Mitigating the obsolescence of quality-specification models in service-based systems

Romina Torres
Universidad Tecnica Federico Santa Maria
Chile
INRIA Paris-Rocquencourt, France
romina@inf.utfsm.cl

Nelly Bencomo
INRIA Paris-Rocquencourt
France
nelly@acm.org

Hernan Astudillo
Universidad Tecnica Federico Santa Maria
Chile
hernan@acm.org

***Abstract*—Requirements-aware systems have addressed the need of reasoning about uncertainty at runtime to support adaptation decisions. Unfortunately, the RE research community has not addressed yet the uncertainty about the QoS of services generated by the market. Currently, requirements of SBS are transformed into specification models using the domain knowledge about the market known at design time. During runtime, the market and therefore the domain knowledge, can change resulting in the obsolescence of the specification models. Obsolete specification models may make the system miss opportunities for self-adaptation to improve its performance. In this paper, we argue that QoS requirements should be specified in a way that avoids its future obsolescence. We propose an approach to address the uncertainty associated to QoS due to the unforeseen behavior of the market during execution. We propose the use of abstract specification models of QoS. During runtime, these abstract specification models are transformed into concrete specification models to determine if the requirements are still satisfied by the current service configuration and consequently executing an adaptation if needed. We have applied our approach in different case studies. Our results showed so far that in 100% of the cases, SBS using our approach are able to detect unsatisfied requirements during runtime and, therefore triggering suitable adaptations.**

***Keywords*-Requirements-awareness, Quality of Service, service-based systems, dynamically adaptive systems, requirements model, model@runtime.**

I. Introduction

The runtime representations of requirements [1] presented by requirements-aware systems [2] act as a base line to drive and reason about dynamically adaptive systems (DAS). Those systems are capable of dealing with different kinds of uncertainty [3], reasoning over their requirements at runtime, monitoring their satisfaction and triggering corrective adaptations when deviations are detected between the system's runtime behaviour and the requirements model.

During the specification of a system, the requirements R are transformed into specification S supported by domain knowledge K [4]. According to Zave and Jackson [4], the specifications S and the relevant domain knowledge K must be sufficient to guarantee that the requirements R are satisfied:

$$S, K \vdash R \tag{1}$$

During execution, it is possible to determine if the requirements are satisfied or not, by monitoring the deviations between the system's behavior and the specification models. The latter is valid just if K has not changed considerably during execution since the specification S were defined. For the specific case of service-based systems (SBS), this assumption cannot always be guaranteed given the unprecedented degree of change in the service market [5]. Even, if the required functionalities of a SBS would not change, the quality specifications which constrain functionalities are likely to change through the time because they are highly dependent on the characteristics of the market represented by K. In this kind of systems, the quantifiable quality specifications S are obtained by observation of what the service market K is offering.

Unfortunately, during execution the ever changing market may provoke the obsolesce of S making impossible for systems to determine if their requirements are being satisfied or not by the current configuration of services. The latter is a problem in the case that the satisfaction of the specifications S are used by systems as a base to drive their adaptations. The system can miss opportunities of adaptations because it is not aware when requirements are becoming unsatisfied.

In this work, we propose an approach to support systems to address the uncertainty of the QoS of the service market, by mitigating the obsolescence of the specifications at runtime in order to avoid the degradation of the adaptation capability of the specification model.

The rest of the article is organized as follows: Section II introduces a motivational example and the background needed to understand following sections; Section III presents our approach; Section IV explains the architecture which supports this approach; Section V explains the current dataset, describes experiments and discusses their results; Section VI contrasts related proposals; and Section VII concludes the paper and draws future works.

II. Motivational Scenario

Consider the following motivational scenario: *our client requires to build a service-based application to send by email, as fast as possible, the city and state for the location where the user is*. To build such application, the architect needs 1) to transform, at design time, the requirements R into an specification model S using the current offering of the service market K and then, 2) at binding time, to select from the service market a proper architecture configuration C (service composition), which satisfies the model, as Figure 1 shows. The process to build the specification model S is

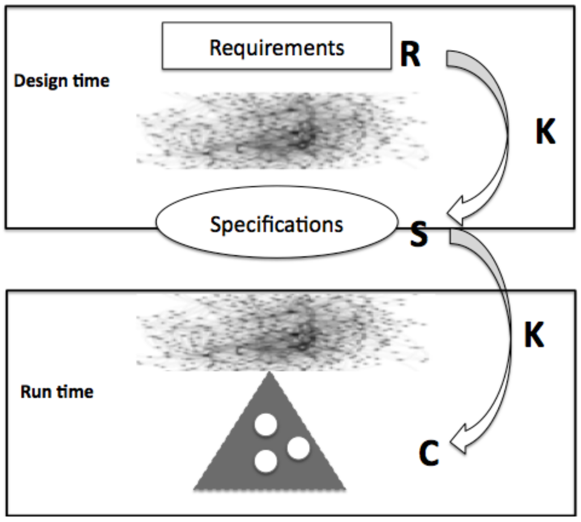


Figure 1. Building the model at design time to drive adaptations at runtime

explained as follows:

1) **From requirements to software requirements**. From the statement, architect identifies three software requirements SRs: SR_1, a service capable of determining the location of an IP address, SR_2, a service capable of returning the state and the city given either a zip code or a location and, SR_3 a service capable of sending emails given an email address and a text. At runtime time, each software requirement will be implemented by one service or a composition of services.
2) **Prioritizing software requirements**. Suppose that, in this example SRs are prioritized as equally important.

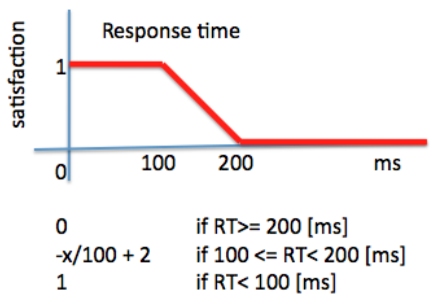


Figure 2. Satisfaction function of the *response time* quality specification by observing the current offering K

3) **From quality requirements to quality specifications**. From the statement, the architect identifies the quality requirements and then how they constrain each SR. In this particular example, from the statement the architect identifies the quality requirement: {*"as fast as possible"*}. Architect transforms the quality constraint of SR_3 by observing the current and relevant offering of the service market, which is in this case, the measurements of the *response time* of the services which are capable to perform the same functionality i required for SR_3. Lets suppose the architect decides for services providing functionality i
   - "fast" are those services with $response\_time \leq$ 100 milliseconds,
   - "no fast" are those services with $response\_time \geq 200$ milliseconds and,
   - for those services whose have $100 \leq response\_time \leq 200$ ms, as Figure 2 shows, there is a function which measures the proportional "fast" degree of them.

   Analogously, the architect defines for SR_1 a range between $[10, 50]$ and for $SR2$ a range between $[40, 100]$. As we can see, the numerical range which represents "fast" is different depending of which part of service market (which kind of functionality) is relevant.
4) **Prioritizing quality specifications**. For each software requirement, quality specifications are prioritized.

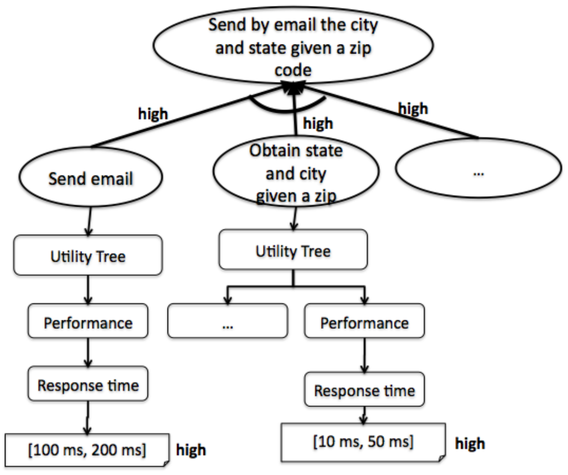


Figure 3. Partial view of the specification model at design time

Figure 3 partially shows the specification model S built at design time which is used to obtain an initial configuration C as well as to trigger the needed adaptations at runtime, when C ceases to satisfy the model S.

Suppose that, for this particular scenario, the architect selects at runtime an initial service composition $C = \{s_u, s_v, s_w\}$ which maximizes the satisfaction of the particular model S at time t. Suppose now that, at time $t+x$ there is enough evidence in K that the service s_w has dropped several times its QoS by increasing, in average, its *response time* from 80 to 230 milliseconds. Then, the system must trigger its adaptation (see Figure 4) in order to replace the infringer service or in some cases the complete composition. Suppose in this case that, the configuration C is replaced by $C' = \{s_u, s_v, s_o\}$, where the *response time* of service s_o is 91 milliseconds.

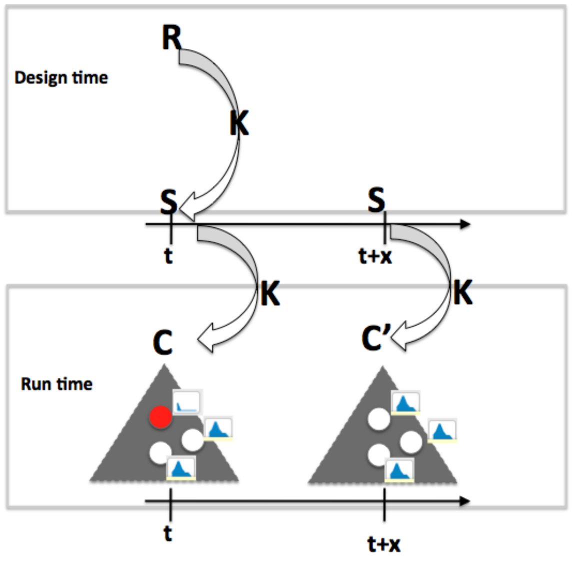


Figure 4. Driving adaptations using the specification model

Suppose now a second scenario. The *response time* measurements of the functionally-equivalent services offering functionality i required to implement SR_3, in general, have decreased. More than 75% of the services have now a $response\_time \leq 107$ milliseconds. The relevant market K to this requirement, has drastically changed, what may change the perception of the architect of what "fast" means in this kind of services (e.g. architect could change its specification from $[100, 200]$ to $[20, 50]$ milliseconds). Therefore, if the specification model S is not updated in this case and each time the assumptions under S was built become falsified (K has drastically changed), then the model S itself will become obsolete and it will be not able to support SBS to drive their adaptation. Unfortunately, the K is continuously and drastically changing because service providers are competing by offering services with similar functionality but different quality and cost attributes [5] [6].

This new kind of dynamism of the service market [5] makes 1) unfeasible for humans manually maintaining their models aware of the market and 2) unfeasible for SBS driving automatically their adaptations under these conditions.

III. Proposal: Mitigating the obsolescence of the specification model

In this Section, we present our approach to mitigate the obsolescence of the specification model S at runtime, which drives the adaptation of SBS under a ever changing market.

We propose to relieve architects from the arduous task of transforming requirements R into measurable specifications S, as well as maintaining synchronized R with S when the service market K is evolving. Indeed, we encourage architects to transform R into an abstract specification model S^* by using "linguistic" variables [7] instead of numerical ones, because the latter are more prone to obsolescence.

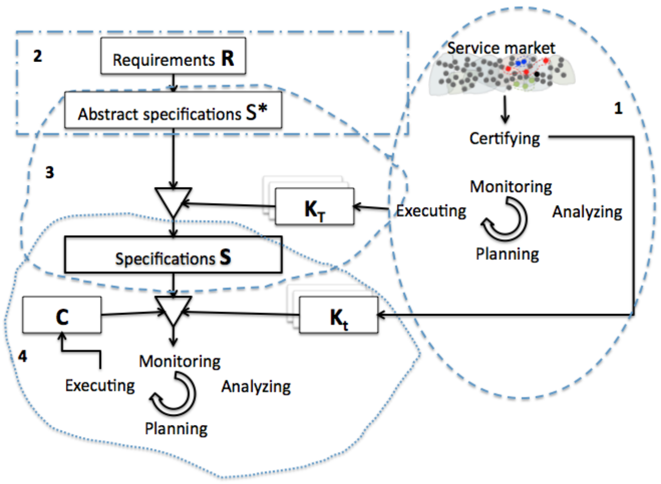


Figure 5. Overview of the obsolescence mitigating process. Area 1 shows how the knowledge domain is obtained; area 2 shows how the requirements are transformed into abstract specifications; area 3 shows how the concrete specification model is generated; and finally the area 4 shows how the SBS drives the adaptation at runtime.

Figure 5 schematically shows the overview of our approach which consists of several subprocesses. Whenever the market has significantly changed, the first subprocess (area 1) is in charge of generating a new view of the knowledge domain K_T. Moreover, this process also provides online the measurements of the services K_t. The second subprocess (area 2) allows to each client to define an abstract specification model S^* from the requirements R. Given this abstract specification S^*, our approach is capable to automatically generate a concrete specification model S by using the current knowledge domain K_T (subprocess marked as area 3) and, secondly, to drive the adaptation whenever there is enough evidence that the current configuration C is non longer satisfying the specification model S (subprocess marked as area 4). In the following sections each subprocess will be explained in details.

A. Subprocess 1: Obtaining the relevant knowledge domain at runtime

Let $CS_i = \{s_i^1, s_i^2, ..., s_i^{n_i}\}$ be a functionally-equivalent service set, which is comprised by $n_i \geq 1$ concrete services that provide the same functionality i than an abstract service sa_i, with $1 \leq i \leq I$ [8]. Let $Q = \{q_1, \ldots, q_M\}$ be the set of quality attributes which allow to distinguish functionally-equivalent services. Let K be the service market composed of all the functionally-equivalent service sets.

We assume that for each functionally-equivalent services set CS_i, it is possible to periodically obtain the measurement for each quality attribute of each service member. Moreover the services can be ordered according to each of these quality attributes as well as they can be categorized in five overlapping groups which allow to classify them comparatively. These groups are the linguistic variables LVs, which in this work we assume they are $LV_j^{[i]} = \{$"poor", "fair", "good", "very good", and "excellent"$\}$. Each linguistic variable is a fuzzy set denoted by μ with a triangular shape whose support a_1, a_2 and its peak a_M are calculated using K_T. μ is defined as follows:

$$\mu(x) = \begin{cases} \frac{x-a_1}{a_M-a1} & \text{if } a_1 \leq x \leq a_M \\ \frac{x-a_2}{a_M-a_2} & \text{if } a_M \leq x \leq a_2 \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where x is the measurement given by K_t.

Let K_τ be the measurements of the quality attributes of all services of the market at the snapshot obtained at time τ (τ could be considered as the time t or time T whichever is applicable).

K_T allows clients to specify its quality specifications by using the LVs, while K_t allows the systems to monitor if the current architecture configuration C is satisfying the concrete specifications S. Notice that the frequency at which the adaptation of K_T is generated is significantly lower than the frequency of K_t (see figure 6).

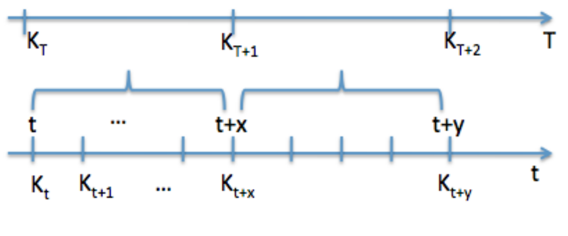


Figure 6.

B. Subprocess 2: Defining the abstract specification model at design time

In this subprocess the architect specifies at design time the abstract specification model S^* by transforming quality requirements into abstract quality specifications by using the linguistic variables LV.

The abstract specification model S^* is constructed from a set of fuzzy conditional statements. These statements are expressions of the form *IF A and B and ... THEN Z* where A, B and Z have fuzzy meaning.

For instance, the concrete specification model which was generated in Figure 1 can be specified as an abstract model by using Ł instead of precise numerical values. The left branch of the model can be specified as an abstract model as follows: *IF the response time of a service capable of sending email is at least "fast" THEN the belonging degree to the acceptable solution set is high*, where "fast" for this kind of service is a linguistic variable whose numerical range is defined in K_T (by simplicity we omit prioritization in the statement). The aim of the specification model is to allows systems to determine if the current architecture configuration C is satisfying the requirements and to support the assessment of replacement configurations in case of an adaptation is needed. Therefore, we represent the abstract specification model S^* as a fuzzy multi-criteria decision making function

$$S^* : \sum_{i=1}^{I} v_i \left(\sum_{j=1}^{J} w_j^{[i]} \delta_{MAC_j^{[i]}}(c_j(C)) \right) \mathbb{I}(C, SR_i) \tag{3}$$

where $V = \{v_1, ..., v_I\}$ and $W^{[i]} = \{w_1^{[i]}, ..., w_J^{[i]}\}$ be the sets of relative importance of each software requirement SR_i, as well as for each software requirement, the relative importance of each quality constraint; $\mathbb{I}(C, SR_i)$ which is an indicator function that returns 1 if there is a service $s \in C$ providing functionality i requested by SR_i or 0 if not; $c_j(C)$ is a function which returns the current measurement value of the service $s \in C$ from K_t if applies; $\delta_{MAC_j^{[i]}}$ is a fuzzy function which returns the membership degree of the current measurement of the $s \in C$ to the minimal acceptable class (MAC) which is a linguistic variable defined in K_T.

C. Subprocess 3: Generating the concrete specification model at runtime

The subprocess marked as 3 in Figure 5 shows a transformation from the abstract specification model S^* into a concrete specification model S. This transformation is executed each time a new K_T is available. The main difference between S^* and S is that we incorporated the information of the relevant knowledge domain K_T in order to obtain the numerical values of the parameters of the model equation 3. This equation has several function $\delta_{MAC_j^{[i]}}$ (one for each quality attribute constrain of each functionally-equivalent set) whose parameters must be obtained from the relevant knowledge domain K_T. Because our model allows to specify the minimal acceptable class, services belonging to better quality levels must also be considered with membership degree of 1. We define each $\delta_{MAC_j^{[i]}}$ as the fuzzy union of the minimal acceptable class (which is one of the five linguistic variables) with those classes which are better, as follows

$$\delta_{MAC_j^{[i]}} = MIN(\mu_{MAC}, \mu_{C_1}, \ldots, \mu_{C_L}) \tag{4}$$

where $\mu_{C_1}, \ldots, \mu_{C_L}$ are those linguistic variables whose linguistic meanings are better than μ_{MAC}. Because, we are assuming triangular fuzzy sets, the function μ_{MAC} is defined as a ramp function as follows (assuming $a_1 \leq a_2$)

$$\mu_{MAC} = \begin{cases} 0 & \text{if } c_j(C) > a_2 \\ \frac{c_j(C) - a_1}{a_2 - a_1} & \text{if } a_1 \leq c_j(C) \leq a_2 \\ 1 & \text{if } c_j(C) < a_1 \end{cases} \tag{5}$$

Each time the subprocess explained in subsection III-A generates a new K_T, a new concrete specification model S will be generated.

D. *Subprocess 4: Driving adaptations at runtime*

The subprocess marked as 4 in Figure 5 shows how the specification model S and the current measurements K_t are used to determine if the current configuration C is satisfying or not the specification model S. If the specification model S using K_t is not satisfied by the current configuration C, then the monitoring component in the area 4 of the Figure 5 sends the violation to the Analyzer component which determines if there is enough evidence to trigger an adaptation or not. The planner component obtain a new configuration C_t by using S and K_t. Then, this new configuration is applied. How the configuration is applied or which adaptation strategy to use (instead of replacement) are out of the scope of this paper.

IV. Approach implementation

In order to maintain the specification models aware of the market, the process 1 explained in the section III-A must be periodically executed. Figure 7 shows the architecture to produce new market views from the current observations. The *functional crawler* component collects from different Web-based catalogs the Web service descriptors. The *QoS certifier* component runs a benchmark tool over the endpoint list obtained by the *functional crawler*, in order to gather the QoS measurements. The *functional clustering* component clusters the Web services (based on their WSDL descriptor files) according to their functionality (if there is not valid categories information available). And the *QoS-fuzzy clustering* which clusters each quality aspect of each functionally-equivalent service set into c classes using a modified fuzzy c-means algorithm (deeper details in [9]). In this work we set up $c = 5$.

Notice in the Figure 5 we have two feedback loops. The first one is located in the market side, which is constantly *monitoring* the changes in the market, *analyzing* if there is enough evidence to generate a new market view, and in the

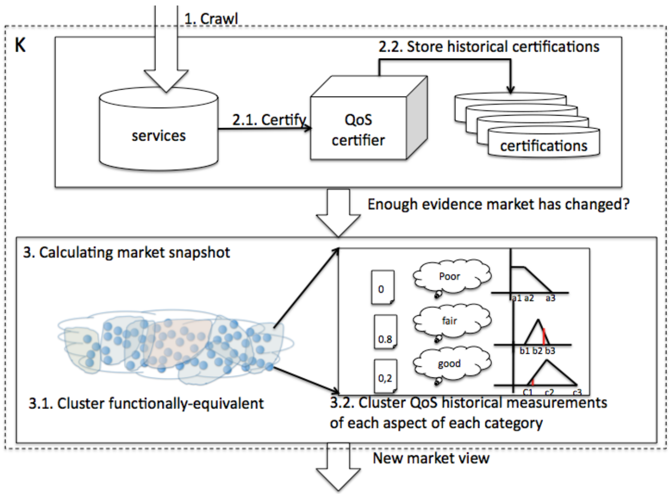

1. Crawl
K
2.2. Store historical certifications
services
2.1. Certify
QoS certifier
certifications
Enough evidence market has changed?
3. Calculating market snapshot
0
0.8
0,2
Poor
fair
good
a1 a2 a3
b1 b2 b3
c1 c2 c3
3.1. Cluster functionally-equivalent
3.2. Cluster QoS historical measurements of each aspect of each category
New market view

Figure 7. Architecture to produce new market snapshots

positive case, in the *planning* stage, generating a new market snapshot which basically allows to update all the parameters of the linguistic variables (fuzzy sets) of each functional-equivalent service set which are informed to the SBS client systems by the *executing stage*.

V. Experiments

In order to show how the approach works, we have developed a basic prototype to study how concrete specification models become obsolete when the market is changing and how new architecture configuration can be driven if the obsolescence is mitigated. The prototype allows (1) to specify a set of software requirements; (2) to prioritize them; (3) for each software requirement to specify its quality constrains by using linguistic variables; (4) to prioritize them as well; (5) to find a valid architecture configuration which maximizes the satisfaction of the model at time t; and finally (6) to show how new configurations are recommended at $t + \Delta t$ and at $t + x\Delta t$ when the configuration recommended at t, C_t does not satisfy anymore the specifications R.

In the following subsections we explain the dataset, what experiments we ran and what conclusions we draw from them.

A. *Dataset*

The dataset consists of a subset of 1500 Web services of the QWS Dataset [1] (all of them valid as of October 2011), which originally included 2507 actual Web service descriptors with nine QoS measurements. The quality aspects are *response time*, *availability*, *throughput*, *successability*, *reliability*, *compliance*, *best practices*, *latency* and *documentation*.

To emulate the market changes, we have created two new market snapshots, where QoS' service specifications

[1]http://www.uoguelph.ca/∼qmahmoud/qws

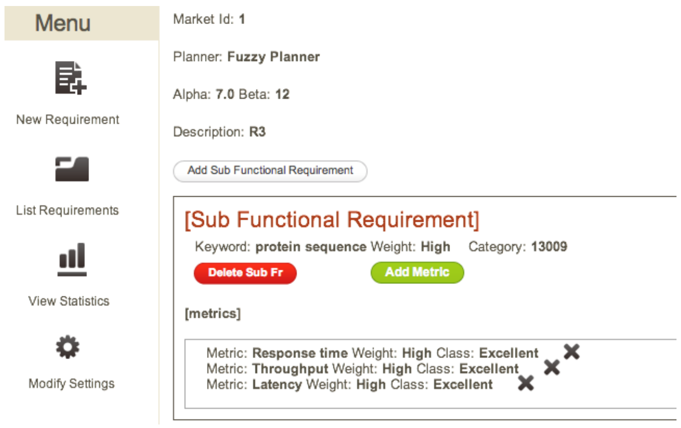
are improved in the first snapshot in a random percentage between 0% and 30%, and in the second snapshot a random percentage between 30% and 50%. All of these modifications are applied to all services. We are not using a black list with the services currently selected by the configurations C of the SBS.

Our prototype follows the approach presented in our previous work [10] to externalize adaptation capabilities by a third application that provides the service of monitoring subscribed contracts (requirements and current architecture configuration in use) and monitoring the changes in the market in order to act as a recommender system, whose objective is to notify subscribed SBS when an adaptation should be executed because its requirements have been not satisfied recurrently, which could mean, probably, its architecture is degrading and it is better to adapt it in order to avoid it becomes obsolete.

B. Case study

We have prepared a set of ten case studies, each one may be composed of multiple software requirements, which themselves may be constrained by multiple quality requirements. Because the objective of these experiments is to study the robustness of the model against the market changes we are not studying prioritization. The reader can assume if a request is divided in several software requirements, these are prioritized as equally important (“high), and if a software requirement is constrained by several quality requirements, these are equally important (“high”) as well. For lack of space, we only show four of the ten cases:

- R1: one service capable of *given a zip code return the country* with at least *response time* “excellent”, at least *availability* “excellent”, and at least *throughput* “excellent”; and a second service capable of *given the latitude and longitude return a map* with at least *throughput* “excellent”, at least *reliability* “excellent”, at least *best practices* “excellent” and at least *latency* “excellent”.
- R3: one service capable of *return the sequence of a protein* with at least *response time* “excellent”, at least *throughput* “excellent”, and at least *latency* “excellent”.
- R6: one service capable of *given a phone number return its information* with at least *response time* “excellent”, at least *throughput* “excellent”, and at least *best practices* “excellent”; and a second service capable of *send a tex by fax* with at least *response time* “excellent”, and at least *availability* “excellent”.
- R8: one service capable of *given a zip code returns the country* with at least *response time* “excellent”, and at least *reliability* “excellent”; and a second service capable of *given a country returns its currency* with at least *response time* “excellent”, at least *availability* “excellent”, and at least *best practices* “excellent”.



Market Id: **1**

Planner: **Fuzzy Planner**

Alpha: **7.0** Beta: **12**

Description: **R3**

[Sub Functional Requirement]

Keyword: **protein sequence** Weight: **High** Category: **13009**

[metrics]

Metric: **Response time** Weight: **High** Class: **Excellent**
Metric: **Throughput** Weight: **High** Class: **Excellent**
Metric: **Latency** Weight: **High** Class: **Excellent**

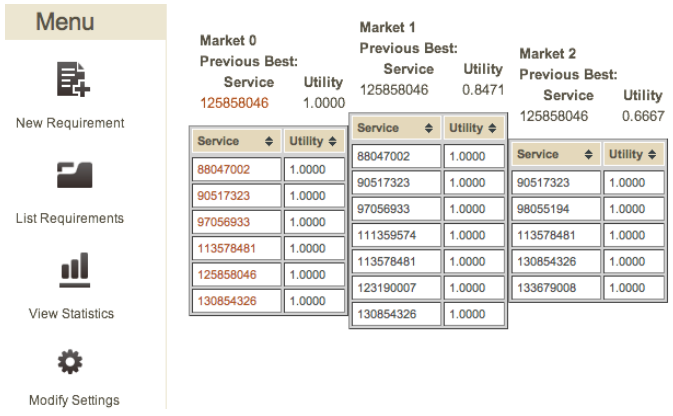
Figure 8. Using the prototype to asses a case study

C. Experiments and discussion

The objective of this experiment is to empirically show that by using our approach of mitigating the obsolescence of the specifications (or in other words, make them market-aware models) specification models maintained at runtime are a valid and effective base to enable systems to reason about them whether their requirements are being satisfied by the current architecture.

Figure 8 shows the user interface of our prototype. Software and quality requirements, prioritization, and minimal acceptable classes are specified. Our prototype computes an architecture at design time (using the first snapshot) choosing one of those whose membership degree to the “acceptable solution” fuzzy set (defuzzifying equation 3) is closest to 1. It is important to notice, that we choose a fuzzy approach instead of a deterministic one in order to avoid always recommend the best, because if we do that, we would increasing the potential demand of some services will face next time the market is evaluated.

Figure 9 shows the results for the request R3, where the service with id 125858046 was selected to implement the requirement R whose membership degree to the “acceptable solution” set was 1. As we can see in the table below the service selected, it is not the only one, which has a membership degree equal to 1. Figure 9 also shows results at runtime: Market 1 ($K1$) and Market 2 (K_2). In K_1, we can see the service with id 125858046 drops its membership degree to the “acceptable solution”, from 1 to 0.8471 and in K_2, it still drops even more its membership degree until 0.6667. We have to remember the K_1 and K_2 snapshots are synthetical data, whose quality was randomly improved from previous market view. Then, it is possible some services do not experiment changes in their quality, or even when they did, the quality fuzzy set drifted in such a way, services still are considered in this case, of “excellent” quality (for instance, the service with id 88047002 with membership degree of 1 to the “acceptable solution” at design time



Market 0
Previous Best:

Service	Utility
125858046	1.0000

Service	Utility
88047002	1.0000
90517323	1.0000
97056933	1.0000
113578481	1.0000
125858046	1.0000
130854326	1.0000

Market 1
Previous Best:

Service	Utility
125858046	0.8471

Service	Utility
88047002	1.0000
90517323	1.0000
97056933	1.0000
111359574	1.0000
113578481	1.0000
123190007	1.0000
130854326	1.0000

Market 2
Previous Best:

Service	Utility
125858046	0.6667

Service	Utility
90517323	1.0000
98055194	1.0000
113578481	1.0000
130854326	1.0000
133679008	1.0000

Figure 9. Case study results

maintains the same degree at runtime K_1).

In Table I we show the results for the four requests that we specified before. The first column shows the id of requests, the second column shows the hypothetical selected architecture configuration and between parenthesis the membership degree to the "acceptable solution" set. The third and four column shows the results obtained for the first and second runtime snapshots respectively. For each request we are divided the results in three rows. the first row of each request, indicates in the second column the solution selected C at design time with its membership degree to the "acceptable solution" set, in the third column asses C again but over K_1 and in the four column asses it again but over K_2. The second row of each request shows in the third column, the recommended adaptation for the system under this new K_1, and in the four column the assessment of this recommendation but over the future K_2. The third row of each request shows in the four column, the recommended adaptation for the system under this new K_2 It is important to notice, we are omitting deeper details as how are they are connected or which one is the expected QoS of the system by using these services with their particular QoS, because this is part of our current work which is discussed in Section VII. We can conclude based on this experiment, if obsolescence of specifications is not mitigated, systems using a model@runtime to drive its architecture adaptation could miss adaptation because the obsolescence of specifications is hiding requirements are becoming unsatisfied. In the 100% of the cases, recommendations at runtime are encouraged to replace the older ones because the current configurations have at runtime a membership degree to the "acceptable solution" lower than the threshold. Threshold should be defined by each SBS owner, but now this is out of the scope of this paper but is part of our future work.

The main contribution of mitigate the obsolescence of specifications against of a market which is constantly changing, is SBS are not missing adaptation opportunities.

Table I
TEST CASES - EACH COLUMN SHOWS THE SERVICES IDS WHICH SATISFY THE REQUEST AND BETWEEN PARENTHESIS IS THE BELONGING DEGREE OF THE PROPOSED SOLUTION TO THE ACCEPTABLE SOLUTION SPACE IN THE CURRENT AND NEXT SNAPSHOTS.

ID	design-time	(1st)run-time	(2nd)run-time
R1	84193574, 4869688 (0.875)	(0.545)	(0.27)
		112625634, 47714966 (1)	(0.5)
			131744503, 121724741 (1)
R3	15103218(1)	(0.667)	(0.501)
		87750611(1)	(0.883)
			84135951(1)
R6	78667380, 23013627 (1)	(0.833)	(0.333)
		87157455, 23013627 (1)	(0.1667)
			79152933, 26808021(1)
R8	112791393, 179771826 (1)	(0.438)	(0.137)
		60150637 47714966(1)	(0.417)
			85415723, 21416266 (1)

VI. Related Work

Ramirez et al. [3] have proposed a taxonomy of potential sources of uncertainty at the requirements, design, and execution phases. The authors reported on existing techniques for mitigating specific types of uncertainty. We deal with the uncertainty of the QoS offering of the service market at runtime ("known unknown"). According to the proposed taxonomy, we are dealing with run-time uncertainty whose source is the incomplete information of the market behavior that we have at design time. Our domain problem could be classified into the kind of concerns tackled by different approaches like RELAX [11] and Requirements Reflection [1]. However, there is not research initiative that specifically addresses this kind of uncertainty.

RELAX is a requirements language addressing the uncertainty in the specification of requirements of self-adaptive systems, which allows analysts to specify which requirements could be relaxed at runtime when the environment changes. RELAX implements key ideas of Requirements Reflection. Similar to RELAX, we also delay decisions until runtime, and we use a language to mark which parts of the requirements are delayed. However, RELAX works at level of specifications of adaptive behavior while we make recommendations of adaptations at the level of adaptive architecture.

Welsh et al. proposed REAssuRE [12], a framework, which monitors when assumptions made at design time are falsified by the current conditions and therefore triggering adaptations. With our approach, different kind of assumptions can be monitored. Using our approach, specifications of the market are monitored to see if the are becoming obsolete and are compared against the QoS offered by the service market. In REAssuRE claims associated to soft goals are made at design time to determine which operationalization alternative is the more suitable decision. During runtime, claims are monitored to check if they are obsolete according to the current environmental conditions. When claims are falsified, REAssuRE allows systems to decide at runtime if an adaptation is needed (i.e. to change to another operationalization). Similarly in our case, when domain knowledge K changes, our approach allows systems to reason and decide at runtime if specifications should be synchronized to mitigate their obsolescence, and then determine if an adaptation is needed (i.e. to change to another proper configuration).

Baresi et al. [13] extended KAOS (Goal-Directed Requirements Acquisition) by including adaptive goals. Goal-based models support the specification of "when" the adaptation should be executed and "what" it means . Besides, authors proposed a runtime infrastructure [14] which constantly monitors the conditions to trigger adaptations. Baresi et al. [15] formalized this model as FLAGS (Fuzzy Live Adaptive Goals for Self-adaptive systems) which represents requirements as runtime entities, distinguishing between crisp and fuzzy goals. Unfortunately, they use stakeholders to define the membership function, which in our case, is not feasible. Under a closed-world assumption [5] their approach works. However, due to the dynamism degree of change of the QoS offered by the service market, these specifications should constantly be updated by stakeholders, which would be an expensive process.

Filieri et al. [16] proposed a formal approach to adaptive software by assuring continuously the satisfaction of non-functional requirements. Similar to our work, they also casted their proposal into the Zave and Jackson approach to requirements [4]. Their approach is exemplified in the context of service-oriented system, focusing specifically in the non-functional requirements, they also assume the domain knowledge regarding the qualities attributes of the services are changing, and therefore their approach is trying to maintain consistent the specifications with the requirements by estimating the knowledge periodically as a way to determine if the requirements are becoming unsatisfied or not. Both proposals allow the system to maintain non-functional properties satisfied by adapting their architecture to new conditions. However, we are specifically proposing a framework which allows analysts to specify requirements in such a way that they are continuously synchronized with the open world in which service-based systems are immersed [5].

VII. Conclusions and further work

In this work we have proposed an approach to support systems to address the uncertainty of the QoS of the service market, by mitigating the obsolescence of the specification models at runtime. The main contribution of this paper is that our approach allows the system to mitigate the degradation of the adaptation capability of the model used during runtime. Until now, the adaptation capability of these models depended on precise numerical quality specifications that become rapidly obsolete against the QoS offered by the ever-changing market.Our proposal supports the reaction capacity of the system to detect requirements dissatisfaction, the maintenance of the consistency of the requirements at runtime to drive adaptations.

As future steps in our research we are considering the following topics:

- **Sensibility adaptation index**: in order to compare the adaptation capability we define the sensibility adaptation index of a model as the percentage of the required adaptations which were recommended using the model.
- **Global quality**: until now we are assuming the global quality of the system under construction and maintenance, can be obtained by ensuring that the quality requirements of the parts are achieved. There are several proposals to obtain a global model, for instance we could the components interact between them under a workflow model [17] where the interaction patterns can be know in advance. Our next step in this area is to apply our approach in this specific service-oriented architecture to reach the global quality specifications and not only the local ones.
- **How often K should be recalculated**: How much evidence the monitor component of the market feedback loop needs, to determine the K is obsolete and needs to be recalculated?

As part of our future work, we will release a benchmark to the community in order to asses similar models proposed by different authors.

Acknowledgment

This work was partially funded by FONDEF (grant D09i1171), UTFSM DGIP 241167 and BASAL FB0821(FB.02PG.11), the EU Marie Curie Project Requirements@runtime and the EU Connect project.

References

[1] N. Bencomo, J. Whittle, P. Sawyer, A. Finkelstein, and E. Letier, "Requirements reflection: requirements as runtime entities," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 199–202. [Online]. Available: http://doi.acm.org/10.1145/1810295.1810329

[2] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, “Requirements-aware systems: A research agenda for re for self-adaptive systems,” in *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, ser. RE ’10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 95–103. [Online]. Available: http://dx.doi.org/10.1109/RE.2010.21

[3] A. Ramirez, A. C. Jensen, and C. B. H. C., “To appear seventh workshop on software engineering for adaptive and self-managing systems (seams 2012),” in *ICSE*, 2012.

[4] P. Zave and M. Jackson, “Four dark corners of requirements engineering,” *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 1, pp. 1–30, Jan. 1997. [Online]. Available: http://doi.acm.org/10.1145/237432.237434

[5] L. Baresi, E. Di Nitto, and C. Ghezzi, “Toward open-world software: Issue and challenges,” *Computer*, vol. 39, no. 10, pp. 36–43, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1109/MC.2006.362

[6] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, “A journey to highly dynamic, self-adaptive service-based applications,” *Automated Software Engineering*, vol. 15, pp. 313–341, 2008, 10.1007/s10515-008-0032-x. [Online]. Available: http://dx.doi.org/10.1007/s10515-008-0032-x

[7] L. A. Zadeh, “Outline of a new approach to the analysis of complex systems and decision processes,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-3, no. 1, pp. 28 –44, jan. 1973.

[8] R. Calinescu, L. Grunske, M. Z. Kwiatkowska, R. Mirandola, and G. Tamburrelli, “Dynamic QoS management and optimization in service-based systems,” *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 387–409, 2011.

[9] R. Torres, H. Astudillo, and R. Salas, “Self-adaptive fuzzy QoS-driven web service discovery,” in *Proceedings of the IEEE International Conference on Services Computing*, ser. SCC ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 64–71. [Online]. Available: http://dx.doi.org/10.1109/SCC.2011.87

[10] R. Torres and H. Astudillo, “Externalizing the autopoietic part of software to achieve self-adaptability,” in *Proceedings of the 2011 IEEE World Congress on Services*, ser. SERVICES ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 107–114. [Online]. Available: http://dx.doi.org/10.1109/SERVICES.2011.69

[11] J. Whittle, P. Sawyer, N. Bencomo, B. Cheng, and J.-M. Bruel, “RELAX: a language to address uncertainty in self-adaptive systems requirement,” *Requirements Engineering*, vol. 15, pp. 177–196, 2010, 10.1007/s00766-010-0101-0. [Online]. Available: http://dx.doi.org/10.1007/s00766-010-0101-0

[12] K. Welsh, P. Sawyer, and N. Bencomo, “Towards requirements aware systems: Run-time resolution of design-time assumptions.” in *ASE*, P. Alexander, C. S. Pasareanu, and J. G. Hosking, Eds. IEEE, 2011, pp. 560–563.

[13] L. Baresi and L. Pasquale, “Adaptive goals for self-adaptive service compositions,” in *IEEE International Conference on Web Services (ICWS)*, july 2010, pp. 353–360.

[14] L. Baresi, S. Guinea, and L. Pasquale, “Integrated and composable supervision of BPEL processes,” in *Proceedings of the 6th International Conference on Service-Oriented Computing*, ser. ICSOC ’08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 614–619.

[15] L. Baresi, L. Pasquale, and P. Spoletini, “Fuzzy goals for requirements-driven adaptation,” in *18th IEEE International Requirements Engineering Conference (RE)*, 27 2010-oct. 1 2010, pp. 125 –134.

[16] A. Filieri, C. Ghezzi, and G. Tamburrelli, “A formal approach to adaptive software: continuous assurance of non-functional requirements,” *Formal Aspects of Computing*, vol. 24, pp. 163–186, 2012, 10.1007/s00165-011-0207-2. [Online]. Available: http://dx.doi.org/10.1007/s00165-011-0207-2

[17] D. Ardagna and R. Mirandola, “Per-flow optimal service selection for web services based processes,” *Journal of Systems and Software*, vol. 83, no. 8, pp. 1512–1523, Aug. 2010.