Statistic-Based Dynamic Complexity Measurement for Web Service System

Chengying Mao
School of Software and Communication Engineering
Jiangxi University of Finance and Economics, 330013 Nanchang, China
E-mail: maochy@yeah.net

Keywords: web services, QoS, variability, execution trace, entropy

Received: January 25, 2016

The existing research mainly concerns on the static complexity measurement for service-based system, but the dynamic features like execution behavior have been ignored. In this paper, we proposed a hierarchical measurement framework for evaluating the complexity of Web services from the dynamic aspect. At the level of single service, fluctuation rate is used to represent the QoS (Quality of Service) change during service invocation. Then, a cumulative distribution function is used to measure the dynamic complexity of service performance. At the system level, execution vectors and the corresponding probabilities can be counted according to the trace set of system dynamic executions. Subsequently, the complexity of dynamic execution behaviors can be calculated by the usage of entropy value. In addition, the rationality of above metrics has been validated by the studies on two real applications.

Povzetek: Predlagan je hierarhični okvir za evalvacijo kompleksnosti dinamičnih spletnih storitev.

1 Introduction

In recent years, Web services have been widely utilized for building distributed system over Internet. Different from the traditional software paradigm, the rapid and on-demand service composition in SOA (Service-Oriented Architecture) cause a series of problems on the analysis, design and maintenance of service system. Since some quantitative information like the complexity metric is beneficial to the software engineering activities, such as project cost estimation, system testing, and fault repairing, it is necessary to well understand the structure and behavior of service-based software system. However, These service systems usually run in the dynamic, heterogeneous and changeable environment. As a result, how to measure the complexities of them is a brand-new and challenge problem.

Almost all existing methods are concerned with system's static complexity. They used the models like BPEL (business process execution language) [1, 2], Petri-Net [3] or their variants to describe system logic. Subsequently, the complexities of basic structure units in the business process of service system are defined as atomic metrics. Based on them, the structure complexity of whole service system is calculated according to the aggregation mode among the service units in composition logic. Although the execution probability of each branch is considered, the probability obtained by counting historical data is still a relatively static value. Therefore, the dynamic features of service system, such as the replacement of service at run-time and the business logic changes in service system evolution, are hard to be depicted by the existing complexity measurement framework. For this reason, we will mainly focus on the complexity of service system from the perspective of dynamic behaviors of service or system running.

In this paper, we proposed a framework to measure the dynamic complexity at both single service level and service composition level. For a single service, its dynamic feature is principally reflected by the quality variation along with execution time. While considering the whole service system, its dynamic behavior mainly lies in invocation sequence of services at each time of system execution. Based on the above consideration, our framework investigates the dynamic complexity of Web services in following two aspects: For a single Web service, QoS (quality of service [4]) records are collected in the run-time environments firstly, and then the uncertainty of QoS performance is measured by statistical analysis. At the service composition level, the execution traces of Web service system are the important information for measuring the dynamic complexity. First, the traces of system dynamic execution are collected by a monitor, which can be implemented by aspect-oriented technology [5, 6]. Then, the records are converted into vectors in accordance with the occurrence of each service. Subsequently, the complexity computation method is provided based on the above execution vectors and Shannon entropy theory. Finally, the feasibility and effectiveness of above two kinds of metrics are validated by two real applications.

The main contributions of this paper can be addressed as follows.

(1) Take the variation of QoS as the dynamic feature of service unit, we propose a statistical metric for depicting the fluctuation of QoS records of a single Web service.

(2) Through collecting the execution traces of service system, the diversification of system execution behaviors

is measured by the entropy of execution vectors.

(3) Studies on two examples and two real applications are performed to validate the effectiveness of our presented measurements.

The structure of the paper is as follows. In the next section, some existing complexity researches on Web services that are closely related with our presented approach are stated. In section 3, the dynamic complexity of Web services is defined and discussed. The method for measuring dynamic complexity of a single service is addressed in section 4. Section 5 gives the measurement of dynamic complexity for service composition. Meanwhile, sections 4 and 5 utilize an example to demonstrate the usage of the proposed methods, respectively. Further, the proposed measurements are validated by two real applications in section 6. Finally, section 7 concludes the paper.

2 Related work

How to understand and measure the complexity of Web services has received much attention in recent years. For a single Web service, some existing object-oriented metrics have been adopted for measuring the complexity of service interface [7]. In the aspect of application, some tools like WSDAudit [8] have been developed for measuring service interface files. These metrics perform the complexity analysis from the perspective of service code, so they can only evaluate the complexity of a service for outside calling. Although they can guide users to design the service invocation code, it can not reflect the dynamic behaviors of Web services.

Besides the consideration of static architecture complexity [9], the studies on the complexity of business process in service system have also been investigated [10]. Especially, Jorge Cardoso [1, 11] has done some very influential work in this direction. Similar to other related studies [12, 13, 14], Cardoso's work mainly concerns on the structural complexity of service system. First, the complexities of basic structure units are defined. Then, system complexity is calculated according to the aggregation mode of these service units. In addition, cohesion/coupling metrics [15, 16] and cognitive weights [17] are also used for measuring the process complexity of service system. However, all above methods are mainly concerned with system complexity in static aspect. That is to say, all above methods have a basic assumption that the composition architecture of Web services is unchanged during the system execution.

Jung *et al*. [18] presented an entropy-based method to measure the uncertainty of process models. Although they have considered the execution probability of each branch, the probability obtained by counting historical data is still a relative static value, so it is difficult to evaluate the dynamic complexity in a specific execution context. In addition, the computation for the complex processes is quite complicated. Recently, Martin Ibl and Jan Čapek use stochastic Petri nets (SPN) to model the business processes [19]. They map all reachable marking of SPN into the continuous-time Markov chain and then calculating its stationary probabilities, the uncertainty of a process model is then measured as the entropy of the Markov chain. Like Jung *et al*.'s work, theirs approach highly depends on the fixed architecture of service system, and only analyze the static profile of possible state. These profiles are not the real behaviors produced at run-time. Therefore, they are still belongs to the static complexity model indeed.

In the past few years, the dynamic complexity measurements for object-oriented design and software have been extensively studied [20, 21]. However, the existing work mainly concerns on the coupling between classes, objects and methods. That is to say, the dynamic feature is reflected by the interactions between objects. Since loose-coupling and distributed are two notable characters of service system, the information coupling between services is not very worth taking into consideration while analyzing the complexity of service system at runtime. In our solution, the diversity (uncertainty) of execution traces is viewed as an indicator of dynamic complexity of service system. Recently, Lavazza *et al*. [22] evaluated quality attributes of Web services through analyzing the specifications in form of XML files. Although their quality attributes include dynamic indicators like coupling, most of them are extended from the traditional static complexity metrics such as LOC or McCabe cyclomatic complexity.

3 Dynamic complexity of web services

The complexity analysis and measurement of software system have been studied over four decades. At the early stage, the related researches mainly focus on the static properties of a software system, that is so-called static complexity. Although the static metrics can quantify the complexity of design or source code of a given application, they are still insufficient in evaluating the dynamic behaviour of the application at runtime [23]. Accordingly, dynamic software metrics gradually become a growing research topic in the filed of software measurement [24], especially for object-oriented software [25].

As stated previously, the static complexity of service system has been investigated from various aspects such as control dependency or data dependency. However, the dynamic complexity for this new emerging software system has not been fully exploited yet. Referring to the definitions of dynamic software measures in [22, 26], here we define the dynamic complexity of a Web service system as the diversity of its execution behaviors in dynamic environment.

Definition 1. The *dynamic complexity of Web services* (a single service or whole service system) can be defined as the variability (or uncertainty) of execution behaviors. That is to say, the less change of execution records means

the lower complexity in dynamic aspect.

In the previous definitions about dynamic complexity, such as Lavazza *et al*.'s work [22, 26], they mainly extend the concepts of static metrics to quantify dynamic features of execution traces or specifications represented by XML. In their theoretical framework, the dynamic software metrics are still reflected by some basic and axiomatic issues such as size, McCabe complexity, coupling, cohesion and so on. However, when we measure the diversity of execution behaviours of Web services, the statistical indicators (e.g., entropy) rather than static metric-like issues are introduced here.

With regard to a single Web service, the dynamic features are mainly reflected by its runtime performance, that is quality of service (QoS). In order to measure the variability of service performance, QoS records should be collected between the same time gap. Then, these records are used for volatility analysis so as to scale the dynamic complexity of service unit. In the paper, the distribution of fluctuation rate and its cumulative function are used to measure the uncertainty of Web service's performance.

For service-based system, it is easy to see that the dynamic complexity involves not only system's static structure but also the run-time scenarios including input data, execution environment and observed results. As a result, in order to precisely measure the dynamic complexity of service system, the execution traces and system behaviors should be recorded firstly. Then, the entropy of execution vector partitions is referred as an indicator of dynamic complexity .

According to the above definition, the dynamic complexity in this paper mainly reflects the performance uncertainty of single service or the diversity of service system execution behaviours. For a single service, service requesters can use the dynamic complexity metric to make scientific decision on service selection. In the application scenario of service selection, it needs to choose the most satisfied one from the functional equivalence set of services. From the perspective of performance stability, services with the lower dynamic complexity should be recommended as the preferred candidates. For the whole service system, the results of dynamic complexity are beneficial for software developers and maintainers to understand the evolution of service system or to perform rational system refactoring.

4 Dynamic complexity for a single service

4.1 Measurement method

For a Web service, its QoS generally includes the following issues [4]: response time, throughout, availability, reliability, etc. For the sake of simplicity, we take response time as an example to illustrate our method here.

Definition 2. For the QoS record set of a given service, its *fluctuation rate* at time slot t_i can be calculated as:

$$r_i = |q_i - \overline{Q_{i,\delta}}|/\overline{Q_{i,\delta}} \tag{1}$$

where q_i is the QoS value of the time slot t_i, $\overline{Q_{i,\delta}}$ is the mean value at the backward δ-step fragment w.r.t. q_i, i.e.,

$$\overline{Q_{i,\delta}} = (q_{i-\delta} + q_{i-\delta+1} + \cdots + q_{i-1})/\delta \tag{2}$$

In particular, $\overline{Q_{i,\delta}} = q_{i-1}$ if $\delta = 1$.

It is not hard to find that, r_i reflects the amplitude of QoS change at the current time slot t_i. In literature [27], Wang *et al*. adopt the difference between q_i and the mean of set $\{q_i\}$ to represent the uncertainty of service performance. However, this mode may fail to express dynamic complexity when the QoS mean values of two services have a big gap. In order to overcome this problem, we adopt the difference between service's current QoS to the mean of the latest consecutive QoSs to describe the fluctuation of service performance.

Here, suppose a sequence of fluctuation rates R=$(r_1, r_2, \cdots, r_n)$ has been calculated by continuous monitoring on a given Web service. In order to measure the dynamic complexity of service performance, the distribution of fluctuation rates should be counted. Given k partitioning points C=$(c_1, c_2, \cdots, c_k)$, the cumulative partitioning probability $p_j (1 \leq j \leq k)$ can be computed according to formula (3). Here, $c_1 < c_2 < \cdots < c_k$.

$$p_j = \frac{|R(c_j)|}{n},\ R(c_j) = \{r_i | r_i \leq c_j, 1 \leq i \leq n\} \tag{3}$$

where $|R(c_j)|$ represents the cardinality of set $R(c_j)$. Obviously, $R(c_j) \subseteq R$, $0 \leq p_j \leq 1$.

Accordingly, the dynamic complexity DC_{qos} of a single service can be expressed as follows.

$$DC_{qos} = 1 - \frac{p_1 + p_2 + \cdots + p_k}{k} \tag{4}$$

The meaning of the above dynamic complexity can be intuitively explained in the way of graphical representation. As shown in Fig.1, X-coord. represents the units from 1 to 5 (i.e., the ordinal number j in formula (3)), Y-coord. represents the corresponding cumulative partitioning probability p_j. The proportion of the grey area relative to the whole 5×1 rectangular area approximately equals to $(p_1 + p_2 + \cdots + p_k)/k$, so the proportion of the remaining white part is DC_{qos}. It is easy to see that, the more low-amplitude fluctuation rate means the lower dynamic complexity. For the two services in Fig. 1, DC_{qos} of service 1 is obviously lower than that of service 2.

4.2 Example One

To validate the rationality of our proposed metric for a single service's dynamic performance, response times of two Web services in a continuous period have been gathered here. All 31 records for each service are illustrated in Fig. 2.

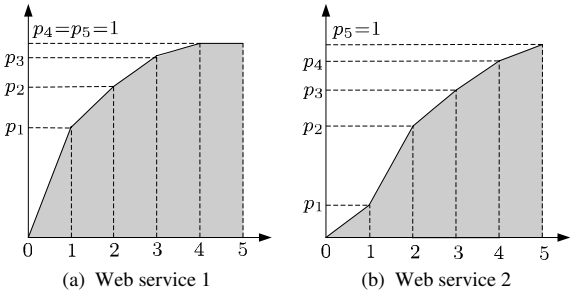


Figure 1: The illustration of dynamic complexity metric.

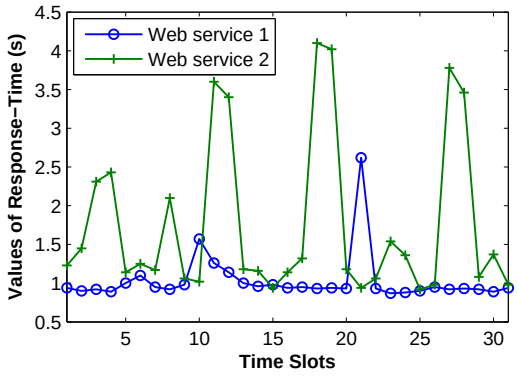


Figure 2: Response times of two sample Web services.

Here, we set δ to 1, so the fluctuation rate is calculated by $|q_i - q_{i-1}|/q_{i-1}$. Based on the above 31 records, 30 rates can be collected for each service, i.e. n = 30. In this example, 9 partitioning points are utilized for dividing the fluctuation rates, i.e. C = (0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5). Then, the partitioning probabilities of two services (i.e., $ws1$ and $ws2$ in Fig. 2) can be calculated according to formula (3). For the first service, $\{p_j\}_{ws1}$ = (0, 0.2, 0.57, 0.73, 0.9, 0.9, 0.97, 1.0, 1.0). For the second one, $\{p_j\}_{ws2}$ = (0, 0.07, 0.1, 0.3, 0.47, 0.7, 0.9, 0.9, 1.0). Therefore, the dynamic complexities of two services are $DC_{qos}(ws1)$ = 0.3037 and $DC_{qos}(ws2)$ = 0.5074, respectively.

From the view of instinctive experience, $ws2$'s change is more frequent and violent than that of $ws1$ in Fig. 2. According to the results in this case, the above dynamic complexity metric can exactly reflects the dynamic variety of two different services' performance.

5 Dynamic complexity for service composition

5.1 Modeling and measurement

When measuring the behaviors of a whole system, execution records (a.k.a. *traces*) are the important source of information for consideration. Based on our previous research [28], the concept of *execution trace* can be defined as below.

Definition 3. The *execution trace* of Web service system can be defined as 2-tuple $et = (\{ws\}^+, fs)$, where $\{ws\}^+ = < ws_i, ws_j, \ldots, ws_k >$ is the ordered sequence of service executions (i, j and k are the service ordinal numbers in whole service set), fs is the final result (or state) of execution sequence $\{ws\}^+$.

Generally speaking, fs is one of the following three states: success (S), wrong result (W) and failure (F, or exception). Take the service system shown in Fig. 3 as an example, an execution of the system may be $(< ws1, ws2, ws3, ws5, ws2, ws4, ws5, ws7 >, W)$.

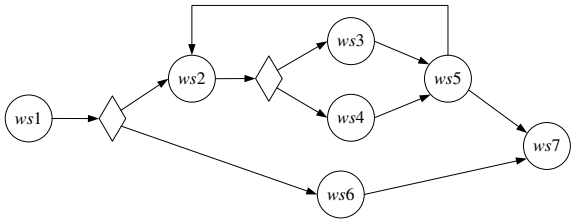


Figure 3: An example Web service system $WS1$.

Definition 4. Given a service system contains m service units, the *execution vector* of the system can be denoted as a vector with length $m + 1$, i.e. $ev = (\{ws\}_m, fs)$. Here, $\{ws\}_m$ is an occurrence list from service 1 to service m.

Obviously, any execution trace et can be converted into an execution vector ev. (1) If ws_j $(1 \leq j \leq m)$ doesn't exist in et, the j-th value in ev (i.e. $ev[j]$) is 0. (2) If ws_j appears only once in et, $ev[j]$=1. (3) If ws_j appears more than one time in et, $ev[j]$=2. Thus, $ev[j] \in \{0, 1, 2\}$. For the last element of ev, $ev[m + 1] \in \{S, W, F\}$.

For the trace related with Fig. 3, the corresponding execution vector can be expressed as $(1, 2, 1, 1, 2, 0, 1, W)$. Here, the j-th $(1 \leq j \leq m)$ number represents the occurrence frequency of ws_j in the given execution trace. According to the above definition, the frequency number could only be the following three choices: 0, 1 or 2.

Definition 5. The *execution partition* of a Web service system is a map structure $ep = (ev, prob)$, where ev is an execution vector of that system, and $prob$ is the probability of occurrence of ev in the set of system execution traces.

In Fig. 3, the diamond frame represents the branch structure. For the sample service system (denoted as $WS1$), the execution partitions in a possible scenario can be gathered and listed in Table 1. In this execution scenario, six execution partitions can run successfully. For the remaining two, service system will produce the wrong result in one case, and system will throw an exception in the other case. For each execution partition, the corresponding probability can be counted through collecting the traces during the dynamic execution of system.

For a Web service system, once the execution traces have been collected and the execution partition have been counted, the complexity indicating system's dynamic execution behaviors can be measured from the perspective of information theory.

Table 1: One possible execution partition set for the service system $WS1$.

No.	$ws1$	$ws2$	$ws3$	$ws4$	$ws5$	$ws6$	$ws7$	fs	$prob$
ep_1	1	1	1	0	1	0	1	S	0.1
ep_2	1	2	2	0	2	0	1	S	0.15
ep_3	1	1	0	1	1	0	1	S	0.15
ep_4	1	2	0	2	2	0	1	F	0.1
ep_5	1	2	1	1	2	0	1	W	0.1
ep_6	1	2	2	1	2	0	1	S	0.1
ep_7	1	2	1	2	2	0	1	S	0.15
ep_8	1	0	0	0	0	0	1	S	0.15

Suppose the execution partition set of a service system is $EP = \{ep_i\}$, where $ep_i = (ev_i, prob_i)$, then the dynamic complexity of system execution can be calculated as below.

$$DC_{exe} = -\sum_{i=1}^{|EP|} prob_i \cdot \log_2 prob_i \tag{5}$$

where $|EP|$ is the cardinality of set EP. Obviously, $0 \leq DC_{exe} \leq 1$, the larger value of DC_{exe} means more complex execution behaviors of a given Web service system.

For the execution traces (ET) of service system $WS1$, its DC_{exe} metric can be computed according to the execution partitions shown in Table 1, that is, $DC_{exe}(ET_{WS1}) = -4 \times 0.1 \times \log_2 0.1 - 4 \times 0.15 \times \log_2 0.15 = 2.9710$.

5.2 Example Two

In order to validate our entropy-based metric for system dynamic execution behaviors, the following two cases are studied here.

(1) *Study on different execution traces*. Here, suppose service system $WS1$ runs in another context, so a new execution trace set ET' of system $WS1$ can be collected. In a similar way, the execution partitions of this kind of execution can also be counted in Table 2. In the current scenario, system behaviors do not exhibit as much diversity as to the execution illustrated in Table 1. Intuitively, the dynamic complexity in the latter execution scenario (i.e., ET') must be lower than that in the former scenario (i.e., ET).

For the second execution trace set ET', its dynamic complexity can also be calculated according to the occurrence probabilities and formula (5).

$DC_{exe}(ET'_{WS1}) = -3 \times 0.1 \times \log_2 0.1 - 2 \times 0.2 \times \log_2 0.2 - 0.3 \times \log_2 0.3 = 2.4464$.

According to the above results, relation $DC_{exe}(ET'_{WS1}) < DC_{exe}(ET_{WS1})$ is workable. It is not hard to find that, the metric computed by our method is in very good agreement with the human cognitive.

(2) *Study on different service systems*. In order to demonstrate the distinguishability of our measurement method for different service systems, we performed the analysis on the other system shown in Fig. 4. In this figure, symbol '||' represents the parallel execution relation, and the diamond symbol is still for the choice relation. Since the execution relation after service $ws2$ is a parallel structure, the execution behaviors of system $WS2$ are much simpler than those of $WS1$ according to the common sense.

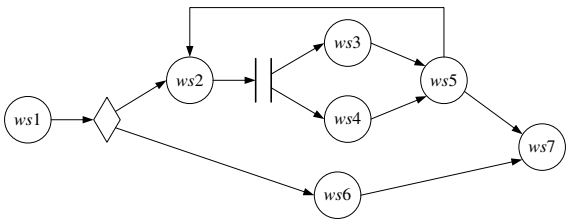


Figure 4: The other example Web service system $WS2$.

Here, suppose the possible traces and the corresponding partitions are shown in Table 3. Since the relation between $ws3$ and $ws4$ is parallel, $ws2$, $ws3$, $ws4$ and $ws5$ must appear at the same time. Meanwhile, there is a loop from $ws2$ to $ws5$, so the occurrence frequencies of services between them should be 1 or 2 in a consistent manner.

For the above traces of system $WS2$, i.e. ET_{WS2}, its dynamic complexity can also be calculated here. Through comparing the metrics of $WS1$ and $WS2$, it is clear that our proposed metrics can precisely reflect the actual situations.

$DC_{exe}(ET_{WS2}) = -2 \times 0.3 \times \log_2 0.3 - 0.4 \times \log_2 0.4 = 1.5710$.

6 Empirical study on real applications

Although the above two examples have demonstrated the feasibility of our proposed methods for measuring the dynamic complexity of Web services, they are not from the scenarios of real applications. Thus, it is necessary to validate our methods on the cases from the real and public benchmarks.

6.1 Study on service's performance

In this section, the following research question should be identified so as to confirm the effectiveness of our proposed

Table 2: The other possible execution partition set for system $WS1$.

No.	$ws1$	$ws2$	$ws3$	$ws4$	$ws5$	$ws6$	$ws7$	fs	$prob$
ep_1	1	1	1	0	1	0	1	S	0.1
ep_2	1	2	2	0	2	0	1	S	0.2
ep_3	1	1	0	1	1	0	1	S	0.1
ep_4	1	2	0	2	2	0	1	F	0.1
ep_5	1	2	1	1	2	0	1	W	0.2
ep_6	1	0	0	0	0	0	1	S	0.3

Table 3: The possible execution partitions for system $WS2$.

No.	$ws1$	$ws2$	$ws3$	$ws4$	$ws5$	$ws6$	$ws7$	fs	$prob$
ep_1	1	1	1	1	1	0	1	W	0.3
ep_2	1	2	2	2	2	0	1	S	0.4
ep_3	1	0	0	0	0	0	1	S	0.3

measurements.

RQ1: Can the dynamic complexity measurement for single Web service precisely depict the fluctuation of QoS performance?

To answer the above question, the public QoS records of Web services are introduced in our experimental analysis. WS-DREAM[1] is a set of QoS datasets which are collected from real-world Web services by Zheng *et al*. [29]. Here, the third subset is used in this study. It contains the real-world QoS evaluation results from 142 users on 4532 Web services on 64 different time slots. For the limit of space, we did not investigate the time-aware QoS records of all users for all services. As similar in [30], the records of user #9 for services #741 and #745 are used as benchmarks in the experiments. The QoS records at 64 different time slots of these two services are illustrated in Fig. 5, where sub-figure 5(a) is for service #741 and sub-figure 5(b) is for service #745. Through intuitively comparing the stability of QoSs (i.e. response times here) of two services, it is easy to see that service #745 is obviously complex than service #741 from perspective of performance.

According to Equation (1), the fluctuation rate vectors of the above two services can be calculated. Since the length of time slots is 64, the length of each fluctuation rate vector should be 63. In this case study, we also set the step-length (δ) in Equation (1) as 1, that is to say, the fluctuation rates of the considered two services are computed by means of $|q_i - q_{i-1}|/q_{i-1}$ $(1 \leq i \leq 63)$. Based on the above calculation, the sequence of fluctuation of services #741 and #745 can be expressed as (0.0035, 0.0035, 0.0534, 0.1486, 0.5119, $\cdots$) and (6.0030, 0.8602, 5.4185, 0.0273, 0.3612, $\cdots$), respectively.

Since the fluctuation rate of service #745 will exceed five, the number of partitioning points which are used to divide the fluctuation rates is assigned with 10. Accordingly, the set of partitioning points is designed as C=(0.01,0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10). Based on this partitioning, the cumulative partitioning probabilities of these two services can be calculated respectively, that is, $\{p_j\}_{\#741}$=(0.0476, 0.0794, 0.1905, 0.3810, 0.5238, 0.8571, 0.9683, 0.9683, 1.0, 1.0) and $\{p_j\}_{\#745}$=(0.0794, 0.0952, 0.2063, 0.4127, 0.5079, 0.5873, 0.7937, 0.9048, 0.9524, 1.0). Further, the dynamic complexities of them can be achieved by applying Equation (4), i.e., $DC_{qos}(\#741)$=0.3984 and $DC_{qos}(\#745)$=0.4460. According to the above measurement results, service #741 is more complex than service #745 with respect to QoS performance. The relation between these two services is consistent with the intuitive judgement.

Through the validation on real QoS dataset named WS-DREAM, we can conform that the proposed metric for dynamic complexity of a single service is feasible and rational.

6.2 Study on execution behaviours of service system

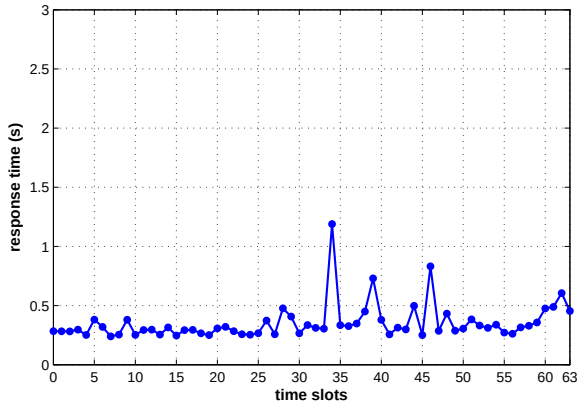
Similarly, we also should investigate the research question on the effect of dynamic complexity measurement for Web service system.

RQ2: Can the dynamic complexity based on execution vector and Shannon entropy scientifically measure the diversification of system execution behaviors?

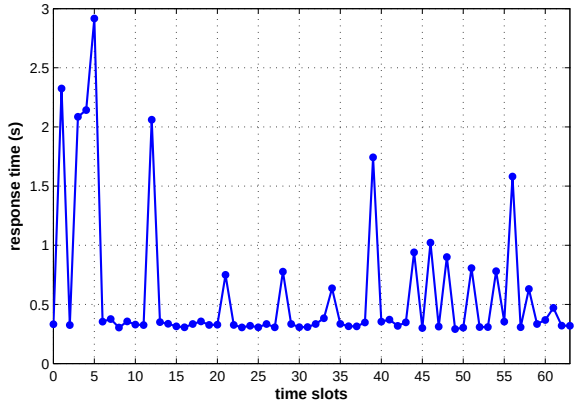
Here, a subject application **`LoanDemo`** from the samples of Oracle BPEL Process Manager [31, 32] is adopted for experimental analysis. The business logic of this service system is described by BPEL, and the main process is depicted by the file **`LoanFlow.bpel`** as shown in the code listing 1 of reference [32]. In the main process, `UnitedLoanService` and `StarLoanService` are two composite services whose execution can be further decomposed by a BPEL file. Here, for the sake of simplification, we only further describe the workflow process of service `StarLoanService` by BPEL code (refers to the code listing 2 in [32]).

The process of whole service system is depicted in Fig.

[1] http://www.wsdream.com/dataset.html



(a) user #9 for services #741



(b) user #9 for services #745

Figure 5: The time period QoS feature of response time.

6(a), in which the oval node represents basic service unit, the black strip stands for the parallel logical node and the diamond box is for the branch logic. It should be noted that, the workflow process includes both external services and basic execution activities, here treat both of them as the basic service units. Compared with the original version (see Fig. 6(a)) of system logic, the updated system (refer to Fig. 6(b)) considers the activity about exception handling and the different treatment for VIP customers. Therefore, two new services (i.e., S13 and S14) are added into the workflow process of the updated system.

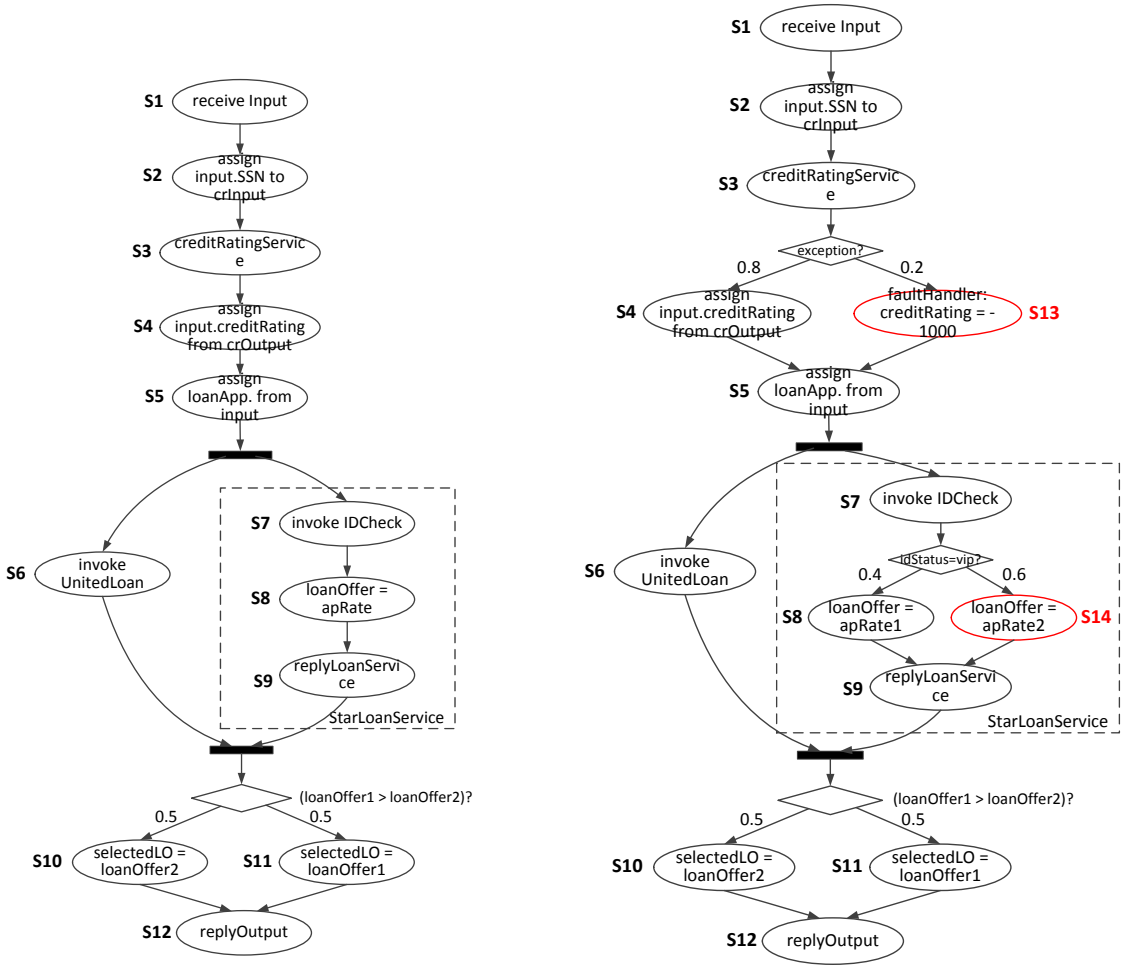
In order to compare the dynamic complexities between the original and updated service systems, we firstly collect (or simulate) the execution traces of each service system separately, and then apply the metric defined in Section 5.1 to measure the complexity of each trace set. Here, we set the number (N) of execution traces as 20, and collect the same size of traces for both service systems. In this case study, we assume each service in `LoanDemo` system can execute successfully, that is, the final results of all execution traces are S (success).

According to the definition 5, the above collected traces of each service system can be formed as execution partitions. The corresponding partitions of original and updated service systems are shown in Tables 4 and 5, respectively. When the size of trace set (N) is equal to 20, the partitioning number ($|EP|$) is 2 for the original system, and is 7 for the updated system. Based on the results in Tables 4 and 5, we can further calculated the dynamic complexities in the light of Equation (5). For the original service system, its complexity $DC_{exe}(ET_{original}) = -0.55 \times \log_2 0.55 - 0.45 \times \log_2 0.45 = 0.9928$. Similarly, for the updated system, the corresponding complexity $DC_{exe}(ET_{updated}) = 2.5016$. Thus, we can say that the updated service system is more complex than the original one from the perspective of dynamic execution behaviors. It it not hard to find that the conclusion is consistent with the subjective judgement.

The above comparison is only performed on a case of execution trace set. It should be noted that, given a number of trace set size, the collected trace sets may have some minor variance. For this reason, we repeat the above analysis 100 times and obtain the entropy distribution of each system version. As shown in Fig. 7(a), the mean of dynamic complexity for the original system is 0.9721, and the corresponding value of the updated system is 2.4125. Since the complexity of the original system ($DC_{exe}(ET_{original})$) is always less than or equal to 1.0, and the complexity of the updated system ($DC_{exe}(ET_{updated})$) is always higher than 1.5, the relation $DC_{exe}(ET_{original}) < DC_{exe}(ET_{updated})$ is always kept regardless of experiment trials.

While revisiting the updated service system, we find that the number of possible execution paths is eight in Fig. 6(b), however the partitioning number ($|EP|$) in Table 5 is only seven. That is to say, the collected execution traces are not sufficient to cover all possible execution scenarios. Due to this reason, we repeat the experiments for the case of N=50. In this setting of trace set size, the dynamic complexities of these two system versions are $DC_{exe}(ET_{original})$=0.9857 and $DC_{exe}(ET_{updated})$=2.5790. Obviously, the relation $DC_{exe}(ET_{original}) < DC_{exe}(ET_{updated})$ can still be kept here. Through comparing Figs. 6(a) with 6(b), we can find that the distribution of dynamic complexity will be more stable with the increase of trace set size. Take the updated service system as an example, the distribution box of updated system's complexity in Fig. 6(b) is obviously shorter than that in Fig. 6(a). That is to say, the complexity result in case of N=50 is more credible than that in case of N=20. However, the relation between the complexities of two system versions usually will not be affected by the size of execution trace set. On the other hand, the larger trace set means greater effort on trace collection and complexity computation. Therefore, we suggest to adopt a medium scale to set the size number (N) of trace set.

Based on the above observations, we can conclude that the measurement based on execution vector and Shannon entropy is suitable to measure the dynamic complexity of Web service system.



(a) The original process

(b) The updated process

Figure 6: The business process of Web service system LoanDemo.

7 Conclusion

Although some studies on the evaluation of process complexity in service system have been investigated, the metrics for understanding the dynamic complexity of system behaviors are still very lack. In this paper, a two-level measurement framework for assessing the dynamic variability of Web services has been presented. At service level, the uncertainty of service QoS is measured by continuous monitoring and fluctuation rate statistic. At system level, the execution traces are classified into vectors, then the entropy of vector probabilities is calculated as the complexity indicator of system execution behaviors. Finally, the effectiveness of the proposed metrics has been evaluated by two real applications.

Acknowledgments

We are grateful to Changfu Xu for collecting the partial experimental results in this paper. This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61462030, the Natural Science Foundation of Jiangxi Province under Grant No. 20151BAB207018, the Science Foundation of Jiangxi Educational Committee under Grant No. GJJ150465, and the Program for Outstanding Young Academic Talent in Jiangxi University of Finance and Economics.

References

[1] J. Cardoso. Complexity Analysis of BPEL Web Processes, *Software Process: Improvement and Practice*, vol. 12, no. 1, 2007, pp. 35-49.

[2] R. M. Parizi and A. A. A. Ghani. An Ensemble of Complexity Metrics for BPEL Web Processes. *Proc. of the 9th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'08)*, Phuket, Thailand, August 6-8, 2008, pp. 753-758.

[3] C. Mao. Complexity Analysis for Petri Net-based Business Process in Web Service Composition, *Proc.*

Table 4: The execution partitions for the service system shown in Fig. 6(a), N=20.

No.	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	fs	$prob$
ep_1	1	1	1	1	1	1	1	1	1	1	0	1	S	0.55
ep_2	1	1	1	1	1	1	1	1	1	1	1	0	S	0.45

Table 5: The execution partitions for the service system shown in Fig. 6(b), N=20.

No.	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	fs	$prob$
ep_1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	S	0.35
ep_2	1	1	1	1	1	1	1	1	1	0	1	1	0	0	S	0.1
ep_3	1	1	1	1	1	1	1	0	1	1	0	1	0	1	S	0.15
ep_4	1	1	1	1	1	1	1	0	1	0	1	1	0	1	S	0.2
ep_5	1	1	1	0	1	1	1	1	1	1	0	1	1	0	S	0.05
ep_6	1	1	1	0	1	1	1	0	1	1	0	1	1	1	S	0.1
ep_7	1	1	1	0	1	1	1	0	1	0	1	1	1	1	S	0.05

of the 5th IEEE Intl Symp. on Service-Oriented System Engineering (SOSE'10), Nanjing, China, June 4-5, 2010, pp.193-196.

[4] D. A. Menasce. QoS Issues in Web Services, *IEEE Internet Computing*, vol. 6, no. 6, 2002, pp. 72-75.

[5] M. Sun, B. Li, and P. Zhang. Monitoring BPEL-based Web Service Composition using AOP, *Proc. of the 8th IEEE/ACIS International Conference on Computer and Information Science (ACIS-ICIS'09)*, Shanghai, China, June 1-3, 2009, pp. 1172-1177.

[6] C. Mao. AOP-based Testability Improvement for Component-based Software. *Proc. of the 31st Annual International Computer Software and Applications Conference (COMPSAC'07)*, Vol. II, Beijing, China, July 24-27, 2007, pp.547-552.

[7] J. L. O. Coscia, M. Crasso, and *et al*. Estimating Web Service Interface Complexity and Quality through Conventional Object-Oriented Metrics, *Proc. of the XV Iberoamerican Conference on Software Engineering (CIbSE'12)*, Buenos Aires, Argentina, April 24-27, 2012, pp. 154-167.

[8] H. M. Sneed. Measuring Web Service Interfaces, *Proc. of the 12th IEEE International Symposium on Web Systems Evolution (WSE'10)*, Timisoara, September 17-18, 2010, pp. 111-115.

[9] J. Zhao. Complexity Metrics for Software Architecture, *IEICE Trans. Inf. & Syst.*, vol. E87-D, no. 8, 2004, pp. 2151-2156.

[10] L. S. González, F. G. Rubio, and *et al*. Measurement in Business Processes: A Systematic Review, *Business Process Management Journal*, vol. 16, no. 1, 2010, pp. 114-134.

[11] J. Cardoso. Business Process Control-flow Complexity: Metric, Evaluation, and Validation, *Int. J. Web Service Research*, vol. 5, no. 2, 2008, pp. 49-76.

[12] K. B. Lassen and W. M. P. Aalst. Complexity Metrics for Workflow Nets, *Information and Software Technology*, vol. 51, 2009, pp. 610-626.

[13] C. Mao. Control and Data Complexity Metrics for Web Service Compositions, *Proc. of the 10th International Conference on Quality Software (QSIC'10)*, Zhangjiajie, China, July 14-15, 2010, pp. 349-352.

[14] A. Khoshkbarforoushha, P. Jamshidi, and *et al*. Metrics for BPEL Process Context-independency Analysis, *Service-Oriented Computing and Applications*, vol. 5, no. 3, 2011, pp. 139-157.

[15] I. T. P. Vanderfeesten, H. A. Reijers, and W. M. P. Aalst. Evaluating Workflow Process Designs using Cohesion and Coupling Metrics, *Computers in Industry*, vol. 59, no. 5, 2008, pp. 420-437.

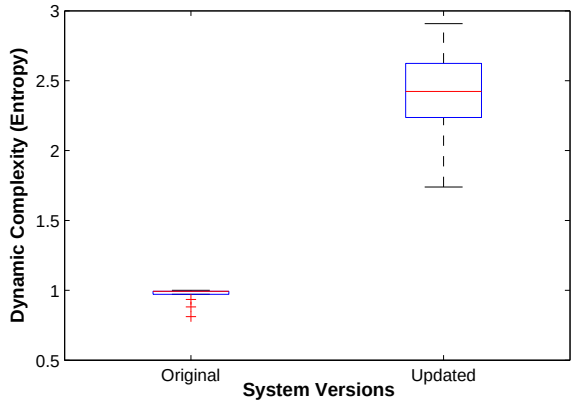
[16] A. Kazemi, A. N. Azizkandi, and *et al*. Measuring the Conceptual Coupling of Services using Latent Semantic Indexing, *Proc. of IEEE 8th International Conference on Services Computing (SCC'11)*, Washington DC, USA, July 4-9, 2011, pp. 504-511.

[17] R. Laue and V. Gruhn. Complexity Metrics for Business Process Models, *Pro. of the 9th International Conference on Business Information Systems (BIS'06)*, Klagenfurt, Austria, May 31 - June 2, 2006, pp. 1-12.

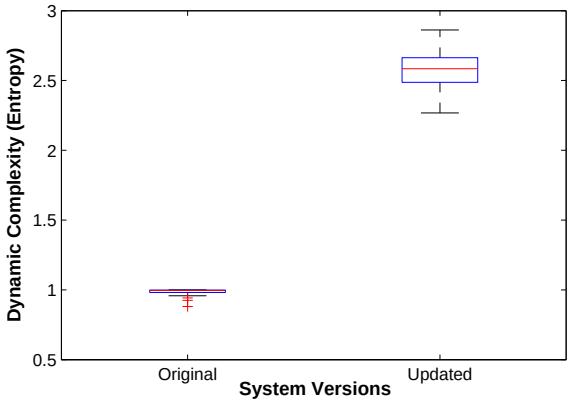
[18] J.-Y. Jung, C.-H. Chin, and J. Cardoso. An Entropy-based Uncertainty Measure of Process Models, *Information Processing Letters*, vol. 111, 2011, pp. 135-141.

[19] M. Ibl and J. Čapek. Measure of Uncertainty in Process Models using Stochastic Petri Nets and Shannon Entropy, *Entropy*, 2016, vol. 18, no. 33, pp. 1-14.

[20] S. Yacoub, H. Ammar, T. Robinson. Dynamic Metrics for Object-Oriented Designs. *Proc. of the 5th Inter-*



(a) The entropy distribution when N=20



(b) The entropy distribution when N=50

Figure 7: The entropy distributions of two service systems (repeated trials = 100).

national Software Metrics Symposium, Boca Raton, USA, November 4-6, 1999, pp.50-61.

[21] E. Arisholm, L. C. Briand, A. Foyen. Dynamic Coupling Measures for Object-Oriented Software. *IEEE Transactions on Software Engineering*, 2004, vol. 30, no. 8, pp. 491-506.

[22] L. Lavazza, S. Morasca, and D. Tosi. Enriching Specifications to Represent Quality in Web Services in a Comprehensive Way, *Proc. of the 9th IEEE International Symposium on Service-Oriented System Engineering (SOSE'15)*, Redwood City, CA, USA, March 30-April 3, 2015, pp. 203-208.

[23] J. K. Chhabra and V. Gupta. A Survey of Dynamic Software Metrics, *Journal of Computer Science and Technology*, 2010, vol. 25, no. 5, pp. 1016-1029.

[24] A. Tahir and S. G. MacDonell. A Systematic Mapping Study on Dynamic Metrics and Software Quality, *Proc. of the 28th IEEE International Conference on Software Maintenance (ICSM'12)*, Riva del Garda, Trento, Italy, September 23-30, 2012, pp. 326-335.

[25] A. Gosain and G. Sharma. Dynamic Software Metrics for Object Oriented Software: A Review, *Advances in Intelligent Systems and Computing*, vol. 340, 2015, pp. 579-589.

[26] L. Lavazza, S. Morasca, and *et al*. On the Definition of Dynamic Software Measures, *Proc. of the 6th International ACM Symposium on Empirical Software Engineering and Measurement (ESEM'12)*, Lund, Sweden, September 19C20, 2012, pp. 39-48.

[27] S. Wang, Z. Zheng, and *et al*. Reliable Web Service Selection via QoS Uncertainty Computing, *Int. J. Web and Grid Services*, vol. 7, no. 4, 2011, pp. 410-426.

[28] C. Mao, I. Tervonen, and J. Chen. Diagnosing Web Services System Based on Execution Traces Pattern Analysis, *Proc. of the 8th IEEE International Conference on e-Business Engineering (ICEBE'11)*, Beijing, China, October 19-21, 2011, pp. 117-122.

[29] Y. Zhang, Z. Zheng, and M. R. Lyu. WSPred: A Time-Aware Personalized QoS Prediction Framework for Web Services, *Proc. of the 22th IEEE Symposium on Software Reliability Engineering (ISSRE'11)*, Hiroshima, Japan, November 29 - December 2, 2011, pp. 210-219.

[30] H. Ma, Z. Hu, and *et al*. Toward Trustworthy Cloud Service Selection: A Time-aware Approach using Interval Neutrosophic Set, *Journal of Parallel and Distributed Computing*, vol. 96, 2016, pp. 75-94.

[31] Oracle Corporation. Oracle BPEL Process Manager, available at http://www.oracle.com/ technology/products/ias/bpel/index.html, accessed in June 2016.

[32] C. Mao. Static Inter-BPEL Program Slicing for Web Services, *Int. J. Simulation and Process Modelling*, vol. 7, no. 3, 2012, pp. 204-216.

Appendix

The business processes of two composite services are shown in the following two code listings. The code is written in BPEL (Business Process Execution Language) which is published by OASIS standard organization to specify actions within business processes with Web services. The process of the whole service application (`LoanDemo`) is shown in the Listing 1, and the process of subsystem `StarLoanService` is described by Listing 2. The code except shadow lines represents the original business logic, and the shadow part means the modified code in the updated version of system.

The two process graphs in Fig. 6 are the corresponding logic representations of the original BPEL code and updated code, respectively. It should be pointed out that, besides the external service units, the actions of task execution in BPEL code are also treated as atomic service nodes in Fig. 6. At the same time, the updated code segments are represented by the oval red nodes in process graph.

Listing 1: The BPEL code of LoanDemo application.

```
<process name="LoanFlow" ...>
 <partnerLinks>
   <-- Include client, creditRatingService, UnitedLoanService and StarLoanService -->
   ...
 </partnerLinks>
 <variables>
   <variable name="input" .../>
   <variable name="crInput" .../>
   <variable name="crOutput" .../>
   <variable name="crError" .../>
   <variable name="loanApplication" .../>
   <variable name="loanOffer1" .../>
   <variable name="loanOffer2" .../>
   <variable name="selectedLoanOffer" .../>
 </variables>

 <sequence>
   <receive name="receiveInput" partnerLink="client" portType="tns:LoanFlow" operation="initiate"
       variable="input" createInstance="yes"/>
   <scope name="GetCreditRating">
     <-- Watch for faults (exceptions) being thrown from creditRatingService -->
     <faultHandlers>
       <catch faultName="services:NegativeCredit" faultVariable = "crError">
         <assign> <copy> <from expression="number(-1000)"/>
                  <to variable="input" part="creditRating"/> </copy>
         </assign>
       </catch>
     </faultHandlers>
     <sequence>
       <assign> <copy> <from variable="input" part="SSN"/> <to variable="crInput" part="SSN"/> </
           copy>
       </assign>
       <invoke name="invokeCR" partnerLink="creditRatingService" portType="services:
           CreditRatingService" operation="process" inputVariable="crInput" outputVariable="
           crOutput"/>
       <assign> <copy> <from variable="crOutput" part="rating"/> <to variable="input" part="
           creditRating"/> </copy>
       </assign>
     </sequence>
   </scope>

   <scope name="GetLoanOffer">
   <sequence>
     <assign> <copy> <from variable="input"> <to variable="loanApplication"/> </copy>
     </assign>
     <flow>
       <sequence>
         <invoke name="invokeUnitedLoan" partnerLink="UnitedLoanService" portType="services:
             LoanService" operation="initiate" inputVariable="loanApplication"/>
         <receive name="receive_invokeUnitedLoan" partnerLink="UnitedLoanService" portType="
             services:LoanServiceCallback" operation="onResult" variable="loanOffer1"/>
       </sequence>
       <sequence>
         <invoke name="invokeStarLoan" partnerLink="StarLoanService" portType="services:LoanService
             " operation="initiate" inputVariable="loanApplication"/>
         <receive name="receive_invokeStarLoan" partnerLink="StarLoanService" portType="services:
             LoanServiceCallback" operation="onResult" variable="loanOffer2"/>
       </sequence>
     </flow>
   </sequence>
   </scope>

   <scope name="SelectOffer" variableAccessSerializable="no">
   <switch> <-- If loanOffer1 is greater (worse) than loanOffer2 -->
     <case condition="bpws:getVariableData('loanOffer1','APR') > bpws:getVariableData('loanOffer2
         ','APR')">
       <assign> <copy> <from variable="loanOffer2" part="APR/> <to variable="selectedLoanOffer"
           part="APR"/> </copy>
       </assign>
```

```
      </case>
      <otherwise>
        <assign> <copy> <from variable="loanOffer1" part="APR"/> <to variable="selectedLoanOffer"
            part="APR"/> </copy>
        </assign>
      </otherwise>
    </switch>
    </scope>

    <invoke name="replyOutput" partnerLink="client" portType="tns:LoanFlowCallback" operation="
        onResult" inputVariable="selectedLoanOffer"/>
  </sequence>
</process>
```

Listing 2: The BPEL code of service `StarLoanService`.

```
<process name="StarLoanService" ...
  <partnerLinks>
      <partnerLink name="LoanFlow" .../>
      <partnerLink name="IDCheckService" .../>
  </partnerLinks>
  <variables>
    <variable name="loanApp" .../>
    <varaible name="sNumber" .../>
    <varaible name="crdRate" .../>
    <variable name="apRate" .../>
    <variable name="loanOffer" .../>
  </variables>

  <sequence>
    <receive name="receive_loanApplication" partnerLink="LoanFlow" portType="services:LoanService"
        operation="initiate" variable="loanApp" createInstance="yes"/>
    <assign><copy><from variable="loanApp" part="SSN"/> <to variable="sNumber"/></copy>
            <copy><from variable="loanApp" part="creditRating"/> <to variable="crdRate"/> </copy>
    </assign>
    <invoke name="invokeIDCheck" partnerLink="idCheckService" portType="services:IDCheckService"
        operation="idCheck" inputVariable="sNumber" outputVariable="idStatus"/>
  </sequence>

  <sequence>
    <if> <condition idStatus = "vip" crdRate="high"/>
      <assign> <copy> <from expression="number(0.0015)"/> <to variable="apRate"/></copy>
      </assign>
    <else>
      <assign> <copy> <from expression="number(0.002)"/> <to variable="apRate"/></copy>
      </assign>
    </if>
    <assign><copy><from variable="apRate"/> <to variable="loanOffer" part="APR"/></copy>
    </assign>

    <invoke name="replyLoanService" partnerLink="LoanFlow" portType="services:LoanServiceCallback"
        operation="onResult" inputVariable="loanOffer"/>
  </sequence>
</process>
```