Requirements Eng (2015) 20:233–252
DOI 10.1007/s00766-015-0227-1

RE 2014

Rationalism with a dose of empiricism: combining goal reasoning and case-based reasoning for self-adaptive software systems

Wenyi Qian[1,2] · Xin Peng[1,2] · Bihuan Chen[1,2] · John Mylopoulos[3] · Huanhuan Wang[1,2] · Wenyun Zhao[1,2]

Received: 20 October 2014 / Accepted: 6 April 2015 / Published online: 14 May 2015

Abstract Requirements-driven approaches provide an effective mechanism for self-adaptive systems by reasoning over their runtime requirements models to make adaptation decisions. However, such approaches usually assume that the relations among alternative system configurations, environmental parameters and requirements are clearly understood, which is often not true. Moreover, they do not consider the influence of the current configuration of an executing system on adaptation decisions. In this paper, we propose an improved requirements-driven self-adaptation approach that combines goal reasoning and case-based reasoning. In the approach, past experiences of successful adaptations are retained as adaptation cases, which are described by not only requirements violations and contexts, but also currently deployed system configurations. The approach does not depend on a set of original adaptation cases, but employs goal reasoning to provide adaptation solutions when no similar cases are available. Case-based reasoning is used to provide more precise adaptation decisions that better reflect the complex relations among requirements violations, contexts, and current system configurations by utilizing past experiences. To prevent case-based reasoning from getting trapped in suboptimal adaptation solutions, an additional case mutation mechanism is introduced to mutate existing adaptation solutions when necessary. We conduct an experimental study with an online shopping benchmark to evaluate the effectiveness of our approach. The results show that our approach outperforms both a requirements-driven approach and a case-based approach in terms of satisfaction level of quality constraints. The results also confirm the effectiveness of case mutation for producing better adaptation solutions. In addition, we empirically investigate the evolution process of adaptation solutions. The evolution analysis reveals some general evolution trends of adaptation solutions such as different evolution phases.

Keywords Self-adaptive systems · Requirements goal models · Goal reasoning · Case-based reasoning

✉ Xin Peng
pengxin@fudan.edu.cn

Wenyi Qian
qianwy@fudan.edu.cn

Bihuan Chen
bhchen@fudan.edu.cn

John Mylopoulos
jm@disi.unitn.it

Huanhuan Wang
huanhuanwang13@fudan.edu.cn

Wenyun Zhao
wyzhao@fudan.edu.cn

1 School of Computer Science, Fudan University, 825 Zhangheng Rd, Shanghai, China

2 Shanghai Key Laboratory of Data Science, Fudan University, 825 Zhangheng Rd, Shanghai, China

3 Department of Information Engineering and Computer Science, University of Trento, Via Calepina, 14, 38122 Trento, Italy

1 Introduction

A self-adaptive system can switch among alternative system configurations so that it can continue to satisfy stakeholder requirements (goals) in a changing and uncertain runtime environment. Requirements-driven approaches [3, 12, 28, 31, 36] provide an effective mechanism for such self-adaptive systems by reasoning over their runtime requirements models [15, 22, 32] to make adaptation decisions.

However, such approaches assume that the requirements of a self-adaptive system are well understood so that they can be used as basis for adaptation. Specifically, it is assumed that the relations between alternative system configurations, environmental parameters and requirements are clearly understood and modeled. Given these models, the adaptation mechanism can reason about alternative system configurations that can work best when requirements or contexts (environmental parameters) change at runtime.

The above assumption, however, is often simply not true when the system is complex and the relations between system configurations, contexts and requirements are not specified precisely. Moreover, an appropriate adaptation decision depends not only on requirements and context changes but also on the configuration currently deployed by the system. In that sense, there can be very complex combinations of requirements violations, contexts and currently deployed system configurations, each of which may demand a different adaptation solution.

By contrast, case-based reasoning (CBR) is able to utilize specific knowledge of previously experienced, concrete problem situations (cases) instead of relying solely on general knowledge [1]. Its underlying idea is that new problems can be solved by reusing successful solutions adopted in the past. This makes CBR a promising problem solving paradigm for problems that are not well understood or ones where relevant knowledge is hard to codify. There have been several attempts to apply CBR to self-adaptive systems [19, 23, 25]. In such approaches, cases are described by low-level symptoms (symbolic or numeric variables) characterizing encountered problems (e.g., requirements failures). Moreover, such approaches assume the availability of a set of initial cases that have been constructed manually.

In this paper, we propose an improved requirements-driven self-adaptation approach that combines goal reasoning and case-based reasoning. In our proposal, past experiences of successful adaptations are retained as adaptation cases and stored in a case base. Adaptation problems in adaptation cases are described not only by requirements violations (a set of quality attributes) and contexts (a set of environmental parameters), but also by currently deployed system configurations. Based on a predefined requirements goal model, both the current system configuration and the adaptation solution (i.e., new configuration to be adopted after adaptation) are specified by goal configurations consisting of specific alternatives for variation points and values for control variables. Such a goal-oriented case representation provides a richer and higher-level representation of adaptation cases than those of earlier proposals. Instead of depending on a set of original adaptation cases, our approach employs goal reasoning to provide adaptation solutions when no similar cases are available. Case-based reasoning, on the other hand, provides more precise adaptation decisions that better reflect the complex relations among requirements violations, contexts and current system configurations by utilizing past experiences (adaptation cases). To prevent the case-based reasoning mechanism from getting trapped in suboptimal adaptation solutions, our approach introduces a mutation operation that mutates existing adaptation solutions when necessary.

To evaluate the effectiveness of our approach, we have conducted an experimental study with an online shopping benchmark. We compare our approach with a requirements-driven approach and a case-based reasoning approach. The results show that our approach outperforms both the requirements-driven approach and the case-based approach in terms of satisfaction level of quality constraints. The results also confirm the effectiveness of case mutation for producing better adaptation solutions. In addition, we have analyzed empirically the evolution process of adaptation solutions of the same kind of adaptation problems to better understand how our approach actually works. The evolution analysis reveals some general evolution trends of adaptation solutions such as different evolution phases.

In a preliminary version of this work [29], we demonstrated the effectiveness of the combination of goal reasoning and case-based reasoning for requirements-driven self-adaptation. In this paper, we extend our earlier work along three directions. First, we improve the case revise mechanism (i.e., one of the four steps performed during case-based reasoning) with case mutation to prevent case-based reasoning traps in suboptimal adaptation solutions. Second, we conduct an extended experimental study, which not only evaluates the effectiveness of our approach but also empirically investigates the evolution process of adaptation solutions, and the role of case mutation during the self-adaptation process. Third, we provide a more comprehensive discussion on related issues, including interesting insights and shortcomings of our approach.

The rest of the paper is structured as follows. Section 2 introduces required preliminaries, including parameterized goal models and case-based reasoning. Section 3 presents the proposed self-adaptation approach. Section 4 evaluates the proposed approach using an online shopping system and empirically investigates the evolution process of adaptation solutions. Section 5 discusses fundamental characteristics and assumptions of our work. Section 6 reviews a number of existing proposals and compares them with ours. Finally, Sect. 7 concludes the paper and outlines future work.

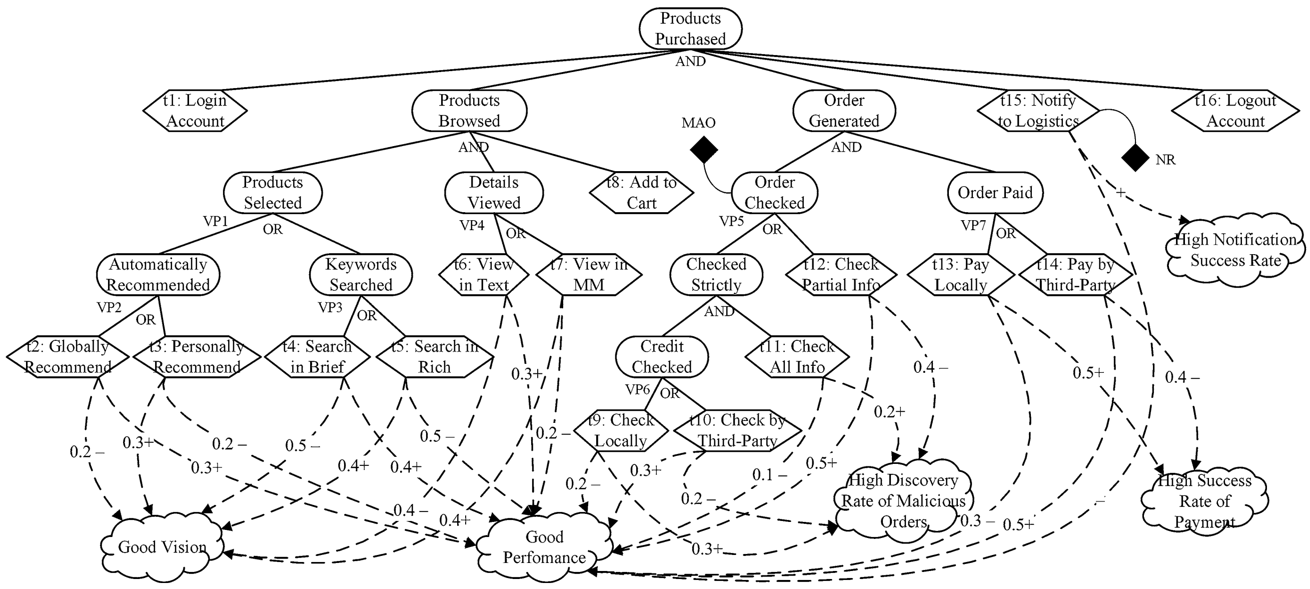

Fig. 1 The parameterized goal model of an online shopping system

2 Preliminaries

In this section, we briefly introduce the required preliminaries, i.e., parameterized goal models and case-based reasoning.

2.1 Parameterized goal models

In goal-oriented requirements analysis, functional requirements are modeled as *hard goals*, while quality requirements are modeled as *softgoals* [27]. In a requirements goal model, goals are refined iteratively into subgoals through AND/OR *decomposition links* until we reach goals that are simple enough to be fulfilled by *tasks* carried out by software or human agents. To satisfy an AND/OR-decomposed goal, all/at least one of its subgoals must be satisfied. Furthermore, goals can be related to each other through quantitative *contribution links* $w+$ and $w-$ where the weight w is between 0 and 1 and $+/-$ indicates that the satisfaction of the source goal contributes to the satisfaction/denial of the target goal [15].

In a goal model, an OR decomposition represents a variation point for system configurations with its alternative subgoals as variants. When used for runtime adaptation of software systems, all the non-system alternative subgoals (i.e., those subgoals that need to be accomplished by human agents) of an OR-decomposed goal are removed and all the remaining subgoals are treated as exclusive alternatives at runtime. This means that at runtime exactly one alternative subgoal of an OR-decomposed goal can be chosen at the same time. Recently, control variables were introduced as attachments to goals or tasks to represent variables that influence the fulfillment of a goal, or the execution of a task [33]. Control variables provide another mechanism for reconfiguration. Goal variation points and control variables are together referred to as parameters, while goal models with variation points and control variables are called parameterized goal models. Here, a goal configuration that satisfies a root-level goal consists of choosing one of the alternatives for every variation point, and also choosing a value for every control variable. Alternative configurations of a system at runtime can then be represented by alternative goal configurations.

Figure 1 shows the parameterized goal model of an online shopping system with seven variation points and two control variables. Hard goals, softgoals and tasks are shaped as rounded rectangles, clouds and hexagons, respectively. In more detail, *Products Selected*, *Automatically Recommended*, *Keywords Searched*, *Details Viewed*, *Order Checked*, *Credit Checked* and *Order Paid* are OR-decomposed goals. Each task is annotated with a symbol for the sake of reference simplicity. Products could be automatically recommended globally (*t2*, according to the sales of products) or personally (*t3*, according to the user's purchase history). When a user searches for products by keyword, search is done in a simple way (*t4*, search for the description of a product through string matching) or in a more sophisticated way (*t5*, search by taking into account customer preferences, extracted from the customer purchase history). The details of products could be viewed in textual mode (*t6*, view only descriptive text) or in multimedia mode (*t7*, view text and pictures). *t2*, *t4* and *t6* help improve the performance but hurt vision enhancement, while *t3*, *t5* and *t7* contribute to these softgoals in the opposite way. An order can be simply checked according to partial information (*t12*, only check the total amount of the order) or fully checked (*t11*, check not only total amount, but also each item in the order, as well as the customer credit). Customer credit can be checked locally based on the customer purchase history (*t9*, rate of customer unpaid or canceled orders) or checked by a third-party online banking system (*t10*). *t9* helps the discovery of malicious orders but penalizes system performance, while *t10* and *t12* contribute to these softgoals in the opposite ways. An order may be paid locally (*t13*, use payment services provided by the shopping system) or by third party (*t14*, use a third-party online banking system for payment). The former helps the success rate of payments but penalizes system performance, while the latter contributes to these softgoals in the opposite ways. The control variable *MAO* represents the minimum amount of money for an order to be checked before further processing, while control variable *NR* represents the number of retries to be attempted if a notification to logistics fails. A candidate goal configuration that satisfies *Products Purchased* is [*t4*, *t6*, *t9*, *t13*, 200 (*MAO*), 2 (*NR*)].

2.2 Case-based reasoning

Case-based reasoning (CBR) is a problem solving paradigm, based on the reuse of previous experience. This experience is represented in the form of cases that capture problem situations along with the outcomes of applying a particular solution [1]. CBR is well suited for problems where requirements and domain knowledge are unavailable. Furthermore, cases can be accumulated in a case base and for a new problem, similarity is used to select from the case base which cases to apply to. Figure 2 shows the CBR process, which comprises four steps, i.e., *Retrieve*, *Reuse*, *Revise* and *Retain*.

Specifically, *Retrieve* retrieves cases whose problems are similar to the problem-at-hand, using a matching operation. Based on retrieved similar cases, *Reuse* generates a solution to the problem-at-hand, for example, by calculating averages of the solutions of similar cases. Then through *Revise*, the solution is applied and outcomes are observed. If the solution fails, it may be repaired using domain-specific knowledge. Finally, *Retain* adds the new case into the case base for future reuse.

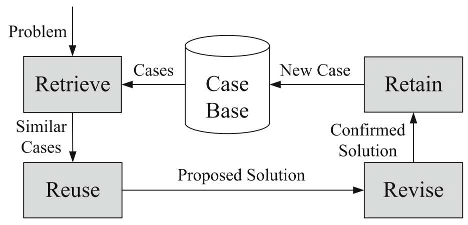

Fig. 2 The process of case-based reasoning

3 Our approach

This section first presents an overview of the proposed approach and then details the underlying techniques used for goal reasoning, and adaptation case representation, retrieval, reuse, revision and retention.

3.1 Overview

We propose an improved requirements-driven self-adaptation approach that combines goal reasoning and case-based reasoning. The objective of our approach is to maximize the satisfaction of quality constraints, which depend on quality attributes that are derived from softgoals during requirements analysis. For example, one quality constraint for the online shopping system (Fig. 1) is that response time should be less than 1000 ms, whose satisfaction depends on the quality attribute response time for softgoal Good Performance.

An adaptation case base is used to store past experiences of successful adaptations, and is initially empty. Our approach takes as input a parameterized goal model. That model is used to provide the representation for adaptation cases and to reason about adaptation solutions when no similar past cases can be found.

Figure 3 presents the adaptation process for our approach, showing the main steps and their corresponding inputs/outputs. The process is a variant of the MAPE-loop used for autonomic systems [17]. The four gray rectangles in the figure correspond to the four main tasks of case-based reasoning, i.e., *Retrieve*, *Reuse*, *Revise*, *Retain*.

Given a base system, i.e., a software system without self-adaptation mechanisms, our approach augments it with an adaptation feedback loop that is periodically executed. In each adaptation process, quality attributes (e.g., response time, success rate, cost) and environmental parameters (e.g., access load, CPU/memory usage) of the system are monitored and analyzed based on collected runtime data. Notice that *Monitor* and *Analyze* are not the main focus of this paper, so we have adopted log analysis techniques for

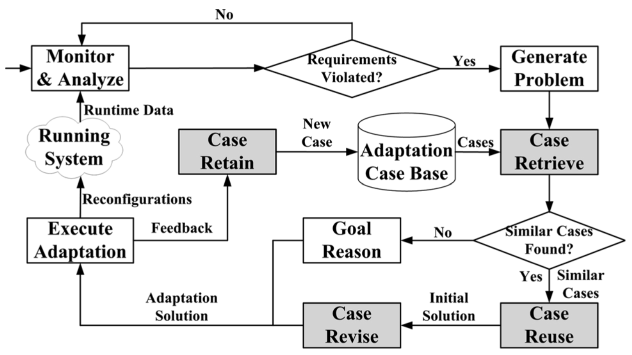

Fig. 3 Approach overview

simplicity. For larger and more complex software systems, more efficient runtime monitoring and analysis techniques such as SIENA [6] and pulse monitoring [16] can be adopted. Using such techniques, our adaptation mechanism evaluates quality attributes and determines whether some requirements are violated, for example, quality constraints are violated or desired quality deteriorates. If requirements violations are detected, the adaptation phase is triggered ('Generate Problem' in Fig. 3), with inputs the current configuration, values of quality attributes and context.

To choose an adaptation, our approach searches for similar adaptation cases from the case base. If similar cases are found, an initial adaptation solution (i.e., a goal configuration) is generated by the case reuse mechanism based on retrieved cases. To avoid the adaptation being trapped in suboptimal adaptation solutions, the initial adaptation solution will be mutated with a certain probability as a means of case revision, which could produce a new solution that has not been tried for similar adaptation problems before. If no similar cases are found, the goal reasoning is used to produce an adaptation solution by reasoning about optimal goal configurations based on the goal model. Based on the adaptation solution produced by case reuse or goal reasoning, the base system is reconfigured. Notice that the *Execute Adaptation* activity is not the main focus of this paper, and such architectural reconfigurations, including replacing, unbinding and binding a component, are currently supported by service-oriented architectures or reflective component models such as Fractal [5].

Once an adaptation is operationalized, changes in quality attributes are observed and constitute feedback. Based on this feedback, the effect of the adaptation is evaluated by the case retain mechanism. If the adaptation is successful, i.e., quality constraints are now satisfied, the adaptation solution is confirmed and a new case is created for the current adaptation and retained in the case base.

3.2 Case representation

Each case represents a successful adaptation where a reconfiguration of the system worked in a specific situation (requirements violations, context, currently deployed configuration). In case-based reasoning, a case usually includes three major parts [21]: the *problem-situation description*, the state of the world and the problem needing to be solving; the *solution*, the stated or derived solution to the problem; the *effect*, the resulting state of the world when the solution was carried out. In our approach, the solution part of an adaptation case specifies a switch to an alternative goal configuration. The effect part of an adaptation describes the outcome of an adaptation case according to the improvement in quality attributes. Therefore, an adaptation case can be represented as follows:

$$Case = \langle Problem, GoalConfig_{\mathrm{new}}, Quality_{\mathrm{new}} \rangle \tag{1}$$

3.2.1 Adaptation problem

As specified in the following equation, the problem description (*Problem*) of an adaptation case consists of the situation of requirements violations ($Quality_{\mathrm{cur}}$) and context ($Context_{\mathrm{cur}}$), and an additional description of the goal configuration currently deployed ($GoalConfig_{\mathrm{cur}}$) when an adaptation problem is raised.

$$Problem = \langle Quality_{\mathrm{cur}}, Context_{\mathrm{cur}}, GoalConfig_{\mathrm{cur}} \rangle \tag{2}$$

In the problem description, $Quality_{\mathrm{cur}}$ describes requirements violations in terms of quality attributes, each corresponding to a softgoal. For each quality attribute, a value is aggregated based on the runtime data collected during a fixed adaptation interval (e.g., 30 s). For example, for softgoal *Good Performance* in Fig. 1, a value of average response time can be aggregated and assigned to it. $Context_{\mathrm{cur}}$ describes a context in terms of environmental parameters. For example, the runtime contexts of a Web system are described by CPU usage, memory usage and access load. $GoalConfig_{\mathrm{cur}}$ denotes the current configuration of the system when requirements violations occur. For example, according to the goal model in Fig. 1, a description of $GoalConfig_{\mathrm{cur}}$ consists of an alternative task chosen for *Products Selected*, *Automatically Recommended*, *Keywords Searched*, *Details Viewed*, *Order Checked*, *Credit Checked* and *Order Paid*, respectively, and a value for *MAO* (minimum amount of order) and *NR* (number of retries), respectively.

3.2.2 Adaptation solution

The solution part ($GoalConfig_{\mathrm{new}}$) of an adaptation case specifies a new configuration of the system. Therefore, it is described in the same way as the current configuration ($GoalConfig_{\mathrm{cur}}$), i.e., by a set of alternative goals/tasks and control variable values.

3.2.3 Adaptation effect

The effect description ($Quality_{\mathrm{new}}$) of an adaptation case records the improvement of quality attributes after the adaptation is carried out. Therefore, it is described in the same way as the quality part ($Quality_{\mathrm{cur}}$) of an adaptation problem, i.e., by a set of quality attribute-value pairs.

3.3 Goal reasoning

Goal reasoning is used to provide adaptation solutions based on the given goal model when no similar cases are available. Intuitively, the adaptation mechanism can prevent a quality attribute (softgoal) from getting worse by increasing its weight (preference rank) such that the goal configurations increasing its satisfaction can be chosen [28]. To reflect the problem of requirements violations, the weights of related quality attributes are tuned dynamically. As a result, an adaptation solution will be generated based on the tuned weights through goal reasoning to try to ameliorate violated requirements. The procedure of weight tuning is described in Algorithm 1. The input of the algorithm is a set of quality attributes *QASet*. Each quality attribute is described using the following representation, where *value*, $cons_{\mathrm{u}}$, $cons_{\mathrm{l}}$, *weight* represent the current value, upper constraint (the value of the quality attribute should be lower than $cons_{\mathrm{u}}$), lower constraint (the value of the quality attribute should be higher than $cons_{\mathrm{l}}$) and current weight of the quality attribute. The upper (lower) constraint can be null if the quality attribute has no upper (lower) constraint. The weights of all quality attributes are initiated to 1 when the system is launched.

$$QualityAttribute = \langle value, cons_{\mathrm{u}}, cons_{\mathrm{l}}, weight \rangle \quad (3)$$

Algorithm 1 identifies quality attributes whose constraints are violated (*VioSet*). If some quality constraints are violated (i.e., *VioSet* is not empty), the weights of related quality attributes are increased by a predefined constant α (e.g., 20 %) so that an adaptation solution that favors such quality attributes can be generated. Otherwise, the current adaptation is triggered by the degradation of the combined utility, so the weight of a selected quality attribute with lowest weight is increased by α. Note that the weight of each quality attribute (corresponding to a softgoal) will be normalized into a relative weight to be used in goal reasoning.

Algorithm 1 Weight Tuning

```
 1: procedure WEIGHTTUNING($QASet$)
 2:     $VioSet = \emptyset, min = +\infty$
 3:     for each $q \in QASet$ do
 4:         if $q.value > q.cons_u \mid\mid q.value < q.cons_l$ then
 5:             $VioSet = VioSet \bigcup \{q\}$
 6:         else if $q.weight < min$ then
 7:             $min = q.weight$
 8:             $temp = q$
 9:         end if
10:     end for
11:     if $VioSet \neq \emptyset$ then
12:         for each $q \in VioSet$ do
13:             $q.weight = q.weight \times (1 + \alpha)$
14:         end for
15:     else
16:         $temp.weight = temp.weight \times (1 + \alpha)$
17:     end if
18: end procedure
```

To use control variables in goal reasoning, we map each task with a control variable onto an OR decomposition as shown in Fig. 4. For each *Task* with a control variable *CV*, three alternative *Task*(*CV*++), *Task*(*CV*−), *Task*(*CV*0) are created, representing the same *Task* but with increasing, decreasing and unchanged *CV*, respectively. For each of them, a contribution link is created to relevant softgoals. The contribution weights of *Task*(*CV*0) to a softgoal are always 0, indicating that there is no influence change on the softgoal if *CV* remains unchanged. If *Task*(*CV*++/–) is selected, *CV* will be increased/decreased with a predefined changing pace. If the value of *CV* goes beyond its value range after a change, the candidate configuration is discarded.

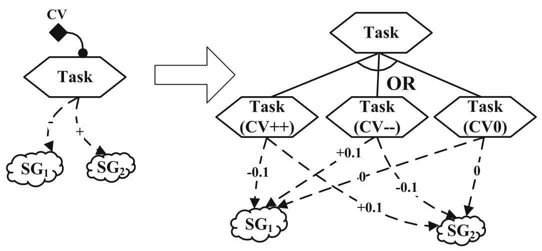

Fig. 4 Mapping of control variables onto OR-decompositions

For a candidate goal configuration, the satisfaction level of a softgoal can be calculated using label propagation algorithms [15]. The overall contribution of the goal configuration to all the softgoals is calculated as the weighted sum of the satisfaction levels of all the softgoals. The weight of a softgoal is the relative weight of its corresponding quality attribute calculated by using the following equation:

$$RelativeWeight(qa) = \frac{qa.weight}{\sum_{q \in QASet} q.weight} \quad (4)$$

The selected adaptation corresponds to the goal configuration with the highest overall contribution.

3.4 Case retrieval

In case-based reasoning, case retrieval is usually based on the similarity between a given problem and that of existing cases. Similarity between two problems (P1 and P2) is calculated as the Euclidean distance of description vectors VP1 and VP2 [1, 26]:

$$Distance(P_1, P_2) = \sqrt{\sum_{i=1}^{n} (diff(V_{P_1}[i], V_{P_2}[i]))^2} \quad (5)$$

The *diff* function in the above equation denotes the difference between the values on the *i*th dimension in P_1 and P_2. For a dimension corresponding to an enumeration type (e.g., alternative goal/task), the difference is 0 if the two values are the same and one otherwise. For a numerical dimension corresponding to a control variable, the difference is computed after normalization. The normalization of a numerical variable is calculated by $\frac{v-MIN}{MAX-MIN}$, where v is its current value, *MIN* and *MAX* are its minimal and maximal values it can assume during system operation.

Based on the case distance measurement, the case retrieval procedure (see Algorithm 2) identifies similar adaptation cases using two thresholds $threshold_h$ and $threshold_l$ ($threshold_l < threshold_h$). Adaptation problems whose distance is lower than $threshold_h$ are regarded as similar problems, while adaptation problems whose distance is lower than $threshold_l$ are regarded as the same kind of problems (i.e., very similar). The inputs of the algorithm are an adaptation problem *curProblem* and a set of adaptation cases *CaseSet*. If there are cases with the same kind of problem (similarity less than $threshold_l$), the most similar case (perfect case) is returned as the result of case retrieval. Otherwise, a set of sufficiently similar cases *SimCases* (similarity less than $threshold_h$) is returned.

Algorithm 2 Adaptation Case Retrieval

```
1:  function RETRIEVE($curProblem, CaseSet, SimCases$)
2:      $min = threshold_l, perfectCase = null$
3:      for each $case \in CaseSet$ do
4:          $dis = Distance(curProblem, case.Problem)$
5:          if $dis < min$ then
6:              $perfectCase = case$
7:              $min = dis$
8:          else if $dis < threshold_h$ then
9:              $SimCases = SimCases \bigcup \{case\}$
10:         end if
11:     end for
12:     return $perfectCase$
13: end function
```

3.5 Case reuse

The case reuse strategy depends on the results of case retrieval. If a perfect case is returned, the solution part of the returned case (i.e., a goal configuration) is directly adopted as the adaptation solution. Otherwise, our approach synthesizes an adaptation solution based on the returned similar cases.

To avoid chaos in combining many less similar cases, our approach only uses the two most similar cases. In addition, another goal configuration is generated using goal reasoning. The three configurations are then synthesized into an adaptation solution. The synthesis is done for each OR-decomposed goal and each control variable involved in the goal configurations. An OR-decomposed goal is synthesized by voting, i.e., alternative goals/tasks adopted by the most goal configurations are selected. If more than one alternative goals/tasks are selected, an alternative is randomly selected as the result. A control variable is synthesized by calculating the average of its values in different goal configurations.

3.6 Case revision

Similar to the local optimum problem in search algorithms, the adaptation process may be trapped in suboptimal adaptation solutions generated from existing similar cases: If a set of effective but not optimal adaptation solutions is generated for a kind of adaptation problem at first, new solutions will always be generated from these existing solutions and may remain suboptimal. In such cases, our adaptation process will continue to generate suboptimal solutions.

Inspired by the mutation operation in genetic algorithm- [35], we introduce case mutation as a means for improving the diversity of adaptation solutions for the same kind of adaptation problem. For an initial adaptation solution, each alternative goal/task or control variable value in its goal configuration is mutated with a probability p: An alternative goal/task is randomly changed to another alternative and a control variable value is randomly increased or decreased.

To avoid repeating failed adaptation solutions, the mutated adaptation solution is compared with past failed adaptation cases. If there is a past failed case with the same kind of adaptation problem and exactly the same adaptation solution, the mutated adaptation solution is discarded and case mutation is reperformed. The above process is repeated *MuTime* times till a mutated solution with no past failure is found or a maximum number of iterations is reached. If a mutated solution cannot be found, the initial adaptation solution is adopted.

Differently from genetic algorithm, we cannot evaluate the effectiveness of a mutated adaptation solution with a fitness function. The only way to precisely evaluate its effectiveness is to execute the mutated adaptation solution. Therefore, the mutated adaptation solution is executed and feedback is collected and evaluated by the case retain mechanism. Each mutated adaptation solution that is found not to be effective is recorded as a failed adaptation case and tagged as *Ineffective Mutation*.

The mutation probability in genetic algorithms can be either constant or dynamically estimated by the fitness value (i.e., effectiveness evaluation of solutions) [34], which is not applicable to our adaptation problem. Therefore, we design ourselves the mutation probability p, which is determined in an adaptive way based on the following observations. First, if similar adaptation problems have occurred many times and the past adaptation solutions make little improvement on related quality attributes, it is more likely that a different adaptation solution would need to be generated for achieving a more effective adaptation. Second, if case mutation has been tried many times for similar adaptation problems and all past mutations failed to generate a better solution, it is more likely that optimal adaptation solutions have been included in existing cases and there is no need to generate different ones.

The mutation probability p for an adaptation problem is calculated by the following equation:

$$p = k * \frac{1 - \frac{1}{\lg N + 1}}{1 + f} * \frac{1}{\lg(N_{\mathrm{IM}} + 1) + 1} \tag{6}$$

where

- k is the base mutation probability (i.e., the highest possible probability), which is a predefined constant between 0 and 1;
- N ($N \geq 1$) is the number of past cases with the same kind of adaptation problems;
- f is the average quality improvement ratio of the N past cases with the same kind of adaptation problems calculated as $\frac{Quality_{\mathrm{new}} - Quality_{\mathrm{cur}}}{Quality_{\mathrm{cur}}}$ (e.g., the improvement ratio of overall utility or response time);
- N_{IM} is the number of past cases that have the same kind of adaptation problems and are tagged as *Ineffective Mutation*.

The first observation is reflected by the influence of N and f to p, i.e., the mutation probability increases as more similar adaptation problems have occurred and less quality improvement has been made. The second observation is reflected by the influence of N_{IM} to p, i.e., the mutation probability decreases as more ineffective mutations have occurred for the same kind of adaptation problems.

3.7 Case retention

For an adaptation solution with the goal configuration *Goal-Config* for an adaptation problem *Problem*, the case retention mechanism evaluates its effectiveness based on the feedback from adaptation execution. It measures related quality attributes of the system during the adaptation period after the adaptation execution. The resulting situation of quality attributes $Quality_{\mathrm{new}}$ is then compared to the situation in the adaptation problem (i.e., $Problem.Quality_{\mathrm{cur}}$). If *Problem* involves violations of quality constraints, the effectiveness of the adaptation solution is judged by the following criterion: No new constraint violations are involved in $Quality_{\mathrm{new}}$ and all of the violated quality attributes in $problem.Quality_{\mathrm{cur}}$ have been improved. Otherwise, the effectiveness is judged by the improvement in the combined utility.

If the adaptation solution is evaluated to be effective, a new adaptation case is created using *Problem* as problem, *GoalConfig* as the solution and $Quality_{\mathrm{new}}$ as the effect. If no past cases with the same problem description exist in the case base, the new case is retained for future reuse, i.e., added to the case base. If there is a past case with the same problem description and the effect of the new case is better than that of the old one, the old case is replaced by the new case. Otherwise, the new case is discarded.

To facilitate the calculation of mutation probability and the evaluation of mutated adaptation solutions, we also create a failed adaptation case for each adaptation solution that is evaluated to be not effective. These failed adaptation cases are stored for case mutation decisions.

4 Experimental study

To evaluate the effectiveness of the proposed approach and empirically investigate the adaptation process, we conduct an experimental study to answer the following three research questions:

- *RQ1*: Can the proposed approach achieve improvement over self-adaptation approaches that involve only goal reasoning or case-based reasoning? (Sect. 4.2)
- *RQ2*: Can case mutation effectively improve the adaptation results? (Sect. 4.3)
- *RQ3*: How do the adaptation solutions for different kinds of adaptation problems evolve? (Sect. 4.4)

For *RQ1*, we compare our approach with adaptation approaches using only goal reasoning or case-based reasoning in terms of satisfaction of quality constraints. For *RQ2*, we evaluate the effectiveness of case mutation by analyzing the quality improvement achieved by mutated adaptation solutions. For *RQ3*, we empirically investigate the evolution process of adaptation solutions for different kinds of adaptation problems.

4.1 Experimental setup

Our experimental study is based on an online shopping system, whose goal model is shown in Fig. 1. In the experiments, twelve environmental parameters and five quality attributes are dynamically recorded into a log file when the system receives a request (e.g., login into the system). For each predefined time interval, the data in the log file are analyzed to obtain the monitored quality attributes and environmental parameters for facilitating runtime adaptations. Table 1 presents the environmental parameters considered in our experiments. These parameters are obtained from Linux's virtual memory statistics. The quality attributes (softgoals) of the online shopping system and their corresponding quality metrics are detailed as follows.

- The metric of softgoal *Good Performance* is response time, which is measured through log analysis as the average response time of each request within a

Table 1 Environmental parameters of the online shopping system

Context variables	Description
thread_num	The number of concurrent access threads
c_procs_r	The number of processes waiting for run time
c_procs_b	The number of processes in uninterruptible sleep
c_memory_swpd	The amount of virtual memory used
c_memory_free	The amount of idle memory
c_memory_buff	The amount of memory used as buffers
c_memory_cache	The amount of memory used as cache
c_io_bi	Blocks received from a block device (blocks/second)
c_io_bo	Blocks sent to a block device (blocks/second)
c_system_in	The number of interrupts per second, including the clock
c_system_cs	The number of context switches per second
c_cpu_id	Time spent idle of total CPU time

predefined time interval. For example, if there are five requests with response 100, 200, 150, 300 and 250 ms, then the average response time is 200 ms.

- The metric of softgoal *Good Vision* is user satisfaction, which is measured through a simulated real-life customer feedback analysis. Here we assume that user satisfaction for a product selecting request is a random value between 0.0 and 0.5 if the variation point *VP*1 in Fig. 1 is bound to *Globally Recommended* or *Search in Brief*, and between 0.5 and 1.0 if *VP*1 is bound to *Personally Recommended* or *Search in Rich*. Similarly, we assume that user satisfaction for a viewing request is a random value between 0.0 and 0.5 if the variation point *VP*4 is bound to *View in Text*, and between 0.5 and 1.0 if *VP*4 is bound to *View in MM*. Then the user satisfaction is the average of the user satisfaction of searching and viewing requests in an interval. For example, if *VP*1 and *VP*4 are bound to *Search in Brief* and *View in Text*, respectively, and there are two product selecting requests whose user satisfactions are 0.2 and 0.3 and two viewing requests whose user satisfactions are 0.1 and 0.2, then the user satisfaction is estimated as the average, i.e., 0.2.
- The metric of softgoal *High Discovery Rate of Malicious Orders* is the rate of identified malicious orders among all malicious orders. Here we assume that the alternative task *Check Locally* of *VP*6 can check out 100 % malicious orders, task *Check by Third Party* of *VP*6 can check out 80 % malicious orders, and task *Check Partial Info* of *VP*5 can check out 50 % malicious orders, and each order has a 30 % chance to be a malicious one. Note that the control variable *MAO* determines which orders will be checked. For example, suppose there are 100 orders and 30 of them are malicious orders. Among these, suppose 20 of them have amount higher than 1000 dollars. Given that *VP*5 is bound to *Check Partial Info* and *MAO* is set to 1000 dollars, the malicious order rate will be $\frac{20*0.5}{30} = 0.33$.
- The metric of softgoal *High Notification Success Rate* is the rate of successfully notified orders among all orders. For example, there are ten orders and eight of them are successfully notified to logistics, and then the notification success rate is 80 %. Note that a notification can be successfully sent after several retries and the control variable *NR* determine the maximum numbers of retries.
- The metric of softgoal *High Success Rate of Payment* is the rate of successful payments. Here we assume that success rate of payment is a random value between 0.5 to 0.75 if the variation point *VP*7 is bound to *Pay by Third Party*, and between 0.75 to 1.0 if the *VP*7 is bound to *Pay Locally*. For example, there are 100 orders paid, and 80 of them are paid successfully, and then the success rate of payment will be 80 %.

Among all the quality attributes, response time is associated with a quality constraint **Cons1**. For the other four quality attributes (i.e., user satisfaction, discovery rate of malicious orders, payment success rate, and notification success rate), we define a combined utility as the weighted sum of their normalized values (between 0 and 1) and associate it with a quality constraint **Cons2**. The weights of the four quality attributes are equally set to 0.25.

- **Cons1**: The average response time should be less than 1000 ms.
- **Cons2**: The combined utility should be larger than 0.55.

Based on the quality attributes monitored at each time interval, the adaptation mechanism determines whether an adaptation is required for requirements violations based on the following two conditions: **Cons1** is violated and **Cons2** is not violated; **Cons2** is violated and **Cons1** is not violated. As our adaptation mechanism is essentially based on dynamic tradeoff among different quality attributes, we simply ignore the condition where both **Cons1** and **Cons2** are violated. Once the adaptation condition is triggered, the adaptation mechanism plans an appropriate adaptation solution using the process presented in Sect. 3, and this solution is executed to reconfigure the system to improve related quality attributes. We say an adaptation is *effective* if the violated requirement (response time or utility) is improved and the satisfied requirement is not violated after adaptation.

As indicated earlier, the online shopping system has seven variation points and two control variables (Fig. 1). For the experiments, we set the ranges of the control variables *MAO* and *NR* being 0 to 2000 and 0 to 10, respectively, with their changing paces being 200 and 1, respectively. The variation points and control variables are initially configured to [*Automatically Recommended*, *Personally Recommend*, *View in MM*, *Checked Strictly*, *Check Locally*, *Pay Locally*, *1000 (MAO)*, *5 (NR)*].

To evaluate the effectiveness of the proposed approach, we conduct four experiments with the following four approaches, respectively, using the same experimental settings.

- *Goal-Based Approach*: Adaptation based on goal reasoning.
- *Case-Based Approach*: Adaptation based on case-based reasoning. Its initial case base is constructed by goal reasoning, i.e., retaining adaptation solutions produced by goal reasoning as cases. Its case reuse strategy is to select and reuse the most similar case.
- *Combined Approach*: Adaptation process that combines goal reasoning and case-based reasoning (i.e., the proposed approach without case mutation).
- *Improved Combined Approach*: The adaptation process proposed here, with case mutation.

In our experimental study, each experiment is executed for 5 h. The adaptation interval is set to 30 s, i.e., the adaptation mechanism is executed every 30 s. In the experiment with the *Case-Based Approach*, the initial training phase for the construction of the original case base takes 30 min. In the experiment with the *Combined Approach* and the *Improved Combined Approach*, the parameters are set as follows: The two case distance thresholds $threshold_{\mathrm{l}}$ and $threshold_{\mathrm{h}}$ are set to 0.8 and 1.2, respectively; and the weight tuning pace of quality attributes (i.e., α in Algorithm 1) is set to 20 %. Specially, for the *Improved Combined Approach*, the base mutation probability (i.e., k in Eq. 6) is set to 0.1, and the repeat times of case mutation (i.e., *MuTime*) is set to five.

All the four experiments are conducted on the same server with a 4-core 3.1 GHz CPU and 8 GB RAM. We use stress testing tool JMeter to simulate concurrent system accesses, and the same simulation script is applied to all the four experiments.

4.2 Comparative evaluation (RQ1)

We analyze the adaptation frequency and the quality attributes of the system during each experiment (i.e., in total 600 intervals) to compare the effectiveness of the four approaches.

Table 2 shows the number and effectiveness of the adaptations selected by using the four approaches. The first column lists the approaches being compared, the second column lists the number of adaptations, the third column lists the number of effective adaptations, and the fourth column lists the effectiveness rate of the adaptations.

We can observe that the *Goal-Based Approach* triggers 286 adaptations, and 101 of them are evaluated to be effective; the *Case-Based Approach* triggers 396 adaptations, and 56 of them are evaluated to be effective; the *Combined Approach* triggers 284 adaptations, and 130 of them are evaluated to be effective; and the *Improved Combined Approach* triggers 274 adaptations, and 158 of them are evaluated to be effective. The *Improved Combined Approach* triggers the least adaptations and achieves the highest effectiveness rate (i.e., 58 %). This shows that it will be less effective if the adaptation mechanism only utilizes one of the two types of reasoning.

Table 3 shows the satisfaction levels of the two constraints of the adaptations selected by using the four approaches. The first column lists the involved approaches. The second and third columns, respectively, list the number/percentage of intervals satisfying **Cons1** and **Cons2**. The fourth column lists the number/percentage of intervals

Table 2 Number and effectiveness of the adaptations selected by using the four approaches

Approach	Adaptation (#)	Eff. (#)	Rate (%)
Goal-based	286	101	35
Case-based	396	56	14
Combined	284	130	46
Improved combined	274	158	58

Table 3 Constraints satisfaction levels of the adaptations selected by using the four approaches

Approach	**Cons1** (%)	**Cons2** (%)	**Cons1**∧**Cons2** (%)
Goal-based	401/67	242/40	181/30
Case-based	485/81	123/20	107/18
Combined	408/68	259/43	193/32
Improved combined	453/76	332/55	254/42

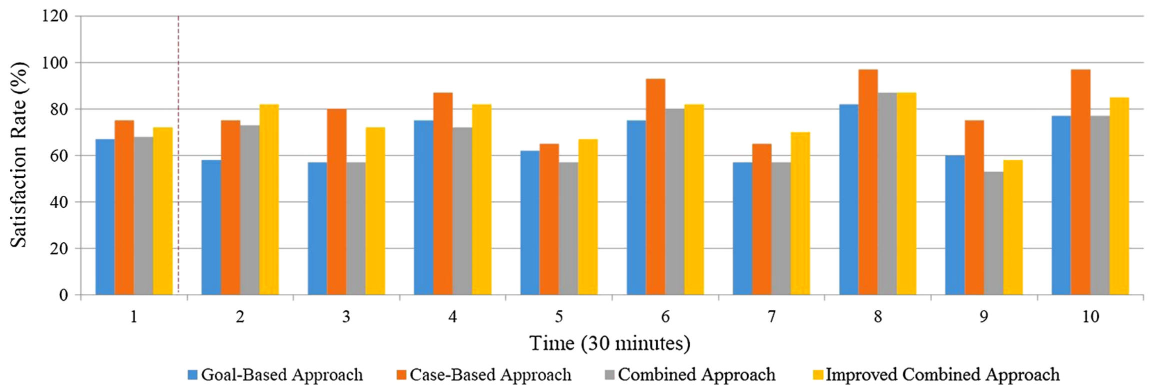

Fig. 5 Distribution of satisfaction rate of **Cons1**

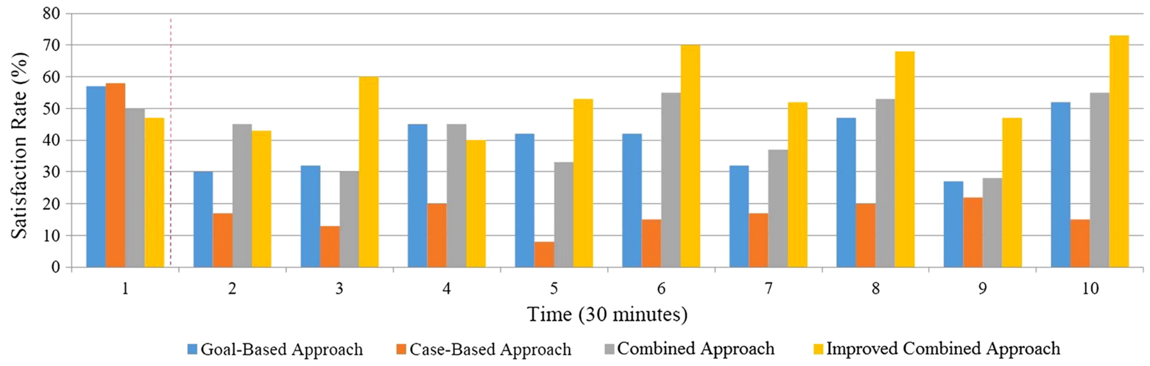

Fig. 6 Distribution of satisfaction rate of **Cons2**

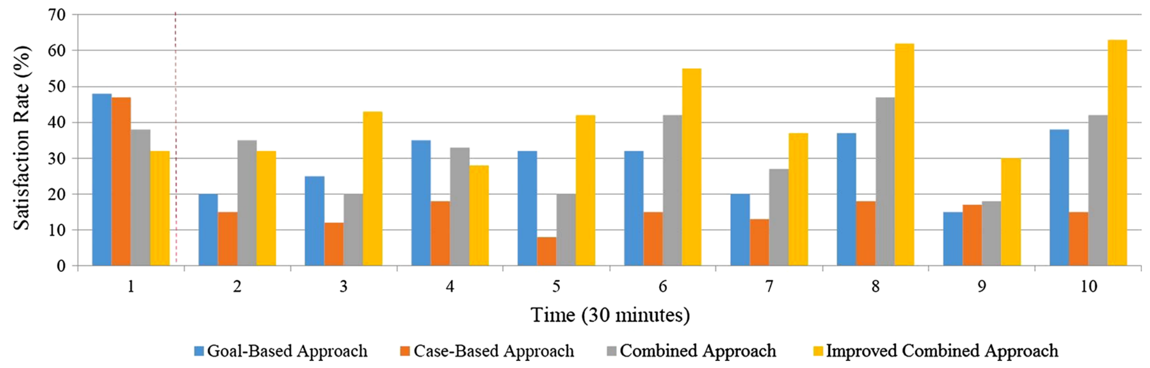

Fig. 7 Distribution of satisfaction rate of **Cons1**∧**Cons2**

satisfying both **Cons1** and **Cons2**. It can be seen that, compared with the other three approaches, the *Improved Combined Approach* achieves comparable satisfaction level as **Cons1** and much higher satisfaction levels than **Cons2** and **Cons1**∧**Cons2**.

To further compare the effectiveness of the four approaches in different phases, we divided the 600 intervals into ten time periods (30 min per period) and analyzed the satisfaction rates of the two constraints during each period. Figures 5, 6 and 7 show the distribution of the satisfaction rate of **Cons1**, **Cons2** and **Cons1**∧**Cons2**, respectively. The X axis denotes the time period, and the Y axis denotes the satisfaction rates of the four approaches. The red dot line represents the ending of the training phase (i.e., the construction of the original case base) of the *Case-Based Approach*.

We can see from Figure 5 that the highest satisfaction rate (97 %) occurs in time periods eight and ten for the *Case-Based Approach*. The lowest satisfaction rate (53 %)

occurs in time period nine for the *Combined Approach*. The *Improved Combined Approach* always achieves a higher satisfaction rate than the *Goal-Based Approach* and *Combined Approach*. However, the *Case-Based Approach* performs the best in seven time periods. After analyzing the solutions generated by the *Case-Based Approach*, we find that this is because during the training phase the *Case-Based Approach* learned many effective solutions for **Cons1** but few for **Cons2**. As a consequence, when tackling a new problem, the *Case-Based Approach* tends to adopt a solution that favors response time. Hence, the satisfaction rate of **Cons1** of the *Case-Based Approach* is often highest, but the satisfaction rate of **Cons2** is often lowest (See Fig. 6). This leads to the lower effectiveness rate of adaptation solutions for the *Case-Based Approach*.

We can see from Fig. 6 that the *Improved Combined Approach* always has the highest satisfaction rate of **Cons2** except for time periods 1, 2 and 4. The highest satisfaction rate (73 %) occurs in time period ten for the *Improved Combined Approach*. The lowest satisfaction rate (8 %) occurs in time period five for the *Case-Based Approach*. As discussed above, the *Case-Based Approach* can not improve the combined utility very well as adaptation cases for this kind of problems are not included in the original case base, making it unable to treat different problems in different environments.

We can see from Fig. 7 that for **Cons1**$\wedge$**Cons2**, the *Improved Combined Approach* performs much better than the other three approaches. The highest satisfaction rate (63 %) occurs in time period ten for the *Improved Combined Approach*. The lowest satisfaction rate (8 %) occurs in time period five for the *Case-Based Approach*. This figure offers evidence that the combination of rationalism and empiricism shows promise in solving the adaptation problem. When we use the *Goal-Based Approach* or the *Case-Based Approach* alone, we can hardly get solutions that are effective for both **Cons1** and **Cons2**. Therefore, the average satisfaction rates of **Cons1**$\wedge$**Cons2** are low (i.e., 30 and 18 %). When we combine goal reasoning and case-based reasoning (i.e., *Combined Approach* or *Improved Combined Approach*), new solutions can be generated when tackling new problems that had not been tackled before, and hence these approaches get higher satisfaction rates for **Cons1**$\wedge$**Cons2**. However, the *Combined Approach* may be trapped in suboptimal adaptation solutions, so its satisfaction rate for **Cons1**$\wedge$**Cons2** is only a littler higher than the *Goal-Based Approach*.

The above analysis answers *RQ1* positively that the proposed approach achieves significant improvement in this experiment over self-adaptation approaches that involve only goal reasoning or case-based reasoning.

4.3 Mutation evaluation (RQ2)

In the experiment with the *Improved Combined Approach*, 29 adaptations generated by case-based reasoning involve case mutation. All of them produce effective mutated solutions that were finally operationalized in the running system. And among the 29 mutated solutions, 19 solve the adaptation problems successfully. For each of these 19 mutated solutions, we analyzed the solutions of the same kind of problem that were adopted before and after it to assess the role of the case mutation mechanism.

Table 4 presents the quality improvement of the 19 mutated adaptation solutions. The second and third columns list the numbers of effective solutions of the same kind of problem before and after the mutation, respectively. The fourth column lists each solution's quality improvement ratio after the mutation, which is calculated by Eq. 7:

$$QualityImprovement = \frac{f_{\text{after}} - f_{\text{before}}}{f_{\text{before}}} * 100\,\% \tag{7}$$

where f_{before} and f_{after} are the average quality improvement ratio of the cases of the same kind of problems before and after the mutation, respectively (the same meaning for f as in Eq. 6).

We can observe from Table 4 that the average quality improvement from those successful mutations is 22 %,

Table 4 Quality improvement of mutated adaptation solutions

ID	S_{before} (#)	S_{after} (#)	*QI* (%)
0	5	22	14
1	3	10	−2
2	6	16	13
3	4	3	55
4	3	2	6
5	6	11	15
6	9	17	8
7	6	7	−8
8	5	8	57
9	5	10	20
10	7	8	19
11	4	5	37
12	9	8	6
13	2	1	−14
14	3	3	13
15	18	1	68
16	7	2	7
17	5	1	−13
18	14	2	117
avg.	6.37	7.21	22

while variation range is very large. For example, the quality improvement from mutation 18 is 117 %, but the quality improvement from mutation 4 is 6 %. This is because the situations of adaptation solutions for different kinds of problems can be quite different before mutation: Some are far from optima and have much room for improvement, while some others are already almost optimal. Besides, the mutation mechanism may sometimes lead to worse solutions (e.g., mutation 13) due to its stochastic nature.

The above analysis and the comparison between the *Improved Combined Approach* and the *Combined Approach* in Sect. 4.2 answer *RQ2* positively that case mutation effectively improves the adaptation results.

4.4 Evolution process analysis (RQ3)

We distinguish three different kinds of adaptations produced by the *Improved Combined Approach*, i.e., adaptations generated by goal reasoning (using the best goal configuration), case-based reasoning (using one perfect case), or synthesized reasoning (synthesizing the two most similar cases and the best goal configuration).

Table 5 shows the frequency of the adaptations generated by different reasoning mechanisms in the experiment (5 h) with the *Improved Combined Approach*. The first column lists the reasoning mechanisms, the second and third list the number of adaptations and the number of effective adaptations, respectively, and the last column lists the effectiveness rate.

We can see that among the 274 adaptations, 158 adaptations are from case-based reasoning, 87 adaptations are from goal reasoning, and 29 adaptations are from synthesized reasoning. The adaptations from synthesized reasoning have the highest effectiveness rate (76 %), while the adaptations from goal reasoning have the lowest effectiveness rate (22 %). Moreover, the effectiveness of case-based reasoning is improved (recall that its effectiveness rate in the *Case-Based Approach* is only 14 %) because the case base in the *Improved Combined Approach* can be incrementally expanded by combining goal reasoning to generate adaptation solutions for new emerging problems (i.e., no similar cases can be found). However, the effectiveness of goal reasoning is decreased (recall that its effectiveness rate in the *Goal-Based Approach* is 35 %) because goal reasoning in the *Improved Combined Approach* can only be used in complex situations where no similar cases exist.

Table 5 Adaptations generated by different reasoning mechanisms

Reasoning mechanism	Adaptation (#)	Eff. (#)	Rate (%)
Case-based reasoning	158	117	74
Goal reasoning	87	19	22
Synthesized reasoning	29	22	76

To further investigate the evolution process of adaptation solutions for different kinds of adaptation problems, we clustered all generated adaptations into groups and the adaptations in each group define a kind of adaptation problem. To this end, we designed a simple adaptation clustering algorithm based on the adaptation problem distance defined in Eq. 4. In the algorithm, every two adaptation problems whose distance is less than the lower distance threshold *threshold*$_l$ are grouped together and regarded as the same kind of problem.

Based on the clustering algorithm, all the 274 adaptations produced by the *Improved Combined Approach* are clustered into 76 groups. The smallest group has only 1 adaptation and the largest group has 36 adaptations. Note that an adaptation may belong to more than one group.

Table 6 shows the details of several adaptation groups. The second column of the table lists all the adaptations of a group in the time sequence, where symbol '−' indicates an adaptation using goal reasoning, symbol '+' indicates an adaptation using synthesized reasoning, symbol '•' indicates an adaptation using case-based reasoning, and an underlined adaptation means that the adaptation is effective. The third to fifth columns of the table list the times of goal reasoning [GR(#)], the times of synthesized reasoning [SR(#)], and the times of case-based reasoning [CBR(#)], respectively.

On average, for each kind of adaptation problem 29 % of the adaptation solutions are generated from goal reasoning, 19 % of the adaptation solutions are from synthesized reasoning, and 52 % adaptation solutions are from case-based reasoning.

After analyzing the sequence of adaptations in each group, we can observe the following general evolution

Table 6 Adaptations in several case groups

ID	Adaptations	GR (#)	SR (#)	CBR (#)
13	$- - - - \underline{\bullet} - - \underline{\bullet} \bullet$	6	0	3
28	$\underline{-} \bullet \underline{\bullet} \underline{\bullet}$	1	0	3
46	$- \underline{-} \underline{+} \underline{+} \underline{\bullet} - \bullet \underline{\bullet} \bullet \bullet \bullet + \underline{\bullet} + \bullet - \bullet \bullet \bullet$	4	4	11
58	$- \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet}$	1	0	5
67	$- \underline{+} \underline{\bullet} \bullet \underline{\bullet} \bullet \underline{\bullet} \underline{\bullet} \bullet \underline{-} \bullet \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet} \underline{\bullet}\bullet \underline{\bullet}$	2	1	20

trend for adaptation solutions. In the beginning, adaptation solutions are usually generated by goal reasoning. In the following stage, adaptation solutions are increasingly generated by synthesized reasoning and case-based reasoning. In the late stage, adaptation solutions are usually generated by case-based reasoning, which indicates that the adaptation solutions of new adaptation problems of the same kind can be stably provided by suitable past cases.

On the basis of this analysis, the evolution process of adaptation solutions for a kind of adaptation problem can be divided into the following three phases:

- **Initialization Phase**: The adaptation mechanism utilizes rationalistic knowledge underlying goal reasoning to incrementally build the initial case base.
- **Accumulation Phase**: Based on an initial case base, similar adaptation cases can be retrieved for new adaptation problems and case-based reasoning is increasingly used. However, because the retained cases are not rich enough, the retrieved cases may not be exactly similar to new problems. As a result, the adaptation mechanism chooses to generate adaptation solutions by synthesizing two most similar cases and a solution produced by goal reasoning. In this phase, both case-based reasoning and synthesized reasoning are gradually used to accumulate more adaptation solutions to further enrich the case base.
- **Maturity Phase**: After the accumulation phase, the case base is rich enough to handle most new adaptation problems. As a result, most adaptation solutions are generated by case-based reasoning, and goal reasoning and synthesized reasoning are only used when encountering new problems that are different from existing ones.

Moreover, we can classify all the 76 groups of adaptation cases into four different types according to the diversity and convergence of their adaptation solutions. Diversity means that different effective adaptation solutions (i.e., goal configurations) for the same kind of adaptation problems are explored and retained in existing adaptation cases. Convergence means that effective adaptation solutions for new adaptation problems of the same kind can always be found from existing cases.

- *Exploratory Non-convergency*: This type of adaptation group can be considered in the **Initialization phase** or the **Accumulation phase**. The solutions in the group are mainly generated from goal reasoning, just beginning to explore the solution space for a kind of adaptation problem. Since goal reasoning often gives a solution with relative low quality, many different solutions of the same kind will be tried.
- *Settling Non-convergency*: This type of adaptation group can be considered in the **Maturity phase**. The solutions of one kind of problem have good diversity but poor convergence. Most solutions in the group are generated from case-based reasoning, using several solutions generated by goal reasoning and synthesized together. But since there will always be new problems, new adaptation solutions will continue to be generated and explored.
- *Precocious Convergency*: This type of adaptation group can be considered in the **Maturity phase**. The solutions of one kind of problem have good convergence but poor diversity. This is because early solutions generated by goal reasoning are effective and continue to be reused for the same kind of problem.
- *Settling Convergency*: This type of adaptation group can also be considered in the **Maturity phase**. Different from *Precocious Convergency*, the solutions of one kind of problem have good diversity and good convergency. The evolution process has tried many different solutions in the solution space, and gradually focused on the best ones.

Figure 8 shows some examples of adaptation groups of different types. The X axis shows the chronological order of adaptations. The Y axis shows the adaptation solution adopted in each adaptation by mapping it to an integer. Such kind of scatter diagram is used to show the diversity and convergency of solutions. Symbols '−', '+' and '•' have the same meanings as in Table 6.

Figure 8a shows the adaptations in group 13 as an example of *Exploratory Non-convergency*. We can see that in the early stage, the solutions are generated mainly using goal reasoning, and different solutions using other reasoning mechanisms arise gradually. Only a few solutions in this group are effective. Figure 8b shows the adaptations in group 46 as an example of *Settling Non-convergency*. We can see that after several adaptations, many problems could be solved by case-based reasoning. But different solutions are still generated in order to solve new problems. Less than half of the solutions in this group are effective. Figure 8c shows the adaptations in group 28 as an example of *Precocious Convergency*. We can see that such kind of problems can always be solved by the same solution generated by case-based reasoning. More than half of the solutions in this group are effective. Figure 8d shows the adaptations in group 67 as an example of *Settling Convergency*. We can see that after several adaptations, most problems could be solved effectively by one solution from case-based reasoning.

Considering the effectiveness rate of solutions from different reasoning mechanisms (Table 5), if an adaptation problem is in group *Settling Convergency*, it can always be

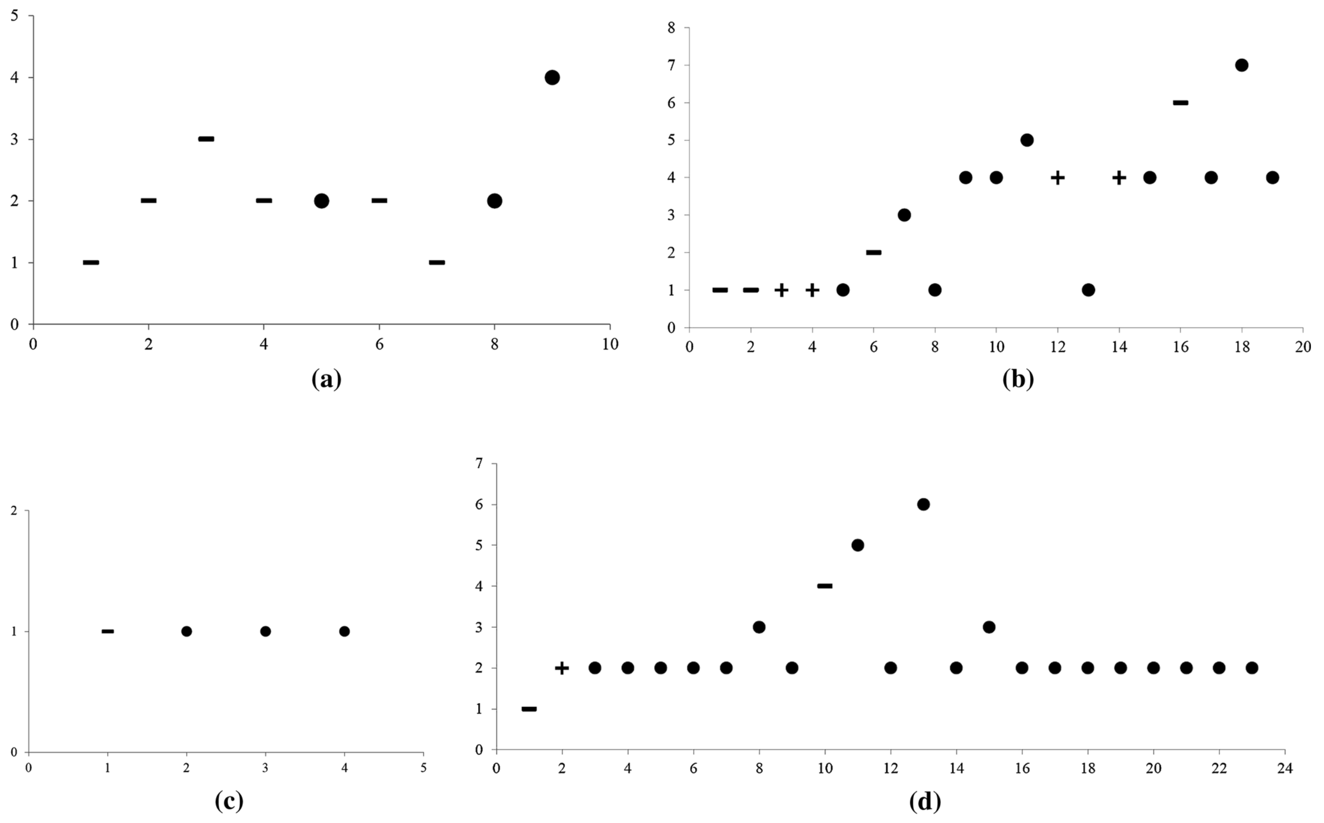

Fig. 8 Examples of different types of adaptation groups by diversity and convergence. **a** Group 13 (exploratory non-convergency), **b** Group 46 (settling non-convergency), **c** Group 28 (precocious convergency), **d** Group 67 (settling convergency)

solved well; if an adaptation problem is in group *Settling Non-convergency* or *Precocious Convergency*, it can be solved well if it is not a new problem; if an adaptation problem is in group *Exploratory Non-convergency*, it may not be solved well. We classified the 76 groups manually and found that 28 groups (37 %) are *Settling Convergency*, 10 groups (13 %) are *Precocious Convergency*, 19 groups (25 %) are *Settling Non-convergency* and 19 groups (25 %) are *Exploratory Non-convergency*.

The above analysis answers *RQ3* from different perspectives: adaptation solutions for the same kind of adaptation problem combine solutions generated from goal reasoning, case-based reasoning, and synthesized reasoning; their evolution process can be divided into three phases, i.e., initialization, accumulation and maturity; their quality can be dynamically evaluated depending on the diversity and convergence of adaptation solutions.

4.5 Threats to validity

The main threat to the internal validity of our experimental study is the experiment environment. We conducted the four experiments on the same server with exactly the same experimental settings to provide an "equal" experiment environment for different experiments. Therefore, we believe the results of the four experiments can reflect the differences of the four approaches.

The main threats to the external validity of our study lie in the fact that we only evaluated the approaches with a small system under a controlled environment. The concurrent system accesses were simulated by JMeter and using a predefined script. The execution scenario described by the script may not cover all the different situations in real system execution. For example, in a scenario where concurrent accesses change very smoothly and regularly the advantage of goal reasoning and case mutation may be not so significant, as the adaptation mechanism can often find effective adaptation solutions from past cases. The experimental results also depended on a series of parameter settings such as thresholds, quality metrics and adaptation interval. Different parameter settings may cause different results, so it is helpful to evaluate the approaches under different parameter settings.

5 Discussion

In this section, we discuss some issues related to our approach and experimental study, including the philosophy of combining rationalism and empiricism, parameter setting, uncertainty and diversity and convergence of adaptation cases.

5.1 Combination of rationalism and empiricism

The philosophy behind our approach is founded on a combination of rationalism (goal reasoning) and empiricism (case-based reasoning). Rationalism enables us to plan a rational solution based on known knowledge about the inherent nature of a system, e.g., the relations between alternative goals/tasks and softgoals in a goal model. This kind of rational reasoning, however, may lead to improper solutions when the knowledge is incomplete. This is usually the case for self-adaptive systems, where complex and ever-changing contexts make it impossible to have a complete model of the relationships between alternative system configurations, environmental parameters and requirements.

Empiricism, on the other hand, enables us to learn from past experiences that embody explorations of problem solving in complex and ever-changing contexts. This kind of experience-based problem solving, however, may provide poor solutions when no similar past experiences are available. Case-based reasoning, which depends on a set of original cases, usually cannot capture a comprehensive case base that can cover all the combinations of related elements in a self-adaptive system.

Combining case-based reasoning with goal reasoning, our approach enables rationalism-based reasoning about suitable solutions for new problems. Such solutions, in turn, are validated by feedback and retained for future reuse. In our approach, approximate solutions produced by goal reasoning can be further selected by feedback-based evaluation and refined by being synthesized with past experiences.

It is worth noting that the case-based approach implemented in the comparative evaluation uses goal reasoning to construct the original case base. Without this kind of automated learning of an initial case set, it is usually hard for case-based reasoning approaches to prepare a comprehensive set of original cases for self-adaptation.

Last but not the least, ineffective cases may also be helpful for case-based reasoning for self-adaptation. Just as with positive experiences provided by effective cases, negative experiences contained in ineffective cases can provide useful hints for future adaptation decision making. For example, we store failed adaptation cases and use them to eliminate mutated adaptation solutions that have been unsuccessfully applied in the past.

5.2 Performance overhead

Our approach involves continuous runtime adaptation planning and execution to seek optimal adaptation solutions and reconfigures the system accordingly. Therefore, the runtime overhead caused by adaptation planning and execution can be a problem when applying the approach in practice.

In our approach, the performance overhead of adaptation planning is mainly caused by case-based reasoning and goal reasoning. Among all the four main steps of case-based reasoning (i.e., *Retrieve*, *Reuse*, *Revise*, *Retain*), *Retrieve* is the most expensive, since it requires to inspect all the adaptation cases in the case base. Our performance testing shows that, when the case base includes 500 and 1000 cases, it takes about 75 and 120 ms, respectively, to find the most similar case. Considering that the case base of a medium-sized system typically includes hundreds to thousands of adaptation cases, this overhead is acceptable for a periodical adaptation mechanism that is executed every few minutes. Goal reasoning needs to enumerate all the candidate goal configurations and choose one with the highest overall contribution from them. The number of candidate goal configurations increases exponentially as the number of goal variation points and control variables increases. Therefore, for a medium- or large-sized system, for example with more than 20 goal variation points and control variables, goal reasoning may need several seconds or more to generate an adaptation solution. For more complex systems with a large number of goal variation points, more efficient algorithms such as genetic algorithm can be used to find near-optimal goal configurations. An advantage is that, after the construction of the initial case base, case-based reasoning is the main source of adaptation solutions and goal reasoning is seldom used (see the evolution analysis in Sect. 4.3).

The overhead of adaptation execution highly depends on the implementation mechanism of runtime reconfiguration. Light-weighted mechanisms such as runtime parameter configuration usually are very cheap, while heavy-weighted mechanisms involving dynamic component creation and replacement with safety assurance are much more expensive.

5.3 Parameter setting

Our approach uses a series of parameters and thresholds, which affect different aspects of the approach.

The setting of the weight tuning pace of quality attributes (i.e., α in Algorithm 1) and the changing pace of control variables reflects the tradeoff between tolerance of system disturbance and significance of adaptation effect. If these two paces are set larger, goal reasoning is likely to produce more radical adaptation solutions, which may make more significant improvement in the desired quality attributes but may also cause great disturbance to other quality attributes. Otherwise, goal reasoning is likely to produce more gentle adaptation solutions with smaller disturbance to related quality attributes.

The two distance thresholds in case retrieval (i.e., $threshold_h$ and $threshold_l$ in Algorithm 2) affect the opportunity of finding reusable past cases and the suitability of the initial adaptation solution. Lower distance thresholds can ensure that the initial adaptation solutions are more suitable for the encountered adaptation problems, but may reduce the opportunity and thus make the adaptation approach depend more on goal reasoning. Higher distance thresholds can increase the opportunity of reusing past adaptation cases, but may produce more ineffective adaptation solutions generated from less similar cases.

The highest mutation probability k in Eq. 6 determines the frequency of case mutation and thus has a great influence on the diversity of adaptation cases. A higher mutation probability can increase the diversity by exploring and retaining more different kinds of adaptation solutions, but may introduce more ineffective adaptation solutions.

The parameters in our experimental study were set based on our observation and experience from the adaptation process. In our future work, we will conduct sensitivity analysis of these parameters to empirically evaluate their impacts on the effectiveness of our approach. We will introduce adaptive parameter tuning, which can, for example, adaptively tune the distance thresholds based on the feedback related to the effectiveness of initial adaptation solutions.

5.4 Uncertainty

Our approach suffers from diverse uncertainties in its evaluation of adaptation solutions and matching of similar past cases.

The effectiveness of an executed adaptation solution is evaluated based on the runtime data about quality attributes collected during a fixed adaptation period. This means that the executed adaptation solution is assumed to have an effect in the adaptation period immediately following its execution. However, the timing delay for the adaptation solution to fully take effect is uncertain, causing the so-called effect uncertainty [10]. Effect uncertainty may cause inaccurate evaluation of the effectiveness of an adaptation solution. For example, an effective adaptation solution may be evaluated to be ineffective due to insufficient timing delay for it to take effect. Therefore, it is worthwhile to integrate effect uncertainty handling mechanisms such as the heuristics-based technique proposed in [10] to alleviate the negative impact of effect uncertainty.

The context (i.e., $Context_{cur}$) of an adaptation problem is specified by a set of predefined environmental parameters such as CPU usage, memory usage, access load. These environmental parameters are usually incomplete and might not characterize sufficiently the operational environment. This in turn causes uncertainty in matching and selecting similar past cases for generating new adaptation solutions. For example, irrelevant adaptation cases may be retrieved as similar cases due to the missing of distinguishing environmental parameters. To alleviate the negative effect of the uncertainty of environmental description, it may be necessary to conduct a sensitivity analysis to carefully identify and select environmental parameters that are essential for a complete description of adaptation problems.

5.5 Diversity and convergence of adaptation cases

The effectiveness and quality of the adaptation solutions for a specific kind of adaptation problem highly depend on their diversity and convergence.

In our approach, adaptation solutions for a specific kind of adaptation problem are initially generated by goal reasoning and then synthesized from past cases with the same kind of problems. Although the adaptation cases of these adaptation problems may have good convergence, they are likely to suffer from the problem of insufficient diversity, making the adaptation be trapped in suboptimal adaptation solutions.

Case mutation can effectively improve the diversity of adaptation solutions by trying unexplored adaptation solutions for a kind of adaptation problems. The exploration may fail to produce a new effective solution and even cause the degradation of quality attributes. For example, among the 36 case mutations in our experimental study, 20 mutations successfully produced effective adaptation solutions, while the other 16 failed. However, the exploration is beneficial and even necessary, as the explored effective adaptation solutions can be much better and repeatedly used in subsequent adaptation problems of the same kind.

However, it is also possible that the adaptation solutions of a kind of adaptation problem have good diversity but poor convergence. This means that, although a large number of different effective adaptation solutions have been explored and retained for a kind of adaptation problem, the adaptation solutions for new adaptation problems of the same kind cannot be found from existing cases. One reason for this may be incomplete adaptation problem descriptions such as missing environmental parameters, which make the adaptation approach unable to distinguish adaptation problems of the same kind that are actually different. Therefore, the adaptation approach may retrieve unsuitable adaptation cases for encountered adaptation problems. The other reason may be incomplete goal model, which means some key configuration points of the system are not captured by its goal variation points and control variables.

Based on the above analysis, it is possible to improve the proposed approach by introducing another layer of feedback control loop for the continuous improvement of the self-adaptation mechanism. Due to incomplete system analysis or continuous evolution of requirements and contexts, the predefined adaptation problem descriptions and goal model of a self-adaptive system may be incomplete, leading to suboptimal runtime adaptations. The new feedback control loop can continuously monitor the convergence and diversity of adaptation solutions, and involve human experts to analyze the overall quality of adaptation cases and consider possible improvement of the adaptation problem descriptions and goal model, for example, by identifying missing environmental parameters or refining existing goals.

6 Related work

Our work falls in the area of self-adaptive system [11, 13, 31]. Since we combine the requirements-driven self-adaptation with the case-based reasoning in our approach, here we focus our discussion on related work belonging to two areas: requirements-driven self-adaptation and case-based self-adaptation.

6.1 Requirements-driven self-adaptation

Requirements-driven self-adaptive systems can be seen as a kind of requirements-aware systems. In such systems, runtime requirements models are used to reason about the satisfaction levels of requirements and to support adaptation decisions [31]. Baresi et al. [3] propose an approach that introduces fuzzy and adaptive goals to embed adaptation countermeasures. Wang et al. [36] propose a self-repairing approach that uses goal reasoning to select the best system configuration. Dalpiaz et al. [12] enrich goal models with context-dependent goal decompositions, goal commitments with time limits and domain assumptions in their self-reconfiguration architecture. Based on the enriched goal models, their approach monitors for and diagnoses runtime failures by comparing monitored behavior of a system to expected and allowed behaviors. Peng et al. [28] propose a self-optimization approach that uses a preference-based goal reasoning procedure to reason about optimal goal configurations based on tuned preference ranks by a feedback controller. Chen et al. [8, 9] propose a requirements-driven approach to achieving survivability assurance of Web systems and quality optimization of composite services. Chen et al. [7, 10] also handle diverse uncertainties (i.e., preference uncertainty, contribution uncertainty and effect uncertainty) in goal-driven self-optimization, and combine requirements and architectural adaptations with model transformation techniques. Fu et al. [14] propose a goal-based monitoring and self-repairing framework for sociotechnical systems, which reasons about runtime failures and repairing solutions based on goal state machines and their interactions. Salehie et al. [30] propose a requirements-driven approach to supporting adaptive security in order to protect variable assets at runtime. Bencomo et al. [4] propose an approach that maps the goal models into Dynamic Decision Networks and uses the Dynamic Decision Networks to automatically make the best adaptation decisions.

These requirements-driven self-adaptation approaches depend on runtime requirements models (usually goal models) to reason about requirements violations and adaptation decisions. The relations between system configurations, environmental parameters and requirements, serving as the basis of runtime reasoning, are assumed to be clearly understood and modeled. Different from our approach, these approaches do not learn the complex and changing relations from past experiences.

6.2 Case-based self-adaptation

Several attempts have been already made to apply case-based reasoning in self-adaptive systems. Montani et al. [2, 24, 25] propose an approach that uses case-based reasoning to diagnose runtime failures of software systems and provide remediation solutions. McSherry et al. [23] propose a hypothesis-driven approach that can diagnose runtime failures and provides recovery suggestions by asking questions and explaining the relevance of the questions to users. Khan et al. [18] suggest applying case-based reasoning for self-configuration in autonomic systems and propose a new similarity measure for case retrieval. In their subsequent work, they improve the approach by restricting the size of case base without harming the reasoning accuracy [19] and applying clustering on the case base to improve the reasoning efficiency [20].

These approaches depend on an original case base, which is usually constructed manually by humans based on historical system operation records or human expertise. They cannot provide a proper adaptation solution if there are no similar cases for a given adaptation problem in the case base. By contrast, our approach can provide proper, perhaps not perfect, adaptation solutions with goal reasoning when no similar cases are available.

7 Conclusions

Requirements-driven self-adaptation approaches can make proper adaptation decisions by reasoning over runtime requirements models, but cannot learn from past experiences the complex relations between system configurations, environmental parameters and requirements.

In this paper, we have proposed an improved requirements-driven self-adaptation approach that combines goal reasoning and case-based reasoning. In our approach, past experiences of successful adaptations are retained in a case base to be reused for future adaptation problems. Both the current system configuration and the adaptation solution of an adaptation case are specified by goal configurations consisting of specific alternatives for variation points and values for control variables. The approach reasons over a predefined requirements goal model to provide adaptation solutions when no similar cases are available. An additional case mutation mechanism is introduced to prevent case-based reasoning from being trapped in suboptimal adaptation solutions. The effectiveness of the approach and the role of case mutation have been evaluated by comparing our approach with a goal reasoning approach and a case-based reasoning approach. Moreover, we have empirically investigated the evolution process of solutions for different kinds of adaptation problems.

Our future work will investigate some key elements of the approach such as the selection of environmental parameters and the strategy of adaptation solution synthesis, as well as the influence of different thresholds settings. Moreover, the approach will be evaluated with respect to a real, as opposed to a simulated, case study.

Acknowledgments This work is supported by National Natural Science Foundation of China under Grant No. 61361120097 and National High Technology Development 863 Program of China under Grant No. 2013AA01A605.

References

1. Aamodt A, Plaza E (1994) Case-based reasoning: foundational issues, methodological variations, and system approaches. AI Commun 7(1):39–59
2. Anglano C, Montani S (2005) Achieving self-healing in autonomic software systems: a case-based reasoning approach. In: SOAS, pp 267–281
3. Baresi L, Pasquale L, Spoletini P (2010) Fuzzy goals for requirements-driven adaptation. In: RE, pp 125–134
4. Bencomo N, Belaggoun A (2013) Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks. In: REFSQ, pp 221–236
5. Bruneton E, Coupaye T, Leclercq M, Quéma V, Stefani J (2006) The FRACTAL component model and its support in java: experiences with auto-adaptive and reconfigurable systems. Softw Pract Exp 36(11–12):1257–1284
6. Carzaniga A, Rosenblum D, Wolf A (2003) Design and evaluation of a wide-area event notification service. In: foundations of intrusion tolerant systems, 2003 (Organically Assured and Survivable Information Systems), pp 283–334
7. Chen B, Peng X, Yu Y, Nuseibeh B, Zhao W (2014) Self-adaptation through incremental generative model transformations at runtime. In: ICSE, pp 676–687
8. Chen B, Peng X, Yu Y, Zhao W (2011) Are your sites down? Requirements-driven self-tuning for the survivability of Web systems. In: RE, pp 219–228
9. Chen B, Peng X, Yu Y, Zhao W (2014) Requirements-driven self-optimization of composite services using feedback control. IEEE Trans Serv Comput 1
10. Chen B, Peng X, Yu Y, Zhao W (2014) Uncertainty handling in goal-driven self-optimization-limiting the negative effect on adaptation. J Syst Softw 90:114–127
11. Cheng BH, De Lemos R, Giese H, Inverardi P, Magee J, Andersson J, Becker B, Bencomo N, Brun Y, Cukic B, et al (2009) Software engineering for self-adaptive systems: a research roadmap. In: Softw Eng Self-Adapt Syst, pp 1–26
12. Dalpiaz F, Giorgini P, Mylopoulos J (2009) An architecture for requirements-driven self-reconfiguration. In: CAiSE, pp 246–260
13. De Lemos R, Giese H, Müller HA, Shaw M, Andersson J, Litoiu M, Schmerl B, Tamura G, Villegas NM, Vogel T et al (2013) Software engineering for self-adaptive systems: a second research roadmap. In: Softw Eng Self-Adapt Syst II, pp 1–32
14. Fu L, Peng X, Yu Y, Mylopoulos J, Zhao W (2012) Stateful requirements monitoring for self-repairing socio-technical systems. In: RE, pp 121–130
15. Giorgini P, Mylopoulos J, Nicchiarelli E, Sebastiani R (2002) Reasoning with goal models. In: ER, pp 167–181
16. Hinchey M, Sterritt R (2006) Self-managing software. Comput 39(2):107–109
17. Kephart JO, Chess DM (2003) The vision of autonomic computing. Comput 36(1):41–50
18. Khan MJ, Awais MM, Shamail S (2007) Achieving self-configuration capability in autonomic systems using case-based reasoning with a new similarity measure. In: ICIC, pp 97–106
19. Khan MJ, Awais MM, Shamail S (2008) Enabling self-configuration in autonomic systems using case-based reasoning with improved efficiency. In: ICAS, pp 112–117
20. Khan MJ, Awais MM, Shamail S (2010) Improving efficiency of self-configurable autonomic systems using clustered cbr approach. IEICE Trans Inf Syst 93(11):3005–3016
21. Kolodneer JL (1991) Improving human decision making through case-based decision aiding. AI Mag 12(2):52
22. Letier E, Van Lamsweerde A (2004) Reasoning about partial goal satisfaction for requirements and design engineering. ACM SIGSOFT Softw Eng Notes 29(6):53–62
23. McSherry D, Hassan S, Bustard D (2008) Conversational case-based reasoning in self-healing and recovery. In: ECCBR, pp 340–354
24. Montani S, Anglano C (2006) Case-based reasoning for autonomous service failure diagnosis and remediation in software systems. In: ECCBR, pp 489–503
25. Montani S, Anglano C (2008) Achieving self-healing in service delivery software systems by means of case-based reasoning. Appl Intell 28(2):139–152
26. Montani S, Anglano C (2008) Retrieval, reuse, revision, and retention in casebased reasoning. Appl Intell 28(2):139–152
27. Mylopoulos J, Chung L, Nixon B (1992) Representing and using nonfunctional requirements: a process-oriented approach. IEEE Trans Softw Eng 18(6):483–497
28. Peng X, Chen B, Yu Y, Zhao W (2012) Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop. J Syst Softw 85(12):2707–2719
29. Qian W, Peng X, Chen B, Mylopoulos J, Wang H, Zhao W (2014) Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation. In: RE, pp 113–122
30. Salehie M, Pasquale L, Omoronyia I, Ali R, Nuseibeh B (2012) Requirements-driven adaptive security: protecting variable assets at runtime. In: RE, pp 111–120

31. Sawyer P, Bencomo N, Whittle J, Letier E, Finkelstein A (2010) Requirements-aware systems: a research agenda for RE for self-adaptive systems. In: RE, pp 95–103
32. Sebastiani R, Giorgini P, Mylopoulos J (2004) Simple and minimum-cost satisfiability for goal models. In: CAiSE, pp 20–35
33. Souza VES, Lapouchnian A, Mylopoulos J (2011) System identification for adaptive software systems: a requirements engineering perspective. In: ER, pp 346–361
34. Srinivas M, Patnaik LM (1994) Adaptive probabilities of crossover and mutation in genetic algorithms. IEEE Trans Syst Man Cybern 24(4):656–667
35. Srinivas M, Patnaik LM (1994) Genetic algorithms: a survey. Comput 27(6):17–26
36. Wang Y, Mylopoulos J (2009) Self-repair through reconfiguration: a requirements engineering approach. In: ASE, pp 257–268