2016 10th International Conference on the Quality of Information and Communications Technology

Implicit Priorities in Adaptation Requirements

João Pimentel
Universidade de Pernambuco
Universidade Federal Rural de Pernambuco
jhcp@ecomp.poli.br

Maria Lencastre
Universidade de Pernambuco,
Recife, Brazil
mlpm@ecomp.poli.br

Jaelson Castro
Universidade Federal de Pernambuco
jbc@cin.ufpe.br

***Abstract*— The increasing need for adaptive systems has led to the creation of many frameworks aiming to support their development. Nonetheless, the implementation of requirements related to adaptation, just as of any other kind of requirement, comes at a cost. In being so, it is necessary to consider requirements' priorities when creating such systems. In this work we analyze the relationship between requirements prioritization and software adaptation. In particular, the following questions are tackled: is the use of requirements-based adaptation frameworks somehow affected by requirements' priorities? Does the use of said frameworks affect requirements' priorities? Through an analysis of selected frameworks, we have found evidences that suggest positive answers for both questions. In this paper we present our findings as well as an initial proposal for interlacing prioritization activities with adaptation activities.**

Keywords—requirements engineering; software adaptation; adaptive systems; requirements prioritization.

I. INTRODUCTION

Software adaptation enables the creation of more flexible, resilient, robust, recoverable and energy-efficient systems [6]. It can be defined as the software capability of accepting environmental changes as well as changes on the system itself. Adaptive systems, which are often implemented with feedback loops [3], pose specific challenges regarding requirements engineering, since the system must be able to perform satisfactorily on a diversity of contexts that may be unforeseen at design time. In fact, in some cases the system itself may perform requirements engineering activities when requirements change at runtime [4].

Adaptation requirements, i.e., requirements related to adaptive behavior, must be subject to traditional requirements engineering activities, such as specification, conflict analysis, refinement, prioritization, verification, and so on. Analyzing the literature on software adaptation, it is possible to identify many proposals to handle the specification of adaptation requirements, with some also handling refinement. However, the remaining activities are often neglected. In this paper we focus on requirements prioritization as pertaining to the development of adaptive systems.

Considering the peculiarities of adaptive systems, such as its high complexity, its dynamic changes, and the need to implement different alternatives for achieving a goal, requirements prioritization may prove to be a key factor for the successful development of said systems. By reasoning on explicit requirements' priorities, it may be possible to better execute the trade-offs that are necessary not only at design time, but also at runtime.

In this work we aim at investigating the effects of requirements prioritization on the development of adaptive systems, as well as the impact of the latter on the former. In order to do so, we analyzed the elements, processes, and examples of requirements-based adaptation frameworks. For the sake of space, in this paper we present observations only of two well-established frameworks: Relax [7], and Zanshin [1][2]. Relax modifies requirements statements with a set of pre-defined operators, while also including additional information regarding environment, monitoring, relationship, and dependencies. Zanshin extends goal models to include monitoring needs, adaptation strategies, and their relationship. Based on this analysis, we propose a model that represents the relationship between prioritization and adaptation.

In the following sections the two frameworks for requirements-based adaptation are briefly described and discussed. Section IV presents a proposal to make explicit the relationship between prioritization and adaptation. Lastly, section V concludes the paper and present future work.

II. RELAX FRAMEWORK

The RELAX approach for adaptation is grounded on the uncertainty that afflicts the execution of complex socio-technical systems. Since lack of resources, unforeseen conditions, and failed assumptions, among other factors, may prevent a system to completely satisfy its requirements, a fuzzy specification is adopted: instead of binary requirements satisfaction (successfully or not), each requirement may present different degrees of satisfaction.

In the RELAX process, each requirement (in the form of a *shall* statement) is analyzed, deciding if it must be RELAXed or if it should be kept as-is (*invariant requirement*). A RELAXed requirement contains a new version of the requirements statement using one of the following operators of the RELAX language: *SHALL; MAY...OR; EVENTUALLY; UNTIL; BEFORE/AFTER; IN; AS EARLY/LATE AS POSSIBLE; AS CLOSE AS POSSIBLE TO;* and *AS MANY/FEW AS POSSIBLE*. Moreover, the following types of uncertainty factors can be defined for each requirement: environmental, monitoring, relationship (between environment and monitoring), and dependencies (between requirements).

978-1-5090-3581-6/16 $31.00 © 2016 IEEE
DOI 10.1109/QUATIC.2016.52

The following requirement of a smart home system [7] illustrates these operators and uncertainty factors: `The fridge **SHALL** detect and communicate with food packages.` After going through the process, see the RELAXed requirement in Figure 1.

This relaxed requirement is more flexible than the original statement, since now the system does not need to detect and communicate with every food package, but with *as many as possible*. In the following sub-section we describe further requirements of this system (extracted from [7]), illustrating the use of the RELAX process and language.

R1.1' The fridge **SHALL** detect and communicate information with **AS MANY** food packages **AS POSSIBLE**.	
UNCERTAINTY FACTORS	**ENVIROMENT:** Food locations, food item information (type, calories) & food state (spoiled, unspoiled).
	MONITORING: RFID readers; Cameras; Weight sensors.
	RELATIONSHIP: RFID tags provide food locations/food information/food state; Cameras provide food locations; Weight sensors provide food information (whether eaten or not).
	DEPENDENCIES: negatively impacts R1.2'; positively impacts R1.4 and R1.6.

Fig. 1 Complete specification of the R1.1' requirement, from [7]

1) Example:

To exemplify RELAX we use a smart-home system focused on healthcare described in [7], with the requirements shown in Figure 2. This system includes a fridge that detects the food it contains (**R1.1**), using this information to create diet plans for its user (**R1.2**). Besides defining the user's diet, the system must take care of the liquid intake of the user (**R1.3**) and raise an alarm in case of prolonged and unexpected inactivity (**R1.5**). Additionally, the system must have low energy consumption (**R1.4**) and low alarm latency (**R1.6**).

R1.1: The fridge SHALL detect and communicate with food packages.
R1.2: The fridge SHALL monitor and adjust the diet plan.
R1.3: The system SHALL ensure a minimum of liquid intake.
R1.4: The system SHALL minimize energy consumption during normal operation.
R1.5: The system SHALL raise an alarm if no activity by Mary is detected for t.b.d. *(to be defined)* hours during normal waking hours.
R1.6: The system SHALL minimize latency when an alarm has been raised.

Fig. 2 Requirements statements of the smart home system, from [7]

After enacting the RELAX process, the first four requirements were modified (see Figure 3).

R1.1': The fridge **SHALL** detect and communicate information with **AS MANY** food packages **AS POSSIBLE**.
R1.2': The fridge **SHALL** suggest a diet plan with total calories **AS CLOSE AS POSSIBLE TO** the daily ideal calories. The fridge **SHALL** adjust the diet plan in line with Mary's actual calorie consumption.
R1.3': The system **SHALL** ensure that Mary's liquid intake is as **AS CLOSE AS POSSIBLE TO** ideal during the course of the day. The system **SHALL** ensure minimum liquid intake **BEFORE** bedtime.
R1.4': The system **SHALL** consume **AS FEW** units of energy **AS POSSIBLE** during normal operation.

Fig. 3 Relaxed requirements of the smart home system, from [7]

For the sake of brevity, the uncertainty factors of these requirements are not presented here, except for **R1.1** which was presented in the previous sub-section. Requirements **R1.5** and **R1.6** remained unchanged (invariant requirements). In the following sub-sections we discuss the RELAX process and language in light of the rationales for these modifications as described in [7].

2) Discussion:

The RELAX language defines a set of operators that, when applied to a given requirements statement, may reduce the constraints imposed by the original requirements.

This relaxation also improves the flexibility of the system, as it now can manage the use of the available resources in order to favor the satisfaction of a requirement or another. For instance, the rationale for **R1.1'** presented in [7] describes that "*it might be preferable to divert resources from the intelligent fridge to support emergency functions…*", where **R1.6** is an example of an emergency function (minimal alarm latency) that should be favored. Implicitly, this indicates that **R1.6** has higher priority than **R1.1**.

During the RELAX process, dependencies between requirements are made explicit, in order to define which other requirements are impacted by the relaxation. For instance, "*relaxing **R1.1** will impair the system's ability to suggest an appropriate diet plan (**R1.2'**)*" [7]. This information indicates that **R1.2** depends on **R1.1**, which is relevant information for prioritization and release planning [8].

As shown for **R1.1'**, some kind of monitoring is required in order to obtain information from the environment. Thus, a requirement may become costlier to implement, due to the implementation of monitoring capabilities. On the other hand, a requirement may become less costly to implement, if the original statement was too strict. For instance, if the system needed to obtain information from every food package (instead of as many as possible), monitoring through RFID would not suffice and more expensive solutions would be required (for instance, video cameras and image processing). In either way, the RELAX process has potential to change the cost of implementing a requirement, which is a relevant factor for requirements prioritization [5][9].

Relaxation implicitly suggests that the requirement is of lower priority than an unchanged (invariant) requirement. This is evidenced by excerpts from [7]: "***R1.5** was considered an invariant and was thus unchanged, since it specified behavior that may be critical to Mary's health*"; "*…treat **R1.6** as an invariant because of its implicit criticality to Mary's health*".

III. ZANSHIN FRAMEWORK

Zanshin is a framework for requirements-based adaptation which makes the feedback loop explicit by defining what needs to be monitored (through indicators) and what can be changed (by means of parameters and adaptation strategies).

The Zanshin process for system identification of adaptive software systems starts with a requirements goal model and

ends with a goal model enriched with adaptation elements. The process represents an incremental refinement of the goal model, in order to include: indicators, parameters, and relations between parameters and indicators.

The goal model variation adopted by Zanshin represents goals, tasks, softgoals, domain assumptions (DAs) and quality constraints (QCs). The space of alternatives for goal satisfaction is represented by Boolean AND/OR refinements. Tasks are directly mapped to functionality in the running system and are satisfied if executed successfully. DAs are satisfied if they hold (affirmed) while the system is pursuing their parent goal. Softgoals represent, usually, non-functional requirements that do not have clear-cut criteria for satisfaction, such as *Low cost* or *Fast response*. Nonetheless, their satisfaction may be metricized by one or more QCs, which can be associated with run-time procedures that check their satisfaction.

In order to better support the development of adaptive systems, [1][2] propose two additional concepts for goal models: indicators (named awareness requirements) and parameters.

- Indicators are requirements about the state of other requirements (such as their success or failure) at runtime. In practice, indicators define which requirements should be monitored at runtime.
- Parameters are variables that affect the fulfillment of indicators. They represent some attribute of the system that may be modified as desired. For example: level of automation, number of servers and of ambulances available.

When indicators fail at runtime, a possible adaptation strategy is to change one or more parameter values in order to improve chances of success. This change is driven by qualitative differential expressions relating parameters and indicators [1] stating, for instance, that increasing the *number of staff members* in a call center increases the likelihood of *successfully taking calls*.

1) Example:

The use of Zanshin is illustrated by a case study based on the London Ambulance Service Computer-Aided Dispatch (CAD) system, as reported in [10]. An excerpt of the goal model is shown on Fig. 4. Its main goal is to *Generate optimized dispatching instructions*. This goal is AND-refined onto three other goals: *Call taking*, *Resource mobilization,* and *Map retrieval*. Furthermore, the following domain assumption was made: *Resource data is up-to-date*. The *Resource mobilization* goal is further AND-refined onto *Determine best ambulances, Inform stations / ambulances, Provide route assistance,* and *Get good feedback.* The system also presents two softgoals, *Fast arrival* and *Fast dispatching*, which are refined with the following quality constraints, respectively: *Ambulance arrive in 8 min* and *Dispatching occurs in 3 min*.

Besides these traditional goal model elements, Fig. 4 also shows some adaptation elements – namely, indicators and parameters. The *i5* and *i11* indicators mandate, respectively, that the *Resource data is up-to-date* domain assumption and the *Dispatching occurs in 3 min* quality constraint should never

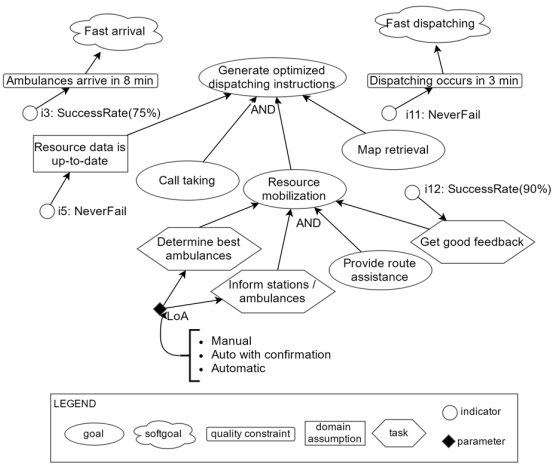

Fig. 4. Excerpt of the goal model of the CAD system, based on [10]

fail – i.e., throughout the entire system execution, these elements should always be satisfied. Moreover, *i3* defines that the *Ambulance arrive in 8 min* quality constraint should have a success rate of at least 75%, whereas *i12* states that the *Get good feedback* task should have a success rate of at least 90%.

Lastly, the *LoA* (Level of Automation) parameter can have any of the following values: *Manual, Auto with confirmation,* or *Automatic*

2) Discussion:

According to [10], the following elements are input for defining the indicators of the system: critical requirements, non-functional requirements, preferable solutions, trade-offs, preemptive adaptation, meta-awareness requirements, and qualitative elicitation. From these, the following are related to prioritization: critical requirements, preferable solutions, and trade-offs.

As reported in [10], the definition of indicators for the CAD system was mostly based on problems that may occur during execution. A few excerpts of the report show, nevertheless, the impact of criticality in the identification of indicators: "*Such a subset of critical goals can be identified in the process and AwReqs specifying the precise achievement rates that are required for these goals will be attached to them*"; "*A constraint on the first part is depicted in the CAD model by quality constraint Dispatching occurs in 3 min and to indicate the criticality of this constraint, AwReq AR11 indicates the constraint should never fail*" [10]. These excerpts describe a rationale for creating indicators (AwReqs): the criticality of the requirements that they refer to.

Since indicators define which requirements should be monitored, they potentially affect the costs of implementing some requirements. Thus, this additional cost should be taken into consideration in an eventual reprioritization.

Similarly to RELAXed requirements, the parameters included in a goal model potentially create new dependencies in the system. This is the case since the same parameter may affect different indicators, creating an indirect dependency between them. It is possible, for instance, that modifying a

parameter in order to improve the success likelihood of a given indicator will have adverse effects on other(s) indicators(s). For instance, decreasing values for *Level of Automation* (LoA) helps the *i3* and *i12* indicators, whereas it hurts the satisfaction of the *i11* indicator (which refers to the *Dispatching occurs in 3 minutes* quality constraint) [10].

Given the potential of parameter changes having both positive and negative impacts in the system, it is helpful to have explicit priorities of the requirements in order to decide which trade-offs to perform whenever an adaptation is needed, as discussed in [11]. Lastly, when defining adaptation strategies, different alternatives for handling the failure of a given indicator can be defined. For instance, two possible reconfigurations were defined if *i11* is not successful (i.e., if an ambulance dispatch takes longer than three minutes). The order on which the different alternatives should be chosen suggests implicit prioritization of these alternatives.

IV. Where does prioritization happen?

. On one hand, requirements priorities seem to play a role on the decisions made during the RELAX process, thus they would be a valuable input. On the other hand, the RELAX process is an analysis process that provides additional information about the requirements, such as cost and dependencies. Since this information is an input of the requirements prioritization activity, a reprioritization may be advisable in order to revisit preferences in light of what was uncovered during the process.

The Zanshin framework, although using different foundations, is similar to RELAX in its relationship with prioritization: requirements' priorities are an input for defining some aspects of adaptation; new information is discovered during the process, which may warrant a reprioritization of the system requirements. Unlike RELAX, Zanshin has an implicit prioritization of alternative adaptations that can be executed, in terms of ordered enumeration of parameters and of different execution orders within adaptation strategies.

Based on these evidence-based observations, we propose that before starting the process of specifying adaptation requirements the original requirements should be prioritized (see Fig. 5). After an iteration of the adaptation process is completed, a reprioritization may be needed. Lastly, within the adaptation process a specific kind of prioritization may be useful: the prioritization of alternative adaptations.

V. Conclusion

By analyzing different requirements-based frameworks for software adaptation, we identified that requirements priorities can aid the task of specifying adaptation. Moreover, we observed that the specification of adaptation is an analysis process where further information is elicited, in such a way that in some cases a reprioritization may be warranted. Lastly, we have found weaker evidence that the prioritization of alternative options for adaptation may be beneficial to better choose which adaptation to perform when more than one alternative is available.

We have used examples from case studies performed and reported by the authors of the respective frameworks, instead of performing new case studies of our own, in order to reduce bias from our part.

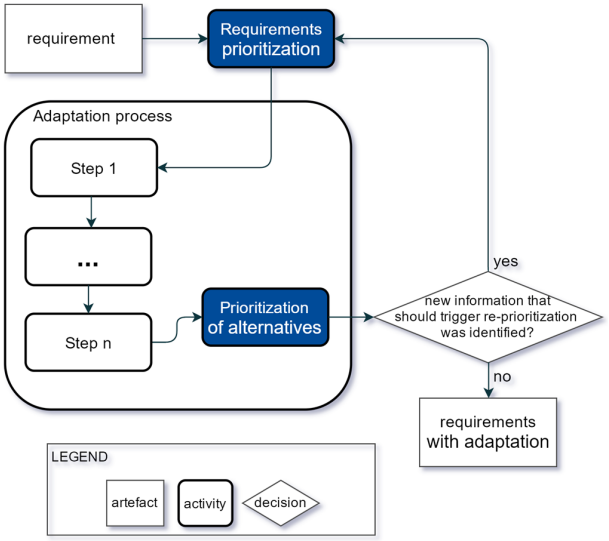

Fig. 5. Overview of the relationship between prioritization and adaptation

References

[1] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "System Identification for Adaptive Software Systems: A Requirements Engineering Perspective," *Conceptual Modeling – ER 2011*, 2011, pp. 346–361.

[2] V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos, "Awareness Requirements," *Software Engineering for Self-Adaptive Systems II*, vol. 7475, R. Lemos, H. Giese, H. A. Müller, and M. Shaw, Eds. Springer, 2013, pp. 133–161.

[3] D. Weyns, M. Usman Iftikhar, and J. Soderlund, "Do external feedback loops improve the design of self-adaptive systems? A controlled experiment," 2013 8th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst., pp. 3–12, May 2013.

[4] D. M. Berry, B. H. C. Cheng, and J. Zhang, "The four levels of requirements engineering for and in dynamic adaptive systems," in 11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ), 2005, p. 8.

[5] P. Berander and A. Andrews, "Requirements Prioritization," in Engineering and Managing Software Requirements,

[6] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, et al, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap," in Software Engineering for Self-Adaptive Systems II, vol. 7475, no. October 2010, R. Lemos, H. Giese, H. Müller, and M. Shaw, Eds. Springer Berlin Heidelberg, 2013, pp. 1–32.

[7] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Chengy, and J. M. Bruel, "Relax: a language to address uncertainty in self-adaptive systems requirement," Requir. Eng. J., vol. 15, no. 2, pp. 177–196, 2010..

[8] P. Carlshamre, K. Sandahl, M. Lindvall, B. Regnell, and J. N. O. Dag, "An industrial survey of requirements interdependencies in software product release planning," Proceedings Fifth IEEE International Symposium on Requirements Engineering. pp. 84–92, 2001.

[9] K. Wiegers and J. Beatty, Software Requirements, 3rd ed. Pearson Education, 2013.

[10] V. E. S. Souza, "Requirements-based software system adaptation," Ph.D Thesis, University of Trento, Italy, 2012.

[11] K. Angelopoulos, V. E. S. Souza, and J. Mylopoulos, "Dealing with Multiple Failures in Zanshin: a Control-Theoretic Approach," Proc. 9th Int. Symp. Softw. Eng. Adapt. Self-Managing Syst. - SEAMS 2014, no. May 2016, pp. 165–174, 2014.