2016 23rd Asia-Pacific Software Engineering Conference

Integrating Goal Model into Rule-based Adaptation

Tianqi Zhao*, Tao Zan†, Haiyan Zhao*, Zhenjiang Hu*† and Zhi Jin*
*Key Laboratory of High Confidence Software Technology, Ministry of Education, China
Institute of Software, School of EECS, Peking University, Beijing, 100871, China
{zhaotq12, zhhy, zhijin}@sei.pku.edu.cn
†National Institute of Informatics
The Graduate University for Advanced Studies
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430
{zantao, hu}@nii.ac.jp

***Abstract*—Goal-oriented adaptation provides a powerful mechanism to develop self-adaptive systems, enabling systems to keep satisfying user goals in a dynamically changing environment. The goal-oriented approach normally reduces the adaptation planning as a global optimization process and leaves the system the task of determining the actions required to achieve the goals. However, the high computation cost of global optimization prevents a self-adaptive system from quickly adjusting itself to the dynamically changing environment at runtime, which is intolerable since efficiency of planning is of utmost importance in most self-adaptive systems. On the other hand, rule-based adaptation has the advantage of efficient planning process since it predefines the adaptation logic by rules instead of leaving the system the task of reasoning. To combine the advantages of both approaches, we propose a novel adaptation framework that can integrate goal model into rule-based adaptation to make user goals to be better satisfied efficiently. We have applied the framework to design a self-adaptive e-commerce website. Our experimental results show that the proposed framework outperforms both the traditional goal-oriented approach and the traditional rule-based approach in terms of adaptation efficiency and effectiveness.**

I. Introduction

Goal-oriented adaptation ([1], [2], [3], [4]) provides a powerful mechanism to develop self-adaptive systems, enabling systems to keep satisfying goals in a dynamically changing environment. It normally reduces the dynamic adaptation as an *optimization process* and leaves the system the task to reason on the actions required to achieve high-level goals, and can be well combined with online learning process to improve the accuracy of adaptation decisions ([5], [6]).

Thanks to the automatic optimization and the learning process, goal-oriented adaptation is capable of dealing with uncertainty and keeping making optimal adaptation decisions even when unforeseen conditions occur. However, searching for optimal decisions is often computationally expensive and encounters less-efficiency problems. For a big system with very large searching space, the optimization process would become impossible since consumed time grows exponentially with the increases of system features. For example, the online e-commerce system in Figure 1b (adapted from [2]) has 26 configurable features, which would yield more than ten billion possible configurations in the searching space. If we would like to take all 26 features together with their enum values into account in the optimization process, it would take hours to search for optimal solutions for each adaptation.

Such less-efficiency problem would prevent a self-adaptive system from reacting timely to arising situations at run-time, which is not tolerable since efficiency of planning is of utmost importance in most self-adaptive systems ([7], [8]). For example, if the aforementioned e-commerce website cannot react timely when it suddenly encounters a traffic peak, its goal of "*response in time*" will fail and its customers might leave.

Moreover, too many features would make the learning process difficult. A self-adaptive system needs to learn from runtime behaviors to cope with the changing environment, and this process requires large numbers of observed data to infer an accurate model to support the optimization process. The more features are involved in the learning process, the more observed data is needed. For a sizable system with too many features, it is often difficult to collect sufficient observed data timely.

To overcome these limitations, we propose to shrink the searching space in the optimization process and the amount of data needed in the learning process by reducing the number of features. Our idea is to divide the whole features into two sets, so that the features in one set can be quickly handled by the goal-oriented adaptation to guarantee certain degree of goal satisfaction, while the features in the other will be handled by the rule-based adaptation ([9], [10], [11]) (which has the advantage of efficient planning process) based on the results got from the goal-oriented adaptation.

To this end, we should (1) find an effective way to divide the features so that the subset of the features we have chosen for the goal-oriented adaptation will lead to good goal satisfaction, and (2) guarantee that propagating the adaptation results to the rest features by the rule-based adaptation will do better, not bringing any bad effect on the results we have got.

Recall that an adaptation rule typically takes the form of "$condition \Rightarrow action$" where *condition* specifies the trigger of the rule, which is often fired as a result of a set of monitoring operations, and *action* specifies an operation sequence to perform in response to the trigger. For instance, in the e-commerce system, we may have the following rule

$$VerificationType = Strict \ \wedge \ Workload = High \Rightarrow PayLog := None; \ PayEncryption := None$$

1530-1362/16 $31.00 © 2016 IEEE
DOI 10.1109/APSEC.2016.55

saying that when "verification is strict" and "workload is high", we should turn off logging and encryption. Generally, the action of an adaptation rule might be conflict with user goals when the environments are changed, so it is not easy to combine the goal-oriented adaptation and the rule-based adaptation. Now which features shall be involved in the goal-oriented optimization? How can we guarantee that each individual rule *preserves* user goals with synergy between the goal-oriented and the rule-based adaptations?

Inspired by the idea in bidirectional model transformation [12] that is originated from the *view updating* mechanism in the database community [13], we propose a novel *view*-based adaptation framework that can seamlessly integrate these two adaptation approaches. Here *view* denotes the planning results of goal-oriented adaptation that will be preserved by rule-based adaptation. The main contribution of our research is summarized as follows:

- We present a novel view-based adaptation framework that can combine the advantages of both learning-based goal-oriented adaptation and the rule-based adaptation.
- The framework [1] has been implemented with a newly developed algorithm for automatic division of the whole feature sets.
- We have applied the framework to design a self-adaptive e-commerce website. Our experimental results show that our framework works more efficiently than the traditional goal-oriented approach, yielding comparative or even better goal-satisfaction degrees.

The rest of this paper is structured as follows. Section II introduces some preliminaries required before Section III presents our framework. Section IV and Section V detail the learning-based analyzing process and the integrated planning process respectively. Section VI evaluates the proposed framework. Section VII introduces and compares some related work before Section VIII draws our conclusions.

II. Preliminaries

In this section, we will introduce the models used in our framework.

A. Feature Model

The proposed framework captures the environment and system information using environment feature model (e.g., Figure 1a) and system feature model (e.g., Figure 1b) respectively.

A feature in the environment feature model specifies an environment characteristic, e.g., f_1 in Figure 1a specifies the workload level, while a feature in the system feature model specifies a system characteristic, for example, f_{24} in Figure 1b specifies the verification type.

Features in a feature model are organized as tree-like hierarchy through parental relationships as follows.

- Mandatory: Mandatory feature has to be included if its parent feature is selected

[1]The framework is available at http://www.prg.nii.ac.jp/members/stefanzan/viewrule.html.

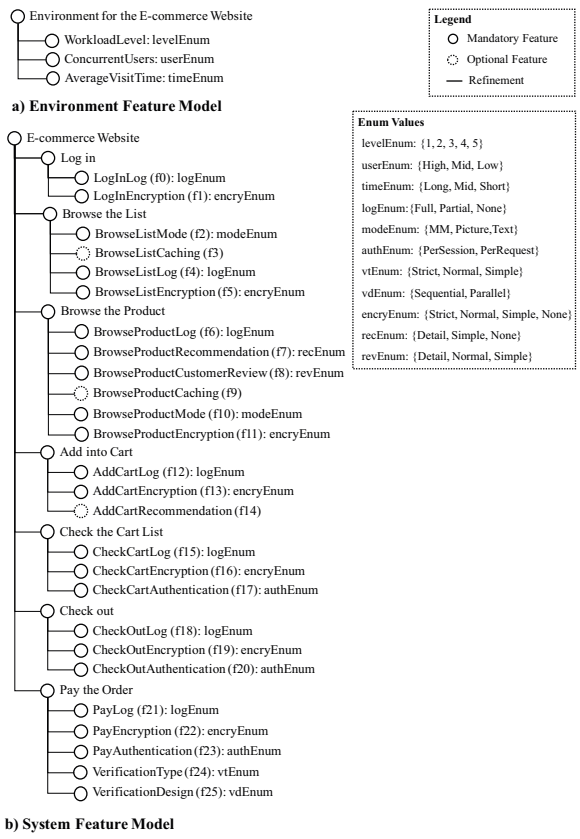


Fig. 1: Example Feature Models or the E-commerce Website

- Optional: Optional feature may be included if its parent feature is selected
- Alternative: Alternative features are organized in alternative group. Exactly one feature in such a group has to be selected if the group's parent is selected. In this paper, we treat the alternative features in a group as the alternative values of the group's parent feature.

A configuration of the system feature model describes a member of the product family defined by the feature model. In a configuration, each system feature is assigned one of its enum values. The alternative group's parent feature takes integer values, e.g., the value of "VerificationType" f_{23} can be "1" = "Strict", "2" = "Normal" or "3" = "Simple". Other features take on boolean values, i.e., a feature value can be either "1" = "selected" or "0" ="not selected".

A configuration of the environment feature model describes a possible environment state. Unlike the system features, the values of environment features cannot be configured by people or the control system. Their values can only be collected by monitor through sensors.

B. Non-functional Goals

Goals can be categorized into functional goals that underlie services the system is expected to deliver, and non-functional goals that refer to expected system qualities such as security, safety, performance and usability [14]. Since functional goals

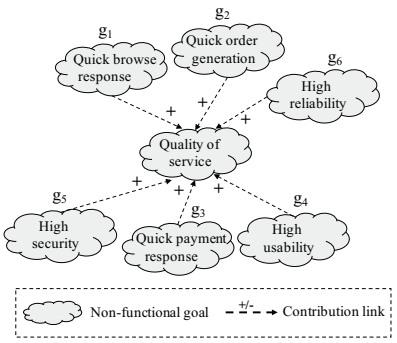


Fig. 2: An Example Non-functional Goal Model

are normally considered as mandatory goals in self-adaptive systems, we focus on non-functional goals in this paper.

When making adaptation plans, non-functional goals are referred as criteria for comparing system configurations and choosing one that yields a highest satisfaction value of the root non-functional goal (e.g., "Quality of service" is the root non-functional goal in Figure 2). The satisfaction value of a non-functional goal ranges from 0 to 100, and can be calculated based on its related metric using a utility function. For example, the related metric of g_3 (shown in Figure 2) is "payment response time" (denoted as m_3); the utility function describes a negative correlation between m_3 and g_3, where a longer response time yields a lower satisfaction value.

III. View-based Adaptation Framework

Figure 3 depicts the proposed adaptation framework, which has the capability to seamlessly integrate goal-oriented and rule-based adaptation. Here we introduce the concept *view* to denote the results of goal-oriented planning, and name the proposed framework as the *view*-based adaptation framework. This view-based adaptation framework follows the MAPE-K adaptation Loop [7], which stands for *Monitor*, *Analyzer*, *Planner* and *Executor* based on *Knowledge*.

In the loop, *Monitor* collects information from the environment and the system, and detects whether changes occur.

Analyzer works to find out which types of environment changes occur, and then notify the results to *Planner*. To facilitate the integrated planning process, *Analyzer* is responsible to divide the system features (**SFs**) into two sets, i.e., the critical system features (**CSFs**) and the non-critical system features (**NSFs**). The critical system features have significant impact on goals, while the non-critical system features do not have such significant impact.

Planner integrates goal-oriented planning and rule-based planning, in the sense that

- Goal-oriented planning will be triggered when either the goal setting (goals and their priorities) has been tuned or critical environment changes occur. When triggered, goal-oriented planning searches the configuration space of **CSFs** to work out *view*, which is the optimal configuration of **CSFs** given the current goal setting.
- Rule-based planning will be triggered whenever changes activate a rule in the adaptation rules set. This process activates an adaptation rule only if the rules's condition is true and action preserves *view*. In this way, rule-based planning can propagate changes to the whole system feature model while preserve *view*, and therefore preserve the goals.

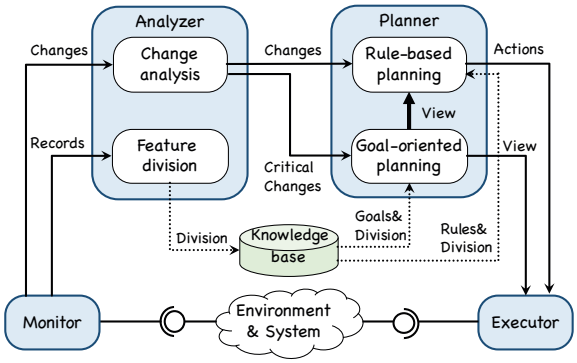


Fig. 3: Overview of View-based Adaptation Framework

Executor is to enforce the plan derived from *Planner* to implementation. It enforces changes at feature level according to the planning results firstly. The changes will then be traced to architecture level changes through model to model transformation ([15], [16]), and/or to implementation level changes through model to code transformation [17].

The detailed analyzing and planning process will be further elaborated in Section IV and Section V respectively.

IV. The Learning-based Analyzer

In this section, we describe the two major processes *Analyzer* performs: 1) *change analysis* that decides the type of environment changes; 2) *feature division* that divides the system features into CSFs and NSFs.

A. Change Analysis

Monitor collects information from the environment and then detects whether changes occur. For example, when *Monitor* has sensed that the workload is in level 5, a change will be detected if the original value of "WorkloadLevel" is not 5.

Analyzer analyzes the changes detected by *Monitor* to decide which types of environment changes occur. Environment changes can be categorized into two types: critical changes that have explicit impact on the system behavior patterns, and non-critical changes that do not have such explicit impact. For example, the change of "WorkloadLevel" is categorized as a critical change since it has explicit impact on the behavior patterns of the website, in the sense that the website could behave well with strict payment verification type and multimedia browse mode when the workload is in low level, but would probably encounter slow response problem when the workload is in high level.

After that, *Analyzer* informs *Planner* its change analysis results, which will trigger different planning process.

B. Feature Division

To facilitate the integrated planning process, *Analyzer* divides the configurable system features set into two sets: CSFs to be handled by the goal-oriented planning process, and NSFs to be handled by the rule-based planning process. CSFs are system features that have significant impact on user goals, while NSFs do not have such significant impact.

To decide whether such "significant" impact exists between system features and goals, the feature division process runs *significance tests* based on records collected by *Monitor*.

A record consists of the observed system feature values and goal satisfaction values at a specific time point. While a feature takes integer value, a goal value is a float that ranges between 0 and 100. Table I shows some example records for our running example, where a record consists of the values of 26 system features (f_1 to f_{26} in Figure 1b) and the satisfaction degrees of 6 non-functional goals (g_1 to g_6 in Figure 2) at a specific time point. The meaning of feature values and the calculation of goal values can be referred to Section II.

TABLE I: Collected Records for Significance Tests

No.	f_0	f_1	f_2	f_3	f_4	...	g_1	g_2	g_3	...
Rec.0	3	3	1	2	2	...	84.9	49.8	7.8	...
Rec.1	2	1	3	2	3	...	83.2	62.2	64.3	...
Rec.2	2	1	1	2	3	...	68.8	70	60	...
Rec.3	2	3	1	2	2	...	89.6	68.4	89.7	...

Significance tests are executed based on the collected records. A significance test will be performed for each goal, and the significance test for goal g_i is conducted according to the following steps:

1. Specify a significance level α;
2. State the model $v_{g_i} = \sum_{j=0}^{m} \beta_j \times v_{f_j}$, where v_{g_i} denotes the satisfaction degree of goal g_i; v_{f_j} denotes the value of feature f_j; and β_j denotes the effect of feature f_j on goal g_i;
3. State the null hypotheses for each feature. The null hypotheses for feature f_j is that v_{f_j} does not have a significant effect on v_{g_i};
4. Compute p value (indicates the probability of the null hypotheses) for the null hypotheses of each feature. The detailed computation can be referred to [18].

Feature f_j is regarded as a significant feature for g_i only if its p-value is less than α. If a feature has significant impact on at least one goal, it will be categorized as a critical system feature. Table II demonstrates an example feature division result for our running example, where "$*$" indicates that a feature has significant impact on the corresponding goal, e.g., f_{13}, f_{16} and f_{24} have significant impact on g_5 while others don't. In this case, the CSFs include f_2, f_3, f_9, f_{10}, f_{13}, f_{16}, f_{24} and f_{25}, while NSFs include the other 18 features.

After finding out CSFs, *Analyzer* will estimate the relationship between goals and CSFs as functions (namely, goal-feature functions) to support the further goal-oriented adaptation. The estimation is conducted based on records collected by *Monitor*,

TABLE II: Results of Significance Tests

	CSFs								NSFs	
	f_2	f_3	f_9	f_{10}	f_{13}	f_{16}	f_{24}	f_{25}	f_0	...
g_1	*			*						...
g_2					*	*				...
g_3							*	*		...
g_4	*			*						...
g_5					*	*	*			...
g_6		*	*							...

where a record consists of the goal values and the CSF values. The estimation does not tie to specific techniques, and can take use of techniques like the linear regression, neutral network and m5 model tree [1], and also other machine learning techniques.

Back to our running example, suppose polynomial regression is chosen, the goal-feature function for g_1 can be learnt through polynomial regression with its critical features f_2 and f_{10} as predictor variables. In this case, the goal-feature function is more likely a segmented function since the impact from features to goals is considerable influenced by the workload level.

$$v'_{g_1} = \begin{cases} 9.41 + 17.91 \times v_{f_2} + 11.09 \times v_{f_{10}} & workloadLevel < 3 \\ 0.91 + 4.37 \times v_{f_2}^2 + 3.65 \times v_{f_{10}}^2 & workloadLevel \geq 3 \end{cases}$$

Similarly *Analyzer* estimates goal-feature functions for other goals.

$$\begin{cases} v'_{g_2} = 9.75 + 1.06 \times v_{f_{13}} + 9.15 \times v_{f_{16}} \\ v'_{g_3} = 48.81 + 16.49 \times v_{f_{24}} - 3.67 \times v_{f_{25}} \\ v'_{g_4} = 72.98 - 12.63 \times v_{f_2} - 11.65 \times v_{f_{10}} \\ v'_{g_5} = 143.69 - 23.20 \times v_{f_{24}} - 12.27 \times v_{f_{13}} - 11.2 \times v_{f_{16}} \\ v'_{g_6} = -85.93 + 46.51 \times v_{f_3} + 44.51 \times v_{f_9} \end{cases}$$

V. The Integrated Planner

One distinguished characteristic of our view-based framework is the integration of *goal-oriented planning* and *rule-based planning* in the sense that the former is responsible for global optimization and the latter is for local adaptation based on rules.

In this section, we will detail the process of *goal-oriented planning* and *rule-based planning* respectively.

A. Goal-oriented Planning

Normally, goal-oriented planning reduces the dynamic adaptation as an optimization process. It searches the configuration space of feature model (or architecture model) to reason on an ideal configuration that can best achieve goals. However, searching for optimal solutions is often computationally expensive, and even become impossible for big systems.

In our framework, goal-oriented planning works on CSFs instead of the whole feature set. Since only CSFs are involved in this process and NSFs are irrelevant, the feature searching space has been greatly shrunk, which yields a much efficient searching process.

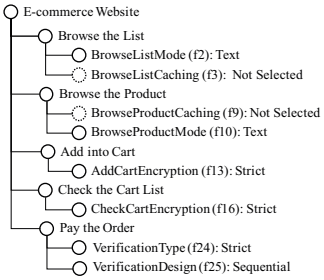


Fig. 4: An Example View of the E-commerce Website

1) The Planning Method: Goal-oriented planning will be triggered by the changes of goal setting and the critical environment changes. A goal setting is comprised of a set of goals and their weights, denoted as

$$G = \{(g_1, \omega_{g_1}), ..., (g_n, \omega_{g_n})\}$$

where g_i is the ith goal and ω_{g_i} is the weight of g_i. ω_{g_i} can be customized by users and be tuned dynamically at run time. When the value of ω_{g_i} is customized as zero, g_i is considered to be irrelevant.

When triggered, the goal-oriented planning process searches the configuration space of CSFs to reason on *view* (denoted as V^*), an ideal configuration that can yield a highest goal satisfaction degree. Defining the overall goal satisfaction degree as a weighted sum of sub goal satisfaction degrees, we got the following optimization problem:

$$V^* = \arg\max_{V} \sum_{(g,\omega_g)\epsilon G} \omega_g \times v'_g(V)$$

where V is a possible configuration of CSFs, i.e., a collection of selected CSF values, and $v'_g(V)$ represents the predicted value of v_g given V by using the learnt goal-feature functions.

Solving this optimization problem yields *view*. *View* will always be preserved until another goal-oriented planning process triggered.

2) Example: Back to our running example, suppose the weights of the six goals are customized as 0.2, 0.15, 0.15, 0.2, 0.2 and 0.1 respectively. An optimization problem will be generated for this case,

$$V^* = \arg\max_V (0.2v'_{g_1}(V) + 0.15v'_{g_2}(V) + 0.15v'_{g_3}(V) \\ +0.2v'_{g_4}(V) + 0.2v'_{g_5}(V) + 0.1v'_{g_6}(V))$$

where v'_{g_i} is the prediction of v_{g_i} that has been learnt in Section IV-B. Here only 8 CSFs (as stated in Section IV-B) are involved in this searching and optimization process, while the other 18 NSFs are irrelevant. Solving this optimization problem given "workloadLevel $\geq$ 3" results in the following solution:

$$V^* = \{(f_2, 3), (f_3, 2), (f_9, 2), (f_{10}, 3), (f_{13}, 1), \\ (f_{16}, 1), (f_{24}, 1), (f_{25}, 1)\}$$

The resulted *View* has been demonstrated in Figure 4, which is a configuration of CSFs that can maximize the goal satisfaction degree given the current goal setting. V^* shall be kept until the next goal-oriented planning process.

B. Rule-based Planning Process

Normally, the adaptation logic of rule-based planning is prescribed by a set of adaptation rules. An adaptation rule takes the form of "$condition \Rightarrow action$", where *condition* specifies the trigger of the rule, which is often fired as a result of a set of monitoring operations, and *action* specifies an operation sequence to perform is response to the trigger. Generally, the action in an adaptation rule might conflict with user goals when the environment or the goal setting has changed unexpectedly, making it not easy to combine the goal-oriented adaptation and the rule-based adaptation.

In our *view*-based framework, rule-based planning is slightly different from that of the traditional rule-based adaptation, in the sense that it will preserve *view* and user goals.

1) The Planning Method: The format of adaptation rules in our framework is the same as traditional adaptation rules, while the trigger of rules is slightly different from that in traditional rule-based adaptation. Here an adaptation rule "$condition \Rightarrow action$" will be triggered under the following conditions:

a The *condition* of rule is true with respect to the monitoring results, which is also the trigger condition of the traditional adaptation rules. It worth noting that a rule can only be triggered when its condition does not conflict with current V^*, since goal-oriented planning has already enforced system changes based on V^*.

b The *action* of rule does not do harm to V^*, which is to guarantee that the rule execution will preserve V^* and enforce changes only on NSFs.

In our rule-based planning process, the triggered rule set is selected not only based on the current environment, but also based on V^*. In this way, it can guarantee that each triggered individual rule *preserves* view and user goals with synergy between goal-oriented and the rule-based adaptations.

2) Example: Returning to our running example, assume the following three rules in the rule set.

(r1) $ConcurrentUsers = Low \wedge WorkloadLevel = 1$
$\wedge VerificationType = Normal$
$\Rightarrow VerificationDesign := Sequential, PayLog := Full$

(r2) $ConcurrentUsers = High \wedge VerificationType = Strict$
$\Rightarrow VerificationDesign := Parallel$

(r3) $ConcurrentUsers = High \wedge WorkloadLevel = 5$
$\wedge BrowseProductMode = Text$
$\Rightarrow BrowseProductLog := Partial$

Let's imagine a scenario where view V^* is as Figure 4 shows, and the environment state is monitored as {"WorkloadLevel=5", "ConcurrentUsers=High", "AverageVisitTime=Long"}. In this scenario, the condition of $r1$ is not true with respect to the environment state and therefore cannot be triggered. Although the condition of $r2$ is true, its action does harms to "f_{25}=1" ("VerificationDesign=Sequential") in V^* so it cannot be triggered. Accordingly, $r3$ is the only rule that should be triggered at this point, so the action of $r3$ is enforced to propagate changes: f_6:=2 ("BrowseProductLog:=Partial").

VI. Evaluation

We aim to answer the following questions:

Q1: Can our approach improve the planning efficiency (Section VI-B)?

Q2: Can our approach improve the planning quality (Section VI-C)?

A. Experiment Design

We have conducted the evaluation based on two e-commerce websites:

- *Ecommer-26*: The e-commerce website with 26 configurable system features (as shown in Figure 1b).
- *Ecommer-18*: The e-commerce website with 18 configurable system features (removing 8 configurable features in Figure 1b).

We have established mapping relationship between the system feature model and system implementation by mapping each system feature value to a code fragment. When a system feature is assigned a specific feature value by *Planner*, the corresponding code fragment of the selected feature value would be automatically weaved to the system implementation by *Executor*.

The e-commerce websites are deployed on a ThinkCentre PC with Intel Core i7 3.60 GHz processor and 8 GB RAM, while the visiting traffic is generated by the PC and a MacBook Air with Intel Core i5 1.8 GHZ processor and 4GB RAM. The traffic is simulated using the stress testing tool JMeter installed in both computers.

The websites may face two kinds of changes at runtime: 1) changes of the environment, i.e., changes of workload, the number of concurrent users and etc; 2) changes of the goal setting, i.e., changes of the weights (priority) of goals.

The adaptation objective of the websites is to optimize the goal satisfaction degree in such dynamically changing environment. The overall goal satisfaction degree here is calculated as a weighted sum of the satisfaction degree of 6 non-functional goals (g_1 to g_6 in Figure 2). Each goal is related with metrics based on which its satisfaction degree can be calculated using a utility function (detailed in Section II-B): g_1 with browse response time, g_2 with order generation time, g_3 with payment response time, g_4 with usability related feature values (including feature "BrowseProductMode", "BrowseProductRecommendation" and etc), g_5 with security related feature values (including feature "VerificationType", "PayAuthentication" and etc), and g_6 with reliability related feature values (including "PayLog", "BrowseProductCaching" and etc).

For comparison, we have imposed different adaptation logics into this e-commerce website:

- **Random**: A base line approach that randomly assigns values to configurable system features.
- **GoalAdap**: The traditional goal-oriented adaptation that searches for a system configuration to optimize goals.
- **RuleAdap**: The traditional rule-based adaptation that is supported by predefined adaptation rules.

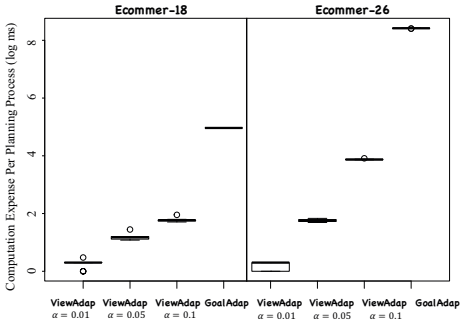


Fig. 5: Time Consumption of the Planning Process

- **ViewAdap**: The proposed view-based adaptation approach that combines **GoalAdap** and **RuleAdap**.

In this experiment, we have implemented an online learning process both in **GoalAdap** and **ViewAdap**, where the goal-feature relationship is learnt from the runtime environment to support the goal-oriented planning.

B. Efficiency Evaluation (Q1)

To evaluate whether the proposed approach improves efficiency, we set the websites in an adaptation scenario where workload changes randomly, and apply **ViewAdap** and **GoalAdap** to make adaptation plans. Here we apply three **ViewAdap** approaches with significance level α set as 0.01, 0.05 and 0.1 respectively. Recall that α greatly impacts the number of CSFs and therefore the performance of **ViewAdap**.

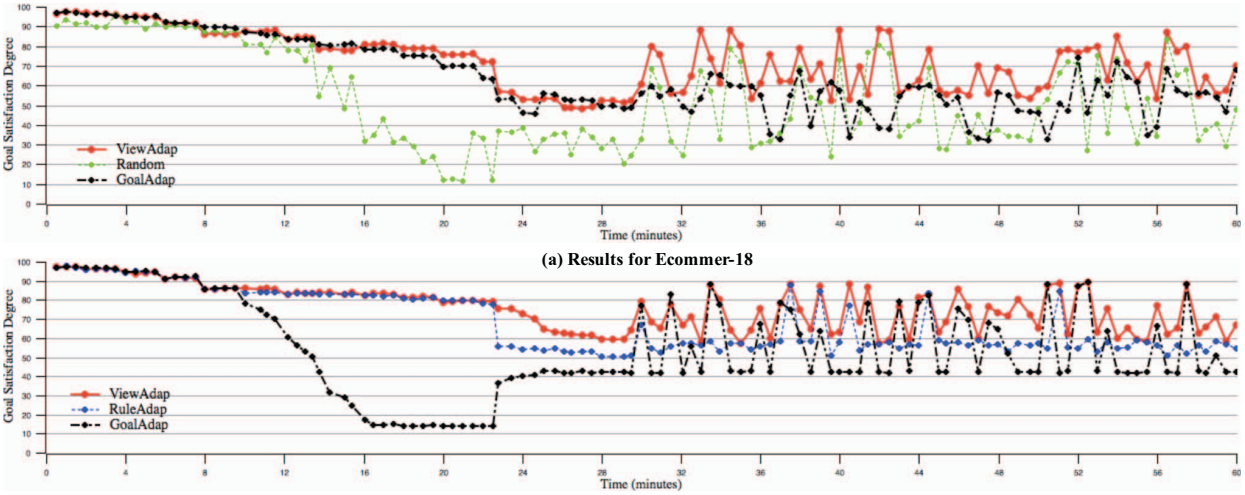
We record the time consumed by each planning process, and summarize the time consumption using box plot as shown in Figure 5. Here the vertical axis represents the computation expense per planning process, which is a logarithm result of the time consumption (in ms) per planning process, for the ease of visibility. The left part of Figure 5 reports the results of *Ecommer-18* while the right part reports the results of *Ecommer-26*. The results show that all the three **ViewAdap** approaches are much more efficient than **GoalAdap**. Precisely, **GoalAdap** spends an average of 92164ms for *Ecommer-18* and an average of 260561437ms for *Ecommer-26* to make an adaptation decision, which is too long and can prevent the self-adaptive system from reacting timely to arising situations at run-time.

Conclusion. The proposed approach can improve the planning efficiency of **GoalAdap**.

C. Effectiveness Evaluation (Q2)

To evaluate whether the proposed approach can improve the adaptation quality, we set an adaptation scenario based on the e-commerce websites, and apply **Random**, **RuleAdap**, **GoalAdap** and **ViewAdap** (α=0.05) to handle them respectively.

Figure 6 reports the experiment results, where the changes occur as follows: 1) the number of concurrent users firstly increases from 0 to 300 in 30 minutes (denoted as *Normal Changing Traffic*), then fluctuates between 100 and 300 for another 30 minutes (denoted as *Random Changing Traffic*); 2) the goal setting is initialized to $[0.35, 0.25, 0.25, 0.05, 0.05, 0.05]$,



(a) Results for Ecommer-18

(b) Results for Ecommer-26

Fig. 6: Goal Satisfaction Degrees in the Adaptation Process

then switches to [0.16, 0.16, 0.25, 0.16, 0.16, 0.16] in the 22nd minute.

In the two plots of Figure 6, the horizontal axis represents the time points, and the vertical axis represents the overall goal satisfaction degrees.

Figure 6a reports the results of **ViewAdap**, **Random** and **GoalAdap** when handling *Ecommer-18*. In the first 30 minutes (with *Normal Changing Traffic*), **ViewAdap** and **GoalAdap** both yield high goal satisfaction degrees and outperform **Random**. Although **GoalAdap** spends about 90 seconds every planning process, it still leads to good results since the workload changes gradually and only small changes can happen in 90 seconds. In the second 30 minutes (with *Random Changing Traffic*), **GoalAdap** performs poorly since unexpected environment changes might happen before its planning process completes and therefore it cannot react to the changes timely. On the contrary, **ViewAdap** can still yield good goal satisfaction degrees, since it is quite efficient and therefore can make timely adaptation decisions.

Figure 6b reports the results of **ViewAdap**, **RuleAdap** and **GoalAdap** when handling *Ecommer-26*. Since the performance of **RuleAdap** greatly depends on the quality of the prescribed adaptation rules, we carefully designed a set of rules according to the initial goal setting. At first, **ViewAdap** and **RuleAdap** lead to comparable goal satisfaction degrees. However, the results of **RuleAdap** drop dramatically after the switch of goal setting in the 22nd minute. This is because the rule set is designed for the initial goal setting that states a higher priority of g_1, g_2 and g_3, and does not fit the new goal setting that assigns all sub goals the same priorities. This reveals the drawback of the traditional rule-based adaptation, which is the static adaptation logic that cannot react to changes of the goal setting. In this scenario, **GoalAdap** performs poorly since it takes more than one day to complete a goal-oriented process, while the experiment only lasts for 1 hour. Therefore, **GoalAdap** did not complete even 1 planning process in this experiment, leading to the results equal to a static approach that never changes its configuration.

The experiment results show that **ViewAdap** outperforms **RuleAdap** since it can react to dynamic changing goal setting, and outperforms **GoalAdap** since its more efficient planning process.

Conclusion. Our approach can lead to comparable or even better adaptation quality than **GoalAdap**.

VII. Related Work

Many efforts have been done on the self-adaptation ([7, 8]). Among them, the most related includes the MAPE-K framework ([19, 20]), the goal-oriented and rule-based adaptation, and the dynamic product lines with feature models. Our work follows the MAPE-K framework but enriches the planning part by integrating the goal-oriented and rule-based planning.

Goal-oriented modelling approaches have been adopted widely in adaptive systems ([21, 22]). [23] uses goal model to specify adaptive requirements, including inferring runtime monitors, and [24] introduces a variability-intensive approach. In [25], user goals are refined into alternative functionalities and a decision-making process is deployed to generate system design to fulfill user goals. Some works ([21, 22]) use utility function to describe the dynamic variability for enabling the online decision-making. Elkhodary et.al [1] proposes a learning based feature-oriented self-adaptation framework FUSION, which prunes the feature space by inter-feature relationship. G.Ghezzi et.al [3] proposes a framework that supports adaptation to maximize the system's ability to meet its non-functional requirements. Chen et.al [2] propose to combine goal-driven self-adaptation and architecture-based adaptation. Qian et.al [6] propose a requirements driven self-adaptation approach that combines goal reasoning and case-based reasoning. These goal-oriented approaches provide solutions to enable adaptation plans matching with changed goals. Our approach integrates goal-oriented planning with rule-based planning approach, which significantly reduce the searching space and therefore dramatically lower the execution

cost of goal-oriented adaptation.

Rule-based adaptation can facilitate efficient adaptation. Rainbow [10] uses rules to specify the desired adaptive behavior. A rule-based framework for self-adaptation was proposed in [26], with the consideration of separation between the application and the adaptation logic. Acher et.al ([11, 27]) use adaptation rules to configure the adaptive system with respect to a particular context. However, the traditional rule-based adaptation cannot well support rule evolution to cope with requirements changes and unexpected environment changes. Our work integrates goal model into rule-based adaptation, which makes the adaptation logic can be dynamically updated.

Dynamic software product line (DSPL) and feature models have provided opportunities to dynamic adaptive systems. Many works ([28, 16, 29]) have been devoted to adopting DSPL to enable self-adaptation by using feature models as the variability model. These works, in some extent, bridge the gap between features and architecture by various approaches. Our work takes the advantage of the properties of context-awareness, and resource-aware decision-making, and consistent dynamic reconfiguration provided by DSPL.

Our work was inspired by the idea in bidirectional transformation ([12, 30, 31, 32]). Bidirectional transformation is a new mechanism for maintaining the consistency of two different information originated from the *view updating* mechanism in the database community ([13]). Different from the existing approaches, this work can be considered as the first attempt to construct adaptive systems in a view-based adaptation way.

VIII. Conclusion

We propse a novel approach to combining the goal-oriented and rule-based adaptation for tackling the inefficiency issue in the traditional goal-oriented adaptation. Our novel view-based framework shows synergy between *goal-oriented planning* and *rule-based planning*, where the former is responsible for global optimization (on a subset of important features) and the latter is for local adaptation based on the adaptation logic. Thus, it combines the strengths of the rule-based and goal-oriented adaptation approaches and enjoys the advantages of both. The application to the construction of an online self-adaptive shopping system shows that our approach outperforms the existing goal-oriented and rule-based approaches.

As a future work, it should be interesting to consider other realistic context models by taking account of behavior of system components. This would make adaptation actions be applicable to wider realistic situation.

IX. Acknowledgement

This research is supported by the National Basic Research Program of China (the 973 Program) under Grant No. 2015CB352201 and the National Natural Science Foundation of China under Grant Nos. 61620106007, 91318301, 61432020, and 61272163.

References

[1] M. Acher, P. Collet, F. Fleurey, P. Lahire, S. Moisan, J.-P. Rigault, and et al. Modeling context and dynamic adaptations with feature models. In *Proceedings of the 4th International Workshop on Models@run.time*, 2009.

[2] M. Acher, P. Collet, P. Lahire, S. Moisan, and J.-P. Rigault. Modeling variability from requirements to runtime. In *Proceedings of 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2011.

[3] L. S. Aiken, S. G. West, and S. C. Pitts. Multiple linear regression. *Handbook of psychology*, 2003.

[4] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 1981.

[5] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Muller, M. Pezze, and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70. 2009.

[6] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Autonomic computing through reuse of variability models at runtime: The case of smart homes. *IEEE Computer*, 42(10):37–43, 2009.

[7] B. Chen, X. Peng, Y. Yu, B. Nuseibeh, and W. Zhao. Self-adaptation through incremental generative model transformations at runtime. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India*.

[8] B. H. Cheng and et al. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, pages 1–26. Springer-Verlag, 2009.

[9] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *ICMT2009*, 2009.

[10] R. de Lemos, H. Giese, H. A. Müller, and et. al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany*, 2010.

[11] S. Dobson, S. Denazis, and et al. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems*, 2006.

[12] A. Elkhodary, N. Esfahani, and S. Malek. Fusion: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the 18th international symposium on Foundations of software engineering*. ACM, 2010.

[13] N. Esfahani, A. M. Elkhodary, and S. Malek. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Transaction of Software Engineering*, 2013.

[14] M. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behaviour. In *Proceedings of the 9th international workshop on Software specification and design*, pages 50–59, 1998.

[15] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven. Using architecture models for runtime adaptability. *IEEE Software*, 23(2):62–70, 2006.

[16] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems*, 29(3):17, 2007.

[17] D. Garlan, S. Cheng, A. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *IEEE Computer*, 2004.

[18] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli. Managing non-functional uncertainty via model-driven adaptivity. In *35th International Conference on Software Engineering, USA*, 2013.

[19] S. A. Hendrickson and A. Van der Hoek. Modeling product line architectures through change sets and relationships. In *Proceedings of the 29th international conference on Software Engineering*, 2007.

[20] Z. Hu, A. Schürr, P. Stevens, and J. F. Terwilliger. Dagstuhl Seminar on Bidirectional Transformations (BX). *SIGMOD Record*, 40(1):35–39, 2011.

[21] L. Kapova and T. Goldschmidt. Automated feature model-based generation of refinement transformations. In *Software Engineering and Advanced Applications. SEAA'09.*

[22] J. O. Kephart and R. Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 2007.

[23] v. A. Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *International Symposium on Requirements Engineering*, 2001.

[24] I. Lanese, A. Bucchiarone, and F. Montesi. A framework for rule-based dynamic adaptation. In *TCG 2010: Lecture Notes on Computer Science 6084*.

[25] I. Lanese, A. Bucchiarone, and F. Montesi. A framework for rule-based dynamic adaptation. In *International Conference on Trustworthly Global Computing*, 2010.

[26] S. Liaskos, A. Lapouchnian, Y. Yu, E. Yu, and J. Mylopoulos. On goal-based variability acquisition and analysis. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 79–88, 2006.

[27] B. Morin, O. Barais, J. Jezequel, F. Fleurey, and A. Solberg. Models@run.time to support dynamic adaptation. *IEEE Computer*, 42(10):44–51, 2009.

[28] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao. Rationalism with a dose of empiricism: Case-based reasoning for requirements-driven self-adaptation. In *22nd International Requirements Engineering Conference*, 2014.

[29] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh. Requirements-driven adaptive security: Protecting variable assets at runtime. In *20th International Requirements Engineering Conference (RE), USA*, 2012.

[30] W. Yu, W. Zhang, H. Zhao, and Z. Jin. Tdl: A traceability description language from feature model to use case for automated use case derivation. In *18th International Software Product Line Conference*, Florence, Italy.

[31] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. S. P. Leite. From goals to high-variability software design. In *Foundations of Intelligent Systems*, 2008.

[32] Y. Yu, Y. Lin, Z. Hu, S. Hidaka, H. Kato, and L. Montrieux. Maintaining invariant traceability through bidirectional transformations. In *34th International Conference on Software Engineering*, 2012.