Requirements-Driven Self-Optimization of Composite Services using Feedback Control

Bihuan Chen, Xin Peng, Yijun Yu, *Member, IEEE* and Wenyun Zhao

Abstract—In an uncertain and changing environment, a composite service needs to continuously optimize its business process and service selection through runtime adaptation. To achieve the overall satisfaction of stakeholder requirements, quality tradeoffs are needed to adapt the composite service in response to the changing environments. Existing approaches on service selection and composition, however, are mostly based on quality preferences and business processes decisions made statically at the design time. In this paper, we propose a requirements-driven self-optimization approach for composite services. It measures the quality of services (QoS), estimates the earned business value, and tunes the preference ranks through a feedback loop. The detection of unexpected earned business value triggers the proposed self-optimization process systematically. At the process level, a preference-based reasoner configures a requirements goal model according to the tuned preference ranks of QoS requirements, reconfiguring the business process according to its mappings from the goal configurations. At the service level, selection decisions are optimized by utilizing the tuned weights of QoS criteria. We used an experimental study to evaluate the proposed approach. Results indicate that the new approach outperforms both fixed-weighted and floating-weighted service selection approaches with respect to earned business value and adaptation flexibility.

Index Terms—QoS, quality tradeoffs, self-optimization, earned business value, process reconfiguration, service selection.

1 Introduction

SERVICE-oriented architecture (SOA) has been an emerging paradigm for developing and integrating business applications. For a service-oriented system, the development focus has shifted from in-house custom application construction to the design of business processes and the selection and composition of services [1]. For Web services orchestration in particular, BPEL (Business Process Execution Language) [2] is widely used to specify the business process of a composite service based on the interactions between the composite service and external Web services.

Apart from functional requirements concerning the business logic, composite services should also fulfill non-functional requirements concerning the quality of services (QoS) [1]. In a changing and uncertain environment, however, composite services cannot always run optimally with statically configured business processes and selected services that only reflect design-time decisions which may not hold at runtime. Therefore, self-optimization through runtime adaptation is a promising way for composite services to better meet their overall QoS requirements [3].

To address the problem of runtime adaptation for composite services, a number of service selection and composition approaches have been recently proposed. Given a statically configured business process consisting of a set of abstract services, these approaches use local [4], [5], [6], global [1], [4], [7], [8], heuristic [9] or hybrid [10], [11] optimizations to dynamically select and bind a concrete service for each abstract service. The objective is to maximize user satisfaction as well as to meet all the QoS constraints. The user satisfaction is usually expressed by a utility function weighing multiple QoS criteria with fixed QoS weights.

However, these approaches are based on static decisions about the quality preferences and the business processes made at design time, i.e., the weights of QoS criteria and the structure of business processes used in service selection and composition are determined at design time and do not change at runtime. Such static decisions may become unfashionable at runtime, making composite services run in a sub-optimal manner.

On the one hand, to better optimize the overall satisfaction of stakeholder requirements, dynamic quality tradeoffs are needed to adapt a system in response to the changing environments [12]. For example, it is meaningless for an order processing service to keep a high preference to risk prevention (which requires a large amount of time and resource for additional processing such as credit verification) when its response time is so long that most users may lose their patience.

On the other hand, considering process variability like optional activities and alternative sub-processes, there are often several alternative configurations that fulfill the same business goals but fit differently in the QoS requirements. Therefore, business processes need

- *X. Peng is the corresponding author.*
- *B. Chen, X. Peng and W. Zhao are with the School of Computer Science and the Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China.*
  E-mail: {bhchen, pengxin, wyzhao}@fudan.edu.cn
- *Y. Yu is with the Center for Research in Computing, The Open University, Milton Keynes, UK.*
  E-mail: y.yu@open.ac.uk

to be flexible so as to be reconfigured to accommodate different preferences to the QoS requirements [13]. For example, following the preference tradeoff decision of response time over risk prevention, the order processing process can be reconfigured to temporarily skip credit verification to improve the response time.

The above analysis motivates the need for requirements-driven self-optimization of composite services. First, dynamic quality tradeoffs should be conducted for maximizing the overall satisfaction of stakeholder requirements. Second, system specifications should be dynamically planned and adapted based on runtime requirements models capturing the solution space for high-level requirements. Third, business processes of composite services should be reconfigured and concrete services be selected dynamically to reflect the adapted quality preferences and system specifications.

Further, apart from the technical benefits such as efficient development and graceful evolution, the main driver for adopting SOA is to create business value for stakeholders not only at design time but also at runtime. Value-based software engineering [14] is proposed to integrate value consideration into software engineering principles and practices. Its emphasis is to incorporate business value achievement into feedback control systems. Therefore, self-optimization for composite services should take into account the business value achievement, which can usually be measured by successfully committed transactions.

Motivated by the ideas of requirements-driven and value-based self-optimization, in this paper, we propose a new approach for self-optimization of composite services with the following characteristics:

- support dynamic quality tradeoffs;
- combine runtime process reconfiguration and service selection;
- use earned business value to reflect the runtime overall satisfaction of stakeholder requirements.

Measuring quality attributes and estimating earned business value using a predefined value formula, our approach makes dynamic quality tradeoffs, i.e., tunes the preference ranks of quality attributes using a feedback controller. The detection of a violation of earned business value triggers our self-optimization process systematically. At the business level, a preference-based reasoner [12] configures a requirements model according to the tuned preference ranks of QoS requirements; and then a process configurator reconfigures the business process according to its mappings from goal configurations. At the service level, service selection is optimized by integrating the selection approach in [10] with the tuned weights of QoS criteria.

To evaluate the effectiveness of our approach, we conducted an experimental study on an order processing service adapted from an IBM's sample example. The results show the improvement of our approach over both fixed-weighted and floating-weighted service selection approaches in terms of earned business value and adaptation flexibility. The study also demonstrates the acceptable performance of our approach and shows the rationality of combining process reconfiguration and service selection for self-optimization of composite services.

The rest of the paper is organized as follows. Section 2 introduces a number of existing proposals and compares them with ours. Section 3 introduces goal models and process variability on top of which we build our proposal. Section 4 presents our requirements-driven self-optimization approach. Section 5 gives the details of our implementation. Section 6 evaluates the proposed approach. Section 7 makes some discussion before Section 8 draws our conclusions.

2 Related Work

Our work falls in the area of self-adaptive service-oriented applications [3]. Here we focus our discussion on the most closely related studies in three areas: service selection and composition, business process adaptation, and requirements-driven self-adaptation.

2.1 Service Selection and Composition

Service selection and composition have been a challenging problem because the number of the candidate services that have the same functionalities but differ in QoS is increasing with the prevalence of SOA, Cloud Computing and Software as a Service.

In the local optimization approaches [4], [5], [6], service selections for different abstract services are independent of each other. Given a utility function weighing multiple QoS criteria, for each abstract service, a concrete service with the highest utility is selected. Though usually efficient, local optimization approaches cannot satisfy end-to-end constraints, and the selection result is usually not globally optimal.

The global optimization approaches [1], [4], [7], [8] turn the optimization problem into a mixed integer programming problem, i.e., maximizing an objective function weighing multiple QoS criteria as well as meeting multiple QoS constraints. They can always satisfy end-to-end constraints and produce an optimal selection result when the problem is solvable, but the time complexity is exponential.

Attempting to overcome the shortcomings of local and global approaches, Yu et al. [9] propose to model this problem as a multi-dimension multi-choice 0-1 knapsack problem and a multi-constraint optimal path problem, and to use efficient heuristic algorithms to find near-optional solutions. Alrifai et al. [10] and Sun and Zhao [11] propose hybrid approaches that first decompose end-to-end QoS constraints into local QoS constraints and then perform local selections, thus combining both the efficiency of local approaches and the optimality of global approaches.

These approaches pay their major attention to the selection algorithm itself with an assumption that

QoS weights given by experts at design time will not change at runtime. Complementary to them, our approach allows the QoS weights to be dynamically tuned to reflect quality tradeoffs using a feedback controller. Our approach further supports dynamic reconfiguration of business processes, i.e., changes the structure and behavior of business processes, from the requirements perspective.

2.2 Business Process Adaptation

To the best of our knowledge, dynamic runtime business process reconfiguration is rarely systematically addressed in literature, which mainly faces two challenges. One is to make business processes adaptable at runtime, i.e., process variability can be well represented at design time as well as be dynamically configured at runtime. The other is to provide appropriate decision and control mechanisms for achieving QoS-driven self-adaptation.

For the first challenge, several advances have been made to model variability [17], [18] and support adaptation [18], [19] of BPEL processes. Topaloglu et al. [17] differentiate architecture-oriented and task-oriented variability by the criterion that whether architectural elements (e.g., type of protocol) or Web services are involved. Charfi and Mezini [19] propose AO4BPEL, an aspect-oriented extension to BPEL, to achieve runtime process adaptation by dynamic weaving. Koning et al. [18] propose VxBPEL, an extension to BPEL, to capture and model variability for BPEL services. They also implement a prototype of runtime adaptation by extending an existing BPEL engine. These approaches focus on how to make business processes adaptable and provide the infrastructure for runtime adaptation. Compared to them, our approach only uses standard BPEL elements to model variability, but we make a step forward to provide supports for the adaptation decision and control mechanisms.

For the second challenge, Lapouchnian et al. [13] propose a requirements-driven approach for the design and configuration management of business processes. They use goal models to capture alternative business process variability and generate BPEL processes. Their approach supports process variability customization based on user-prescribed preference ranks of relevant QoS requirements. While this approach supports limited process variability types and only deployment-time customization, thus cannot be used for headless (unsupervised) self-adaptation, our approach focuses on runtime process reconfiguration and service selection without human intervention.

2.3 Requirements-Driven Self-Adaptation

Self-adaptive systems should be requirements-aware because these systems are increasingly running under poorly-understood environments [16]. Several advances have been made on requirements-driven self-adaptation. Wang et al. [20] propose a requirements monitoring and diagnosing approach, which finds a set of goal configurations that are free of failures by goal reasoning and then selects the best one for repairing the system from failures. Salehie et al. [21] propose a requirements-driven approach to support adaptive security for protecting variable assets at runtime. They respectively focus on self-repairing and self-protecting while our work focuses on self-optimization.

In earlier work we have proposed a general framework for value-based and requirements-driven self-optimization [12], where a PID controller is proposed to achieve dynamic quality tradeoffs and a preference-based reasoner is proposed to reason about the optimal goal configuration. Following this, we propose to integrate dynamic quality tradeoffs with dynamic functional tradeoffs, i.e., sacrificing desirable components to assure crucial components, for the survivability of Web systems [22]. Compared with our earlier work, in this paper, we apply the similar value-based feedback control loop with a deeper exploration of earned business value and the application domain changed to self-optimization for composite services. Further, we revise the PID controller in [12] by combining a PD controller to stabilize the preference tuning process by avoiding overshoots, and use dynamic quality tradeoffs to drive both process reconfiguration and service selection. To the best of our knowledge, our work is the first one that combines requirements-driven process reconfiguration with service selection on the basis of dynamic quality tradeoffs.

3 Baseline

We use requirements goal models [15] to capture business goals and explore business alternatives because they provide sufficient modeling and reasoning support at an appropriate abstraction level [16] (Section 3.1). And we identify three types of process variability to reflect business process variations (Section 3.2). We also establish a mapping schema from goal variations to process variations for mapping goal reconfigurations to process reconfigurations (Section 3.3).

3.1 Goal Models

Goal-oriented requirements engineering (GORE) [15] as a methodology has been widely used in the past two decades. In GORE, stakeholders are usually modeled by actors and their intentions are represented by goals. Goal models are used to capture intentions of stakeholders on the system-to-be and explore alternative ways to satisfy these intentions. Once the relationships between agents and requirements are elicited, a goal model is constructed to represent AND/OR refinements of requirements.

Requirements that have clear-cut satisfaction criteria are modeled as hard goals or tasks in goal models, and others that do not have are modeled as soft goals. For example, Fig. 1 shows the goal model of order

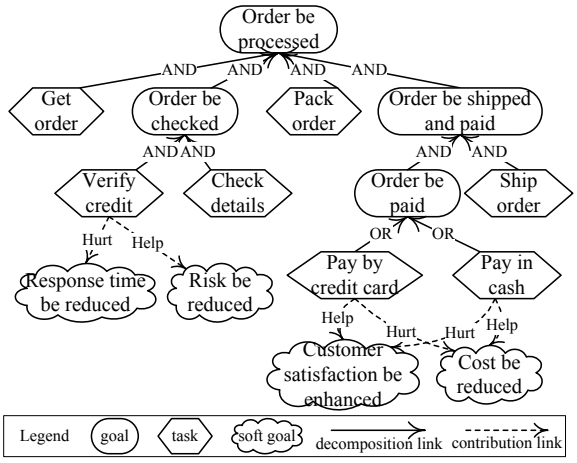


Fig. 1. Goal model of order processing.

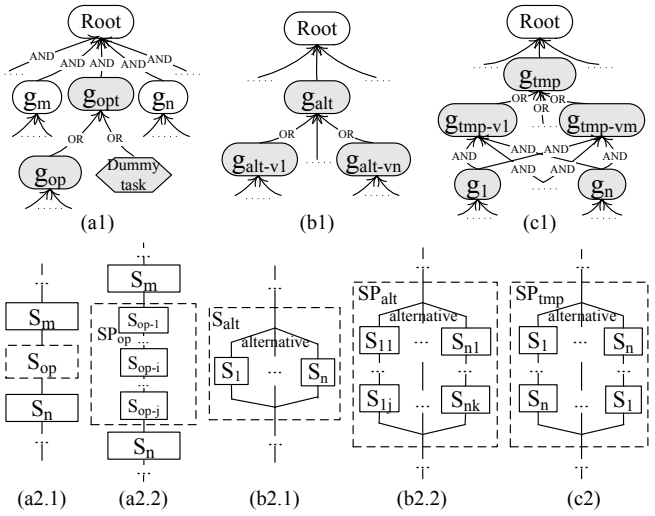


Fig. 2. Mapping from goal variations to process variations (gray ones for variability-related goals).

processing, where *Order be checked* is a hard goal that is either fulfilled or not and *Customer satisfaction be enhanced* is a soft goal that can be fulfilled or not to some degree. Most quality requirements can be seen as soft goals, which have no optimal but only "good enough" satisfactions. Such uncertainty of quality requirements makes soft goals suitable for expressing the criteria for comparing alternative design choices [23].

Goals can be refined into subgoals by AND/OR decomposition links until leaf-level tasks can be accomplished by either a software component or a human agent. To fulfill an AND/OR-decomposed goal, all/at least one of its subgoals should be fulfilled. Therefore, goal models can capture alternative ways for fulfilling high-level goals by OR-decompositions. For example, *Get order*, *Order be checked*, *Pack order* and *Order be shipped and paid* in Fig. 1 have to be all fulfilled to fulfill *Order be processed*; and *Order be paid* can be fulfill either by *Pay by credit card* or *Pay in cash*.

Goals can be related to soft goals through a series of contribution links such as *Help*, *Hurt*, *Make*, *Break*. A *Help*/*Make* link indicates that the fulfillment of the source goal contributes to the partial/full fulfillment of the target goal, whilst a *Hurt*/*Break* link indicates that the fulfillment of the source goal contributes to the partial/full denial of the target goal. Usually, each subgoal of an OR-decomposed goal has different contributions to certain soft goals, implying different quality tradeoff decisions. For example, *Pay by credit card* helps to enhance customer satisfaction for its convenience but hurts to reduce the cost due to the expense paid to third-party payment service providers; its alternative goal *Pay in cash* contributes to these two soft goals in the other way around.

3.2 Process Variability

A business process often has several process variants, each of them constituting an adjustment of a reference process model to accommodate specific requirements in the process context [24]. Such process variability forms the basis of runtime process reconfiguration in our approach. For example, both regular verification (e.g., checking order details such as the quantity and total amount of products) and credit verification (e.g., checking if customers cheat in online shopping) are required during regular order processing to reduce the transaction risk. However, when the load of service requests is high, it may be reasonable to skip credit verification to improve throughput. Variability of a system usually can be characterized by variation points and variants. A variation point is a certain part in a system that can vary, and the alternatives for such a variation point are called variants [18].

To support runtime process reconfiguration, three types of variability in business processes are identified as follows:

- **Optional**: An optional service or sub-process can be either invoked (if bound) or skipped (if unbound) during the execution of a process.
- **Alternative**: An alternative service or sub-process has several variant services or sub-processes, one and only one of which can be bound at runtime.
- **Temporal**: A set of services constituting a sub-process can be invoked in different orders. A sub-process with temporal variability can be seen as a special kind of alternative sub-process.

3.3 Mapping from Goal Variations to Process Variations

Fig. 2 shows our mapping schema from goal variations to the three types of process variations for mapping goal configurations to process reconfigurations. Fig. 2 (a1), (b1), and (c1) give the goal variations, and Fig. 2 (a2.1) and (a2.2), (b2.1) and (b2.2), and (c2) show the corresponding process variations.

For an optional goal, e.g., g_{op} in Fig. 2 (a1), an OR-decomposition is introduced with a new goal g_{opt} as its parent goal and a virtual *Dummy task* (i.e., nothing to achieve) as its sibling goal. Thus, an optional goal

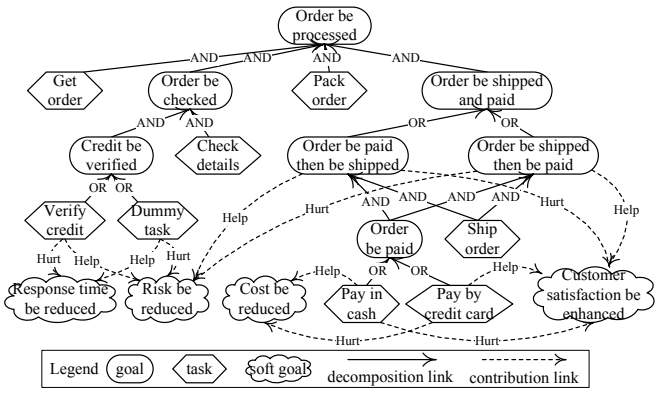


Fig. 3. Goal model of order processing with variability.

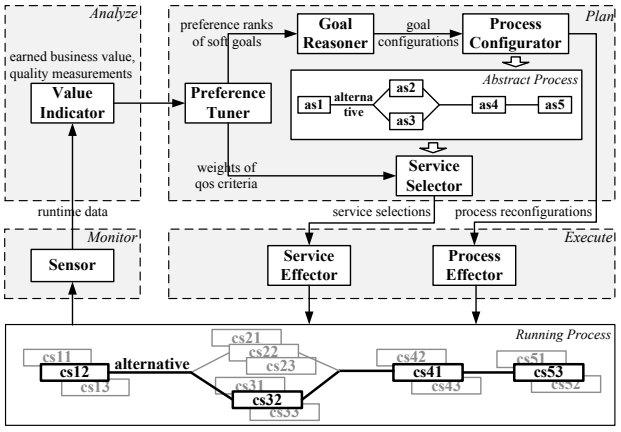


Fig. 4. Overview of Our Approach.

can be bound or unbound during process execution, and can be mapped to an optional service S_{op} in Fig. 2 (a2.1) or an optional sub-process SP_{op} in Fig. 2 (a2.2).

For an OR-decomposed goal with alternatives, e.g., g_{alt} in Fig. 2 (b1), its subgoals g_{alt-v1} to g_{alt-vn} can be seen as its variants, and can be mapped to the alternative service S_{alt} in Fig. 2 (b2.1) or the alternative sub-process SP_{alt} in Fig. 2 (b2.2) depending on if these subgoals correspond to individual services or composite services.

For a temporal goal, e.g., g_{tmp} in Fig. 2 (c1), it is OR-decomposed into subgoals g_{tmp-v1} to g_{tmp-vm}, each of which represents a possible execution order of the same set of goals g_1 to g_n. Thus, g_{tmp} can be regarded as an alternative goal, and its subgoals can be mapped to the alternative sub-process SP_{tmp} in Fig. 2 (c2).

Fig. 3 illustrates the goal model of order processing after introducing temporal and optional variability and corresponding contribution links into the goal model in Fig. 1. The two temporal variant goals *Order be paid then be shipped* and *Order be shipped then be paid* represent two different execution orders of payment and shipping. The former helps to reduce risk but hurts to enhance customer satisfaction, whereas the latter contributes to them in the other way around. And the optional goal *Verify credit*, if bound, can help to reduce risk but hurt to reduce response time.

With these goal variants and their contribution links represented in goal models and the preference ranks of QoS requirements dynamically tuned (Section 4.3), we can then reason about and reconfigure goal variations and map goal configurations to process reconfigurations at runtime based on this mapping schema (Section 4.4). For example, if customer satisfaction is preferred to cost and thus *Order be paid* is configured to *Pay by credit card*, the business process will be reconfigured to execute the credit card payment service.

4 Our Approach

In this section, we first present an overview of our approach, and then explain in detail the key techniques involved in our approach, including runtime value indicator, dynamic quality tradeoffs, and process reconfiguration and service selection.

4.1 Approach Overview

To achieve self-optimization for composite services, our approach continuously seeks opportunities to improve the earned business value (i.e., the indicator of overall satisfaction of QoS requirements) by making dynamic quality tradeoffs and then performing process reconfiguration and service selection accordingly. Fig. 4 shows the overview of our approach, with a mapping of its main components to the MAPE (Monitor, Analyze, Plan, Execute) control loop [25].

In particular, a **value indicator** is executed at regular *intervals* (e.g., one minute) at runtime to measure the quality attributes from runtime data collected by sensors and estimate the earned business value on the basis of a predefined value formula (Section 4.2).

Taking runtime quality measurements and earned business value as feedbacks, a control-theoretic **preference tuner** is used to make dynamic quality tradeoffs, i.e., tune the preference ranks of relevant quality attributes, to reflect the relative importance of those quality attributes (Section 4.3).

Following the dynamic quality tradeoffs, our self-optimization is triggered by a violation of expected earned business value and is conducted at the process and service levels respectively (Section 4.4). The adaptations at the two levels, i.e., process reconfiguration and service selection, are decoupled: the former only concerns abstract services and their process flows and the latter only concerns concrete services and service replacement.

At the process level, requirements-driven process adaptation is conducted. A preference-based **goal reasoner** configures the requirements goal model according to the tuned preference ranks of QoS requirements (soft goals). The goal configurations are then mapped to process reconfigurations by **process configurator** based on the mapping schema in Fig. 2.

At the service level, a **service selector** is used to find an optimal combination of concrete services from can-

didate services for the abstract services, with the objective of maximizing the utility function and meeting the QoS constraints. The utility function is computed on relevant QoS criteria with floating weights tuned through dynamic quality tradeoffs.

Note that our approach currently focuses on the **Analyze** (value indicating) and **Plan** (preference tuning, process reconfiguration and service selection) parts which are detailed in the following subsections, and simplifies the **Monitor** and **Execute** parts which are usually supported by service-oriented middlewares.

4.2 Runtime Value Indicator

The objective of self-optimization for composite services can be regarded as maximizing the value propositions of stakeholders [12], [22]. Following this value-based perspective, we propose to use a value indicator to measure the earned business value and then to determine whether the objective is met. To this end, it is required to define a quantitative value measurement that can provide comprehensive and timely feedback for the self-optimization process.

For composite services, earned business value usually can be measured as the accumulative value earned from every successfully committed transaction (e.g., completing an order processing) in a time interval (e.g., one minute). Further, an earning rule, such as the 0-100 rule and 50-50 rule, should be specified. If the 0-100 rule is used, no value will be earned until a transaction is successfully committed. On the contrary, the 50-50 rule means 50% of the value will be earned as long as the transaction is started.

In order to measure earned business value comprehensively, both business capabilities (i.e., what services a system has provided) and quality of services (i.e., how well the services are provided) should be taken into account because they both heavily influence earned business value. For example, if a business process is finished successfully but takes a long time, or it is completed quickly but suffers a failure, the earned business value will be limited due to either poor quality or weak capability.

On the other hand, for measuring earned business value timely, some long-term influence factors should be captured as an instant positive or negative factor to earned business value rather than be captured when it actually makes the influence. For example, customer satisfaction often takes a long time and a slow process to be reflected in the business value of an organization; and it is too late to take actions to re-establish customer satisfaction once it produces notable bad influence (e.g., customer losing, sales decreasing, etc.).

Based on the above analysis, it can be seen that a predefined application-specific value formula should be provided to measure the earned business value on the basis of business market analysis, risk analysis, business losing trend analysis, etc. by business experts from the economic perspective.

For example, in order processing, different quality attributes can contribute to the earned business value from different perspectives: reducing response time and enhancing service availability are demanded for a higher system throughput under a given bandwidth; reducing cost is aimed for lower third-party service cost; reducing risk is targeted at a higher rate of valid orders; and enhancing customer satisfaction is aimed for a larger number of placed orders. Among these quality attributes, risk and customer satisfaction are long-term influence factors to earned business value. In this study, we use the 0-100 earning rule, and simplify the formula as

$$value = \$3 \times \#sucOrders - \$3 \times \#sucOrders \times risk - \$3 \times \#sucOrders \times (1 - satisfaction) - cost$$

on the assumption that every successfully processed order will produce a value of \$3 for the organization. For example, in a time interval, there are 10 successfully processed orders which are instantly influenced by the response time and availability, the risk is 10%, the customer satisfaction is 70%, the cost is \$2.8, then the earned business value is \$15.2.

Besides earned business value, the value indicator also measures quality attributes such as availability and response time by analyzing the collected runtime data (e.g., failure/success, response time of a service invocation, etc.). Both earned business value and individual quality measurements are used as feedbacks for the planning process (see Fig. 4).

4.3 Dynamic Quality Tradeoffs

To achieve dynamic quality tradeoffs, our approach dynamically tunes the preference ranks of quality attributes. While many tuning techniques have been proposed in the literature, such as analytic hierarchy process [26], multi-criteria analysis method [27] and machine learning-based technique [28], they more or less involve human intervention, making it infeasible to our headless self-adaptation approach. Therefore, the difficulty here is how to tune the preference ranks quantitatively and autonomously.

Intuitively, there is an approximately proportional relationship between the preference rank and expectation of a quality attribute, i.e., the better a quality attribute is expected, the higher its preference rank should be. For example, at the service level, if a service suffers failures frequently, the preference rank of availability should be increased such that another service with higher availability will be selected and then the actual availability will become close to its expectation; at the process level, if a process responses extremely slowly, the preference rank of response time should be increased such that another process with quicker response will be reconfigured and then the actual response time will become close to its expectation. Therefore, the error between the expected and actual

quality measurement reflects the degree of preference rank tuning; the error should be accumulated continuously to tune the preference rank precisely; and the error can be minimized by reflecting the tuned preference rank into the system behavior and structure. Thus a control-theoretic feedback controller [29] can be used for such preference tuning.

Earlier we have proposed a preference tuning algorithm using a PID (proportional-integral-derivative) controller [12]. However, we observe that the integral part in the PID controller tends to accumulate a larger error, i.e., compensating one error may introduce another in the reverse direction if a large change occurs in quality measurement. To avoid this overshoot and stabilize the tuning process, the integral part can be frozen in such situations, which is known as the PD (proportional-derivative) controller. Hence, we revise our earlier version of preference tuner by combining the PID controller with a PD controller. The tuner is designed to strengthen the previous tuning directions if obtaining positive feedbacks, and adjust the previous tuning directions if obtaining negative feedbacks.

Algorithm 1 The Procedure of Preference Tuning

```
1: procedure PREFERENCE-TUNING(qm)
2:     for i ← 1, l do
3:         exp[i] ← average of all the past qm[i]
4:         e_k[i] ← (exp[i] − qm[i])/exp[i] × isPositive[i]
5:         if |e_k[i]| > 1.0 then
6:             conVar[i] ← conVar[i]+k_p×(e_k[i]−e_{k−1}[i])+
   k_d × (e_k[i] − 2 × e_{k−1}[i] + e_{k−2}[i])
7:         else
8:             conVar[i] ← conVar[i]+k_p×(e_k[i]−e_{k−1}[i])+
   k_i × e_k[i] + k_d × (e_k[i] − 2 × e_{k−1}[i] + e_{k−2}[i])
9:         end if
10:        e_{k−2}[i] ← e_{k−1}[i], e_{k−1}[i] ← e_k[i]
11:        rank[i] ← INI_R[i] + INI_R[i] × conVar[i]
12:        if rank[i] > 10 then
13:            rank[i] ← 10
14:        else if rank[i] < 1 then
15:            rank[i] ← 1
16:        end if
17:    end for
18:    return rank
19: end procedure
```

Algorithm 1 shows the procedure of preference tuning. It takes as input the actual quality measurements, and returns the tuned preference ranks. The algorithm has an iterative loop over all the quality attributes that are concerned. It first computes the expected quality measurement as an average of all the past actual quality measurements (Line 3) because the expectation cannot be fixed and should be continuously updated from the old value to the new one that reflects the runtime environments. Then it obtains the current error signal as an incremental or decremental percentage (Line 4). Depending on whether a quality attribute is positive (i.e., the higher the better, e.g., availability) or negative (i.e., the lower the better, e.g., response time), the current error signal is multiplied by 1 or -1. Then the algorithm executes the PD controller if the quality measurement suffers a sudden large change (Line 5-6); otherwise, it executes the PID controller (Line 7-8). The control variable, representing the incremental or decremental percentage of a preference rank, accumulates the errors in an incremental manner such that only three past error signals are stored. Finally, it updates the error signals (Line 10) and tunes the preference rank according to the control variable and the initial preference rank (Line 11-16).

Note that the preference rank is a value between 1 and 10. And k_p, k_i and k_d are the controller parameters of proportional control, integral control and derivative control respectively. These controller parameters may vary from one control system to another, and can be specified at design time by the Ziegler Nichols method [29]. For example, in our study, k_p, k_i, k_d are respectively set to 0.5, 0.3, 0,2.

TABLE 1
Quality Model of Order Processing

Quality Attributes	Aggregation	Service	Process
Cost	Summation	√	√
Response Time	Average	√	√
Availability	Average	√	
Risk	Average		√
Customer Satisfaction	Average		√

4.4 Process Reconfiguration and Service Selection

QoS plays a significant role in both process-level and service-level optimization. Therefore, a quality model is needed to specify what quality attributes are concerned at each level and how to aggregate quality measurements from runtime executions. Table 1 shows the quality model of an order processing process. The quality attributes concerned at the service level are cost, response time and availability, while those concerned at the process level are cost, response time, risk and customer satisfaction. An aggregation function (e.g., summation, average, minimum) is used to measure each quality attribute of a BPEL process from its execution records in an interval.

In addition, QoS can be either provided by service providers directly (e.g., cost), recorded from execution monitoring (e.g., response time), fed back from customers (e.g., customer satisfaction) [5], or even given by domain experts (e.g., risk).

4.4.1 QoS-Driven Self-Optimization at the Two Levels

The adaptations at the process level and the service level can be regarded as adaptations on high-level strategies and low-level tactics respectively. Naturally, once a tactic-level adaptation is enough for the self-optimization, no further strategy-level adaptation is needed; if no tactic-level adaptation is suitable for the

self-optimization, a strategy-level adaptation should be planned and a corresponding tactic-level adaptation should also be made. Hence, service-level self-optimization should be conducted first; if it fails, then process-level self-optimization should be involved.

Based on the above analysis, we propose the procedure of our self-optimization process in Algorithm 2. It takes as input the given goal model and the claimed QoS constraints. The procedure is triggered at regular intervals (e.g., one minute). It first analyzes the runtime data to obtain the current earned business value and the actual quality measurements relevant to the service and process level in the previous interval (Line 2), gets the tuned preference ranks of relevant quality attributes by invoking Algorithm 1 (Line 3-4), and calculates the expected earned business value as an average of all the past earned business values (Line 5). The procedure terminates if the current earned business value is beyond the expectation by α; otherwise, it normalizes the preference ranks into weights of QoS criteria at the service level (Line 7) and tries to find a new service selection for the current process configuration (Line 8-9). If possible, the procedure puts the new selection into effect and terminates (Line 10-12); if not possible, the procedure tries to find a new process configuration (Line 14-15). If found, the procedure needs to find a service selection for the new process configuration and put them into effect (Line 16-19); otherwise, the procedure terminates.

Algorithm 2 The Procedure of Self-Optimization

```
1:  procedure SELF-OPTIMIZATION(gm, c)
2:      value, qm^ser, qm^pro ← analyze runtime data
3:      r^ser ← Preference-Tuning(qm^ser)
4:      r^pro ← Preference-Tuning(qm^pro)
5:      expValue ← average of all the past value
6:      if (value − expValue)/expValue < α then
7:          w ← normalize r^ser
8:          curCon ← get current process configuration
9:          sel ← Service-Selecting(c, w, curCon)
10:         curSel ← get current service selection
11:         if sel ≠ curSel then
12:             put sel into effect
13:         else
14:             goalCon ← Goal-Reasoning(gm, r^pro)
15:             con ← map goalCon to process configuration
16:             if con ≠ curCon then
17:                 sel ← Service-Selecting(c, w, con)
18:                 put con and sel into effect
19:             end if
20:         end if
21:     end if
22: end procedure
```

Note that there are four possible adaptation results in our self-optimization process: (a1) no adaptation is needed because earned business value satisfies its expectation; (a2) a new service selection is produced (service-level optimization); (a3) a new process configuration and its corresponding service selection are produced (process-level and service-level optimization); and (a4) no new service selection or process configuration can be produced although an adaptation is needed (optimization failure). Besides, the setting of α should involve considerations on the system's tolerance of earned business value fluctuation. The lower the tolerance is, the larger α should be. Hence, it should be determined through analysis of the business goal and policy. For example, in our study, α is set to -0.05.

4.4.2 Process Reconfiguration

The purpose of process reconfiguration is to configure the process variations such that the process can best satisfy the QoS requirements. To this end, we use a preference-based reasoner [12] to dynamically configure the runtime goal model. The reasoner takes as input the given goal model and the tuned preference ranks of soft goals. The reasoner first encodes the goal model elements into CNF (Conjunctive Normal Form) proposition formulas to feed a SAT solver and then has an iterative step to invoke the solver for finite times. It tries initially to find a configuration that can satisfy all the soft goals. If not possible, the lowest ranked soft goals will be removed from the encoding so as to satisfy the remaining soft goals, and so on, until either a configuration is found or all soft goals have been removed. It always terminates with a valid configuration because we assume the root hard goal is satisfied by design. Detailed descriptions about the reasoning algorithm can be found in [12].

For example, if response time and customer satisfaction are preferred to risk and cost, *Dummy task* will be selected for *Credit be verified* since it helps to reduce response time, which means the process will be reconfigured to skip the optional credit verification service; *Pay by credit card* and *Order be shipped then be paid* will be selected for *Order be paid* and *Order be shipped and paid* respectively since they help to enhance customer satisfaction, which means the process will be reconfigured to execute the alternative credit card payment service and to execute the shipment service before payment service.

4.4.3 Service Selection

Given the QoS constraints, QoS weights and a process configuration, the service selection is to find an optimal combination of concrete services for the abstract services in a business process, with the purpose of satisfying the constraints and maximizing a weighted utility function.

A number of service selection methods have been proposed. Here we integrate the method proposed in [10]. It uses mixed integer programming (MIP) to find the optimal decomposition of global QoS constraints into local ones. It then uses a local selection method to find for each abstract service a concrete service that satisfies the local constraints and maximizes

a local utility function. Details about the selection algorithm can be found in [10].

The utility function used in [10] is shown in Eq. 1 with CS being a combination of the candidate concrete services, $u(q_k(CS))$ being the utility value of CS on the k-th QoS criterion and w_k ($\sum_{k=1}^{n} w_k = 1$) being the weight of k-th QoS criterion.

$$U(CS) = \sum_{k=1}^{n} u(q_k(CS)) \times w_k \quad (1)$$

Different from fixed weights in [10], our approach allows QoS weights to be changed by dynamic quality tradeoffs, making the selection process better reflect the changing quality preferences. Note that each QoS weight w_k is the normalization of the corresponding preference rank r_k of a quality attribute by Eq. 2.

$$w_k = r_k \; / \; \sum_{k=1}^{n} r_k \quad (2)$$

5 Implementation

We implement the proposed approach in an extensible way such that a generic infrastructure is provided for self-optimization of composite services. The application-independent components, i.e., preference tuner, goal reasoner, process configurator and service selector are implemented as plug-ins. The application-specific components, i.e., sensor, value indicator, service effector and process effector, are programmed as RMI interfaces using RMI-IIOP technology to decouple the generic parts and the BPEL engine and to make them distributed and stand-alone. Thus, developers should provide their own implementation to these interfaces for applying our approach.

For example, when we apply the approach to order processing, sensors are implemented with log4j for recording the runtime data such as response time of a service invocation; value indicator is realized as introduced in Section 4.2; and service/process effectors are realized by dynamically updating process configuration and service selection result in XML files. Thus, the BPEL engine can use XPath at runtime to extract process configuration information to customize business processes and extract service selection information to bind selected concrete services.

Besides, currently we adopt a light-weight way, i.e., using standard BPEL elements, to support the three types of variability in executable business processes. Generally, they are realized through the **if-elseif** activity. Specifically, alternative variability is realized by putting in corresponding condition bodies the alternative services (or sub-processes); optional variability is realized with only one condition body being the optional service (or sub-process); temporal variability is realized by putting in corresponding condition bodies a set of services with possible execution orders. Note that condition expressions in the **if-elseif** activities are equality matches between the possible configurations of the variations and the current configuration.

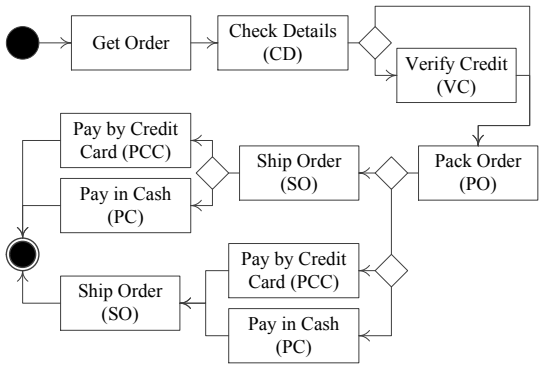


Fig. 5. BPEL Process of Order Processing.

6 Experimental Study

To evaluate the proposed approach, we conducted an experimental study on an order processing service to answer the following two questions:

- **Q1**: Can a composite service achieve better overall satisfaction (in terms of earned business value) and more adaptation flexibility by combining process reconfiguration and service selection? (effectiveness evaluation)
- **Q2**: How can our approach scale with the growth of goal models and business processes? (performance evaluation)

6.1 Experimental Setting

The order processing process is adapted from an IBM's sample example[1] by extending the following variations to make reconfiguration possible: making credit verification optional, adding two alternative payment services, and adding the temporal variability between shipment and payment.

The order processing goal model is shown in Fig. 3, its quality model is shown in Table 1, and the corresponding BPEL process is illustrated in Fig. 5. The BPEL process and its candidate concrete services were developed in Oracle JDeveloper 10.1.3.1 and deployed on Oracle BPEL Process Manager 10.1.3.1.0. The experiments were conducted on a ThinkPad R400 laptop with 2 Intel Core2 Duo 2.53 GHz processors and 2GB RAM, running Windows XP.

For simplicity, we assume that five candidate concrete services have been discovered for each abstract service in the BPEL process in Fig. 5 except for service "Get Order" which can be seen as a local service and has only one concrete service and thus is not considered in service selection. Table 2 lists the claimed QoS of each candidate concrete service. The first, second,

1. http://pic.dhe.ibm.com/infocenter/adiehelp/v5r1m1/topic/com.ibm.etools.ctc.bpel.doc/samples/orderprocessing/orderProcessing.html

TABLE 2
Claimed QoS of the Candidate Concrete Services

S	C($)	R(ms)	A(%)	S	C($)	R(ms)	A(%)
CD_1	0.01	500	90	VC_1	0.02	1000	90
CD_2	0.02	400	92	VC_2	0.04	800	92
CD_3	0.03	300	94	VC_3	0.06	600	94
CD_4	0.04	200	96	VC_4	0.08	400	96
CD_5	0.05	100	98	VC_5	0.10	200	98
PO_1	0.01	1000	87	SO_1	0.01	500	87
PO_2	0.02	800	90	SO_2	0.02	400	90
PO_3	0.03	600	93	SO_3	0.03	300	93
PO_4	0.04	400	96	SO_4	0.04	200	96
PO_5	0.05	200	99	SO_5	0.05	100	99
PCC_1	0.02	500	90	PC_1	0.01	1000	87
PCC_2	0.04	400	92	PC_2	0.02	800	90
PCC_3	0.06	300	94	PC_3	0.03	600	93
PCC_4	0.08	200	96	PC_4	0.04	400	96
PCC_5	0.10	100	98	PC_5	0.05	200	99

third and fourth columns respectively list the service name, cost, response time and availability. It can be observed that a service with higher cost can guarantee lower response time and higher availability. Besides, the QoS constraints on the cost, response time and availability of the BPEL process in Fig. 5 were set to \$0.25, 3100ms and 93% respectively.

We conducted the experiments with three kinds of approaches using the same experimental setting:

- **Static**: this is a trivial approach with the business process configured and concrete services selected statically at design time (i.e., no optimization).
- **Single**: this approach configures the business process statically at design time, but makes dynamic quality tradeoffs and selects concrete services at runtime (i.e., only service-level optimization).
- **Double**: this is the approach proposed in this paper with both the business process reconfigured and concrete services selected dynamically.

For the BPEL process in Fig. 5, there are 2^3 possible process configurations and at least 5^4 possible service selections. Due to this large possibility, we conducted the experiments with the initial process configuration and service selection only being some typical ones.

Detailedly, the initial process configuration of order processing is determined by the initial preference rank setting for [*Risk be reduced*, *Response time be reduced*, *Cost be reduced*, *Customer satisfaction be enhanced*]. Its two typical settings are [6, 3, 6, 3] and [3, 6, 3, 6], which have reverse preferences over the soft goals. If the preference rank is set to [6, 3, 6, 3], the three goal variations in Fig. 3 will be configured to *Verify credit*, *Order be paid then be shipped* and *Pay in cash*, and the process will be correspondingly configured to [Check Details, Verify Credit, Pack Order, Pay in Cash, Ship Order], referred as $Config_1$. If the preference rank is set to [3, 6, 3, 6], the three goal variations in Fig. 3 will be configured to *Dummy task*, *Order be shipped then be paid* and *Pay by credit card*, and the process will be correspondingly configured to [Check Details, Pack Order, Ship Order, Pay by Credit Card], referred as $Config_2$.

Similarly, the initial service selection of order processing is determined by the initial QoS weight setting for [cost, response time, availability]. Its four typical settings are [0.50, 0.25, 0.25], [0.25, 0.50, 0.25], [0.25, 0.25, 0.50] and [0.33, 0.33, 0.33], which either prefer one of the three QoS criteria or have equal preferences. For example, if the QoS weight is set to [0.50, 0.25, 0.25], the service selection with $Config_1$ will be [CD_1, VC_1, PO_5, PC_5, SO_5], simplified as [1, 1, 5, 5, 5], and the service selection with $Config_2$ will be [CD_3, PO_5, SO_5, PCC_1], simplified as [3, 5, 5, 1].

In this study, we conducted 12 experiments, each of which has a running of 60 minutes with 10 concurrent threads. Among them, 8 experiments were conducted for the **static** approach with the initial process being configured to $Config_1$ and $Config_2$, and the initial services being selected according to the four typical QoS weight settings. For the **single** and **double** approaches, 2 experiments were respectively conducted with the initial process being configured to $Config_1$ and $Config_2$, and the initial services being selected according to one typical QoS weight setting [0.33, 0.33, 0.33] since the QoS weight will be changed at runtime by our dynamic quality tradeoffs technique.

6.2 Effectiveness Evaluation (Q1)

In each experiment, the self-optimization interval was set to one minute, i.e., our self-optimization process was triggered every one minute. We analyzed earned business value and the quality attributes in the quality model in Table 1 as the *numerical* indicators, and recorded the runtime adaptation actions as an *adaptation process* indicator. We measured the numerical indicators every one minute as introduced in Section 4.2 by system log analysis, and collected the adaptation process indicator by adaptation log analysis. Note that in the experiments risk and customer satisfaction of different process configurations were stochastically distributed with different density to simulate real-life risk analysis and customer feedbacks.

6.2.1 Numerical Indicators

Table 3 shows the numerical results of the 12 experiments in detail. The first column lists the involved approach; the second, third and fourth columns respectively list the initial process configuration, weight setting and service selection; the fifth and sixth columns respectively list the average earned business value and number of successfully processed orders in one minute; and the other columns respectively list the average cost, response time, availability, risk and customer satisfaction in one minute.

TABLE 3
Numerical Results of the Three Approaches

App.	Con.$_{ini}$	Weight$_{ini}$	Sel.$_{ini}$	Val.($)	Ord.(#)	Cos.($)	Re.(ms)	Ava.(%)	Ris.(%)	Sat.(%)
Static	$Config_1$	[0.50,0.25,0.25]	[1,1,5,5,5]	15.30	9.38	1.79	1358	95.71	9.00	69.82
		[0.25,0.50,0.25]	[4,3,5,5,5]	14.45	9.42	2.68	1292	97.10	9.52	69.91
		[0.25,0.25,0.50]	[5,2,5,5,5]	15.68	10.12	2.68	1386	97.21	9.91	70.57
		[0.33,0.33,0.33]	[5,2,5,5,5]	15.84	10.22	2.74	1405	97.65	9.92	69.82
	$Config_2$	[0.50,0.25,0.25]	[3,5,5,1]	14.55	11.33	1.88	1084	95.59	30.70	79.36
		[0.25,0.50,0.25]	[5,5,5,5]	15.72	12.33	3.18	508	98.95	29.88	81.14
		[0.25,0.25,0.50]	[5,5,5,5]	15.49	12.72	3.24	489	98.44	30.59	79.71
		[0.33,0.33,0.33]	[5,5,5,3]	15.32	11.90	2.66	687	96.72	30.12	80.52
Single	$Config_1$	[0.33,0.33,0.33]	[5,2,5,5,5]	16.25	10.50	2.79	1363	97.56	9.27	69.82
	$Config_2$	[0.33,0.33,0.33]	[5,5,5,3]	16.10	12.05	2.65	794	96.96	28.72	80.62
Double	$Config_1$	[0.33,0.33,0.33]	[5,2,5,5,5]	16.93	11.73	2.94	952	96.87	20.73	77.33
	$Config_2$	[0.33,0.33,0.33]	[5,5,5,3]	17.20	11.90	2.53	797	96.86	20.38	75.78

For the **static** approach, the 8 experiments have similar results in terms of earned business value. Besides, the experiments with the same process configurations differ in cost, response time and availability because of their different QoS weight settings, but are similar in risk and customer satisfaction because these two quality attributes are only relevant at the process level. And the higher the weight is set to a quality attribute, the better the quality attribute is, i.e., the service selections are better in the preferred quality attributes, but are worse in others. Moreover, the experiments with $Config_1$ have lower risk and cost whereas the experiments with $Config_2$ have higher customer satisfaction and lower response time because $Config_1$ prefers *Risk be reduced* and *Cost be reduced* whereas $Config_2$ prefers *Customer satisfaction be enhanced* and *Response time be reduced*. This shows that these process configurations and service selections are competitive with each other from the business value perspective and have their own preferences over the quality attributes, and thus it is really hard and maybe even impossible for business analysts to decide at design time what process configuration and service selection should be used at runtime.

For the **single** and **double** approaches, we can observe that the **single** approach outperforms the **static** approach in terms of earned business value by 5.76% in average; and our approach outperforms the **static** approach by 11.58% and the **single** approach by 5.50% in average. On the other hand, in terms of the quality attributes, we can observe that our approach is not totally superior to the **static** and **single** approaches as well as not totally inferior to them because our approach focuses on the overall optimization from the business value perspective rather than on the quality attributes. This shows that a composite service cannot always run optimally with statically designed processes (i.e., the **static** approach) or with only dynamic service selection (i.e., the **single** approach), and should also dynamically adapt its process structure and behavior to achieve better overall satisfaction (in terms of earned business value.

In summary, the evaluation of numerical indicators answers **Q1** positively that a composite service can achieve better overall satisfaction in terms of earned business value by combining both process reconfiguration and service selection. Further, it also demonstrates that it is reasonable to combine them for the purpose of self-optimization for composite services.

6.2.2 Adaptation Process Indicator

Fig. 6 illustrates the self-optimization process of the **single** and **double** approaches, with the *X* axis denoting discrete time intervals of one minute and the *Y* axis denoting the earned business value in each time interval. Here we only show the comparison result on the initial process configuration $Config_1$ because we got similar result on $Config_2$. On the curve, the adaptation actions as introduced in Section 4.4.1, including service-level optimization (a2), process-level and service-level optimization (a3) and optimization failure (a4), are respectively recorded as dark gray, light gray and black points. For the **single** approach, optimization failure means no new service selection result is found; for the **double** approach, optimization failure means neither new process configuration nor new service selection result is found.

From the curve of the **single** approach (Fig. 6(a)), we can observe that it is often the case that no new service selection result can be found when the earned business value drops (e.g., from time 13 to 15, from time 51 to 55) although an adaptation is really needed to improve the earned business value. Among the 36 adaptation actions in this experiment, 25 of them failed. As a result of such optimization failures, the earned business value was often below the expected level. This further shows that only dynamic service selection is not enough for the optimization of composite services.

From the curve of the **double** approach (Fig. 6(b)), we can observe that optimization failure hardly happens because the business process will be reconfig-

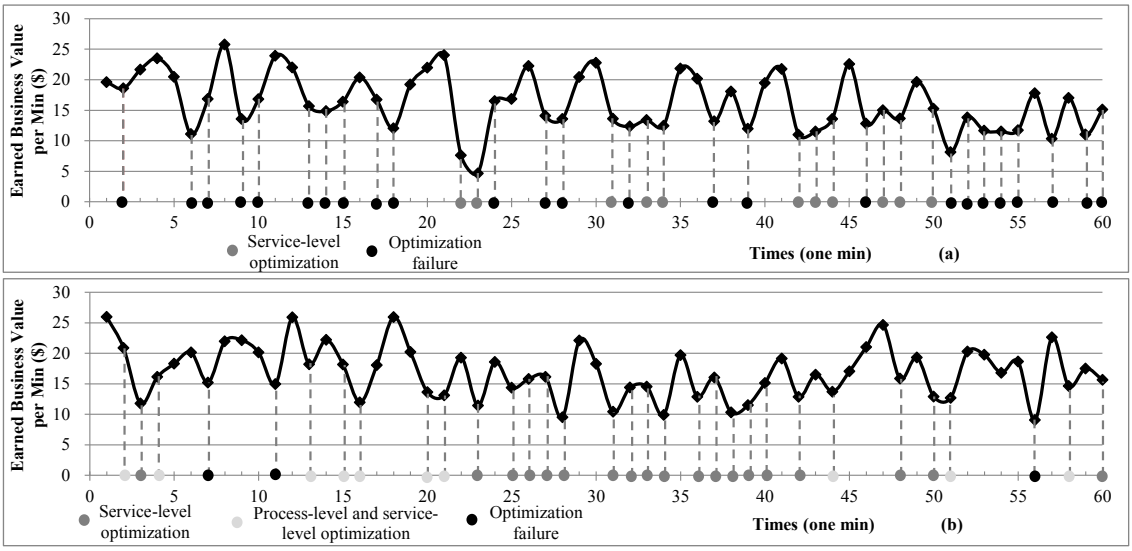


Fig. 6. The Process of Self-Optimization: (a) the Single Approach, (b) the Double Approach.

ured once no service selection result can be found. In this experiment, only 3 of the 32 adaptation actions suffered a failure, and 10 of them involved process-level optimization. This shows that self-optimization by combining process reconfiguration and service selection gives the adaptation process more flexibility.

In summary, the evaluation of this adaptation process indicator answers **Q1** positively that a composite service can achieve better overall satisfaction in terms of more adaptation flexibility by combining process reconfiguration and service selection.

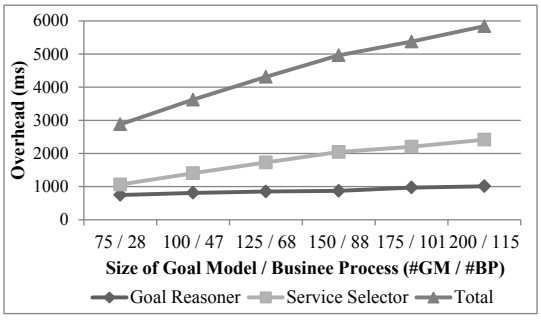


Fig. 7. Performance of our approach.

6.3 Performance Evaluation (Q2)

The service selector and the goal reasoner are the two most time-consuming parts of our approach. They use MIP and SAT, both of which are generally NP-hard problems, for searching and reasoning respectively. Their performance is mainly determined by the sizes of business processes and goal models. To evaluate the performance of our approach, we conducted a set of experiments with randomly generated goal models whose sizes varied from 75 to 200 goals, corresponding business processes whose sizes varied from 28 to 115 abstract services, and the goal/process variations whose numbers varied from 9 to 24. In addition, the number of soft goals was fixed to 20 soft goals; every abstract service had 100 candidate concrete services; and the number of QoS constraints was fixed to 3.

As shown in Fig. 7, the goal reasoner takes around 1 second to produce the configuration result for the size of 200 goals, and increases gently; the service selector takes around 2.5 seconds to produce the selection result for the size of 115 abstract services, and increases faster than goal reasoner. Since our approach performs at most two service selections and one goal reasoning in one self-optimization loop, the curve of "total" shows this maximum overhead. And the total overhead is less than 6 seconds for the size of 200 goals and 115 abstract services, which is still feasible in our approach.

This answers **Q2** positively that our approach has an acceptable performance overhead, which means that approach can scale well with the growth of goal models and business processes and can be effectively applied to real-life composite services.

7 Discussion

We make several assumptions to better illustrate the main focus of this paper. First, we assume that the concrete services for an abstract service share identical interfaces and thus are interchangeable. If they do not, however, they can become interchangeable by using service substitution techniques [30]. Second, we focus on sequential processes because processes with more complex structures such as parallel and loop can be reduced to sequential ones with loop unfolding [31] and loop peeling [32] techniques. Third, all possible process configurations are currently in-built to one business process with the standard BPEL element **if-elseif**. With the increasing number of variations, the business process will be too complex to be understood and maintained. Thus, more flexible techniques such as VxBPEL [18] and AO4BPEL [19] should be explored

to support the graceful definition and adaptation of process variability in BPEL processes. These assumptions mostly affect the specific mechanisms for service selection and adaptation execution, which are not the main focus of this paper.

In this paper we focus on self-optimization rather than personalization since the adaptation is conducted for all the composite service instances with the purpose of maximizing the overall requirements satisfaction. The preference ranks of quality attributes are tuned not for the satisfaction of individual composite service requesters, but for the overall requirements satisfaction from the perspective of composite service providers. However, being combined with the preference profiles of individual service requesters, our approach can be adapted to support personalization by reconfiguring the business process and selecting optimal concrete services for individual service requesters based on their own quality preference profiles.

Instead of using utility function based on quality attributes, we exploit earned business value to measure the overall satisfaction of stakeholder requirements. The definition of the value formula reflects the current business strategies of the composite service provider. For example, for a company providing order processing service, it may focus on the direct profits obtained from successfully processed orders during some time, but may emphasize more on the customer experience and feedback during other time. Such changes of the value formula will further guide the change of self-optimization strategies and provide a way to map business strategies to IT infrastructures.

Further, when applying our approach, if an appropriate value formula can be defined is the main threat to validity. On the one hand, to capture and reflect the changes of business strategies, online analytical processing (OLAP) [33] and business intelligence (BI) [34] can be combined with our approach. On the basis of the large amounts of business data about a composite service, OLAP and BI can provide predictive analysis on business strategies and then guide the definition or adjustment of the value formula. On the other hand, to capture and reflect some long-term influence factors such as reputation and customer satisfaction, specific analysis like market investigation and customer losing trend analysis should be conducted by business experts to help define the value formula. However, if the definition of value formula is incorrect (e.g., violating the stakeholders' original intentions) or less sensitive to the changing environments (e.g., failing to properly capture long-term influence factors), our approach may also take undesired or delayed adaptation actions since its indicator is inappropriate; and if the relevant business data is unavailable or the specific analysis is impractical, our approach will be not applicable.

Last but not the least, we evaluate the effectiveness of our approach on only one composite service. Surely, experiments on more real-life composite services are needed to further evaluate the feasibility of our approach, which is one of our future work. We cannot guarantee the same improvement of about 12% in earned business value with other composite services, which actually may be better or worse depending on application-specific factors such as the number of process variation points, the value measurement and the environments. However, we believe that in general our approach can produce certain improvement since it can exploit a larger adaptation space by considering dynamic quality tradeoffs and process variability.

8 Conclusions

In this paper we have proposed a requirements-driven self-optimization approach for composite services. Firstly, it combines high-level process reconfiguration and low-level service selection to provide more adaptation flexibility. Secondly, it supports dynamic quality tradeoffs to reflect the changing environments by employing a feedback controller to tune the preference ranks of relevant quality attributes. Thirdly, it uses earned business value as the objective and trigger for self-optimization.

An experimental study on an order processing service indicated that our approach outperformed both fixed-weighted and floating-weighted service selection approaches in terms of earned business value and adaptation flexibility, and demonstrated the acceptable performance overhead of a combined adaptation of process reconfiguration and service selection.

In future, we intend to improve this work by integrating more advanced process variability mechanisms, e.g., VxBPEL [18] in order to achieve even more flexible process variability definition and adaptation. We also plan to apply our approach to more real-life composite services to further validate its feasibility.

Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant No. 61361120097, National High Technology Development 863 Program of China under Grant No. 2013AA01A605, and ERC Advanced Grant 291652 - ASAP.

References

[1] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, "QoS-driven runtime adaptation of service oriented architectures," in *Proc. 7th Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng.*, 2009, pp. 131–140.

[2] OASIS. (2007, April) Web services business process execution language, v-2.0. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

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

[4] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, 2004.

[5] Y. Liu, A.H. Ngu, and L. Zeng, "QoS computation and policing in dynamic Web service selection," in *Proc. 13th Int'l Conf. World Wide Web Alternate Track Papers and Posters*, 2004, pp. 66–73.

[6] D.A. Menascé and V.K. Dubey, "Utility-based QoS brokering in service oriented architectures," in *Proc. IEEE Int'l Conf. Web Service*, 2007, pp. 422–430.

[7] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q.Z. Sheng, "Quality driven Web services composition," in *Proc. 12th Int'l Conf. World Wide Web*, 2003, pp. 411–421.

[8] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 369–384, 2007.

[9] T. Yu, Y. Zhang, and K.J. Lin, "Efficient algorithms for web services selection with end-to-end qos constraints," *ACM Trans. Web*, vol. 1, no. 1, 2007.

[10] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proc. 18th Int'l Conf. World Wide Web*, 2009, pp. 881–890.

[11] S.X. Sun and J. Zhao, "A decomposition-based approach for service composition with global QoS guarantees," *Inf. Sci.*, vol. 199, pp. 138–153, 2012.

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

[13] A. Lapouchnian, Y. Yu, and J. Mylopoulos, "Requirements-driven design and configuration management of business processes," in *Proc. 5th Int'l Conf. Business Process Management*, 2007, pp. 246–261.

[14] B. Boehm, "Value-based software engineering: Reinventing "earned value" monitoring and control," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 2, pp. 4–, 2003.

[15] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and using nonfunctional requirements: A process-oriented approach," *IEEE Trans. Softw. Eng.*, vol. 18, no. 6, pp. 483–497, 1992.

[16] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for RE for self-adaptive systems," in *Proc. 18th IEEE Int'l Requirements Eng. Conf.*, 2010, pp. 95–103.

[17] N.Y. Topaloglu and R. Capilla, "Modeling the variability of Web services from a pattern point of view," in *Proc. European Conf. Web Services*, 2004, pp. 128–138.

[18] M. Koning, C. Sun, M. Sinnema, and P. Avgeriou, "VxBPEL: Supporting variability for Web services in BPEL," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 258–269, 2009.

[19] A. Charfi and M. Mezini, "AO4BPEL: An aspect-oriented extension to BPEL," *World Wide Web*, vol. 10, no. 3, pp. 309–344, 2007.

[20] Y. Wang and J. Mylopoulos, "Self-repair through reconfiguration: A requirements engineering approach," in *Proc. 24th IEEE/ACM Int'l Conf. Automated Software Eng.*, 2009, pp. 257–268.

[21] M. Salehie, L. Pasquale, I. Omoronyia, R. Ali, and B. Nuseibeh, "Requirements-driven adaptive security: Protecting variable assets at runtime," in *Proc. 20th IEEE Int'l Requirements Eng. Conf.*, 2012, pp. 111–120.

[22] B. Chen, X. Peng, Y. Yu, and W. Zhao, "Are your sites down? requirements-driven self-tuning for the survivability of Web systems," in *Proc. 19th IEEE Int'l Requirements Eng. Conf.*, 2011, pp. 219–228.

[23] J. Mylopoulos, L. Chung, S. Liao, H. Wang, and E. Yu, "Exploring alternatives during requirements analysis," *IEEE Softw.*, vol. 18, no. 1, pp. 92–96, 2001.

[24] A. Hallerbach, T. Bauer, and M. Reichert, "Capturing variability in business process models: The Provop approach," *J. Softw. Maint. Evol.*, vol. 22, no. 6-7, pp. 519–546, 2010.

[25] J.O. Kephart and D.M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[26] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Softw.*, vol. 14, no. 5, pp. 67–74, 1997.

[27] H.P. In, D. Olson, and T. Rodgers, "Multi-criteria preference analysis for systematic requirements negotiation," in *Proc. 26th Int'l Computer Software and Applications Conf. Prolonging Software Life: Development and Redevelopment*, 2002, pp. 887–892.

[28] P. Avesani, C. Bazzanella, A. Perini, and A. Susi, "Facing scalability issues in requirements prioritization with machine learning techniques," in *Proc. 13th IEEE Int'l Requirements Eng. Conf.*, 2005, pp. 297–306.

[29] G.F. Franklin, J.D. Powell, and A.E. Naeini, *Feedback Control of Dynamic Systems*, 5th ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2006.

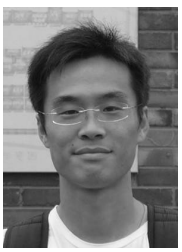
[30] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani, "PAWS: A framework for executing adaptive Web-service processes," *IEEE Softw.*, vol. 24, no. 6, pp. 39–46, 2007.

[31] J. Cardoso, A.P. Sheth, J.A. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and Web service processes," *J. Web Sem.*, vol. 1, no. 3, pp. 281–308, 2004.

[32] D.F. Bacon, S.L. Graham, and O.J. Sharp, "Compiler transformations for high-performance computing," *ACM Comput. Surv.*, vol. 26, no. 4, pp. 345–420, 1994.

[33] A. Berson and S.J. Smith, *Data Warehousing, Data Mining, and Olap*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[34] B. Liautaud, *E-Business Intelligence: Turning Information into Knowledge into Profit*. New York, NY, USA: McGraw-Hill, Inc., 2000.



Bihuan Chen is a PhD student of the School of Computer Science at Fudan University, China. He received the BSc from Fudan University in 2009. His PhD work mainly concerns on requirements engineering, self-adaptive systems and service computing.

Xin Peng is an associate professor of the School of Computer Science at Fudan University, China. He received his PhD degree in computer science from Fudan University in 2006. His current research interests include requirements engineering, self-adaptive system, software product line, software maintenance and reverse engineering.

Yijun Yu is a Senior Lecturer in Computing at the Open University, UK since 2006. Earlier he was a post-doctorate Research Associate at the Department of Computer Science, University of Toronto in Canada (2003-2006) and post-doctorate researcher at ELIS, Ghent University in Belgium (1999-2002). He received PhD degree at CS Department at Fudan University in China (1998). Dr Yu's research interest is in developing automated and efficient software techniques and tools to better support human activities in software engineering: from analysis and design, to implementation, monitoring and maintenance. He is an Associate Editor of the Software Quality Journal, and serves on several program committees of prestigious international conferences such as Requirements Engineering (RE'10-13), Software Maintenance (CSMR'12-13, ICSM'10), Information Systems (CAiSE'14), Engineering Secure Software and Systems (ESSoS'11), Engineering Adaptive Software Systems (EASy'12, EASSy'13).

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