Dealing with Multiple Failures in Zanshin: A Control-Theoretic Approach

Konstantinos Angelopoulos
University of Trento, Italy
angelopoulos@disi.unitn.it

Vítor E. Silva Souza
Federal University of Espírito Santo, Brazil
vitorsouza@inf.ufes.br

John Mylopoulos
University of Trento, Italy
jm@disi.unitn.it

ABSTRACT

Adaptive software systems monitor the environment to ensure that their requirements are being fulfilled. When this is not the case, their adaptation mechanism proposes an adaptation (a change to the behaviour/configuration) that can lead to restored satisfaction of system requirements. Unfortunately, such adaptation mechanisms don't work very well in cases where there are multiple failures (divergence of system behaviour relative to several requirements). This paper proposes an adaptation mechanism that can handle multiple failures. The proposal consists of extending the *Qualia* adaptation mechanism of *Zanshin* enriched with features adopted from Control Theory. The proposed framework supports the definition of requirements for the adaptation process prescribing how to deal at runtime with problems such as conflicting requirements and synchronization, enhancing the precision and effectiveness of the adaptation mechanism. The proposed mechanism, named *Qualia+* is illustrated and evaluated with an example using the meeting scheduling exemplar.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications

Keywords

requirements engineering, adaptive systems, multiple failures, optimization, *Zanshin*

1. INTRODUCTION

Adaptive software systems are designed so that they fulfill their requirements in multiple ways (through multiple configurations/behaviours). When a failure is detected (i.e., a requirement of the system is not fulfilled), the system adapts by switching to an alternative behaviour. Unfortunately, such mechanisms treat each failure on its own merits and don't work very well in cases where multiple failures occur. The reason is simple: in case of failures F, F', the chosen adaptations A, A' may be conflicting, as A may call for a behaviour that exacerbates F', and vice versa with A'. This problem is common in Control Theory, known as multivariable control problem for Multiple Inputs Multiple Outputs (MIMO) [5, 10, 17].

Current approaches either deal with requirement failures individually (e.g., *Qualia* [21]) or apply reconfiguration strategies using rules (e.g., Rainbow [6]). Both approaches have their pitfalls and drawbacks, as they can result in behaviour that continues to fail to fulfill requirements and/or is unstable because of existing dependencies among requirements. Therefore, while treating one failure by tuning some parameters of the system other requirements may be negatively impacted, resulting in immediate or eventual failure of other requirements rather than the one being treated. On the other hand, rule-based adaptation strategies, e.g., [13, 6], don't scale very well as the number of combinations that need to be considered grows combinatorially with the number of concurrent failures. More specifically, if we had to define at design time adaptation rules for up to k possible failures then in order to cover all possible combinations of failures we would require $O(2^k)$ rules.

The main objective of this paper is to propose an adaptation mechanism that can handle multiple failures (i.e., multiple failing requirements). Our proposal considers at the same time all failing requirements and attempts to select an adaptation that is coherent in the sense that it reduces the overall degree of failure, taking into account priorities among requirements. Our proposal supports the definition of adaptation requirements provided by stakeholders. For example an adaptation requirement may state that the adaptation should be conservative in that it does not change parameters in a way that could harm non-failing requirements. Such adaptation requirements are taken into account as the adaptation mechanism considers iteratively current failures, selects an adaptation, applies it, and observes results. The ultimate goal of this approach is to handle dependencies among requirements in a way that is consistent with stakeholder expectations about the adaptation process itself.

The rest of the paper is structured as follows. Section 2 presents the research baseline for this work. Section 3 describes the use of the Analytic Hierarchy Process (AHP) to prioritize requirements and the definition of adaptation requirements using concepts from the *Zanshin* framework. Section 4 depicts the *Qualia+* adaptation mechanism and the extensions applied on *Zanshin*. In Section 5 we evaluate the new mechanism comparing it with older version of

Qualia. Finally, in Section 6 we compare the related work with our proposal and concluding in Section 7.

2. BASELINE

This paper builds on concepts adopted from Goal-Oriented Requirements Engineering, such as goals for modelling stakeholder requirements, softgoals for modelling non-functional requirements, and AND/OR refinements that refine a goal G into simpler goals G_1, $...G_n$ whose satisfaction implies the satisfaction of G. Tasks are actions that an actor can take (the system itself, or an external actor) to fulfill or operationalize a goal. For example, in Figure 1, *Schedule meeting* is a top-level goal, while *Good participation* is a softgoal. The goal Schedule meeting is AND refined into task *Characterize meeting* and subgoals *Collect timetables* etc. (4 subgoals in all).

Given this model, we use *Zanshin* [19] for the design of adaptive systems. Based on requirements and inspired by feedback control theory, the *Zanshin* framework consists of three steps: Awareness Requirements Engineering, System Identification and Evolution Requirements Engineering.

Awareness Requirements Engineering : the first step is concerned with the elicitation of Awareness Requirements, hereafter *AwReqs*, which are requirements that impose constraints over the success/failure of other requirements. *AwReqs* determine the monitoring component of the feedback loop that monitors and adapts a base system. For instance, Figure 1 includes requirement Q: *At least 90% of participants attend meetings*. A possible *AwReq* $AR6$ over this requirement is: *Q should have 75% success rate*.

System Identification : the second step extracts indicators from elicited *AwReqs* and identifies system parameters that, when changed, affect such indicators. These parameters can be either Control Variables (CVs) or Variation Points (VPs). The CVs are variables bounded within a limited range of values that could be either continuous or discrete. One the other hand, VPs are result of the OR-decompositions of the goal models and represent the functional variability of our system without though the extensive detail that architectural models do. The nature of this change is represented by differential relations. Back to the Meeting Scheduler example, indicator I_6 (meeting attendance) can be extracted from $AR6$. Then, parameter FhM, representing *from how many* participants one should collect timetables before scheduling a meeting, is identified. The impact that a parameter has on an indicator is characterized qualitatively by a monotonicity relation. For instance, the relation $\Delta\left(I_6/FhM\right) > 0$ represents the fact that if you increase FhM, I_6 should also increase (knowing more participants' timetables results in better scheduling and, hence, better attendance). In this case the monotonicity of this relation is isotone. On the other hand, the relation $\Delta\left(I_3/FhM\right) < 0$ represents the fact that if you increase FhM, I_3 is expected to decrease (antitone monotonicity).

Evolution Requirements Engineering : if *AwReqs* are requirements for the monitoring component of the feedback loop, Evolution Requirements (hereafter *EvoReqs*) represent requirements for the adaptation component. *EvoReqs* specify required changes to other requirements when certain conditions apply (e.g., the failure of an *AwReq*). For instance, say there is a domain assumption D in the Meeting Scheduler's requirements model that says *Participants use the system calendar* to keep track of their free/busy hours. If *AwReq* $AR7$: *D should always be true* fails, one way of adapting is to replace assumption D with a task T: *Enforce participants' use of system calendar*. This requirement effectively defines an evolution of other requirements in the requirements model.

The *Zanshin* framework[1] takes as input the requirements model for the base system, including the elements described above, and acts as a feedback controller that monitors for failures and determines adaptation actions to be carried out. *Zanshin* supports two kinds of adaptation: *evolution*, described above, and *reconfiguration*. Reconfiguring consists on finding an alternative specification with new values for system parameters (identified during System Identification) in order to improve indicators of failing *AwReqs* and take the system back to an acceptable state. For instance, if $AR6$ fails, which means I_6 needs improving, *Zanshin* might instruct the base system to increase FhM.

Although *Zanshin* can be integrated with other reconfiguration mechanisms, it provides its own named *Qualia* [21]. *Qualia* is an extensible adaptation mechanism that defines a basic algorithm composed of 8 procedures (parameter selection, parameter change, and so on), which can be replaced by more advanced procedures if desired. The basic algorithm uses qualitative information about indicators, parameters and their interrelations (defined by differential relations) in order to randomly select which parameter to tune from the set of parameters that can positively affect a given indicator of a failing *AwReq*. Advanced procedures may require more precise information from the model (e.g., parameters related to an indicator be given an order of magnitude of their effect), allowing *Zanshin* to handle multiple levels of precision for the adaptation mechanism.

This paper tackles one of *Qualia*'s main limitations: it deals with one failure at a time, on a first-come-first-served fashion. Moreover, it ignores the case in which the adaptation action chosen for one indicator affects negatively other indicators that its *AwReq* may or may not be failing concurrently. For instance, for the Meeting Scheduler we identify the differential relations $\Delta\left(I_6/FhM\right) > 0$ and $\Delta\left(I_3/FhM\right) < 0$ that share the same parameter characterized by opposite monotonicity. Consequently if $AR3$ fails by decreasing the FhM parameter, I_6 would be affected negatively.

3. REQUIREMENTS FOR ADAPTATION

Handling multiple requirements failures requires tradeoffs. To achieve this, we extend *Zanshin* in a way that it can dynamically put together adaptation strategies using priorities over the requirements as criteria for resolving runtime conflicts among requirements that could not be eliminated at design time. We also propose the specification of *Adaptation Requirements* (*AdReqs*) . These requirements are defined by reusing the *Zanshin* framework, namely the notation used to specify *AwReqs* and *EvoReqs*.

3.1 Prioritizing Requirements

Requirements are prioritized in order to support the selection among alternative adaptations during the adaptation process. If R, R' are both failing, it is important to know

[1]See `https://github.com/sefms-disi-unitn/Zanshin`.

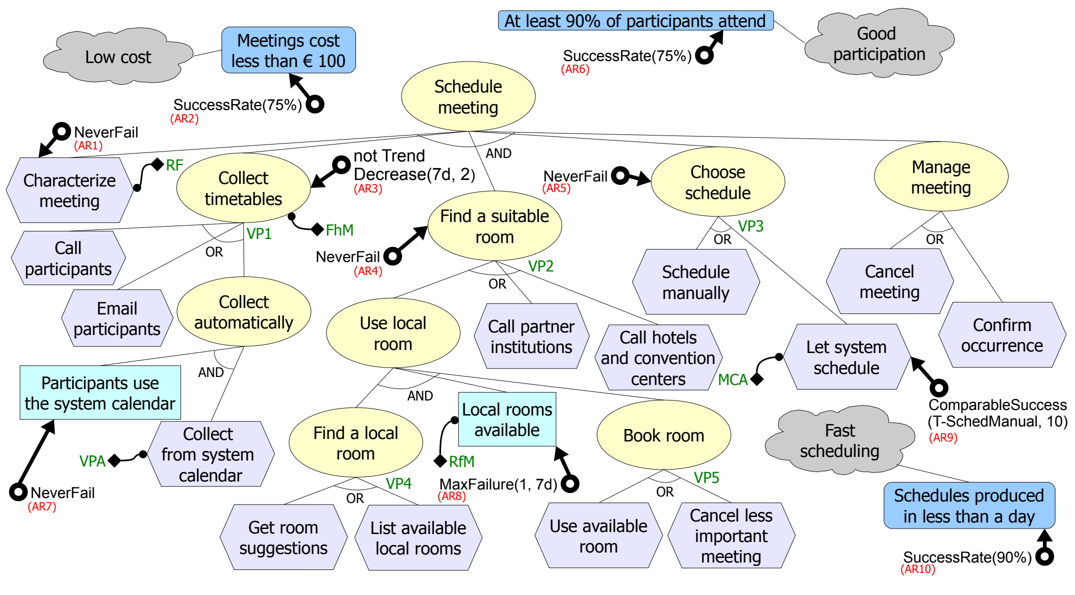

Figure 1: Meeting Scheduler Goal Model

which of the two has higher priority. To make our adaptation mechanism more precise in dealing with failures, we actually require information on how much higher priority does R have over R', a little or a lot.

Given such information, we can adopt the Analytic Hierarchy Process (AHP) [14] which has been proven to be an effective method for prioritizing objectives [8]. Other applications of the AHP prioritize requirements as a means to select adaptations that address failures for requirements of highest priority. [7]. In our case the prioritization is useful for selecting which requirement failure should be fixed and which one should not because it would create further failures that should lead to changes to other requirements.

The process for prioritizing *AwReqs* and their associated indicators includes the following steps. First, after the system identification process is carried out and the qualitative relations among indicators and the parameters are elicited, we identify which indicators present potential conflicts. By the term 'conflict' we mean that two indicators are influenced by the same parameter in opposite directions. For example if I_1 and I_2 are both influenced by parameter P_4 and $\Delta\left(I_1/P_4\right) > 0$ and $\Delta\left(I_2/P_4\right) < 0$ their differential relations mean that if $AR1$ and $AR2$ are failing and we can treat them only by tuning P_4 then we cannot fix both of them. Then, by using the scale presented in Table 2 we compare all the pairs of indicators and assign a value to each pair. For the purpose of illustration consider that we have four indicators I_1, I_2, I_3 and I_4 and the result of the pairwise comparisons is shown in Table 1.

-	I_1	I_2	I_3	I_4
I_1	1	3	1/5	1
I_2	1/3	1	7	1/5
I_3	5	1/7	1	1/5
I_4	1	5	5	1

Table 1: Pairwise Comparison Values

For an effective use of AHP we apply two heuristics that work as guidelines for assigning consistent priority values:

Heuristic 1: Indicators associated with hard-goals are preferred over those of soft-goals.

Heuristic 2: Indicators of hard-goals that are closer to the top-goal are preferred over lower level ones.

The purpose of these heuristics is to give higher priority to the functional integrity of the system over satisfaction of the non functional requirements. The process continues by calculating eigenvalues and then normalizing sums of rows. The final result is shown in equation 1.

$$\frac{1}{4}\cdot\begin{pmatrix}0.87\\0.75\\0.84\\1.45\end{pmatrix}=\begin{pmatrix}0.22\\0.18\\0.21\\0.36\end{pmatrix}\begin{pmatrix}I1\\I2\\I3\\I4\end{pmatrix} \qquad (1)$$

We have now assigned weights over each indicator that represent their relative and we have a numerical guide to perform comparisons when needed. For instance, in the case where the system has to choose between fixing either I_1 and I_3 or I_4, even if I_4 is ranked higher than the other two their aggregated weight is higher and therefore should be preferred.

3.2 Adaptation Requirements

Our proposal includes a component that, given a requirements model and differential relations among requirements and system parameters is able to dynamically compose adaptation strategies that can handle multiple failures. As a first step we prioritized the indicators of the target system with weights that also measure their overall contribution to the correct operation of the system. This, combined with the qualitative relations from the system identification process allows *Zanshin* to automatically compose adaptation strategies, reconfiguring the system and maximizing the value of the satisfied indicators.

Relative Intensity	Definition	Explanation
1	Of equal importance	The two indicators are not conflicting
3	Slightly more important	Experience and judgement slightly favors one indicator over the other
5	Essentially more important	Experience strongly favors one indicator over another
7	Very much more important	An indicator is strongly favored and its dominance is demonstrated in practice
9	Extremely more important	The evidence favoring one over the other is of the highest possible validity
2,4,6,8	Intermediate values	When compromise needed
Reciprocals: If indicator I has one of the above numbers assigned to it when compared with requirement I', then I' has the reciprocal value when compared with I.		

Table 2: Scale For Pairwise Comparisons

Our framework deals with the following kinds of failure:

- *Single Failure (SF):* Only one indicator is failing and needs to be fixed.
- *Multiple Independent Failures (MIF):* Many indicators are failing and either they don't have any common parameters or the common parameters have the same monotonicity with every failing indicator.
- *Multiple Dependent Failures (MDF):* Many indicators are failing and all or groups of them share parameters with opposite monotonicity.
- *Priority Conflict (PC):* A failing indicator can be fixed only by tuning a parameter that harms a non-failing indicator of higher priority.
- *Synchronization Conflict (SC):* An indicator is currently being treated and in the meantime a new failing indicator requires tuning of a parameter that has a negative impact on the one being treated.

In Control Theory a common way to resolve such conflicts is through compensation [4]. This method involves the addition of control mechanisms called compensators whose role is to compensate for deficient performance. In other words, compensators make trade-offs on output values or modify the reference input to achieve an acceptable result. *Zanshin* is able to accommodate compensation mechanisms by defining *EvoReqs* that refer to the adaptation process itself. In this work we exploit this capability of *Zanshin* to achieve conflict resolution and minimize the error caused by failing *AwReqs*.

Therefore, we model in *Zanshin* additional *AwReqs* and *EvoReqs* whose subject is the adaptation process itself, rather than the base-system. Figure 2 presents a simple goal model that prescribes how conflicts should be resolved during the adaptation process. The task *Adapt Conservatively* instructs the framework not to harm non-failing *AwReqs* while fixing the failing ones. The alternative task *Adapt with Compensation* represents the compensation mechanism we mentioned earlier. More specifically, the adaptation framework is allowed to harm a non-failing *AwReqs* of higher value for fixing another one, only if there is a possible action that would increase the indicator of the *AwReq* being harmed. Along the same lines, *Adapt Optimistically* does not consider priority conflicts as hazards for the base system because there is the assumption that the non-failing *AwReqs* are tolerant enough to a potential negative impact. We refer to these requirements as *AdReqs*.

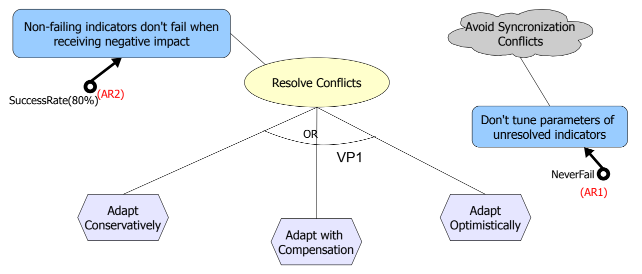
Non-failing indicators don't fail when receiving negative impact
Avoid Syncronization Conflicts
SuccessRate(80%)
(AR2)
Resolve Conflicts
OR
VP1
Don't tune parameters of unresolved indicators
NeverFail
(AR1)
Adapt Conservatively
Adapt with Compensation
Adapt Optimistically

Figure 2: Adaptation Requirements Goal Model

To monitor the success/failure of requirements for the adaptation process depicted in Figure 2 we use *AwReqs*. In this case AR2 imposes the constraint that non-failing *AwReqs* should not fail when receiving negative impact. As we would do for any goal model of a target system we define an adaptation strategy to overcome failures of this Awareness Requirement. Given the differential relation $\Delta\,(I_2/VP1)$ $[AdaptConservatively \rightarrow Adaptwith\ Compensation \rightarrow AdaptOptimistically] < 0$ (the arrows indicate growing enumeration values [20]) the strategy switches among the possible values of the parameter $VP1$.

```
AwReq AR2: Non-failing indicators don't fail when
    receiving negative impact
-Checked at: every 5 minutes
-Adaptation Strategy 3.1: ChangeParam(VP1, Adapt with
     Compensation)
  -Applicability Condition: this is the first failure
-Adaptation Strategy 3.2: ChangeParam(VP1, Adapt with
     Compensation)
  -Applicability Condition: AS3.1 applied last, more
      than 5 minutes ago
-Adaptation Strategy 3.3: ChangeParam(VP1, Adapt
    Optimistically)
  -Applicability Condition: no failure for more than
      1 hour
```

Listing 1: Adaptation Strategy for Adaptation Requirements

Independently of the value of parameter $VP1$, the adaptation framework is required to perform trade-offs among indicators of failing and non failing *AwReqs*. For some indicators there is a parameter to change in order to be bring it closer to fulfilment. For others there won't be any, at least ones that don't harm higher priority indicators. Hence, when potential conflicts are detected during system identification process *EvoReqs* operations (e.g. abort, retry, replace etc) should be assigned to every indicator that may conflict with others. For instance, for the Meeting Scheduler $AR6$ and $AR10$ are both dependent on the parameter

FhM through the differential relations $\Delta(I_6/FhM) > 0$ and $\Delta(I_{10}/FhM) < 0$. Consequently, if $AR6$ has higher priority than $AR10$ and $AR10$ fails, but the framework applies conservative adaptation, a predefined *EvoReqs* operation for $AR10$ could be $Replace(AR10\_1day, AR10\_2days)$. This way we acquire compensation in accordance with requirements set by the stakeholders.

Given the fact that the changes to parameter values do not take effect immediately and in the meantime more failures may take place, it is important to apply a form of synchronization to the adaptation process. In Figure 2 the *Avoid Synchronization Conflicts* and the $AR1$ satisfy this need. This goal states that when a failing indicator is being fixed no parameter can be tuned in a way that will affect the failing indicator negatively. For example, if $AR8$ is failing and the adaptation framework increases RfM to fix it. However, this change may require several time to take effect and before that happens. $AR2$ fails. In order to fix $AR2$ the parameter $VP2$ must be decreased, but that would affect negatively $AR8$ before the latter has been fixed. To avoid such situations that could lead the adaptation process into non-converging adaptations, we set as a requirement not to apply changes that affect unresolved indicators for the shake of fixing other failures.

4. ADAPTATION PROCESS FOR MULTIPLE FAILURES

In the previous section we presented a set of features such as prioritization and compensation mechanisms that can help us handle multiple concurrent failing indicators and compose dynamically adaptation strategies. This section describes the additions to the *Zanshin* framework that implement these features and explains the steps that the adaptation process carries out.

Figure 4 depicts the conceptual architecture of the extended *Zanshin* framework. A *Monitor Component* examines the log files that are produced at runtime from the base system if any failing *AwReqs* are detected the *Failure Manager* and the *Adaptation Manager* are informed. The *Failure Manager* groups failing indicators according to the presence of conflicts with other indicators and informs the *Decision-Maker* component. The *Adaptation Manager* is responsible for configuring the adaptation process by monitoring the requirements model such as the one in Figure 2. When an *AwReq* fails, it selects an adaptation strategy for this failure. Then, it informs the *Decision-Maker* about the selected parameter values of the adaptation process goal model in order to perform the trade-offs accordingly. The *Decision-Maker* exploits input from other components and the indicators' value derived from the AHP to select which indicators should be tuned and which should be compensated. The *Strategy Manager* converts decisions to adaptation strategies by putting together all the required operations. Finally, the *Adapt* component executes the operations that are prescribed in the adaptation strategy.

To give a better understanding of how the framework operates, the diagram of the Figure 3 presents every step of the adaptation process. The steps are:

1. All the *AwReq* failures are collected by the failure manager;

2. The indicators of the failing *AwReqs* are separated with criteria related to the conflicts they may be part of;

3. The decision-maker exploiting the differential relations and the values assigned to each indicator resolves any conflicts that may exist by deciding what action should be performed;

4. For every indicator that is selected to be fixed there might be more than one parameter available and one parameter may be able to fix more than one selected indicator. The framework selects the minimum number of parameters that should be tuned to fix the chosen indicators.

5. The values for the selected parameters are calculated;

6. The values of the selected parameters are changed and the *EvoReqs* operations for the indicators that can't be treated are executed;

7. The framework waits for the changes to take effect;

8. After the wait time the indicators are evaluated again;

9. In each cycle, the process learns from the outcome of this change and the recorded data could be used to derive quantitative relations among indicators and parameters;

10. Finally, if there are no more failing *AwReqs* after the evaluation the process terminates successfully.

11. Otherwise, the framework will look for violations on the requirements for the adaptation process and if any are found the defined adaptation strategy will be executed.

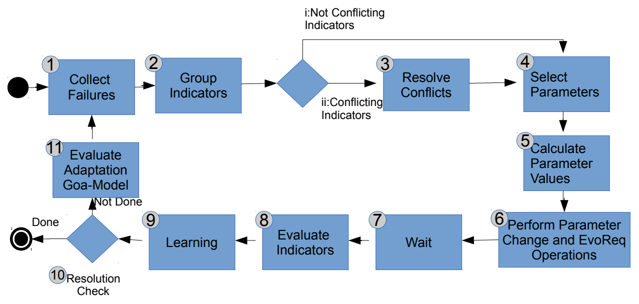
i:Not Conflicting Indicators
1 Collect Failures
2 Group Indicators
ii:Conflicting Indicators
3 Resolve Conflicts
4 Select Parameters
5 Calculate Parameter Values
6 Perform Parameter Change and EvoReq Operations
7 Wait
8 Evaluate Indicators
9 Learning
10 Resolution Check
Done
Not Done
11 Evaluate Adaptation Goa-Model

Figure 3: *Zanshin*'s Adaptation Process

The strategy manager composes a new strategy with all the actions that will apply the new values to the parameters. These actions are executed by Requirement evolution operations on the target system. The framework waits for an amount of time for the changes to take place and then evaluates the indicators that were treated. There is a step that the framework is performing learning in order to derive quantitative relations among parameters and indicators, but it's part of our future research agenda and is not examined in this paper. Finally, the algorithm terminates if all the failing *AwReqs* are fixed and if not the policy manager controls

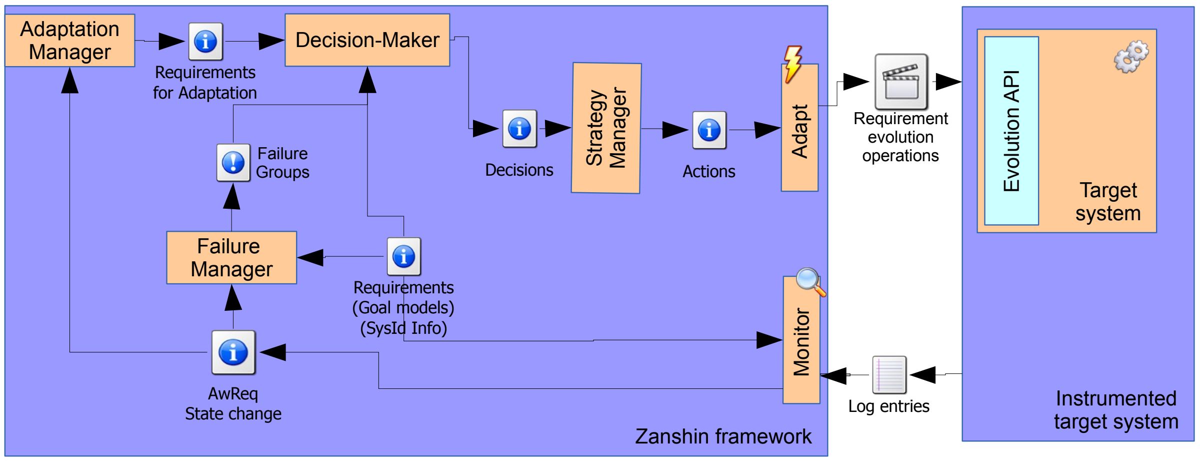

Figure 4: *Zanshin* Architecture

the status of of the adaptation requirements and switches policy if there are any failures.

Computationally the most complex step of the adaptation process is the one that groups failing indicators. The *Failure-Manger* has to compare all the failing indicators pairwise. The output is two disjoint sets of indicators. The indicators of the first set do not present any conflicts with each other neither with those of the other set. On the other hand, the indicators of the the second set do present conflicts with other indicators of the same subset. The *Decision-Maker* is responsible for finding a subset of this set where the indicators don't conflict with each other and the sum of their priority values is the maximum. This is a well-studied NP-hard combinatorial optimization problem [11] for which greedy algorithms have been proposed [15] and perform sufficiently well. We note that as a task of our research agenda is to improve the decision-making performance of the framework by exploiting log data and deriving quantitative relations among indicators and parameters through. Then, faster quantitative optimization approaches will be applicable.

5. EVALUATION

This section explains how the *Qualia+* mechanism works through the Meeting Scheduler case study we introduced in Section 2. Then we demonstrate several cases that the older version of *Zanshin* wouldn't be able to handle as effectively as the extended one does.

5.1 Meeting Scheduler Case Study

The first step for building an adaptive system that will be managed by our proposed framework is to perform system identification and elicit the differential relations among *AwReqs* and the parameters of the system's goal model. From the goal model of the Meeting Scheduler depicted in Figure 1 we elicit a set of differential relations presented in Table 3. We then apply AHP as discussed earlier and show the result in Table 4. The final step for having all the prerequisites for our frameworks input is to assign to each *AwReq* an *EvoReq* operation to be executed in case it cannot be fixed by changing a parameter value due to the presence of conflict(s). For our case study the assigned *EvoReq* operations are listed in Table 5.

$$order\,(RF) : listonly \prec short \prec full \tag{2}$$
$$order\,(VP2, AR10) : partner \prec hotel \prec local \tag{3}$$

$$\Delta\,(I_1/RF) < 0 \tag{4}$$
$$\Delta\,(I_2/RfM) < 0 \tag{5}$$
$$\Delta\,(I_2/VP2) < 0 \tag{6}$$
$$\Delta\,(I_3/FhM) < 0 \tag{7}$$
$$\Delta\,(I_4/RfM) > 0 \tag{8}$$
$$\Delta\,(I_4/VP2) > 0 \tag{9}$$
$$\Delta\,(I_5/MCA) > 0 \tag{10}$$
$$\Delta\,(I_5/VP3) < 0 \tag{11}$$
$$\Delta\,(I_6/RF) > 0 \tag{12}$$
$$\Delta\,(I_6/FhM) > 0 \tag{13}$$
$$\Delta\,(I_6/VPA)\,\{false \rightarrow true\} > 0 \tag{14}$$
$$\Delta\,(I_6/MCA) < 0 \tag{15}$$
$$\Delta\,(I_6/VP1) < 0 \tag{16}$$
$$\Delta\,(I_6/VP3) < 0 \tag{17}$$
$$\Delta\,(I_7/VPA)\,\{false \rightarrow true\} < 0 \tag{18}$$
$$\Delta\,(I_8/RfM)\,[0, enough] > 0 \tag{19}$$
$$\Delta\,(I_8/VP2) > 0 \tag{20}$$
$$\Delta\,(I_9/MCA) > 0 \tag{21}$$
$$\Delta\,(I_9/VP3) > 0 \tag{22}$$
$$\Delta\,(I_{10}/RF) < 0 \tag{23}$$
$$\Delta\,(I_{10}/FhM) < 0 \tag{24}$$
$$\Delta\,(I_{10}/VP1) > 0 \tag{25}$$
$$\Delta\,(I_{10}/VP2) > 0 \tag{26}$$
$$\Delta\,(I_{10}/VP3) > 0 \tag{27}$$

Table 3: Differential relations elicited for the Meeting Scheduler example [18]

AwReq	**Priority Value**
AR6	1.63
AR5	1.58
AR4	1.17
AR1	1.09
AR8	0.93
AR2	0.8
AR3	0.7
AR7	0.76
AR9	0.64
AR10	0.6

Table 4: Priority Values of AwReqs

In our previous work [18] we state that some *EvoReq* operations can act either at instance level or class level. For example, when a requirement R is replaced by a requirement R' at an instance level, it means that future runs of the base system will use R, not R'. On the other hand, a class-level change means that subsequent executions of the base system will see only R'. As presented in Table 5, $AR2$, $AR3$, $AR6$, $AR8$ and $AR10$ can be replaced by other requirements with weaker quality constraints. For example, for *Good participation*, instead of expecting 90% participation we could lower expectations to 80%. For the *AwReqs* $AR1$, $AR4$ and $AR5$ we don't weaken requirements but rather postpone dealing with them, using the Retry(time) operation. For AR7 and AR9 we use the *EvoReq* operations Warning() and Abort() respectively. The first one prints a warning message and the second one suspends the requirement altogether.

Now that all the required input for *Zanshin* has been specified we present a case of multiple failures and how these are resolved. The adaptation requirements for the framework are those presented in Figure 2 and the predefined value of $VP1$ is *Adapt Optimistically*. The monitor component checks periodically every 1 hour if there are any failures. In the first scenario the monitor detects failures of $AR1$, $AR2$. Then the Failure Manager collects the parameters than can tune failing indicators. According to Table 3 for $AR1$ the only option is to decrease RF (required fields to organize a meeting) and for $AR2$ either decrease RfM (Rooms for Meetings available) or decrease $VP2$. There are though priority conflicts with non-faillling *AwReqs* a) $AR1$ conflicts with $AR6$ and b) $AR2$ conflicts with $AR4$ and $AR8$. The Decision-Maker takes into account the adaptation goal *Adapt Optimistically* and ignores the priority conflicts. Therefore, the Strategy Executor composes an adaptation strategy that executes two operations $decrease(RF)$ and $decrease(RfM)$. The framework will wait for the effects to take place and then examines if the indicators still need improvement. The previous *AwReqs* are not failing anymore, but $AR4$ and $AR8$ are adversely affected as they are now failing. Moreover, the Adaptation Manager because of these new failures switches to *Adapt with Compensation*. The Decision Manager then decides to increase the parameter RfM to improve $AR4$ and $AR8$ and executes the assigned *EvoReq* operation for $AR2$ since it has the lowest priority and no other parameter to be reconfigured. The result is a new strategy with the operations $increase(RfM)$ and $Replace(AR2, AR2\_200euro)$. The outcome is that the new $AR2$ is not failing anymore and $AR4$ and $AR8$ are not failing.

AwReq	***EvoReq* operation**
AR1	Retry(50000ms)
AR2	Replace(AR2,AR2_200euro)
AR3	Replace(AR3,AR3_14d)
AR4	Retry(1day)
AR5	Retry(10000)
AR6	Replace(AR6,AR6_80%prt)
AR7	Warning()
AR8	Replace(AR8,AR8_14d)
AR9	Abort()
AR10	Replace(AR10,AR10_3days)

Table 5: Evoreq operations for AwReqs

5.2 Improved Adaptation

The previous version of *Zanshin* and *Qualia* adaptation mechanism were ignoring the fact that there are cases where multiple failures cannot be handled independently by treating individually and sequentially indicators of failing *AwReqs*. To test our approach we implemented a simulation of the meeting scheduler example. The participants were implemented as agents with their own calendars and the rooms for the meetings registered to our implementation's database have the same characteristics with the meeting rooms of the University of Trento and its collaborating institutions. To test our system we initialize the calendars of every participant in order to create more realistic conditions. Then the user of the system has the role of the meeting organizer and initiates meeting requests specifying the room requirements and the participants. The participants may have conflicts with their own schedules or not. A monitoring mechanism is measuring every indicator after every meeting and updates it's status as failing or healthy. For comparison reasons we ran the same simulations using *Qualia* and *Qualia+* and below we demonstrate some of the failing scenario that took place and how each algorithm dealt with them.

Scenario 1: The monitor detects failures for $AR4$ and $AR8$.

***Qualia*:** The framework will treat first the failure that was detected first, in this case $AR4$. The parameter RfM is increased due to the differential relation (7) of Table 3. If after the change $AR4$ is not failing a new adaptation session starts for $AR8$ increasing $VP2$ due to the differential relation (18). When $AR8$ ceases to fail as a consequence of the parameters increment $AR2$ fails because of the negative impact the changes (differential relations (4) and (5)). Then *Qualia* would decrease again either RfM or $VP2$ (or both) in order for $AR2$ to recover. It is obvious that such an adaptation mechanism will fall into an infinite loop doing and undoing the same changes.

***Qualia+*:** The framework is instructed how to adapt in these cases using *AdReqs* as described in the example of the previous subsection.

Scenario 2: The monitor detects failures for $AR5$, $AR6$ and $AR9$.

***Qualia*:** The framework treats again the failures sequentially changing parameters that would result again in an infinite loop.

***Qualia+*:** The Failure Manager finds MCA (maximum conflicts allowed among the participants' time-schedules) as only available parameter which means that the Decision Maker has to choose between increasing MCA to improve $AR5$ and $AR9$ and worsen $AR6$ or decrease MCA to improve $AR6$ and worsen $AR5$ and $AR9$. The choice is based on the priority values of the indicators shown in Table 4 and since (1.58+0.64>1.63) AR5 and AR9 are preferred. The finally result would be a strategy with the operations: a) increase(MCA) and b) $\textit{Replace}(AR6, AR6\_80\%prt)$. This way we compensate for not improving *AwReq* while meeting adaptation requirements.

We note that adaptation strategies could have also been composed defining rules that specify in what order should *EvoReqs* operations be executed and under what circumstances. However, such rules are outside the scope of requirements and hard for stakeholders to conceptualize, and therefore define. *Qualia+* allows stakeholders to define the policies by which such conflicts should be resolved during the adaptation process. Moreover, the dynamic composition of adaptation strategies means that the adaptation process does not need to go off-line when adaptation requirements are changed.

6. RELATED WORK

In the field of software-adaptation a variety of approaches have been proposed to handle multiple objectives that need to be achieved. First, Rainbow[6] exploits Utility Theory to perform trade-offs among conflicting objectives maximizing the overall utility of the system. In the context of the same work, improved adaptation strategies are proposed using the concept of preemption [13]. The revised adaptation mechanism schedules better available strategies and can interrupt them when needed if new failures, more important, come into play. These strategies intended to be based on human experience and expertise of a control-based process. Consequently, there is no guarantee that after the addition or removal of one or more requirements they will still be effective. On the other hand, the basic idea of our work is to automate the composition of the adaptation strategies based on stakeholder requirements and preferences.

The work of Kramer and Magee [9] proposes another Architecture-based solution for self-adaptive systems. Adaptations in this framework are determined at three layers. The bottom layer, called *Component Control*, monitors the base-system and detects failing components. When a failure is noticed a request for adaptation actions is sent to the upper layer, called *Change Management*. This layer is responsible for an adaptation plan that will reconfigure and restore the failing system. If a plan is not available then the top layer, called *Goal Management* is invoked to produce new adaptation plans when new goals are introduced to the system. In this work, as in ours, adaptation strategies are composed dynamically. Our proposal attempts to go farther by exploiting requirements models that prescribe not only how the base-system should operate but also how it should adapt, even if there are no available adaptation actions, through *EvoReqs*. On the other hand architecture models facilitate software adaptation because they offer adaptation options that are simply not there in the requirements models we use. In our previous work [1] we advocate the combination of Requirements-based and Architecture-based approaches to better support adaptation. This is also part of our research agenda.

Another Requirements-based approach is *RELAX* [23], that captures uncertainty declaratively with modal, temporal and ordinal operators applied over *SHALL* statements (e.g.,"the system *SHALL ... AS CLOSE AS POSSIBLE* to ..."). A similar approach, but based on the goal-oriented language *KAOS* [3], is FLAGS [2]. This approach extends the linear temporal logic (LTL) used in *KAOS* with fuzzy relational and temporal operators, allowing some goals to be satisfied even if values are close to but not equal to the desired ones. *FLAGS* also proposes an operationalization of its models in a service-oriented infrastructure. Although *FLAGS* deals with conflict resolution and synchronization during the adaptation process as our work does, the differential relations, the prioritized requirements and *AdReqs* offer a more pragmatic basis for stakeholders to express their requirements on the adaptation process.

Multiple failures are also handled by Vromant et al in [22]. This work adopts and extends the Monitor-Analyze-Execute-Plan (MAPE) adaptation model focusing on interacting feedback loops. The baseline exemplar is a traffic monitoring system that is composed of several groups of agent-cameras that depend on each other to operate and provide accurate results. When several agents are not operating properly, several adaptation loops are being executed and co-ordinated in order to produce effective adaptation plans. Even though the approach presents an interesting way to adaptation mechanism to schedule adaptation actions, the focus is only on detecting which agents are not responding and how the the nodes of the system should be regrouped so that the system will keep providing the required service. In other words the dependencies are among system components and not the requirements as in our case.

Another Requirements-based and control theoretic approach is presented in [12]. In this work the authors implement a PID controller that monitors the set values of the satisfaction of goal model's softgoals and when the output declines finds a different configuration over the goal model. A preference ranking is used to assist the trade-off process among conflicting softgoals. This configuration is a result of reasoning supported by the OpenOME tool [16]. The implementation of the PID controller enhances the precision to the adaptation process, however the approach takes into account only failing softgoals. Moreover, there is no consideration about how lower priority failures are to be treated, while in our proposal we introduce the concept of compensation to achieve that. Nevertheless, this work is closest to the goal-oriented requirements framework we are using here.

7. CONCLUSIONS

The main contribution of this paper is a requirements-based adaptation mechanism that can handle multiple concurrent requirements failures. To accomplish this, we have extended the *Zanshin* framework with two basic new features. The first is the concept of *AdReqs*, which are requirements about the adaptation process itself. Like all requirements, these come from the stakeholders and define policies the adaptation process has to comply with. The second feature is a decision making mechanism that takes into account *AdReqs* to decide which requirements should be improved and which have to be compensated temporarily or permanently by *EvoReqs* operations. The new adaptation mechanism, called *Qualia+*, makes it possible for stakeholders to reflect their needs and preferences for the adaptation process by assigning priorities and compensation operations to base system requirements. Moreover, the fact that adaptation strategies are composed dynamically allows stakeholders to change *AdReqs* during system operation.

Our proposal needs to be further evaluated to ensure its effectiveness on several case studies. Moreover, we plan to extend *Zanshin* so that it can learn during its operations quantitative relations between indicators and control parameters, to replace qualitative relations provided by requirements engineers and domain experts.

8. ACKNOWLEDGMENTS

This work has been supported by the ERC advanced grant 267856 “Lucretius: Foundations for Software Evolution” (April 2011 – March 2016, http://www.lucretius.eu).

9. REFERENCES

[1] K. Angelopoulos, V. E. S. Souza, and J. a. Pimentel. Requirements and architectural approaches to adaptive software systems: A comparative study. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS ’13, pages 23–32, Piscataway, NJ, USA, 2013. IEEE Press.

[2] L. Baresi, L. Pasquale, and P. Spoletini. Fuzzy Goals for Requirements-driven Adaptation. In *Proc. of the 18th IEEE International Requirements Engineering Conference*, pages 125–134. IEEE, 2010.

[3] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed Requirements Acquisition. *Science of Computer Programming*, 20(1-2):3–50, 1993.

[4] R. Dorf. *Modern control systems.* Pearson Prentice Hall, Upper Saddle River, NJ, 2008.

[5] J. Doyle and G. Stein. Multivariable feedback design: Concepts for a classical/modern synthesis. *Automatic Control, IEEE Transactions on*, 26(1):4–16, 1981.

[6] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. *Computer*, 37(10):46–54, 2004.

[7] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *Software, IEEE*, 14(5):67–74, 1997.

[8] J. Karlsson, C. Wohlin, and B. Regnell. An evaluation of methods for prioritizing software requirements, 1998.

[9] J. Kramer and J. Magee. A rigorous architectural approach to adaptive software engineering. *J. Comput. Sci. Technol.*, 24(2):183–188, 2009.

[10] A. Morse and W. Wonham. Status of noninteracting control. *Automatic Control, IEEE Transactions on*, 16(6):568–581, 1971.

[11] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[12] X. Peng, B. Chen, Y. Yu, and W. Zhao. Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. *Journal of Systems and Software*, 85(12):2707–2719, 2012.

[13] R. Raheja, S.-W. Cheng, D. Garlan, and B. Schmerl. Improving architecture-based self-adaptation using preemption. In *Proceedings of the First International Conference on Self-organizing Architectures*, SOAR’09, pages 21–37, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] R. Saaty. The analytic hierarchy process—what it is and how it is used. *Mathematical Modelling*, 9(3–5):161 – 176, 1987.

[15] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2–3):313 – 322, 2003.

[16] R. Sebastiani, P. Giorgini, and J. Mylopoulos. Simple and minimum-cost satisfiability for goal models. In A. Persson and J. Stirna, editors, *16th International Conference on Advanced Information Systems Engineering*, volume 3084 of *Lecture Notes in Computer Science*, pages 20–35, Riga, Latvia, June 2004.

[17] S. Skogestad. *Multivariable feedback control : analysis and design.* John Wiley, Hoboken, NJ, 2005.

[18] V. Souza, A. Lapouchnian, K. Angelopoulos, and J. Mylopoulos. Requirements-driven software evolution (online first). *Computer Science - Research and Development*, pages 1–19, 2012.

[19] V. E. S. Souza. *Requirements-based Software System Adaptation.* Phd thesis, University of Trento, Italy, 2012.

[20] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos. System Identification for Adaptive Software Systems: a Requirements Engineering Perspective. In M. Jeusfeld, L. Delcambre, and T.-W. Ling, editors, *Conceptual Modeling – ER 2011*, volume 6998 of *Lecture Notes in Computer Science*, pages 346–361. Springer, 2011.

[21] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos. Requirements-Driven Qualitative Adaptation. In R. Meersman et al., editors, *On the Move to Meaningful Internet Systems: OTM 2012*, volume 7565 of *Lecture Notes in Computer Science*, pages 342–361. Springer, 2012.

[22] P. Vromant, D. Weyns, S. Malek, and J. Andersson. On interacting control loops in self-adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS ’11, pages 202–207, New York, NY, USA, 2011. ACM.

[23] 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*, 15(2):177–196, 2010.