing service (switch to Type 2). Lookup is performed either in the existing pool of services of the system or using a web service search, such as Woogle or Seekda<sup>5</sup>. The lookup operation tries to match/compare user's keywords and service descriptions to locate available services by showing a list of possible services to the user or choose the most relevant competing service taking into account the users' preferences.

**Example:** *Arriving at Amsterdam airport, the system proposes alternative hotels by using services e.g. Booking.com.*

**Service Selection** ( $S_{confirmation}$ ): The system actively

<sup>5</sup><http://db.cs.washington.edu/>; <http://webservices.seekda.com/>

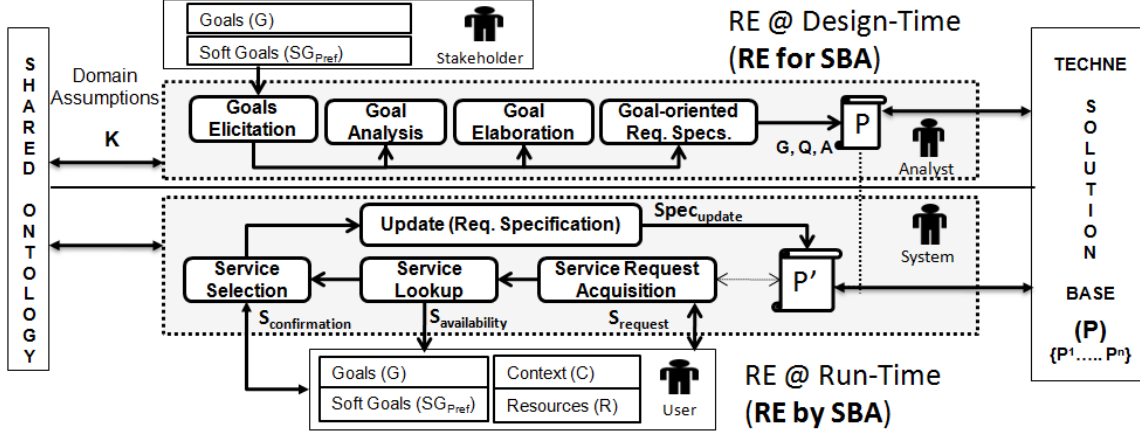


Figure 3. Continuous Adaptive Requirements Engineering (CARE) Framework. Adapted from [11].

involves the user where necessary (switch to Type 3). The RRA is populated with a resulting list of services, which is shown to the user for selection and confirmation. In case the system is proposing a service based on user’s preferences e.g. *LowCost* or based on the competing advantage of an existing service performance e.g. *ResponseTime* (switch to Type 2), the best possible service is shown to the user for the confirmation. At this step, the user confirmation is used to refine the current request (RRA) i.e. adding details about the selected services that operationalizes the users’ request.

**Example:** *The system attempts to update the itinerary to accommodate flight cancellations. It proposes to the user a bus trip from Austria to Norway. The user reject this and selects a train via Prague instead.*

**Update Specification ( $Spec_{update}$ ):** Once the RRA has been completely filled using the users’ input (switch to Type 3) or monitored information (switch to Type 2), this activity is performed once the RRA is complete. An update to the initial specification  $P'$  ensures either refinement of existing or addition of new requirements. The operations mentioned earlier e.g. *AddGoal*, *AddGoal/Plan*, *SuspendGoal/Plan*, *ResumeGoal/Plan*, *RelaxGoal/Plan*; are triggered to perform an update in the **Techne** model. We represent it with Grey nodes in the model as shown in the lower part of the Fig. 2 i.e. Run-time representation of the users’ goals and preferences.

**Example** *The alternative sub goal is added Stay at friend’s Place, which is operationalized by looking up services e.g. Google maps, and transportation service, motivated by the user’s preference e.g., taxi is more convenient than bus and metro. This changes the cost preference. In this case, an alternative to the goal FindAccommodation is added, a change in preference e.g. Taxi > Bus > Metro is accommodated considering Convenience > Cost. This then implies new services might be selected.*

To summarize, **CARE** performs **Continuous RE** by reappraising requirements model and reconfiguring their op-

erationalization along the adaptation types (see section III) involving the user, where needed.

## VII. DISCUSSION

In [8], the authors propose five research challenges for requirements at run-time:

- 1) Run-time representations of requirements
- 2) Evolution of the requirements model and its synchronization with the architecture
- 3) Dealing with uncertainty
- 4) Multi-objective decision-making
- 5) Self-explanation

How does our proposed framework address these challenges? We maintain run-time representations of requirements using the run-time requirement artifact (RRA), a mechanism for users to provide new or changed requirements to the system. The system maintains the requirements model (expressed in **Techne**) in a repository of alternative solutions forms part of the system’s archival base for adaptation. In **CARE**, adaptation is accomplished by (in Types 1-3) reappraisal and reconfiguring the requirements and operationalizations respectively; and evolution of the model is accomplished by (in Type 4), the model is taken offline and re-evaluated incrementally. The monitoring data acts as input to the next search for a solution.

The **Techne** evaluation process is paraconsistent, allowing multiple conflicting requirements in the original model. When a solution is created, uncertain or conflicting requirements are omitted from  $P$ . This search takes two forms: a naive approach, which finds globally optimal sets of tasks satisfying mandatory goals; and a local search approach, which finds local maxima. Multiple objectives are evaluated using the preference and optionality relationships (we don’t use numeric objective functions).

Finally, our proposed framework for continuous RE: **CARE** together with **Techne** modeling and reasoning can help addressing the challenges for RE at run-time. Although,

our proposal does not say much about ensuring transparency to the user for the adaptations made. However, we explicitly call for making proposals to users in the case where the new implementation is not obvious, and we provide a mechanism for user-driven adaptation.

#### A. Future Research Work

The core ontology for RE conceptualizes a requirements *problem*, in which a solution consists of some set of tasks and domain assumptions which satisfy goals and softgoals. Our framework has not yet considered the full potential of the new core ontology concepts. One area we think is particularly useful to investigate is the role of domain assumptions in inconstant systems. We think it is useful to consider how these assumptions might change.

A related avenue of research is into handling of incremental model updates. Our current research is focused on algorithms for incremental (and ideally rapid) re-analysis of a requirements model. Subsequently, this aims at improving the performance of our **CARE** framework for continuous RE.

These systems can also be characterized as moving from closed-world assumptions (classical RE) into open-world assumptions. With this shift comes consequences for system performance. While adaptivity increases, so too does the reasoning challenge. No longer can the system assume that absence of evidence for a particular concept implies that concept is false. New RE reasoning techniques are necessary to manage the resulting scaling and consistency challenges that poses. Good lessons here can be learned from AI, which has long struggled with these issues.

A comprehensive suite of extended examples and scenarios would be a great asset to the research domain. In addition to forcing researchers to agree on what constitute acceptable scope for requirements-aware systems, such a shared repository would make cross-framework comparisons much simpler. A comparable repository was started for SEAMS workshops<sup>6</sup>.

#### REFERENCES

- [1] B. H. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, *Software Engineering for Self-Adaptive Systems: A Research Roadmap*. Springer, 2009, vol. LNCS 5525, pp. 1–26.
- [2] I. J. Jureta, S. Faulkner, and P. Thiran, “Dynamic requirements specification for adaptable and open service-oriented systems,” in *International conference on Service-Oriented Computing*, Berlin, Heidelberg, 2007, pp. 270–282.
- [3] M. S. Feather, S. Fickas, A. V. Lamsweerde, and C. Ponsard, “Reconciling system requirements and runtime behavior,” in *International workshop on Software specification and design*, Washington, DC, USA, 1998, p. 50.
- [4] W. Robinson, “A Roadmap for Comprehensive Requirements Monitoring,” *Computer*, vol. 43, no. 5, pp. 64–72, 2009.
- [5] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, and J.-M. Bruel, “RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems,” in *International Conference on Requirements Engineering*, Atlanta, Aug. 2009, pp. 79–88.
- [6] L. Baresi and L. Pasquale, “Live goals for adaptive service compositions,” in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, New York, NY, USA, 2010, pp. 114–123.
- [7] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, “Requirements reflection: requirements as runtime entities,” in *International Conference on Software Engineering*, 2010, pp. 199–202.
- [8] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, “Requirements-Aware Systems,” in *International Conference on Requirements Engineering*, Sydney, 2010.
- [9] I. J. Jureta, A. Borgida, N. A. Ernst, and J. Mylopoulos, “Techne: Towards a New Generation of Requirements Modeling Languages with Goals, Preferences, and Inconsistency Handling,” in *International Conference on Requirements Engineering*, Sydney, Australia, Sep. 2010.
- [10] N. A. Ernst, J. Mylopoulos, A. Borgida, and I. J. Jureta, “Reasoning with Optional and Preferred Requirements,” in *International Conference on Conceptual Modeling (ER)*, Vancouver, Nov. 2010.
- [11] N. A. Qureshi and A. Perini, “Requirements Engineering for Adaptive Service Based Applications,” in *International Conference on Requirements Engineering*, Sydney, Australia, 2010.
- [12] I. J. Jureta, J. Mylopoulos, and S. Faulkner, “Revisiting the Core Ontology and Problem in Requirements Engineering,” in *International Conference on Requirements Engineering*, Barcelona, Sep. 2008, pp. 71–80.
- [13] B. H. Cheng, D. M. Berry, and J. Zhang, “The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems,” in *International Conference on Requirements Engineering: Foundation for Software Quality*, Porto, Portugal, Jun. 2005.
- [14] W. Tsai, Z. Jin, P. Wang, and B. Wu, “Requirement engineering in service-oriented system engineering,” in *IEEE International Conference on e-Business Engineering (ICEBE) 2007.*, Oct. 2007, pp. 661–668.
- [15] N. A. Qureshi and A. Perini, “Engineering adaptive requirements,” in *Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Vancouver, BC, May 2009, pp. 126–131.
- [16] A. Dardenne, A. van Lamsweerde, and S. Fickas, “Goal-directed requirements acquisition,” *Sci. Comput. Program.*, vol. 20, no. 1-2, pp. 3–50, 1993.
- [17] S. Liaskos, S. A. Mcilraith, and J. Mylopoulos, “Goal-based Preference Specification for Requirements Engineering,” in *International Conference on Requirements Engineering*, Sydney, Sep. 2010.
- [18] S. Fickas and M. S. Feather, “Requirements monitoring in dynamic environments,” in *RE '95: Proceedings of the Second IEEE International Symposium on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 1995, p. 140.

<sup>6</sup><http://seams.self-adapt.org/wiki/Exemplars>