This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

Architecture-Based Behavioral Adaptation with Generated Alternatives and Relaxed Constraints

Bihuan Chen, Xin Peng, Yang Liu, Songzheng Song, Jiahuan Zheng, and Wenyun Zhao

Abstract—Software systems are increasingly required to autonomously adapt their architectural structures and/or behaviors to runtime environmental changes. However, existing architecture-based self-adaptation approaches mostly focus on *structural* adaptations within a *predefined* space of architectural alternatives (e.g., switching between two alternative services) while merely considering *quality* constraints (e.g., reliability and performance). In this paper, we propose a new architecture-based self-adaptation approach, which performs *behavioral* adaptations with *automatically generated* alternatives and supports *relaxed functional* constraints from the perspective of business value. Specifically, we propose a technique to automatically generate behavioral alternatives of a software system from the currently-employed architectural behavioral specification. We employ business value to comprehensively evaluate the behavioral alternatives while capturing the trade-offs among relaxed functional and quality constraints. We also introduce a genetic algorithm-based planning technique to efficiently search for the optimal (sometimes a near-optimal) behavioral alternative that can provide the best business value. The experimental study on an online order processing benchmark has shown promising results that the proposed approach can improve adaptation flexibility and business value with acceptable performance overhead.

Index Terms—Behavioral adaptation, relaxed constraints, business value, quantitative verification, genetic algorithm.

1 Introduction

Software systems are increasingly operating in highly open and dynamic environments with the prevalence of Internet computing and services computing. Consequently, traditional static architectural design decisions, often assuming predictable and well-understood environments, will become less satisfiable. To quickly respond to runtime environmental changes, systems are required to autonomously and dynamically adapt their architectures [1], [2]. Such architectural adaptations can be structural, behavioral, or both.

Structural adaptations focus primarily on the architectural topology of a system and its constituent elements (e.g., components or services), which are usually coarse-grained. For example, by switching between two alternative services that implement the same interface but differ in the offered qualities, structural adaptations can improve the overall quality of a system. On the other hand, *behavioral adaptations* mainly target the behaviors in each architectural element and interactions among the architectural elements of a system, which are usually fine-grained. For example, by changing the interaction protocol between two services from synchronous to asynchronous mode, behavioral adaptations can improve the system performance at the price of potentially decreased consistency. Both kinds of adaptations can affect the behaviors of a system. The main difference is that structural adaptations regard architectural elements as "black boxes", while behavioral adaptations treat them as "white boxes".

A number of advances have been made in the engineering of self-adaptive systems [1], [3], [4]. The majority of them deal with structural adaptations, e.g., replacing components and connectors [5], [6] and reconfiguring the redundant patterns of alternative services [7], [8]; and only a few of them involve behavioral adaptations, e.g., skipping optional components or services [9], [10] and tuning behavioral parameters [11], [12]. However, some challenges still remain.

Existing approaches seldom deal with behavioral adaptations, which leads to a gap in architecture-based self-adaptations. Besides, they usually presume a predefined space of architectural alternatives. However, the alternative space is often too large for system architects to define the potentially exponential alternatives completely [13]; and it is laborious for architects to verify their correctness. Therefore, one challenge in self-adaptive systems is *how to systematically handle behavioral adaptations and automatically capture behavioral alternatives of a system* so that the system can be flexibly adapted.

In addition, existing approaches merely consider quality constraints (e.g., reliability and performance) to maximize a weighted utility. However, functional constraints are either assumed to be always satisfied and thus are not considered, or treated as clear-cut ones (i.e., either satisfied or not) to ensure the functional correctness. Some functional constraints are indeed *critical*, which must be absolutely satisfied to respect laws or to ensure safety-related properties even at the price of quality degradation. Due to the probabilistic nature of system behaviors, others can be *relaxed*, i.e., sacrificed in a tolerance range to favor other more critical constraints [14], [15]. Therefore, another challenge in self-adaptive systems is *how to comprehensively evaluate architectural alternatives of a system with respect to both quality and functional constraints* so that the system can be effectively adapted.

To address these challenges, in this paper, we propose a new architecture-based self-adaptation approach, which focuses on behavioral adaptations with the automatically gen-

- *B. Chen, X. Peng, J. Zheng and W. Zhao are with the School of Computer Science and the Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China. X. Peng is the corresponding author. E-mail: {bhchen, pengxin, jiahuanzheng13, wyzhao}@fudan.edu.cn*
- *Y. Liu and S. Song are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. This work was mainly done when B. Chen was a visiting student in NTU. E-mail: {bhchen, yangliu, songsz}@ntu.edu.sg*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

erated behavioral alternatives from the currently-employed architectural behavioral specification, and comprehensively considers relaxed functional and quality constraints in terms of business value. This approach can efficiently search for the optimal (sometimes a near-optimal) behavioral alternative that can adapt to the monitored environmental changes, and map the specification of the planned behavioral alternative to the running system to achieve runtime adaptations.

The main contributions of our work are as follows.

- We propose a technique to automatically generate the behavioral alternatives of a system at the specification level from the currently-employed architectural behavioral specification. In particular, we formally define *adaptation operations* to adapt the specification, *critical constraints* to eliminate invalid behavioral alternatives, and *correction rules* to correct certain behavioral deviations. Then, our technique is to apply adaptation operations and correction rules on the specification while satisfying critical constraints.
- We apply business value [16] as the utility metric to comprehensively evaluate behavioral alternatives while capturing the tradeoffs among relaxed functional and quality constraints. In particular, the satisfaction of each constraint is predicted through runtime quantitative verification; and the business value that can be earned by employing a behavioral alternative is predicted by a predefined value proposition, which associates the constraints from the perspective of business value. This predicted business value for each behavioral alternative serves as the evaluation criterion during the runtime adaptation planning.
- We propose a genetic algorithm-based planning technique to efficiently search for the optimal behavioral alternative that can offer the best business value. Due to the stochastic nature of genetic algorithms, a near-optimal one could be found. Our technique is realized by adapting a genetic algorithm while integrating behavioral alternatives generation and business value prediction to the genetic process.
- We conduct an experimental study on an online order processing benchmark to evaluate the effectiveness and scalability of our approach, which has shown promising results that our approach can improve adaptation flexibility and business value with acceptable performance overhead.

Structure. Section 2 shapes the motivation. Section 3 introduces the preliminaries. Section 4 presents the overview. Sections 5, 6 and 7 elaborate behavioral alternatives generation, business value prediction and genetic algorithm-based planning. Section 8 evaluates our approach, followed by the discussion in Section 9. Section 10 reviews related work before Section 11 draws our conclusions.

2 Motivating Example

We use an online order processing benchmark to shape our motivation. Fig. 1 shows the component-based architecture model, which depicts the structural view. For the behavioral view, *Order Processing* interacts with the other components to process customer orders. *Order Checking* checks customers credit and products inventory. If order checking is passed, *Order Payment* authenticates the PIN and payments; and *Order Scheduling* reserves the inventory and selects a shipper.

These software components may have alternative behaviors, which makes it possible for them to operate with different behavioral specifications. For example, *Order Checking* can operate with the credit checking and inventory checking being performed in different sequential orders or in parallel, resulting in three behavioral alternatives. Since each component might have multiple behavioral alternatives, the space of compositional behavioral alternatives can be exponential. As a result, it is often impossible for system architects to exhaustively identify and verify all behavioral alternatives at design time. One potential remedy is to automatically generate and verify the behavioral alternatives at the specification level from the currently-employed specification at runtime, and then map the planned behavioral alternative to the running system. To scale the adaptation planning with such exponential behavioral alternatives, the optimal behavioral alternative needs to be efficiently planned.

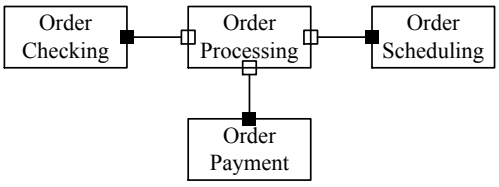


Fig. 1. Architecture Model of an Online Order Processing Benchmark

Behavioral alternatives of a component can have different impacts on quality constraints with different runtime environments. For example, checking inventory before credit favors performance and cost when the pass rate of inventory checking is lower than that of credit checking because there will be no need to check credit if inventory checking is not passed. Therefore, the impacts of behavioral alternatives on quality constraints need to be dynamically predicted according to the monitored environmental changes to facilitate the runtime adaptation planning.

Behavioral alternatives can satisfy functional constraints differently as well. For example, if order payment and order scheduling are executed in parallel rather than in sequence, the constraint "successful payment before scheduling" will be violated with a certain probability; but the throughput of order processing will be greatly increased. Some functional constraints are non-critical, which means their violations are acceptable if the corresponding loss is in a certain tolerance range. Because of the probabilistic nature of constraint violations, it is worthwhile to relax some non-critical functional constraints for allowing a broader range of possible alternatives to be considered during adaptations. Therefore, the impacts of behavioral alternatives on non-critical functional constraints need to be dynamically predicted as well.

Moreover, the satisfaction of quality and functional constraints affects the throughput of the order processing system, which is ultimately reflected in the business value (i.e., real money) that can be earned. In that sense, the goal of runtime adaptations can be regarded to maximize the business value. Hence, business value can be seen as a suitable utility metric to comprehensively evaluate behavioral alternatives while capturing the trade-offs among relaxed functional and quality constraints.

3 Preliminaries

We use the architecture description language Wright# [17] to model the behaviors of a system due to the well-supported

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing


```
Component OrderProcessing {
  var check=0; var pay=0; var schedule=0; var process=0;
  Port checkRI=req!→get?→Skip;
  Port payRI=req!→get?→Skip;
  Port scheduleRI=req!→get?→Skip;
  Computation=ProOrder(); if(check==1&&pay==1&&schedule==1)
     {{process==1}→Skip} else {{process==-1}→Skip};

  ProOrder()=ChkOrder(); if(check==1){PayOrder()};
     if(check==1&&pay==1){SchOrder()};
  ChkOrder()=checkRI:req!→checkRI:get?check→Skip;
  PayOrder()=payRI:req!→payRI:get?pay→Skip;
  SchOrder()=scheduleRI:req!→scheduleRI:get?schedule→Skip;
}
```

Listing 1. Behavioral Specification of Component *Order Processing*

verification capabilities in our PAT framework [18], which is a self-contained and extensible simulation and model checking framework. Wright# adopts an architecture view of components and connectors [19]. Each component (connector) is defined as a 4-tuple, which specifies the variables, the initial valuation of the variables, the behaviors, and the interfaces. The architecture model is defined as a 5-tuple, which specifies a set of communication channels, a set of components, a set of connectors, a set of instances of components and connectors, and the attachments from components' interfaces to connectors' interfaces.

To capture the probabilistic behaviors of a system, we extend Wright# by integrating probabilistic choices introduced in PCSP# [20], and changing the semantic model of Wright# from Labeled Transition System to Markov Decision Process to support probabilistic choices. Part of the syntax to model behaviors of components and connectors is as follows.

$$
\begin{array}{ll}
P ::= Stop \mid Skip & \text{– primitives} \\
\quad \mid e\{prog\} \rightarrow P & \text{– event prefixing} \\
\quad \mid ch!exp \rightarrow P \mid ch?x \rightarrow P & \text{– channel communication} \\
\quad \mid P \ ; \ Q & \text{– sequence composition} \\
\quad \mid P \parallel Q & \text{– parallel composition} \\
\quad \mid if(con) \ \{P\} \ else \ \{Q\} & \text{– conditional choice} \\
\quad \mid while(con) \ \{P\} & \text{– loop expression} \\
\quad \mid Q & \text{– process reference} \\
\quad \mid pcase\{p_0 : P_0 \ ... \ p_n : P_n\} & \text{– probabilistic choice}
\end{array}
$$

Process $Stop$ does nothing and process $Skip$ terminates. Process $e \rightarrow P$ engages in event e first and then behaves as process P. If e is attached with program $prog$, $prog$ will be executed atomically with e. Process $ch!exp \rightarrow P$ evaluates expression exp and puts the value into channel ch, and then behaves as process P. Process $ch?x \rightarrow P$ gets the value from channel ch and assigns it to variable x, and then behaves as process P. Process $P \ ; \ Q$ behaves as P until its termination and then behaves as Q. Process $P \parallel Q$ executes P and Q in parallel independently (here we do not consider event synchronization for simplicity). Conditional choice $if(con) \ \{P\}$ $else \ \{Q\}$ behaves as P if con evaluates to true; otherwise it behaves as Q. Loop expression $while(con) \ \{P\}$ runs P iteratively until con evaluates to false. Recursion is supported by process reference. Probabilistic choice $pcase\{p_0 : P_0 \ ... \ p_n : P_n\}$ behaves as P_i with the probability p_i ($\sum_{i=0}^{n} p_i = 1$). The operational semantics is similarly defined as in PCSP# [20].

For example, Listing 1 gives the behavioral specification of component *Order Processing* in Wright#. First, a set of variables for storing processing results are defined; e.g., the result of order checking will be stored in `check`. Then, the required interfaces for order checking, payment and scheduling are defined. The behavior is defined in `Computation` as a sequence composition of order checking `ChkOrder`, order payment `PayOrder`, and order scheduling `SchOrder`. Each of these processes invokes the corresponding provided interface and stores the result in the corresponding variable.

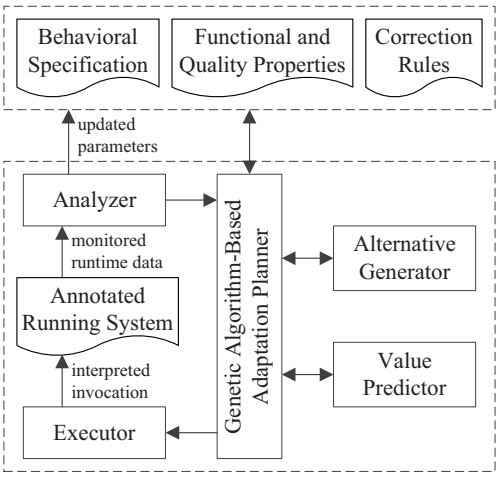


Fig. 2. The Overview of Our Approach

4 Approach Overview

The overview of the proposed approach is shown in Fig. 2. It consists of an *Analyzer*, a *Genetic Algorithm-Based Adaptation Planner* and an *Executor*, which construct an adaptation control loop [21]. Such a loop is periodically executed (e.g., every two minutes) to continuously adapt the behaviors of the running system to the monitored environmental changes.

At design time, system architects need to provide an architectural model of the *running system*, which defines its initial component-based *behavioral specification* in Wright#. The model parameters such as probability values (e.g., pass rate of credit checking) and quality values (e.g., response time of inventory checking) can be initially specified based on predictions or historical knowledge. Architects also need to provide *functional and quality properties* of the system, which specify critical constraints that must be satisfied and non-critical constraints that can be relaxed. Moreover, architects need to provide *correction rules*, which are used to correct some correctable behavioral deviations for the generated behavioral alternatives. On the other hand, components need to be implemented by following an adaptive component model [22]. Implementation-level classes, methods and fields need to be annotated with the names of their corresponding specification-level processes, events and variables to establish a mapping between implementation and specification. This mapping facilitates the runtime reconfiguration of system behaviors via interpreting the adapted behavioral specification.

At runtime, the *Analyzer* periodically updates the parameters in the behavioral specification based on the monitored runtime data to reflect runtime environmental changes. Specific techniques that support dynamic updating of runtime models, e.g., KAMI [23], can be applied, which is beyond the scope of this paper. For simplicity, here we use log analysis.

The *Genetic Algorithm-Based Adaptation Planner* is then invoked to efficiently search for the optimal behavioral alternative. During the search, *Alternative Generator* applies adap-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

tation operations and correction rules on the currently-employed behavioral specification to generate behavioral alternatives that satisfy critical constraints. Meanwhile, based on the predicted satisfaction of relaxed constraints via runtime quantitative verification, the business value that can be earned by each behavioral alternative is predicted by *Value Predictor* according to a system-specific value proposition. Thus the planner is designed to find a behavioral alternative with the best predicted business value; and our approach is conceived to continuously improve the business value.

If a new behavioral alternative is found, the *Executor* will reconfigure the running system by interpreting the new behavioral specification while executing the implementations that are associated with the specification by annotations. In particular, following the adaptive component model in [22], every component separates the computation logic from the control logic; and the control logic is realized by interpreting the behavioral specification to facilitate the adaptation logic. Such a model can be seen as a partial implementation of the software model in the paradigm of Internetware [2]. The *Executor* currently supports systems implemented in Java, but it can be extended to support systems implemented in other object-oriented languages that support annotations.

Here we focus on the planning part where our main contributions lie in, i.e., *Alternative Generator*, *Value Predictor* and *Genetic Algorithm-Based Adaptation Planner*, which will be respectively elaborated in Sections 5, 6 and 7.

5 Behavioral Alternatives Generation

Specification-level behavioral alternatives of a system can be considered as the behavioral specifications derived from the currently-employed specification by applying *adaptation operations* and *correction rules* while ensuring the satisfaction of *critical constraints*. In this section, we first introduce adaptation operations, critical constraints and correction rules, and then present the alternatives generation procedure.

5.1 Adaptation Operations

We define four typical kinds of adaptation operations in this paper, which are not intended to be exhaustive but rather to illustrate the feasibility of the proposed approach.

- *Switching between sequential and parallel execution*. A set of components or operations can be executed in sequence or in parallel. For example, order scheduling and order payment can be executed in sequence or in parallel. The parallel execution can improve response time and throughput, but may schedule an order even if its payment fails, which causes a loss if the customer refuses to pay on delivery.
- *Changing sequential execution orders*. A set of components or operations can be executed in different sequential orders. For example, the two operations inventory checking and credit checking in order checking can be executed in different orders, having different favors in response time and cost depending on their checking pass rates.
- *Skipping or including an optional functionality*. A component or an operation can be optional, which can be skipped or included without influencing the core functionalities but affecting the quality attributes. For example, caching can be either enabled or disabled during inventory checking for a better response time depending on the cache hit rate.
- *Tuning a parameter*. A configuration parameter of a component can be tuned to change the behavior. For example, the maximum number of retries when payment timeout happens can be tuned to have different favors in the cost and success rate of order payment.

According to the syntax of the architectural specification language Wright# as introduced in Section 3, these adaptation operations are defined as the following adaptation operators, where the left-hand and right-hand sides are behavioral specifications before and after applying the adaptation operators denoted by $\longmapsto$.

$$
\begin{array}{rcll}
P\ ;\ Q & \longmapsto & P \,||\, Q & \text{– switch to parallel execution} \\
P \,||\, Q & \longmapsto & P\ ;\ Q & \text{– switch to sequential execution} \\
P\ ;\ Q & \longmapsto & Q\ ;\ P & \text{– change the sequential order} \\
P & \longmapsto & Skip^{[P]} & \text{– skip an optional process} \\
Skip^{[P]} & \longmapsto & P & \text{– include an optional process} \\
v = val_1 & \longmapsto & v = val_2 & \text{– tune a variable}
\end{array}
$$

where P and Q are processes; $Skip^{[P]}$ is a special $Skip$ that has a placeholder for the skipped process that could be included later; and v is a variable whose initial value is val_1.

5.2 Critical Constraints

We distinguish two kinds of critical constraints, i.e., *constructive* and *specification constraints*, for their own purposes.

Constructive constraints stem from the supported adaptation operations, and are often driven by consistency rules from a technical or business perspective. For example, two processes may not be allowed to change their sequential order or switch to parallel execution due to data dependency; a mandatory process cannot be skipped; and a variable cannot be tuned to an invalid value. The purpose of these constraints is to avoid invalid combinations of process elements during the generation of behavioral alternatives.

To formally express such constructive constraints, we define the following propositions, which can be extended with the newly supported adaptation operations.

$$
\begin{aligned}
Prop ::= {} & P \gg_R Q \mid P \not\gg_R Q \mid P \bowtie_R Q \mid P \not\bowtie_R Q \\
& \mid P_R^{\bullet} \mid v_l^h s \mid Prop \wedge Prop \mid Prop \vee Prop
\end{aligned}
$$

where P, Q and R are processes; and v is a variable. Specifically, $P \gg_R Q$ (resp. $P \not\gg_R Q$) means that P must (resp. must not) sequentially precede Q within R. $P \bowtie_R Q$ (resp. $P \not\bowtie_R Q$) means that P must (resp. must not) parallelize with Q within R. $P_R^{\bullet}$ means that P is a mandatory process within R. $v_l^h s$ means that v must be in the range of l and h with a tuning step s. Conjunction ($\wedge$) and disjunction ($\vee$) are logical operators over the propositions.

For example, the constructive constraint of *Order Processing* in Listing 1 is illustrated as follows.

$$
\begin{aligned}
& ChkOrder^{\bullet}_{ProOrder} \wedge ChkOrder \gg_{ProOrder} PayOrder \wedge \\
& PayOrder^{\bullet}_{ProOrder} \wedge ChkOrder \gg_{ProOrder} SchOrder \wedge \\
& SchOrder^{\bullet}_{ProOrder} \wedge SchOrder \not\gg_{ProOrder} PayOrder
\end{aligned}
$$

On the other hand, specification constraints accompany the initial behavioral specification to check the satisfaction of critical properties. Such constraints are also used to further check the validity of the generated behavioral alternatives.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

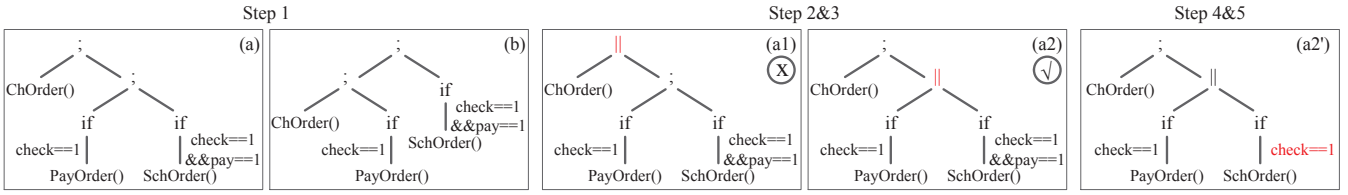


Fig. 3. An Illustrative Example of the Generation Procedure of Behavioral Alternatives

Deadlock-freeness analysis, (probabilistic) reachability analysis, and (probabilistic) linear temporal logics [24] are supported by our model checker for Wright# [17] to define and check specification constraints. For example, the critical constraint "successfully-paid orders should be scheduled eventually" can be defined in LTL as $\Box\,(paid \rightarrow \Diamond scheduled)$.

Notice that it is sufficient to only use specification constraints for the validity checking of generated behavioral alternatives. However, this is a time-consuming process. Consequently, we introduce high-level constructive constraints, which are easier to express at design time and are more efficient to check at runtime, as a preliminary step to quickly eliminate those invalid behavioral alternatives without involving the time-consuming model checking process.

5.3 Correction Rules

Adaptation operations may introduce some behavioral deviations as side effects, which violate specification constraints but can be corrected by some rules. The currently supported adaptation operations only affect the execution sequence of processes, thus probably making the conditional expression in a conditional choice, which actually specifies the precondition for a process to execute, incorrect. For example, the underlined conditional expression in Listing 1 indicates that only if order checking and payment are both passed, order scheduling can be executed. However, after switching the sequential execution of $PayOrder$ and $SchOrder$ to parallel, that expression will evaluate to false, and thus $SchOrder$ will not be executed. In fact, now the correct expression should indicate that if order checking has been passed, order scheduling can be executed with order payment in parallel.

To guide the correction of such conditional expressions for generated behavioral alternatives, we define the following *correction rules*, which can be extended with the newly supported adaptation operations that may introduce other correctable behavioral deviations.

$$P \gg_R Q:\ Q \longrightarrow [con]Q$$
$$P \not\gg_R Q:\ [con]Q \longrightarrow Q$$

where $P \gg_R Q$ and $P \not\gg_R Q$ are propositions, and con is a conditional expression. The first rule means if P sequentially precedes Q in R, con is a precondition for Q to execute. The second rule means if P does not sequentially precede Q in R, con is not a precondition for Q to execute. For example, the correction rules for *Order Processing* are as follows.

$$PayOrder \gg_{ProOrder} SchOrder:\ SchOrder \longrightarrow [pay == 1]SchOrder$$
$$PayOrder \not\gg_{ProOrder} SchOrder:\ [pay == 1]SchOrder \longrightarrow SchOrder$$

Notice that similar to constructive constraints, correction rules are also optional. If no correction rules are specified, behavioral alternatives that have correctable behavioral deviations will not be considered because they violate specification constraints. By introducing and applying correction rules, the space of behavioral alternatives can be enlarged and thus the adaptation can be more flexible.

5.4 Alternatives Generation Procedure

By applying adaptation operations on processes or variables and ensuring the satisfaction of critical constraints, alternatives of a behavioral specification can be generated. Because tuning a variable v is straightforward, i.e., each allowed values as constrained by $v_l^h s$ results in a valid behavioral alternative, we elaborate the generation procedure on processes.

The procedure consists of five steps, which takes as inputs a behavioral specification, a process, critical constraints and correction rules, and returns a set of valid alternatives and a set of invalid ones. We illustrate the procedure step by step with the behavioral specification in Listing 1.

In *Step 1*, all equivalences of the input process are generated based on the associative law[1] of ; and || and the commutative law[2] of || in order to ensure the completeness of generated alternatives. For example, Figs. 3 (a) and (b) are the two equivalences of process $ProOrder$ in *Order Processing*, which show their abstract syntax trees for clarity.

In *Step 2*, all the matched adaptation operators (see Section 5.1) are applied on the equivalences to generate alternatives. In detail, the part of a process that matches the left-hand side of an operator is adapted to the one formed by the right-hand side. For example, Figs. 3 (a1) and (a2) give the generated alternatives after applying "switch to parallel execution" on Fig. 3 (a); and others are omitted for clarity.

In *Step 3*, all the alternatives generated in the previous step are checked with respect to constructive constraints (see Section 5.2). Specifically, based on the syntax of ; and ||, a set of implied propositions can be derived as follows.

$$R = P\ ;\ Q\ \Rightarrow\ \{P \gg_R Q,\ Q \not\gg_R P,\ P \not\bowtie_R Q,\ Q \not\bowtie_R P\}$$
$$R = P\ ||\ Q\ \Rightarrow\ \{P \not\gg_R Q,\ Q \not\gg_R P,\ P \bowtie_R Q,\ Q \bowtie_R P\}$$

On the basis of the basic implications from ; and ||, all the implied propositions of an alternative can be derived recursively. The derived propositions are then used to check if an alternative satisfies constructive constraints. For example, Fig. 3 (a1) fails to satisfy the constructive constraint defined in Section 5.2, but Fig. 3 (a2) satisfies it.

In *Step 4*, the alternatives that satisfy constructive constraints are corrected by applying correction rules. In detail, given the rule $P \gg_R Q : Q \longrightarrow [con]Q$, if $P \gg_R Q$ is implied by R, con is inserted into the conditional expression of Q's residing conditional choice (if such a conditional choice does not exist, a conditional choice is added first); given the rule $P \not\gg_R Q : Q \longrightarrow [con]Q$, if $P \not\gg_R Q$ is implied by R, con is removed from the conditional expression of Q's residing conditional choice (if such a conditional expression

1. $P\ ;\ (Q\ ;\ R) = (P\ ;\ Q)\ ;\ R,\ and\ P\ ||\ (Q\ ||\ R) = (P\ ||\ Q)\ ||\ R$
2. $P\ ||\ Q = Q\ ||\ P$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

becomes empty, the conditional choice is also removed). For example, Fig. 3 (a2′) corrects (a2) by removing $pay == 1$ from the conditional expression of $SchOrder$'s residing conditional choice based on the correction rules in Section 5.3.

Finally, in *Step 5*, the corrected alternatives are checked with respect to specification constraints (see Section 5.2). For example, Fig. 3 (a2′) satisfies the specification constraint defined in Section 5.2. The satisfied alternatives are returned as valid alternatives, while all the others generated in *Step 2* are returned as invalid alternatives. The invalid ones are also useful because some valid alternatives might be derived from invalid ones by further applying adaptation operators.

This procedure only generates alternatives by applying *one* adaptation operation on *one* process. By iterating this procedure over generated alternatives and all processes, we can generate many valid alternatives.

6 Business Value Prediction

Critical constraints serve as the criteria for the generation of valid behavioral alternatives (see Section 5), while relaxed constraints (referred to as constraints hereafter in this section) serve as the criteria for the evaluation of generated behavioral alternatives. In particular, the satisfaction of functional constraints reflects to what extent a system ensures its desired properties when offering its functionalities; and the satisfaction of quality constraints characterizes how well a system offers the functionalities. They are two complementary aspects to evaluate behavioral alternatives.

To unify them, a weighted objective function can be simply applied as the utility metric, which weighs the satisfaction of quality and functional constraints. Despite that it is a widely-used technique in quality-driven self-adaptation approaches, it lacks an explicit explanation of the underlying meaning of the utility value especially when functional and quality constraints are weighed together. As a consequence, it is difficult to know if such a kind of utility really reflects the intention of stakeholders.

The above analysis suggests to measure the overall satisfaction of constraints from the perspective of stakeholders. On one hand, the satisfaction of constraints is ultimately reflected in the form of business value that can be earned by stakeholders. For example, Amazon, EBay and Google claim that a slight increase in user-perceived response time translates into concrete revenue loss [25]. On the other hand, by the principles of value-based software engineering [16], the ultimate goal of runtime adaptations can be interpreted as maximizing the value proposition of stakeholders [26], [27]. Thus, we propose to use business value as the utility metric to unify functional and quality constraints.

Business value depends on the offered functionalities as well as their qualities, which makes it a suitable utility metric for unifying functional and quality constraints. For example, a quick response time of order processing can increase throughput, which has a positive impact on business value; and the violation of the functional constraint of "successful payment before scheduling" usually implies a direct loss of business value. Besides, business value can be explained as the profit earned by stakeholders; and thus its calculation is based on more objective criteria, which are closely related to and really cared by stakeholders.

To predict the business value that can be earned by each behavioral alternative, a system-specific value proposition, unifying functional and quality constraints, needs to be predefined. However, the derivation of the value proposition is beyond the scope of this paper, which can be done by business analysis techniques [28]. Generally, the value proposition can be defined through the cooperation between architects and business experts, i.e., architects list the constraints, and business experts analyze their impact on business value. It is often composed of two parts: revenue and cost. Revenue can be measured based on the committed transactions (e.g., successfully processed orders), while cost includes the operating expense, the loss of improper operations, and etc.

Here we use the constraints and value proposition of the order processing system to illustrate business value prediction. First, we define four constraints for order processing:

- RC_1: the probability (i.e., p_1) of successful order scheduling but failed order payment should be as low as possible.
- RC_2: the probability (i.e., p_2) of successful order processing should be as high as possible.
- RC_3: the end-to-end response time (i.e., q_1) of order processing should be as quick as possible.
- RC_4: the cost (i.e., q_2) of order processing should be as low as possible.

RC_1 and RC_2 are functional constraints, while RC_3 and RC_4 are quality constraints. p_1 and p_2 can be computed by probabilistic reachability analysis [24], while q_1 and q_2 can be computed by reward-bounded reachability analysis [24] with the quality values specified as rewards defining over the events in the behavioral specification. Such analyses are supported by the model checker for Wright# [17], [20]. Since we only consider deterministic systems in this paper, the analysis result is a single value, which would be a range for non-deterministic systems [24].

Traditional reward-bounded reachability analysis computes the expected reward as a weighted sum of the cumulative rewards along all the possible paths reaching the specified states. This analysis can be directly applied to quality attributes such as cost. Differently, the reward cumulative function for response time should be maximum instead of summation when two processes are in parallel composition. For example, if the response time of $P1$ and $P2$ is 50 ms and 80 ms, the response time of $P1 \parallel P2$ should be 80 ms. Thus, we revise the reward analysis for response time by using a maximum cumulative function for parallel compositions.

Then, we define the value proposition for order processing as $value = 1000/q_1 \times (p_2 \times 1.8 - p_1 \times 8.5 - q_2)$, where $1000/q_1$ is an approximation of the throughput of order processing (i.e., the number of processed orders in one second) under the assumption that there is no resource competition among components. Hence, $1000/q_1 \times p_2 \times 1.8$ represents the revenue earned from successfully processed orders with the assumption that each successfully processed order can produce a profit of 1.8 dollars; $1000/q_1 \times p_1 \times 8.5$ represents the cost caused by the orders that are scheduled but are not paid successfully with the assumption that such an order will suffer a loss of 8.5 dollars; and $1000/q_1 \times q_2$ is the cost of operating order processing. Hence, $value$ is a prediction of the business value that will be earned in one second by employing a behavioral alternative. For example, if p_1 is 0.0476, p_2 is 0.8549, q_1 is 484.85 ms, and q_2 is 0.4254 dollars,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

value will be 1.46 dollars. Different behavioral alternatives often have different favors in the constraints, and thus have different predicted business values.

7 Genetic Algorithm-based Planning

Environmental changes will affect model parameters such as probability and quality values, which makes the system run in a less-optimal fashion. To timely mitigate the loss of business value, the behavioral alternative that best suits the current environment needs to be efficiently planned at runtime.

Exact methods (e.g., linear programming [29]) have been commonly applied in self-adaptive systems (e.g., [6], [8]) to plan an adaptation solution. However, due to the potentially exponential space of generated behavioral alternatives and the expensive computation time of runtime quantitative verification, exact methods need exponential computation time and thus become infeasible for our approach. For partially searching the alternative space and thus reducing the number of quantitative verification, metaheuristic methods [30] seem promising to search the optimal behavioral alternative within an acceptable computation time. Here we propose to use genetic algorithm [31], which has been the most widely-applied metaheuristic method in search-based software engineering [32]. Notice that we have conducted a preliminary study to compare generic algorithm with particle swarm optimization [33] (see details in Section 8), and leave it as our future work to systematically compare with other metaheuristic methods (e.g., hill climbing and simulated annealing).

Here we adapt a single-objective genetic algorithm with specific considerations on its optimality and performance to better suit our problem. In general, the genetic algorithm begins with an *initialization* step, which generates a set of *individuals* to form the initial population and evaluates their fitness by a *fitness function*. A *selection* operator is then used to select better individuals as parents, and *crossover* and *mutation* operators are applied on the parents to generate the offspring. After evaluating the offspring's fitness, it selects the best individuals from the offspring and current population to form the population of the next generation. This process is repeated until good enough solutions are found or a fixed number of generations is reached. In the following sections, we will formalize our planning problem with emphasis on the specific adaptations to this genetic process.

7.1 Individual and Initialization

An individual (or a solution) consists of a sequence of genes. In our planning problem, an individual represents a behavioral alternative, where each gene represents a variable or a process in a behavioral specification. Therefore, the length of an individual is application-specific. To shorten the length of an individual, only *tunable variables* (over which constructive propositions $v_l^h s$ are defined) and *adaptable processes* (which have sequence/parallel compositions, or optional processes) are encoded as genes. For example, all the variables in Listing 1 are not encoded as genes; and only process $ProOrder$ is encoded as a gene because it has sequence compositions.

Randomly generated initial population might sometimes densely sample in some regions of the alternative space but sparsely sample in others [31]. Here we use a different strategy, i.e., applying the alternatives generation procedure (see Section 5) *uniformly* on the genes of the current behavioral specification; and only the valid behavioral alternatives are put into the initial population.

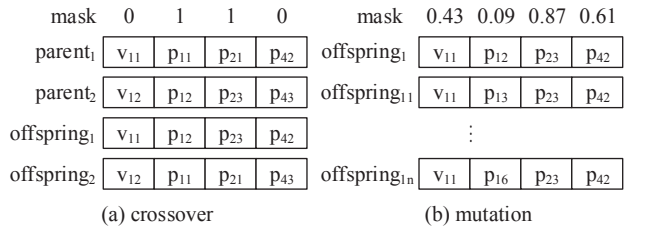


Fig. 4. Genetic Operators Crossover and Mutation

Moreover, we always put into the initial population the behavioral alternative that was found during the previous adaptation planning procedure. On one hand, it can prevent the algorithm from finding a behavioral alternative that is even worse than the previous one, which may happen in the original algorithm due to the stochastic nature. On the other hand, it can also prevent unnecessary adaptations especially when environments are stable. In such stable environments, the algorithm should always return the same behavioral alternative, and thus no adaptation is needed. However, the original algorithm might find a different and less satisfiable behavioral alternative due to the stochastic nature, which causes unnecessary adaptations.

7.2 Fitness Function and Evaluation

It is expensive to evaluate an individual in our planning problem. For an individual that represents a valid behavioral alternative, it verifies the functional and quality constraints by quantitative verification, and predicts the business value according to the value proposition (see Section 6) that serves as the single fitness function. For an individual that represents an invalid behavioral alternative, its fitness is zero, i.e., it has no business value.

To reduce the number of expensive evaluations and thus reduce the computation time, we store in a hashtable all the generated individuals and their predicted business values over all the generations so that an individual is evaluated at most once. It is realized at the price of a lookup step for each individual, which is often less expensive.

7.3 Selection, Crossover and Mutation Operators

A binary tournament selection operator is used to select the better parent individuals for generating the next population via crossover and mutation. Two individuals are uniformly selected from the previous population, and the one that has a higher business value is selected as a parent. This process is repeated until the number of selected parents equals the size of the population.

A uniform crossover operator is applied to exchange the genes of two parent individuals (e.g., $parent_1$ and $parent_2$) to generate two offspring (e.g., $offspring_1$ and $offspring_2$). An example is shown in Fig. 4 (a). According to a uniformly generated binary mask for each gene, the gene from $parent_1$/$parent_2$ is inherited to $offspring_1$/$offspring_2$ if the mask is zero; otherwise, the gene from $parent_1$/$parent_2$ is inherited to $offspring_2$/$offspring_1$.

The mutation operator is applied on the offspring generated by crossover. It mutates the genes of one individual to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

generate offspring, and its mutation rate decides how many genes will be mutated. Here the alternatives generation procedure (see Section 5) is used as the mutation operator. An example is shown in Fig. 4 (b). Based on a uniformly generated decimal mask for each gene, the alternatives generation procedure is applied on the gene if the mask is smaller than the mutation rate (e.g., 0.1). As several adaptation operators can be matched and both valid and invalid behavioral alternatives are returned by this procedure, multiple offspring are generated. We do not discard the invalid ones as they might have genes that can be used to generate the optimal behavioral alternative. Instead, we allow them to stay in the population with the fitness being zero and a low probability to be selected (only when two invalid behavioral alternatives are selected during tournament selection).

8 Experimental Study

To evaluate the proposed approach, we conducted an experimental study to answer the following research questions:

- *RQ1*: What improvement can be achieved by the proposed approach? (Section 8.2)
- *RQ2*: How the proposed approach scales with the growth of model sizes? (Section 8.3)

8.1 Experimental Setup

We conduct the study on an online order processing benchmark. It is adapted from the subject that is used in our previous work [10] and is originally an IBM's sample example. The relaxed functional and quality constraints and the value proposition are introduced in Section 6, while the behavioral specification, critical constraints and correction rules are not completely shown here, but are available at our website[3] together with all the experimental data. The specification has 4 relaxed constraints, 7 critical constraints, 4 correction rules and 13 processes. JMeter 2.11 was used to simulate system accesses with a load of 15 users; and Badboy 2.2 was used to record the testing plan. The experiments were executed on a desktop with 3.00 GHz Intel Core2 CPU and 4 GB RAM.

Recalling that the adaptation control loop is periodically executed instead of being triggered by threshold-based rules because our approach continuously seeks opportunities to improve the system. In our experiments, the adaptation interval was set to two minutes.

For each of the following four approaches, we conducted an experiment with the same experimental setting.

- *Naive*: the baseline approach without self-adaptation.
- *Exact*: the adaptation approach that uses an exact method to exhaustively search for the optimal alternative, which is widely used in existing self-adaptation approaches (e.g., [6], [8]). First, we apply our alternatives generation procedure on every adaptable process to exhaustively generate valid alternatives, and then use each adaptable process as a variable and the value proposition as the objective function to perform the optimization.
- *PSO*: the adaptation approach that applies particle swarm optimization to partially search the alternative space (similar adaptations to GA as introduced in Section 7 are also applied to PSO),
- *GA*: the adaptation approach that uses our adapted generic algorithm to partially search the alternative space.

3. http://www.se.fudan.edu.cn/behavioral-adaptation

8.2 Effectiveness Evaluation (RQ1)

To compare the effectiveness across the approaches, we conducted the experiments under four scenarios that reflected various environmental changes for Internet-scale systems. *Predicted value* (i.e., the business value that could be earned in one second, which was predicted according to the value proposition during planning) and *actual value* (i.e., the business value that was actually earned during one adaptation interval) were collected at runtime and served as the key indicators of effectiveness.

8.2.1 Scenario 1: Changes of Operational Profiles

Fig. 5 (a) shows the scenario: the cache hit rate in inventory checking decreased from 0.80 to 0.30 from time 5 to 6; and the inventory checking pass rate decreased from 0.95 to 0.80 from time 10 to 11. Figs. 5 (b) and (c) illustrate the adaptation effect of four approaches in terms of predicted and actual value; and the behavioral alternatives planned by *Exact* and *GA* are also recorded on the curves.

When the cache hit rate decreased from time 5, the response time of inventory checking increased, making inventory checking slower than credit checking. To respond to the change, *Exact* changed their sequential order so that credit was checked before inventory, which improved the overall response time. With the further decrease of cache hit rate at time 6, inventory checking became slower. *Exact* decided to skip the cache checking in inventory checking to improve its response time. Differently, *GA* did not generate the optimal behavioral alternative at time 5 but at time 6 because of its stochastic nature. Nevertheless, *GA* can ensure that the behavioral alternative generated at the next interval will not be worse with respect to business value than the one employed at the current interval, which mitigates the stochastic nature to some extent. The values of the cache hit rite after time 7 were not available because the cache checking was skipped.

With the decrease of inventory checking pass rate at time 10, the number of orders that failed to pass the inventory checking and thus did not need further credit checking increased. Both *Exact* and *GA* changed the sequential order of inventory checking and credit checking to check inventory before credit, which improved the overall response time.

8.2.2 Scenario 2: Changes of Qualities

Fig. 5 (d) shows the scenario: the response time of inventory reservation in order scheduling increased from 100 to 400 ms from time 5 to 6; and the response time of order payment increased from 200 to 450 ms from time 10 to 11. Figs. 5 (e) and (f) illustrate the adaptation effect of four approaches in terms of predicted and actual value.

All the approaches suffered a loss of business value due to the quality degradation of inventory reservation at time 5 and 6. However, the initial behavioral alternative had already been the optimal one in such environments, thus no behavioral alternative was generated by both *Exact* and *GA*. Similarly, no behavioral alternative was generated by *Exact* and *GA* by the increase of the response time of payment at time 10. However, the further increase at time 11 negatively

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

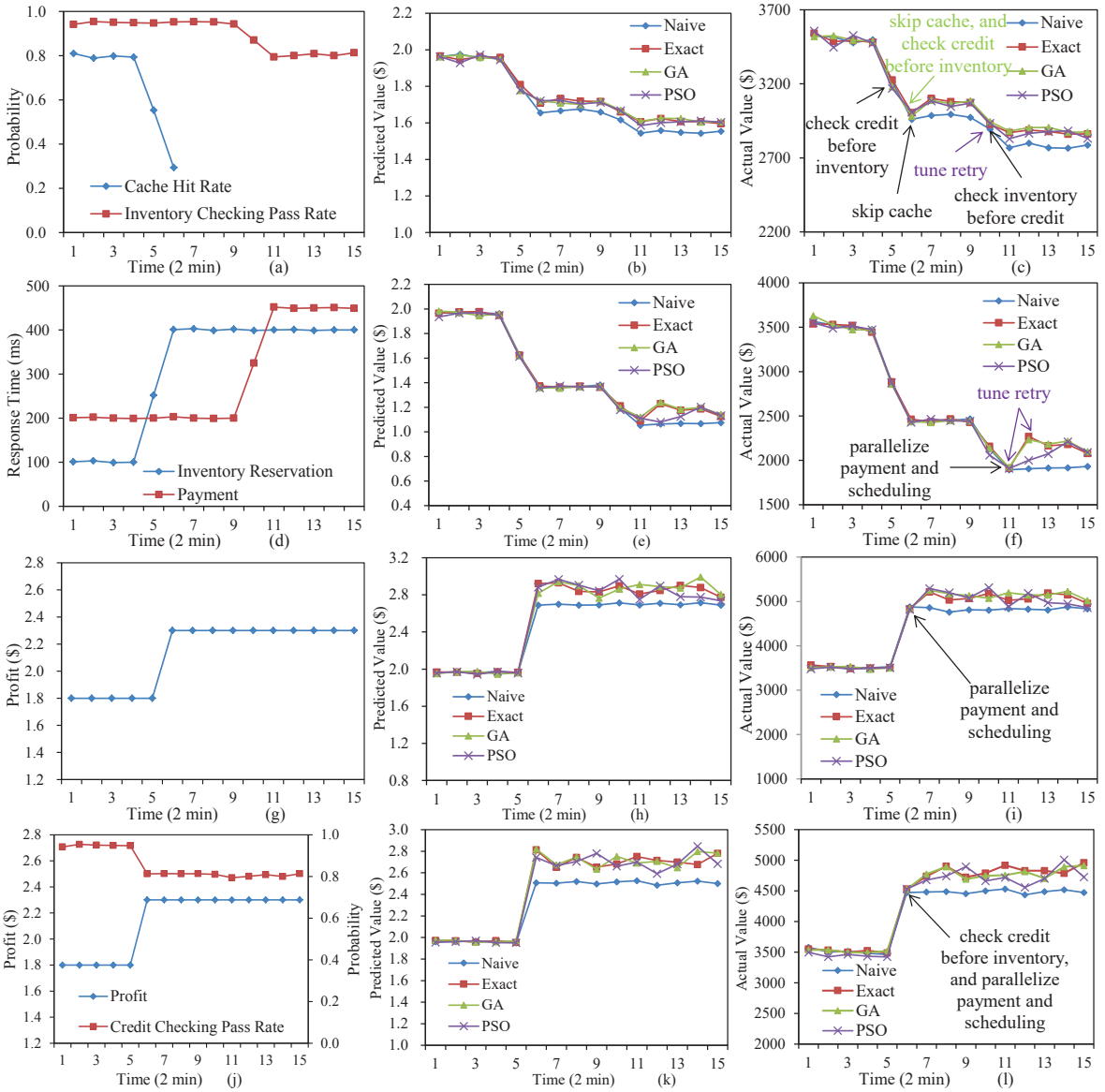


Fig. 5. Adaptation under Environmental Changes: (a-c) Operational Profile, (d-f) Quality, (g-i) Value Proposition, and (j-l) Multiple Changes

affected the throughput of order processing. Both *Exact* and *GA* decided to switch the sequential execution of payment and scheduling to the parallel execution, which improved the throughput with an acceptable loss for the orders that were scheduled but not paid successfully.

8.2.3 Scenario 3: Changes of the Value Proposition

Fig. 5 (g) shows the scenario: the profit from every successfully processed order increased from 1.80 to 2.30 dollars at time 6. Figs. 5 (h) and (i) illustrate the adaptation effect of the approaches in terms of predicted and actual value.

With the increase of profit at time 6, the business value from more successfully processed orders by the parallel execution of payment and scheduling would compensate the loss from the orders that were scheduled but not paid successfully. To respond to the change, both *Exact* and *GA* decided to switch the sequential execution of payment and scheduling to the parallel execution.

8.2.4 Scenario 4: Multiple Environmental Changes

The previous scenarios had only one environmental change at a time. In this last scenario, multiple changes happened simultaneously, which is often the case for Internet-scale systems. As shown in Fig. 5 (j), the credit checking pass rate decreased from 0.95 to 0.80, and the profit increased from 1.80 to 2.30 dollars at time 6. Figs. 5 (k) and (l) illustrate the adaptation effect in terms of predicted and actual value.

Similar to the previous scenarios, with these changes at time 6, *Exact* and *GA* generated the same optimal behavioral alternative, i.e., they changed the sequential order of inventory and credit checking to check credit before checking inventory, and switched the sequential execution of payment and scheduling to the parallel execution.

8.2.5 Comparing GA with PSO

To enlarge the cross-comparison of our GA-based approach, we also conducted preliminary experiments to compare *GA* with one of the other metaheuristic methods, i.e., *PSO* [33]. In particular, *PSO* found a sub-optimal behavior alternative in two of the previous four scenarios. In the first scenario as in Fig. 5 (c), *PSO* further tuned the number of retries in order payment from 2 to 1 at time 10, which was unnecessary and hence caused the decrease of business value from time 10 to

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

15. Similarly, in the second scenario as in Fig. 5 (f), *PSO* further tuned the number of retries to 1 at time 11 and tuned it back to 2 at time 12. These unnecessary adaptations caused the temporary decrease of business value at time 12 and 13.

In this case, *GA* was better than *PSO*. However, the effectiveness of metaheuristic methods usually relies on the optimization problem to be solved. Due to space limitations, we only showed a preliminary study here; and we plan to conduct a systematic comparison across different metaheuristic methods using more systems in the future.

8.2.6 Summary

Despite the stochastic nature, *GA* did not trigger any unnecessary adaptations when runtime environments were stable (e.g., from time 7 to 15 in Fig. 5 (i)) as we always put the previously optimal behavioral alternative into the initial population. This is also consistent with the fact that the previously optimal behavioral alternative should always be the best one when environments are stable. Besides, compared with *Exact*, *GA* generated the optimal and sometimes a near-optimal behavioral alternative with a shorter computation time (which will be detailed in Section 8.3). In addition, the adaptations were flexible in the sense that all the behavioral alternatives were automatically generated. As explained by the causes and effects of the adaptations in the four scenarios, the generated behavioral alternatives were also reasonable and understandable, and hence were meaningful with respect to architectural design. Last, it can be observed from Figs. 5 (c), (f), (i) and (l) that *GA* achieved a higher business value than *Native* by applying adaptations (e.g., from time 12 to 15 in Fig. 5 (f)); and *GA* achieved almost the same business value as *Exact*.

These observations answer *RQ1* positively that our approach can automatically generate behavioral alternatives and hence improve adaptation flexibility; and it can improve business value (i.e., the effectiveness of adaptations).

8.3 Scalability Evaluation (RQ2)

The scalability of our approach is mainly determined by the *Planner* and *Executor*. Thus, to evaluate the scalability of our approach with respect to more complex systems, we evaluated their performance with larger system models that were semi-automatically generated to ensure their complexity.

To this end, we specify a set of specification templates at the process and component level. Process-level templates include sequence and parallel composition with various types of atomic processes and one (critical or relaxed) specification constraint. Here we do not provide constructive constraints and correction rules because the most time-consuming step is to verify specification constraints. Component-level templates ensure the interaction among components, whose behaviors can be defined by process-level templates. Based on the templates, we can automatically generate system models in an iterative way, i.e., randomly choose a component template to interact with a component in the system model and then define its behaviors by randomly composing process-level templates. Then we manually define system-level quality constraints (e.g., cost), and simplify the value proposition as the overall satisfaction of relaxed constraints.

For the planner, two main factors affect its performance: (1) the number of properties that need to be verified, which determines the time complexity of quantitative verification, and (2) the number of processes and variables that could be adapted, which determines the size of the alternative space. To evaluate the performance, we compared the computation time for planning a behavioral alternative and its optimality of *GA* and *PSO* with *Exact*. Here the optimality is measured by comparing the predicted value of the planned alternative via *GA* and *PSO* with the one via *Exact*.

Fig. 6 (a) reports the computation time of *Exact*, *PSO* and *GA*, and the optimality of *GA* and *PSO*, with models whose number of properties (i.e., specification constraints) varies from 2 to 20. The number of processes and variables that can be adapted was fixed to 4. We can observe that their computation time increased linearly with the number of properties; and the computation time of *GA* increased more slowly than *Exact* and *PSO*, and outperformed *Exact* by 40%. In addition, *GA* and *PSO* achieved 99% optimality in average.

Fig. 6 (b) reports the computation time of *Exact*, *PSO* and *GA*, and the optimality of *GA* and *PSO*, with models whose number of adaptable processes and variables varies from 5 to 14. We fixed the number of properties to 4. It can be seen that the computation time of *Exact* increased exponentially; and it took around 1,240 seconds when the number of processes and variables (that can be adapted) climbed to 14 (i.e., with 16,384 behavioral alternatives). However, the computation time of *GA* and *PSO* increased linearly; and it took around 16 and 9 seconds when the number of processes and variables (that can be adapted) climbed to 14. Furthermore, *GA* achieved 97% optimality in average; while *PSO* achieved 92% and its optimality decreased as the growth of models.

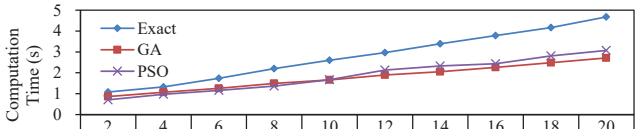
For the executor, it requires additional overhead because of the dynamic system execution through the interpretation of the specification of the generated behavioral alternative. To evaluate the overhead, we compared our interpreted execution with pure Java execution using systems whose number of method calls varies from 50 to 500. Figs. 7 (a) and (b) respectively present their execution time and memory consumption. Our approach introduced an overhead in execution time of around 6.5% and an overhead in memory consumption of around 5.5%, which is still acceptable given the improved adaptation flexibility and business value.

These observations from Figs. 6 and 7 answer *RQ2* positively that our approach is promising to scale well with more complex systems in terms of computation time and optimality of the generated behavioral alternative; and it introduces acceptable execution overhead.

8.4 Threats to Validity

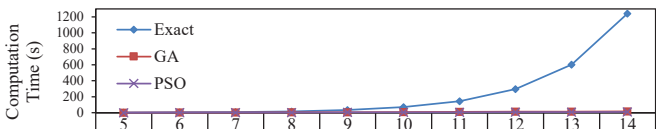
One major threat to the external validity of our experimental study lies in the fact that the effectiveness of our approach was only evaluated on one single system. As a consequence, the presented results might not be really representative for more complex systems. The reason for using such a system is to clearly explain the causes and effects of the underlying adaptation process so that we can gain sufficient confidence in the proposed approach instead of being lost in the hybrid causes and effects. On the other hand, we partially mitigate this problem by a scalability evaluation on the main components of our approach, which also shows promising results. We plan to conduct evaluation on more real-life systems.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing



	2	4	6	8	10	12	14	16	18	20
Exact	1.08	1.32	1.74	2.20	2.60	2.97	3.39	3.78	4.16	4.68
GA	0.86	1.07	1.26	1.49	1.66	1.89	2.05	2.26	2.49	2.71
GA Opt.	0.999	0.998	0.998	0.997	0.999	0.999	0.998	0.997	0.995	0.999
PSO	0.71	0.97	1.16	1.37	1.67	2.13	2.33	2.43	2.81	3.07
PSO Opt.	0.999	0.995	0.992	0.988	0.989	0.985	0.992	0.993	0.992	0.991

Number of Properties (#) (a)



	5	6	7	8	9	10	11	12	13	14
Exact	1.8	4.2	7.9	16.0	34.6	70.6	145	296	604	1243
GA	1.2	2.0	3.1	4.4	5.9	7.4	9.6	12.2	13.6	16.2
GA Opt.	0.978	0.976	0.979	0.986	0.982	0.979	0.977	0.980	0.982	0.981
PSO	0.9	1.6	2.4	3.3	4.3	5.3	6.1	7.0	8.0	9.4
PSO Opt.	0.959	0.951	0.949	0.942	0.934	0.925	0.902	0.902	0.878	0.854

Number of Adaptable Processes and Variables (#) (b)

Fig. 6. Comparison of the Computation Time and Optimality of the GA and PSO Approaches with the Exact Approach

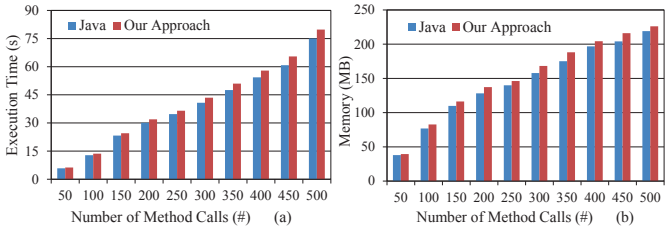


Fig. 7. Overhead of Our Approach with Respect to Execution Time and Memory Consumption

9 Discussion

Our approach is architecture-centric and business-oriented, which can be seen as a preliminary framework for the implementation of Internetware systems [2]. However, to fully fulfill the paradigm of Internetware, our next step is to equip each system element with an autonomous adaptation mechanism for cooperative and distributed adaptations. Here we discuss some threats to the application of our approach.

First, the automatically generated behavioral alternatives may become less understandable from the perspective of architectural design or violate architects' design principles especially when the running system is tremendously adapted. On one hand, this threat can be mitigated to some extent by expressing architects' design principles as a part of the constructive constraints. In this way, undesired behavioral alternatives can be eliminated during the generation procedure. On the other hand, consider that adaptation operations may have their own cost (e.g., impacts on the understandability and stability of a system), we intend to integrate such cost into the value proposition quantitatively. By this means, the system can be adapted in a cost-effective way, and may not be tremendously adapted due to the adaptation cost.

Second, the practical adoption of architecture description languages is questionable, especially for Internet-scale systems. However, "software architecture for the whole life cycle" is a core principle of Internetware [2] and runtime adaptations [34]. In that sense, one possible remedy is to extend our approach to support widely-used architecture description languages (e.g., xADL [35]) or industrial modeling languages (e.g., BPMN[4]). Moreover, we plan to provide some methodological support for system architects to express behavioral specifications, constraints, and correction rules and to map the specification and implementation in a more efficient way; and we intend to provide explicit management interfaces for system architects to update models at runtime in case the specification volatilizes or non-critical constraints become critical.

4. http://www.omg.org/spec/BPMN/2.0/

Third, even though our scalability evaluation has shown acceptable performance overhead, we are still investigating possibilities to make further improvements. One possibility is to apply parametric model checking techniques [36], [37], [38], where variables are used to model probabilistic behaviors and qualities. An expression is computed for each constraint, and can be efficiently computed by replacing variables with monitored real values. Another possibility is to use incremental runtime verification techniques [39], which can exploit the results from previous analyses. On the other hand, parallel genetic algorithms [31] can be used to take advantage of multiple processors. Hence, the evaluation of the population can be performed in parallel.

Last, an appropriate system-specific value proposition is needed for the evaluation of behavioral alternatives. On one hand, some quality attributes (e.g., reputation) usually have implicit influences on business value, and often take a long time to be reflected in business value. For example, a poor reputation will cause customer losing, which in turn gradually affects business value. To capture the implicit and long-term factors instantly, specific analysis (e.g., customer losing trend analysis) should be conducted by business experts to make them an explicit part of the value proposition. However, if the value proposition is not correctly defined, our approach might trigger adaptations that aim at the maximization of the incorrect value proposition. As the value proposition is designed by architects and business experts to reflect the actual intention of stakeholders, its goodness can be reflected in the real profit earned by stakeholders and thus can be dynamically tuned, using the earned value as an indicator, to better capture the intention.

10 Related Work

We refer the readers to [1], [3], [4], [40] for a more detailed discussion of the state of the art in self-adaptive systems. We will focus on the most related work, i.e., architecture-based and requirements-driven self-adaptation, and formal methods and genetic algorithms in self-adaptive systems.

10.1 Architecture-Based Self-Adaptation

Oreizy et al. [34] proposed the concept of architecture-based runtime software adaptations, which is followed by a number of promising approaches. Garlan et al. [5] proposed a customizable self-adaptation infrastructure Rainbow. Floch et al. [41] designed an adaptation middleware MADAM for mobile applications. Morin et al. [42] employed model-level aspects to encapsulate variants and manage the variability by weaving. Elkhodary et al. [6] proposed a feature-oriented

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

self-adaptation framework FUSION that can learn the impacts of adaptation decisions on system's goals. D'Ippolito et al. [43] proposed a multi-tier control framework for self-adaptive systems, which can adapt a system's functionality by degradation and enhancement. Chen et al. [44] proposed to combine requirements and architectural adaptations by model transformations at runtime.

These approaches assume that architectural alternatives can be predefined at design time, but our approach can automatically generate the behavioral alternatives at the specification level. One exception is [43], which can dynamically determine the functionality to be degraded or enhanced, but needs to predefine different environment models. Moreover, most of these approaches target structural adaptations such as switching among alternative components, while our approach focuses on behavioral adaptations such as changing the execution order of several components. In addition, our approach allows probabilistic violation of relaxed functional constraints, which is not considered in other approaches.

10.2 Requirements-Driven Self-Adaptation

Several advances have been made on requirements-driven self-adaptation [14]. Dalpiaz et al. [45] proposed a conceptual architecture to provide systems with self-configuration capability. Wang and Mylopoulos [46] and Fu et al. [47] proposed requirements monitoring and diagnosing approaches to support self-repairing. Our previous work [10], [26], [48], [49] developed several self-optimization techniques. Salehie et al. [50] proposed a self-protection approach for adaptive security. In addition, Whittle et al. [15] and Baresi et al. [51] developed RELAX and FLAGS to formally specify requirements in fuzzy logic to support uncertainty in requirements for self-adaptive systems. This enables self-adaptive systems to temporarily relax the satisfaction of non-critical requirements when environments change. Then, Cheng et al. [52] integrated RELAX with goal modeling and introduced a set of adaptation tactics to deal with the uncertainty; and Baresi and Pasquale [53] successfully applied FLAGS to self-adaptive service compositions.

These approaches use requirements models as the knowledge base of adaptation, while we use architectural models, which reveal more details about the implementations. Further, they also presume a predefined adaptation space. Differently, the alternatives are automatically generated in our approach. Instead of adopting fuzzy logic as in RELAX and FLAGS, we use formal models to specify the overall architecture and then use quantitative verification to evaluate the constraints, which can provide sufficient trustworthiness for the underlying adaptation planning process.

10.3 Formal Methods in Self-Adaptive Systems

Formal methods have been attracting increasing interest for rigorously ensuring the properties of self-adaptive systems [4], [54], [55]. In particular, formal models are used to model the systems; models could be updated at runtime [23], [56], [57]; and quantitative verification is applied at runtime [58] to detect or predict requirements violations [8], [59] as well as to plan adaptation actions [9], [11], [27], [60].

Filieri et al. [59] proposed KAMI to provide continuous assurance of reliability and performance requirements based on Markov models. KAMI can updates model parameters by Bayesian estimation according to runtime observations. It can be integrated into our approach to update the probability and quality values in the behavioral specification.

Filieri et al. [60] also proposed a control-theoretic technique to support reliability requirements, where alternative implementations are modeled through different transitions exiting a given state with different probabilities in Discrete-Time Markov Chains. Such probabilities are tuned dynamically as a response to the changes in reliability requirements and transition probabilities.

Calinescu et al. [11] proposed a framework for the development and runtime operation of autonomic IT systems. It uses Markov chains to model system behaviors and tunes behavioral parameters by an exhaustive quantitative verification to maximize a multi-objective utility policy.

Calinescu et al. [8] proposed QoSMOS for QoS management of service-based systems. It uses formal specification of QoS requirements with probabilistic temporal logics, supports model-based QoS evaluation by quantitative verification, exploits KAMI to support updating of the QoS models, and supports system adaptations through service selection and resource assignment.

Ghezzi et al. [9] proposed a framework ADAM to manage non-functional uncertainty. It exploits Markov Decision Processes to model self-adaptive systems that have alternative and optional functionality implementations. According to the aggregated quality values up to the current state, it finds the execution path with the highest probability to satisfy non-functional requirements, and enables adaptations by switching to alternative implementations or skipping optional functionalities.

Moreno et al. [27] developed a proactive latency-aware adaptation approach, which takes into account the stochastic behavior of the environment. It uses Markov Decision Processes to specify the adaptation decisions and uses probabilistic model checking to resolve the decisions at runtime.

Our approach shares the common ground of quantitative verification with these approaches. However, they only consider quality constraints, i.e., either to ensure one quality dimension or to trade-off multiple quality dimensions. Differently, our approach makes trade-offs among functional and quality constraints holistically from the perspective of business value. Moreover, these approaches assume that architectural alternatives are predefined at design time, while our approach can automatically generate them.

It is worth mentioning that, for those approaches using high-level models (e.g., component or goal models as in Sections 10.1 and 10.2), it is relatively easy to construct models but difficult to handle fine-grained behavioral adaptations. Differently, for the approaches using low-level formal models (e.g., Discrete-Time Markov Chains), it is difficult to construct models especially for complex systems but natural to support fine-grained behavioral adaptations.

10.4 Genetic Algorithms in Self-Adaptive Systems

Genetic algorithms have been recently used in self-adaptive systems due to the performance in a large search space. Canfora et al. [61] employed genetic algorithms to achieve QoS-aware service composition by selecting a concrete service for

each abstract service. Similarly, Pascual et al. [62] used genetic algorithms for runtime adaptations of mobile applications by deciding whether a feature needs to be enabled or not. Tan et al. [63] used genetic algorithms to partially explore a labeled transition system to find a near-optimal recovery plan in case of runtime failures.

These approaches assume a predefined alternative space and an absolute satisfaction of functional constraints. Differently, we propose to generate the alternatives automatically and regard the probabilistic violations of relaxed functional constraints as the trade-offs with other constraints.

11 Conclusions

We have proposed a new architecture-based self-adaptation approach in this paper, which focuses on behavioral adaptations and supports relaxed functional constraints. It can automatically generate behavioral alternatives of a system at the specification level from the currently-employed behavioral specification. It regards the probabilistic violation of relaxed functional constraints as the trade-offs with other constraints from the perspective of business value. It employs a genetic algorithm-based planning technique to efficiently search for the optimal behavioral alternative that can offer the best business value. The experimental study on an online order processing benchmark has shown promising results of the effectiveness and scalability of our approach.

In the future, we plan to provide some methodological support so that system architects can provide the behavioral specification, constraints and correction rules and perform the annotation in a more convenient way. Besides, we intend to extend our approach to support widely-used architecture description languages and industrial modeling languages to improve applicability. We also plan to support more adaptation operations, and explore the costs of different adaptation operations and their impacts on the effectiveness of adaptations. Moreover, we hope to investigate the possibilities of using parametric or incremental model checking techniques and parallel genetic algorithms to further improve the scalability of our approach.

Acknowledgments

This work is supported by the National High Technology Development 863 Program of China under the Grant No. 2015AA01A203; and is supported in part by the National Research Foundation, Prime Minister's Office, Singapore under its National Cybersecurity R&D Program (Award NRF2014NCR-NCR001-30) and administered by the National Cybersecurity R&D Directorate.

References

[1] B. H. Cheng, R. Lemos, H. Giese, and et al., "Software engineering for self-adaptive systems: A research roadmap," in *Software Engineering for Self-Adaptive Systems*, 2009, pp. 1–26.

[2] H. Mei, G. Huang, and T. Xie, "Internetware: A software paradigm for internet computing," *Computer*, vol. 45, no. 6, pp. 26–31, 2012.

[3] E. Di Nitto, C. Ghezzi, A. Metzger, M. Papazoglou, and K. Pohl, "A journey to highly dynamic, self-adaptive service-based applications," *Automated Software Engineering*, vol. 15, no. 3-4, pp. 313–341, 2008.

[4] R. Lemos, H. Giese, H. Müller, and et al., "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, 2013, pp. 1–32.

[5] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer*, vol. 37, no. 10, pp. 46–54, 2004.

[6] A. Elkhodary, N. Esfahani, and S. Malek, "FUSION: A framework for engineering self-tuning self-adaptive software systems," in *FSE*, 2010, pp. 7–16.

[7] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "QoS-driven runtime adaptation of service oriented architectures," in *ESEC/FSE*, 2009, pp. 131–140.

[8] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE Trans. Softw. Eng.*, vol. 37, no. 3, pp. 387–409, 2011.

[9] C. Ghezzi, L. S. Pinto, P. Spoletini, and G. Tamburrelli, "Managing non-functional uncertainty via model-driven adaptivity," in *ICSE*, 2013, pp. 33–42.

[10] B. Chen, X. Peng, Y. Yu, and W. Zhao, "Requirements-driven self-optimization of composite services using feedback control," *IEEE Trans. Serv. Comput.*, vol. 8, no. 1, pp. 107–120, 2015.

[11] R. Calinescu and M. Kwiatkowska, "Using quantitative analysis to implement autonomic it systems," in *ICSE*, 2009, pp. 100–110.

[12] H. Ma, F. Bastani, I.-L. Yen, and H. Mei, "Qos-driven service composition with reconfigurable services," *IEEE Trans. Serv. Comput.*, vol. 6, no. 1, pp. 20–34, 2013.

[13] R. Adler, I. Schaefer, M. Trapp, and A. Poetzsch-Heffter, "Component-based modeling and verification of dynamic adaptation in safety-critical embedded systems," *ACM Trans. Embed. Comput. Syst.*, vol. 10, no. 2, pp. 20:1–20:39, 2011.

[14] 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.

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

[16] B. Boehm, "Value-based software engineering: Reinventing "earned value" monitoring and control," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, 2003.

[17] J. Zhang, Y. Liu, J. Sun, J. S. Dong, and J. Sun, "Model checking software architecture design," in *HASE*, 2012, pp. 193–200.

[18] J. Sun, Y. Liu, J. S. Dong, and J. Pang, "Pat: Towards flexible verification under fairness," in *CAV*, 2009, pp. 709–714.

[19] R. Allen and D. Garlan, "A formal basis for architectural connection," *ACM Trans. Softw. Eng. Methodol.*, vol. 6, no. 3, pp. 213–249, 1997.

[20] J. Sun, S. Song, and Y. Liu, "Model checking hierarchical probabilistic systems," in *ICFEM*, 2010, pp. 388–403.

[21] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, "Engineering self-adaptive systems through feedback loops," in *Software Engineering for Self-Adaptive Systems*. Springer-Verlag, 2009, pp. 48–70.

[22] X. Peng, Y. Wu, and W. Zhao, "A feature-oriented adaptive component model for dynamic evolution," in *CSMR*, 2007, pp. 49–57.

[23] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Model evolution by run-time parameter adaptation," in *ICSE*, 2009, pp. 111–121.

[24] C. Baier and J.-P. Katoen, *Principles of model checking*. The MIT Press, 2008.

[25] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in *NSDI*, 2013, pp. 313–328.

[26] X. Peng, B. Chen, Y. Yu, and W. Zhao, "Self-tuning of software systems through dynamic quality tradeoff and value-based feedback control loop," *J. Syst. Softw.*, vol. 85, no. 12, pp. 2707–2719, 2012.

[27] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: A probabilistic model checking approach," in *ESEC/FSE*, 2015, pp. 1–12.

[28] K. G. Palepu, *Business analysis and valuation*. Cengage Learning EMEA, 2007.

[29] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization*. Wiley New York, 1988.

[30] X.-S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[31] R. L. Haupt and S. E. Haupt, *Practical genetic algorithms*. John Wiley & Sons, 2004.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2016.2593459, IEEE Transactions on Services Computing

[32] M. Harman, "The current state and future of search based software engineering," in *FOSE*, 2007, pp. 342–357.

[33] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *ICNN*, 1995, pp. 1942–1948.

[34] P. Oreizy, N. Medvidovic, and R. N. Taylor, "Architecture-based runtime software evolution," in *ICSE*, 1998, pp. 177–186.

[35] E. M. Dashofy, A. V. d. Hoek, and R. N. Taylor, "A highly-extensible, xml-based architecture description language," in *WICSA*, 2001, pp. 103–.

[36] C. Daws, "Symbolic and parametric model checking of discrete-time markov chains," in *ICTAC*, 2005, pp. 280–294.

[37] E. Hahn, H. Hermanns, and L. Zhang, "Probabilistic reachability for parametric markov models," *Int. J. Softw. Tools Technol. Transfer*, vol. 13, no. 1, pp. 3–19, 2011.

[38] A. Filieri, C. Ghezzi, and G. Tamburrelli, "Run-time efficient probabilistic model checking," in *ICSE*, 2011, pp. 341–350.

[39] V. Forejt, M. Kwiatkowska, D. Parker, H. Qu, and M. Ujma, "Incremental runtime verification of probabilistic systems," in *RV*, 2012, pp. 314–319.

[40] O. Nierstrasz, M. Denker, and L. Renggli, "Model-centric, context-aware software adaptation," in *Software Engineering for Self-Adaptive Systems*, 2009, pp. 128–145.

[41] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using architecture models for runtime adaptability," *IEEE Softw.*, vol. 23, no. 2, pp. 62–70, 2006.

[42] B. Morin, O. Barais, G. Nain, and J.-M. Jezequel, "Taming dynamically adaptive systems using models and aspects," in *ICSE*, 2009, pp. 122–132.

[43] N. D'Ippolito, V. Braberman, J. Kramer, J. Magee, D. Sykes, and S. Uchitel, "Hope for the best, prepare for the worst: Multi-tier control for adaptive systems," in *ICSE*, 2014, pp. 688–699.

[44] 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.

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

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

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

[48] 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.

[49] W. Qian, X. Peng, B. Chen, J. Mylopoulos, H. Wang, and W. Zhao, "Rationalism with a dose of empiricism: Combining goal reasoning and case-based reasoning for self-adaptive software systems," *Requir. Eng.*, vol. 20, no. 3, pp. 233–252, 2015.

[50] 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.

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

[52] B. H. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *MODELS*, 2009, pp. 468–483.

[53] L. Baresi and L. Pasquale, "Adaptive goals for self-adaptive service compositions," in *ICWS*, 2010, pp. 353–360.

[54] R. Calinescu and S. Kikuchi, "Formal methods @ runtime," in *FOCS*, 2011, pp. 122–135.

[55] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad, "A survey of formal methods in self-adaptive systems," in *C3S2E*, 2012, pp. 67–79.

[56] R. Calinescu, Y. Rafiq, K. Johnson, and M. E. Bakir, "Adaptive model learning for continual verification of non-functional properties," in *ICPE*, 2014, pp. 87–98.

[57] A. Filieri, L. Grunske, and A. Leva, "Lightweight adaptive filtering for efficient learning and updating of probabilistic models," in *ICSE*, 2015, pp. 200–211.

[58] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, pp. 69–77, 2012.

[59] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Form. Asp. Comput.*, vol. 24, no. 2, pp. 163–186, 2012.

[60] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio, "Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements," in *ASE*, 2011, pp. 283–292.

[61] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *GECCO*, 2005, pp. 1069–1075.

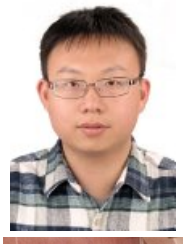
[62] G. Pascual, M. Pinto, and L. Fuentes, "Run-time adaptation of mobile applications using genetic algorithms," in *SEAMS*, 2013, pp. 73–82.

[63] T. H. Tan, M. Chen, É. André, J. Sun, Y. Liu, and J. S. Dong, "Automated runtime recovery for qos-based service composition," in *WWW*, 2014, pp. 563–574.

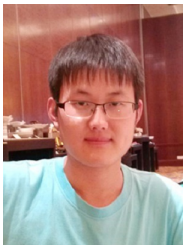
Bihuan Chen received his Bachelor and PhD degrees in computer science from Fudan University in 2009 and 2014. Now he is a post-doctoral research fellow in Nanyang Technological University. His research currently focuses on self-adaptive systems, program analysis, and software testing.

Xin Peng received his Bachelor and PhD degrees from Fudan University in 2001 and 2006, where he is now a full professor of the School of Computer Science. His research interests include self-adaptive system, mobile and cloud computing, software maintenance and evolution. He serves on several program committees of prestigious international conferences in software engineering such as ICSM, RE, ICSR, SPLC, and ICPC. His work won the Best Paper Award at the 27th International Conference on Software Maintenance (ICSM11).

Yang Liu received his Bachelor and PhD degrees in computer science from National University of Singapore (NUS) in 2005 and 2010, and continued with his postdoctoral research in NUS. Since 2012, he joined Nanyang Technological University as an Assistant Professor. His current research focuses on software engineering, formal methods and security, and particularly specializes in software verification using model checking techniques, which led to the development of a state-of-the-art model checker, Process Analysis Toolkit.



Songzheng Song received his PhD degree in computer science from National University of Singapore in 2013. After that, he worked as a postdoctoral research fellow in Nanyang Technological University, and now he works in Autodesk as a software engineer. His current research interests focus on formal methods and multi-agent systems.



Jiahuan Zheng is working toward the PhD degree in the School of Computer Science at Fudan University. His PhD work mainly concerns on self-adaptive systems and mobile computing.

Wenyun Zhao received his Master's degree in computer science from Fudan University in 1989. He is a full professor of the School of Computer Science at Fudan University. His current research interests include software reuse, software product line, software component, and software architecture.