Rationalism with a Dose of Empiricism: Case-Based Reasoning for Requirements-Driven Self-Adaptation

Wenyi Qian*†, Xin Peng*†, Bihuan Chen*†, John Mylopoulos‡, Huanhuan Wang*†, and Wenyun Zhao*†
*School of Computer Science, Fudan University, Shanghai, China
†Shanghai Key Laboratory of Data Science, Fudan University, China
‡Department of Information Engineering and Computer Science, University of Trento, Italy
{qianwy, pengxin, bhchen, huanhuanwang13, wyzhao}@fudan.edu.cn, jm@disi.unitn.it

***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 behaviours, environmental parameters and requirements are clearly understood, which is often simply not true. Moreover, they do not consider the influence of the current behaviour 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 behaviours. 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. And case-based reasoning is used to provide more precise adaptation decisions that better reflect the complex relations among requirements violations, contexts, and current behaviours by utilizing past experiences. Our experimental study with an online shopping benchmark shows that our approach outperforms both requirements-driven approach and case-based reasoning approach in terms of adaptation effectiveness and overall quality of the system.**

I. Introduction

A self-adaptive system can switch among alternative behaviours, so that it can continue to satisfy stakeholder requirements (goals) in a changing and uncertain runtime environment. Requirements-driven approaches [1], [2], [3], [4], [5] provide an effective mechanism for such self-adaptive systems by reasoning over their runtime requirements models to make adaptation decisions.

However, such approaches usually assume that the requirements of a self-adaptive system are well understood so that they can be used as the basis for an adaptation mechanism. In particular, the relations among alternative behaviours, environmental parameters and requirements are clearly understood and modelled. Given these models, the adaptation mechanism can reason about alternative behaviours 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 among behaviours, contexts and requirements are not precisely specified. Moreover, an appropriate adaptation decision depends not only on requirements and context changes but also on the current behaviour deployed in the system. In that sense, there can be very complex combinations of requirements violations, contexts, and currently deployed behaviours (current configuration of the system), 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 [6]. Its underlying idea is that new problems can be solved by reusing solutions of the past to similar problems. This makes CBR a promising problem solving paradigm when the problem is not very well understood or the knowledge is hard to express. Several attempts have been made to apply CBR in self-adaptive systems [7], [8], [9], where cases are described by low-level symptoms (symbolic or numeric variables) characterizing encountered problems (e.g. requirements failures). And they depend on a set of original cases that are manually constructed by humans.

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 and stored in a case base. Adaptation cases are described by not only requirements violations (a set of quality attributes) and contexts (a set of environmental parameters), but also currently deployed behaviours. Based on a predefined requirements goal model, both the current behaviour and the adaptation solution (i.e. new behaviour to be adopted after adaptation) in a case are specified by goal configurations consisting of specific alternatives for variation points and values for control variables. Such a goal-oriented case representation provides richer and higher-level representation for adaptation cases. 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 behaviours by utilizing past experiences (adaptation cases).

To evaluate the effectiveness of our approach, we conduct an experimental study with an online shopping benchmark. The results show that our approach outperforms both a requirements-driven approach and a case-based reasoning

978-1-4799-3033-3/14 © 2014 IEEE

RE 2014, Karlskrona, Sweden

Accepted for publication by IEEE. © 2014 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

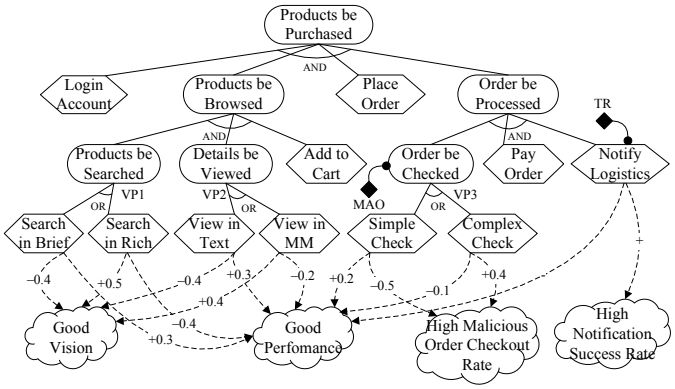

Fig. 1. The Parameterized Goal Model of an Online Shopping System

approach in terms of adaptation effectiveness and overall quality of the system.

The rest of the paper is structured as follows. Section II introduces the required preliminaries including parameterized goal models and case-based reasoning. Section III presents the proposed self-adaptation approach. Section IV evaluates the proposed approach using an online shopping system and discusses related issues. Section V reviews a number of existing proposals and compares them with ours. Finally, Section VI concludes the paper and outlines our future work.

II. Preliminaries

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

A. Parameterized Goal Models

In goal-oriented requirements analysis, functional requirements are modeled as *hard goals*, and quality requirements are modeled as *softgoals* [10]. In a requirements goal model, goals are iteratively refined 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 fulfill 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 [11]. $+/-$ indicates that the satisfaction of the source goal contributes to w-level satisfaction/denial of the target goal.

In a goal model, OR-decomposed goals represent a kind of variation points of system behaviours with their alternative subgoals as the variants, and thus provide a mechanism for reconfiguration. Recently, control variables were introduced as attachments to goals or tasks to represent variables that influence the fulfilment of a goal, or the execution of a task [12]. Control variables provide another mechanism for reconfiguration. Goal variation points and control variables are together referred to parameters, and such goal models 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. The alternative behaviours of a system at runtime can then be represented by alternative goal configurations.

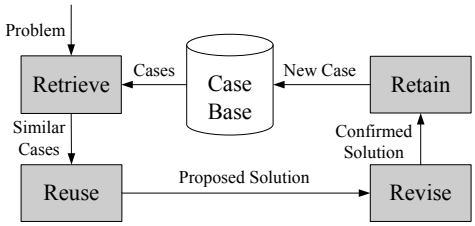

Fig. 2. The Process of Case-Based Reasoning

Figure 1 shows the parameterized goal model of an online shopping system with three variation points and two control variables. For example, the variation point *Products be Searched* can be satisfied by either *Brief Search* (simply search with keywords) or *Rich Search* (search with recommendation), which have reverse contributions to *Good Usability* and *Good Performance*. The control variable *MAO* represents the minimum amount for an order that needs to be checked before further processing, while the control variable *NR* represents the number of retries to be attempted if a notification to logistics fails. Then a candidate goal configuration that satisfies *Products be Purchased* is [*Rich Search*, *View in MM*, *Complex Check*, 200 (*MAO*), 2 (*NR*)].

B. Case-Based Reasoning

Case-based reasoning (CBR) is a problem solving paradigm, which is able to use the knowledge of previously experienced, concrete problem situations (cases) to solve new problems [6]. CBR is well suited for problems where requirements and domain knowledge is simply unavailable. Further, CBR is actually an incremental and sustained learning process that retains and reuses knowledge from past experiences [6].

Figure 2 shows the CBR process, which comprises four steps, i.e. *Retrieve*, *Reuse*, *Revise* and *Retain*. A case base is used to store previously learned cases in the form of problem-solution pairs.

In particular, *Retrieve* retrieves cases whose problems are similar to the problem at hand, using a matching operation. Based on the retrieved similar cases, *Reuse* is conducted to produce a solution to the given problem, for example, by calculating averages of the solutions of similar cases. Then in the *Revise* step, the solution to the given problem is tested for success, e.g. by being applied to real-world problems or evaluated by experts. If failed, the solution may also be repaired using domain-specific knowledge such as rules. Finally, *Retain* retains the learned new case by incorporating it into the case base for future reuse.

III. Our Approach

This section first presents an overview of the proposed approach, and then details the underlying techniques.

A. Overview

Our approach combines requirements-driven self-adaptation and case-based reasoning. An adaptation case base is used

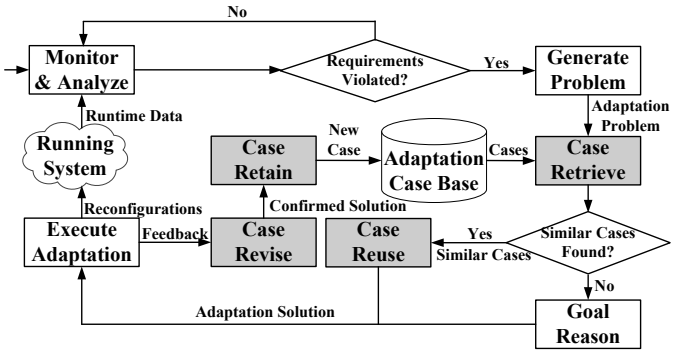

Fig. 3. Approach Overview

to store past experiences of successful adaptations, which is initially empty. On the other hand, a requirements goal model is used to provide a richer and higher-level representation for adaptation cases and reason about adaptation solutions when no similar past cases can be found. Figure 3 presents an overview of our approach, showing the main steps and input/output in the process. The four gray rectangles in the figure correspond to the four main tasks of case-based reasoning, i.e. *Retrieve*, *Reuse*, *Revise*, *Retain*.

Based on a running system, our approach creates an adaptation control 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. Then the approach evaluates the quality attributes and determines whether some requirements are violated, for example, quality constraints are violated or desired quality decreases. If requirements violations are detected, an adaptation problem is raised, which describes the detected quality attributes, contexts, and the current behaviour adopted by the system.

To find an adaptation solution for an adaptation problem, the approach retrieves similar adaptation cases from the case base. If similar cases are found, an adaptation solution (i.e. goal configuration) is synthesized from them (case reuse) for the current adaptation problem. If not found, a goal reasoner is used to produce an adaptation solution by reasoning about optimal goal configurations based on the predefined goal model. Based on the adaptation solution produced by case reuse or goal reasoning, the approach adapts the running system by reconfiguring the runtime architecture and parameters.

After an adaptation solution is executed, the changes of related quality attributes of the system are collected as feedback. Based on the feedback, the effect of the adaptation is evaluated by the case revision mechanism. If the adaptation is successful, i.e. desired quality attributes are improved, the adaptation solution used is confirmed and sent to the case retention mechanism. Then a new case is created for the current adaptation and retained in the case base.

B. Case Representation

Each case in our approach represents a successful adaptation in which a proper switch of alternative system behaviours is made under a specific situation (requirements violations, contexts, and currently deployed behaviour). In case-based reasoning, a case usually includes three major parts [13]: 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 another alternative goal configuration. And the effect part of an adaptation describes the outcome of an adaptation case according to the improvement of quality attributes. Therefore, an adaptation case can be characterized by the following representation:

$$Case =< Problem, GoalConfig_{new}, Quality_{new} > \quad (1)$$

1) Adaptation Problem: As specified in the following equation, the problem description ($Problem$) of an adaptation case consists of the situation of requirements violations ($Quality_{cur}$) and contexts ($Context_{cur}$), and an additional description about the goal configuration currently adopted ($GoalConfig_{cur}$) when an adaptation problem is raised.

$$Problem =< Quality_{cur}, Context_{cur}, GoalConfig_{cur} > \quad (2)$$

Among the problem description, $Quality_{cur}$ denotes the situation of requirements violations specified by a set of quality attributes, each corresponding to a softgoal. And for each quality attribute a value is aggregated based on the runtime data collected during a fixed adaptation period (e.g. 3 minutes). For example, for the softgoal *Good Performance* in Figure 1, a value of average response time can be aggregated and assigned to it. $Context_{cur}$ denotes the situation of contexts specified by a set of predefined environmental parameters. For example, the runtime contexts of a Web system usually can be described by CPU usage, memory usage, access load etc. $GoalConfig_{cur}$ denotes the current goal configuration that is adopted for the root goal of the system when an adaptation problem is raised. It is specified by a set of alternative goals/tasks and control variable values according to a predefined goal model. For example, according to the goal model in Figure 1, a description of $GoalConfig_{cur}$ consists of an alternative task chosen for *Products be Searched*, *Details be Viewed*, and *Order be Checked* respectively, and a value for *MAO* (minimum amount of order) and *NR* (number of retries) respectively.

2) Adaptation Solution: The solution part ($GoalConfig_{new}$) of an adaptation case specifies a switch to another alternative goal configuration of the system. Therefore, it is described in the same way of a current goal configuration ($GoalConfig_{cur}$), i.e. by a set of alternative goals/tasks and control variable values.

3) Adaptation Effect: The effect description ($Quality_{new}$) of an adaptation case records the improvement of quality attributes after the adaptation is carried out. Therefore, it is described in the same way of the quality part ($Quality_{cur}$) of an adaptation problem, i.e. by the values of a set of quality attributes.

C. Goal Reasoning

Goal reasoning is used to provide adaptation solutions based on a predefined goal model when no similar cases are available. To reflect the problem of requirements violations, the weights of related quality attributes (softgoals) are dynamically tuned to improve 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 $value$, $cons_u$, $cons_l$, $weight$ represent the current value, upper constraint (the value of the quality attribute should be lower than $cons_u$), lower constraint (the value of the quality attribute should be higher than $cons_l$), and current weight of the quality attribute. The upper (lower) constraint is assigned to positive (negative) infinity if the quality attribute has no upper (lower) constraint. The weights of all the quality attributes are initiated to 1 when the system is launched.

$$QualityAttribute =< value, cons_u, cons_l, weight > \quad (3)$$

The algorithm 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%). Otherwise, it means that the current adaptation is triggered by the degradation of the combined utility, so the weight of a selected quality attribute with the 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 = ∅, min = +∞
3:      for each q ∈ QASet do
4:          if q.value > q.cons_u || q.value < q.cons_l then
5:              VioSet = VioSet ⋃ {q}
6:          else if q.weight < min then
7:              min = q.weight
8:              temp = q
9:          end if
10:     end for
11:     if VioSet ≠ ∅ then
12:         for each q ∈ VioSet do
13:             q.weight = q.weight × (1 + α)
14:         end for
15:     else
16:         temp.weight = temp.weight × (1 + α)
17:     end if
18: end procedure
```

To involve control variables in goal reasoning, we transform each task with a control variable into an OR-decomposition as shown in Figure 4. For each task $Task$ with a control variable CV, three alternative tasks $Task(CV{+}{+})$, $Task(CV{-}{-})$, $Task(CV0)$ are created, representing the same $Task$ but with

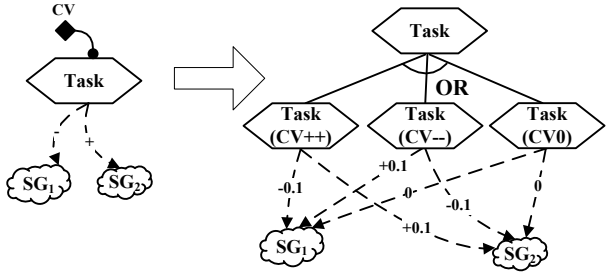

Fig. 4. Transform Control Variable into OR-Decomposition

increasing, decreasing, and unchanged CV respectively. For each of them, a contribution link is created with related softgoals. The contribution weights of $Task(CV0)$ 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. And if the value of CV goes beyond its value range after increase/decrease, the current goal configuration under consideration is discarded.

For a candidate goal configuration, the satisfaction level of a softgoal can be calculated using label propagation algorithms [11]. And 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 as the following equation.

$$RelativeWeight(qa) = \frac{qa.weight}{\sum\limits_{q \in QASet} q.weight}$$

Based on the contribution calculation, the goal configuration with the highest overall contribution is chosen as the result of goal reasoning and used as the solution for the current adaptation problem.

D. Case Retrieval

Case retrieval is based on the distance between a given adaptation problem and the problem description of each case in the adaptation case base. The distance of two adaptation problem P_1 and P_2 is calculated as the Euclidean distance of the their n-dimention description vectors V_{P_1} and V_{P_2}:

$$Distance(P_1, P_2) = \sqrt{\sum_{i=1}^{n} (diff(V_{P_1}[i], V_{P_2}[i]))^2} \quad (4)$$

The $diff$ function in the above equation denotes the difference of the values on the ith dimension in P_1 and P_2. For a dimension corresponding to a enumeration type (e.g. alternative goal/task), the difference is 0 if the two values are the same and 1 if not. For a dimension corresponding to a number type (e.g. control variable), the difference is computed after normalization. The normalization of a numerical variable can be calculated by $\frac{v-MIN}{MAX-MIN}$, where v is its current value, MIN and MAX are its minimal and maximal values during the system operation.

Based on the case distance measurement, the case retrieval procedure (see Algorithm 2) identifies similar adaptation cases with a predefined higher distance threshold $threshold_h$ and a predefined lower threshold $threshold_l$ ($threshold_l < threshold_h$). The inputs of the algorithm are an adaptation problem $curProblem$ and a set of adaptation cases $CaseSet$. It returns a perfect adaptation case or a set of similar-enough cases $SimCases$. If there are some cases whose problem descriptions are very similar (distance lower than $threshold_l$) to the given problem, the most similar case of them is returned as the result of case retrieval. Otherwise, a set of similar-enough cases $SimCases$ (distance lower than $threshold_h$) is returned as the result.

Algorithm 2 Adaptation Case Retrieval

```
1: function RETRIEVE(curProblem, CaseSet, SimCases)
2:     min = threshold_l, perfectCase = null
3:     for each case ∈ 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 ⋃ {case}
10:        end if
11:    end for
12:    return perfectCase
13: end function
```

E. Case Reuse

The case reuse strategy of our approach depends on the result 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 the chaos brought by mixing up too many cases, our approach only picks up to two most similar cases among returned cases. Besides, another goal configuration is generated using goal reasoning. The goal configurations of the two selected cases together with the generated goal configuration are 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.

F. Case Revision and Retention

For an adaptation solution with the goal configuration $goalConfig$ for an adaptation problem $problem$, the case revision mechanism evaluates its effectiveness. Based on the feedback from adaptation execution, the evaluation measures related quality attributes of the system during the adaptation period after the adaptation execution. The obtained situation of quality attributes $Quality_{new}$ is then compared with the situation in the adaptation problem (i.e. $problem.Quality_{cur}$). 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 $Quality_{new}$ and all of the violated quality attributes in $problem.Quality_{cur}$ have been improved. Otherwise, the effectiveness is judged by the improvement of the combined utility.

If the adaptation solution is evaluated to be effective, a new adaptation case is created with $problem$ as the problem, $goalConfig$ as the solution, and $Quality_{new}$ 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.

IV. Experimental Study

To evaluate the effectiveness of the proposed approach, we conduct an experimental study based on an online shopping system, whose goal model is shown in Figure 1, to answer the following two research questions:

- *Q1*: Can the proposed approach achieve improvements over self-adaptation approaches that involve only goal reasoning or case-based reasoning? (Comparative Evaluation)
- *Q2*: Can goal reasoning and case-based reasoning be effectively combined in the adaptation process of the proposed approach? (Process Analysis)

For Q1, we will compare the proposed approach with other two approaches involving only goal reasoning or case-based reasoning, to evaluate if the proposed approach could perform better. The comparison will be conducted in two ways: the effectiveness of the adaptation solutions and the improvement of response-time and utility of the system. For Q2, we will analyze how the proposed approach carries out during the self-adaptation process, to check if the combination of goal reasoning and case-based reasoning could really work well.

A. Experimental Setup

In the experiments, twelve environmental parameters and four quality attributes were monitored to facilitate runtime adaptations. Table I presents the environmental parameters considered in our experiments. These parameters were obtained by Linux's virtual memory statistics. The quality attributes for the softgoals of the online shopping system and their corresponding metrics are detailed as follows.

- The metric for the softgoal *Good Performance* is response time, which was measured by log analysis as the average response time of each request in an interval. For example, there are five requests with the response time being 100

TABLE I
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

ms, 200 ms, 150 ms, 300 ms and 250 ms, then the average response time is 200 ms.

- The metric for the softgoal *Good Usability* is user satisfaction, which was measured by a simulated real-life customer feedback analysis. Here we assumed that user satisfaction for a searching request was a random value between 0.0 and 0.5 if the variation point $VP1$ in Figure 1 was bound to *Brief Search*, and between 0.5 and 1.0 if the $VP1$ was bound to *Rich Search*. Similarly, we assumed that user satisfaction for a viewing request was a random value between 0.0 and 0.5 if the variation point $VP2$ was bound to *View in Text*, and between 0.5 and 1.0 if the $VP2$ was bound to *View in MM*. And the average of all the user satisfaction of both searching and viewing request will be the final quality metric. For example, if $VP1$ and $VP2$ are bound to *Brief Search* and *View in Text* respectively, and there are two searching 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 $\frac{0.2+0.3+0.1+0.2}{4} = 0.2$.
- The metric for the softgoal *High Malicious Order Checkout Rate* is the rate of identified malicious order in all malicious orders. Here we assumed that the alternative task *Simple Check* of $VP3$ can check out 50% malicious orders, *Complex Check* can check out 80% malicious orders, and each order has a 10% chance to be a malicious order. Besides, the control variable MAO determines which orders will be checked. For example, there are 100 orders and 10 of them are malicious orders. Among these 10 malicious orders, 8 orders' amount is higher than 1000 dollars. Given that $VP3$ is bound to *Simple Check* and the MAO is set to 1000 dollars, the malicious order rate will be $\frac{8*50\%}{10} = 0.4$.
- The metric for the softgoal *High Notification Success Rate* is the rate of successfully-notified orders in all orders. For example, there are 10 orders and 8 of them are successfully notified to logistics, then the notification success rate is 80%. Note that a notification can be successfully sent after several retries and the control variable NR determines the maximum numbers of retries.

For response time, we associated it with a quality constraint. We defined a combined utility of the other three quality attributes (i.e. user satisfaction, malicious order checkout rate, and notification success rate) as the weighted sum of their normalized values (between 0 and 1) for another quality constraint. And the weights of the three quality attributes were set to 0.4, 0.4, 0.2, respectively.

Measuring the values by monitoring quality attributes in each interval, the adaptation mechanism determines whether an adaptation is required for requirements violations based on the following two conditions:

- if the response time is larger than 1000 ms
- if the combined utility is smaller than 0.45 and the response time is smaller than 600 ms

Once one of the two conditions is met, the adaptation mechanism will determine and execute an appropriate reconfiguration to improve related quality attributes. We say an adaptation is *effective* if the violated requirements (the response time or utility) are improved in the following interval.

The behavior of the online shopping system has three variation points and two control variables (see Figure 1). In the experiments, we set the range of the control variables MAO and NR being 0 to 2000 and 0 to 10, with the changing pace being 200 and 1. The initial behavior of the system was set to [*Brief Search*, *View in Text*, *Simple Check*, *1000 (MAO)*, *5 (NR)*].

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

- *Goal-Based Approach*: The self-adaptation approach that uses goal reasoning to produce adaptation solutions.
- *Case-Based Approach*: The self-adaptation approach that uses case-based reasoning to produce adaptation solutions.
- *Proposed Approach*: The proposed self-adaptation approach that combines goal reasoning and case-based reasoning.

The original case base of the *Case-Based Approach* is constructed by goal reasoning, i.e. retaining adaptation solutions produced by goal reasoning as cases. And the case reuse strategy is to select and reuse the most similar case.

We conducted each of the experiments for 3.5 hours. The adaptation interval was set to one minute, i.e. the adaptation mechanism was periodically executed every one minute. In the experiment with the *Case-Based Approach*, the initial training phase for the construction of the original case base took about 55 minutes. In the experiment with the *Proposed Approach*, the two similarity thresholds $threshold_l$ and $threshold_h$ were set to 0.5 and 1.0 respectively. All the three experiments were conducted on the same server with a 4-core 3.1 GHz CPU and 8 GB RAM. We used JMeter to simulate concurrent system accesses and the same simulation script was applied to all the three experiments.

TABLE II
ADAPTATION FREQUENCY OF THE THREE APPROACHES

Approach	Adaptation (#)	Eff. (#)	Rate (%)
Goal-Based Approach	62	33	53
Case-Based Approach	48	31	65
Proposed Approach	49	35	71

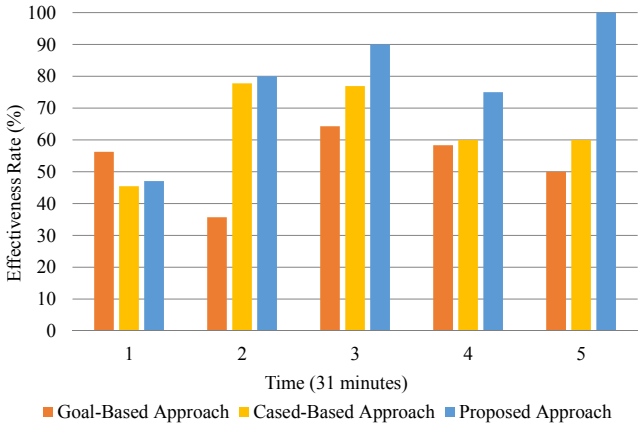

Fig. 5. Distribution of the Effectiveness Rate of the Three Approaches

B. Comparative Evaluation (Q1)

To fairly compare the effectiveness of the three approaches, we collected their frequency of adaptations which were triggered in the last 155 minutes of 3.5 hours (i.e. after the training phase of the *Case-Based Approach*). Table II shows the adaptation frequency of the three approaches. The first column lists the involved approaches, 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* triggered 62 adaptations, and 33 of them were evaluated to be effective; the *Case-Based Approach* triggered 48 adaptations, and 31 of them were evaluated to be effective; and the *Proposed Approach* triggered 49 adaptations, and 35 of them were evaluated to be effective. Moreover, the *Goal-Based Approach* achieved the lowest effectiveness rate (i.e. 53%), and the *Proposed Approach* achieved the highest effectiveness rate (i.e. 71%). This shows that it will be less effective if the adaptation mechanism only utilizes the rationalism knowledge underlying goal reasoning or the empiricism knowledge underlying case-based reasoning.

In addition, we divided the 155 minutes into five time slots, and analyzed the effectiveness rate of the adaptations of the three approaches during each time slot. Figure 5 shows the distribution of the effectiveness rate of the three approaches. The X axis denotes the time slot, and the Y axis denotes the effectiveness rate of the three approaches.

We can see that the *Case-Based Approach* mostly achieved a higher effectiveness rate than the *Goal-Based Approach* because it can provide more precise adaptation decisions, but it always achieved a lower effectiveness rate than the *Proposed Approach* because it lacks the rationalism support. In addition, the *Proposed Approach* always achieved the highest effectiveness rate except for the first time slots. This is because the *Proposed Approach* did not have a rich enough case base in the beginning to handle new emerging problems. However, in the long run, the *Proposed Approach* is more effective than both the *Case-Based Approach* and the *Goal-Based Approach* because it combines the rationalism knowledge underlying goal reasoning and the empiricism knowledge underlying case-based reasoning.

Furthermore, Figure 6 and 7 respectively show the response time and utility of the three approaches. The X axis denotes the adaptation interval, and the Y axis denotes the response time and utility per adaptation interval. The triangle/square marks on these curves mean that an effective/ineffective adaptation was triggered at that adaptation interval. The vertical dashed lines represent the end of the training phase of the *Case-Based Approach*. The horizontal dashed lines represent the threshold of response time (i.e. 1000 ms) and utility (i.e. 0.45).

We can observe from the curves that the *Proposed Approach* triggered the adaptations effectively in most times, i.e. either to reduce the response time, or to increase the utility at the price of performance degradation. The *Goal-Based Approach* and *Case-Based Approach* were less effective. Numerically, the rate of satisfying the constraint of response time (i.e. less than 1000 ms) in the last 155 minutes was 89% (138/155) for the *Proposed Approach*, 90% (140/155) for the *Goal-Based Approach*, and 90% (140/155) for the *Case-Based Approach*. On the other hand, the rate of achieving a better utility (i.e. larger than 0.45) when the response time is less than 1000 ms was 51% (79/155) for the *Proposed Approach*, 43% (67/155) for the *Goal-Based Approach*, and 44% (68/155) for the *Case-Based Approach*. This shows that the proposed approach outperforms *Goal-Based Approach* and *Case-Based Approach* in utility while keeping almost the same violation rate of the constraint of response time.

The above observations from Table II and Figure 5, 6 and 7 answer *Q1* positively that the proposed approach can improve the effectiveness of adaptations and the overall utility by combining the rationalism underlying goal reasoning and the empiricism underlying case-based reasoning.

C. Process Analysis (Q2)

To analyze the adaptation process of the proposed approach, we distinguish the reasoning mechanisms of adaptations. Specifically, adaptations can be generated by case-based reasoning (using the perfect case), or goal reasoning (using the best goal configuration), or synthesized reasoning (synthesizing the two most similar cases and the best goal configuration).

Table III shows the frequency of the adaptations of different reasoning mechanisms in the proposed approach in the whole 3.5 hours. The first column lists the reasoning mechanisms, the second and third list the number of adaptations and the number of effective adaptations, and the last column lists the effectiveness rate.

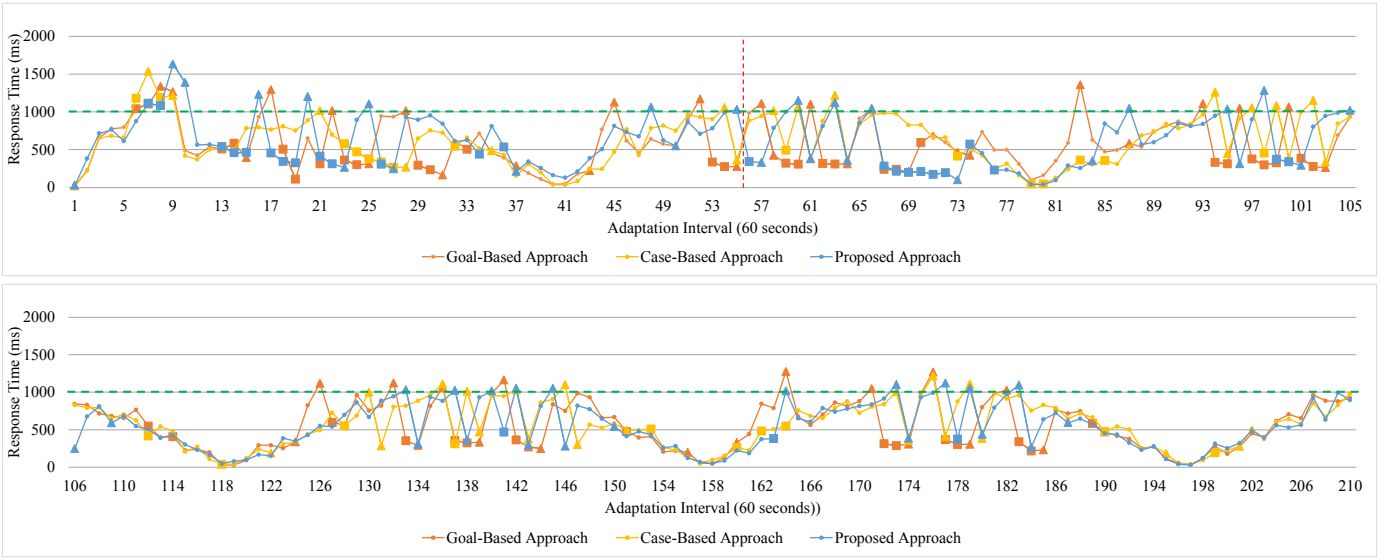

Fig. 6. The Self-Adaptation Process: Adaptation Interval vs. Response Time

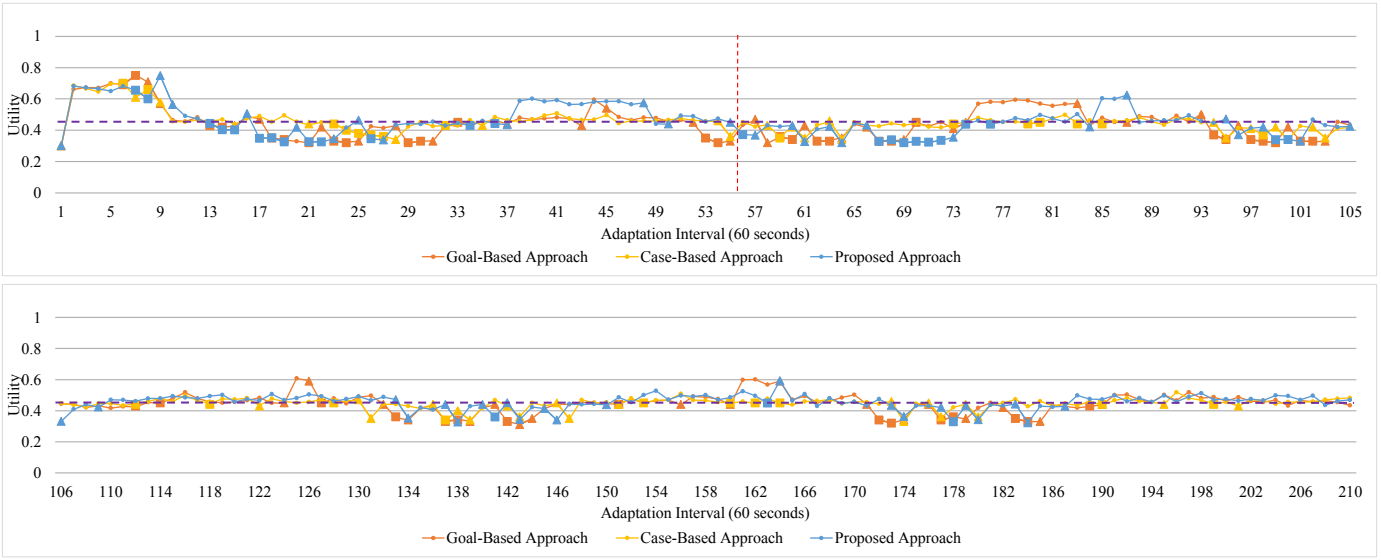

Fig. 7. The Self-Adaptation Process: Adaptation Interval vs. Utility

TABLE III
REASONING MECHANISMS USED IN THE ADAPTATIONS WITH THE PROPOSED APPROACH

Reasoning Mechanism	Adaptation (#)	Eff. (#)	Rate (%)
case-based reasoning	26	23	88
goal reasoning	41	17	41
synthesized reasoning	7	7	100

We can see that among the 74 adaptations, 26 adaptations were from case-based reasoning, 41 adaptations were from goal reasoning, and 7 adaptations were from synthesized reasoning. The adaptations from synthesized reasoning had the highest effectiveness rate (i.e. 100%), and the adaptations from goal reasoning had the lowest effectiveness rate (41%). Moreover, the effectiveness of case-based reasoning was improved (recalling that its effectiveness rate in the *Case-Based Approach* is only 65%) because the case base in the *Proposed Approach* could 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 was decreased (recalling that its effectiveness rate in the *Goal-Based Approach* is 53%) because the *Proposed Approach* would solve a similar problem using case-based reasoning or synthesized reasoning, which reduced the effectiveness rate of goal reasoning.

Furthermore, we divided the 3.5 hours into 7 time slots, and analyzed the distribution of the adaptations that were generated by the three reasoning mechanisms. The results are shown in Figure 8. For example, in the 3rd time slot, there were totally 15 adaptations, and 1 of them was generated by case-based reasoning, 2 of them were generated by synthesized reasoning and 12 of them were generated by goal reasoning.

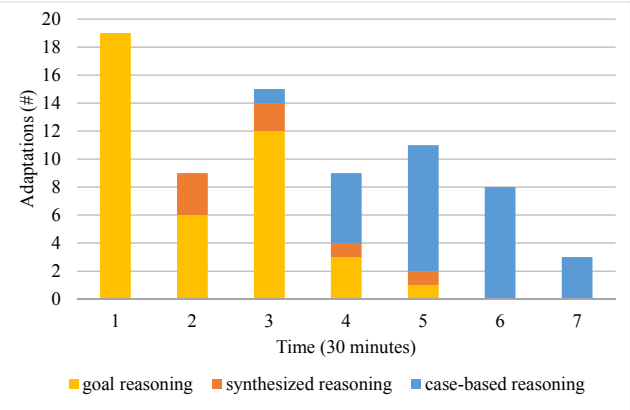

Fig. 8. Distribution of the Adaptations in the Proposed Approach

The trend of the distribution shows that in the beginning the generation of adaptation solutions mostly relied on goal reasoning because the case base was initially empty, and thus too small to handle the emerging problems; then gradually relied more on synthesized reasoning and case-based reasoning because the case base reached a certain size but was still not rich enough; and finally tended to rely on case-based reasoning because the case base became rich enough to handle the most emerging problems.

Specifically, such an adaptation process can be divided into the following three phases:

- **Initialization phase**: Because the case base is initially empty, the adaptation mechanism needs to utilize the rationalism knowledge underlying goal reasoning to help build the case base incrementally and sustainedly.
- **Accumulation phase**: After the initialization phase, there are several cases in the case base. Therefore, similar cases can be retrieved from the case base when there happens a new problem, and case-based reasoning is gradually used. However, because the case base is not rich enough, the retrieved cases may be not very similar to the new problem. As a result, the solutions of the two most similar cases will be synthesized with the solution generated by goal reasoning to get a suitable solution for the new problem. Therefore, both case-based reasoning and synthesized reasoning are gradually used to accumulate adaptations solutions to further enrich the case base.
- **Solving phase**: After the accumulation phase, the case base has been rich enough to handle the most emerging problems. As a result, most adaptation solutions are generated by case-based reasoning, and the goal reasoning and synthesized reasoning are gradually not used.

The above observations from Table III and Figure 8 answers *Q2* positively that the proposed approach properly combines goal reasoning and case-based reasoning for producing effective adaptation solutions. Specifically, when no similar cases can be found by case-based reasoning, either goal reasoning or synthesized reasoning can be utilized to produce effective adaptation solutions and thus enrich the case base. When the case base is rich enough, case-based reasoning is utilized to provide more precise adaptation decisions. In that sense, the three reasoning mechanisms are complementary to each other.

D. Discussion

It's 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 automatic learning of original cases, it is usually hard for case-based reasoning approaches to prepare a comprehensive set of original cases for a self-adaptive system.

Our approach uses a few constants and similarity thresholds, which are usually application-specific and can be specified by experts. Specifically, the setting of α in Algorithm 1 and the changing pace of control variables depends on the tradeoff between tolerance of system disturbance and effectiveness of adaptations. If it is set too large, goal reasoning will produce radical adaptations, probably leading to system disturbance. Otherwise, if it is set too small, goal reasoning will produce less effective adaptations. The setting of $threshold_h$ and $threshold_l$ affects the quality of the reused case in case-based reasoning. If they are set too large, more cases that are not very similar to the problem will be taken into account, which will worsen the solution from case-based reasoning. Otherwise, if they are set too small, less cases with higher similarity will be referenced, which limits the effectiveness of case-based reasoning. In the future, we will conduct sensitivity analysis of these parameters to empirically evaluate their impacts on the effectiveness of our approach

Last but not the least, ineffective cases may also be helpful for case-based reasoning for self-adaptation. Just as the positive experiences provided by effective cases, negative experiences contained in ineffective cases may also provide useful hints for future adaptation decision making, e.g. to avoid applying an adaptation solution that has been proved to be ineffective. Therefore, it is worthwhile to take such ineffective cases into account to see whether they can provide useful experiences for runtime adaptations.

V. Related Work

Requirements-driven self-adaptive systems are 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 [1]. Baresi et al. [2] propose an approach that introduces fuzzy and adaptive goals to embed adaptation countermeasures. Wang et al. [3] propose a self-repairing approach that uses goal reasoning to select a best system configuration. Dalpiaz et al. [4] 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 behaviour of a system to expected and allowed behaviours. Peng et al. [5] 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. [14] propose a requirements-driven self-optimization approach to achieve survivability assurance for Web systems. And their another work [15] proposes a self-adaptation approach that combines requirements and architectural adaptations with model transformation techniques. Salehie et al. [16] propose a requirements-driven approach to support adaptive security in order to protect variable assets at runtime. Fu et al. [17] propose a goal-based monitoring and self-repairing framework for socio-technical systems, which reasons about runtime failures and repairing solutions based on goal state machines and their interactions. Bencomo et al. [18] propose an approach that maps the goal models into Dynamic Decision Networks, and uses the Dynamic Decision Networks to automatically make the best decisions for self-adaptive systems.

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

Several attempts have been made to apply case-based reasoning in self-adaptive systems. Montani et al. [9] propose an approach that uses case-based reasoning to diagnose runtime failures of software systems and provide remediation solutions. McSherry et al. [7] propose a hypothesis-driven approach than can diagnose runtime failures and provides recovery suggestions by asking questions and explaining the relevance of the questions to users. Khan et al. [8] propose a self-adaptation approach using case-based reasoning, which can restrict the size of the case base without harming accuracy. 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 (may not be perfect) adaptation solutions with goal reasoning when no similar cases are available.

VI. 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 among system behaviours, 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 behaviour 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. The effectiveness of the approach has been evaluated by comparing with a goal reasoning approach and a case-based reasoning approach.

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. We also consider to apply the proposed approach in more complex Web-based systems to further evaluate its effectiveness and scalability.

VII. Acknowledgment

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

References

[1] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for RE for self-adaptive systems," in *RE*, 2010, pp. 95–103.

[2] L. Baresi, L. Pasquale, and P. Spoletini, "Fuzzy goals for requirements-driven adaptation," in *RE*, 2010, pp. 125–134.

[3] Y. Wang and J. Mylopoulos, "Self-repair through reconfiguration: A requirements engineering approach," in *ASE*, 2009, pp. 257–268.

[4] F. Dalpiaz, P. Giorgini, and J. Mylopoulos, "An architecture for requirements-driven self-reconfiguration," in *CAiSE*, 2009, pp. 246–260.

[5] 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*, vol. 85, no. 12, pp. 2707–2719, 2012.

[6] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Communications*, vol. 7, no. 1, pp. 39–59, 1994.

[7] D. McSherry, S. Hassan, and D. Bustard, "Conversational case-based reasoning in self-healing and recovery," in *ECCBR*, 2008, pp. 340–354.

[8] M. J. Khan, M. M. Awais, and S. Shamail, "Enabling self-configuration in autonomic systems using case-based reasoning with improved efficiency," in *ICAS*, 2008, pp. 112–117.

[9] S. Montani and C. Anglano, "Achieving self-healing in service delivery software systems by means of case-based reasoning," *Applied Intelligence*, vol. 28, no. 2, pp. 139–152, 2008.

[10] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Transactions on Software Engineering*, vol. 18, no. 6, pp. 483–497, 1992.

[11] P. Giorgini, J. Mylopoulos, E. Nicchiarelli, and R. Sebastiani, "Reasoning with goal models," in *ER*, 2002, pp. 167–181.

[12] V. E. S. Souza, A. Lapouchnian, and J. Mylopoulos, "System identification for adaptive software systems: A requirements engineering perspective," in *ER*, 2011, pp. 346–361.

[13] J. L. Kolodneer, "Improving human decision making through case-based decision aiding," *AI Magazine*, vol. 12, no. 2, p. 52, 1991.

[14] B. Chen, X. Peng, Y. Yu, and W. Zhao, "Are your sites down? Requirements-driven self-tuning for the survivability of Web systems," in *RE*, 2011, pp. 219–228.

[15] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao, "Self-adaptation through incremental generative model transformations at runtime," in *ICSE*, 2014, pp. 676–687.

[16] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh, "Requirements-driven adaptive security: Protecting variable assets at runtime," in *RE*, 2012, pp. 111–120.

[17] L. Fu, X. Peng, Y. Yu, J. Mylopoulos, and W. Zhao, "Stateful requirements monitoring for self-repairing socio-technical systems," in *RE*, 2012, pp. 121–130.

[18] N. Bencomo and A. Belaggoun, "Supporting decision-making for self-adaptive systems: From goal models to dynamic decision networks," in *REFSQ*, 2013, pp. 221–236.