Article Info

Article history:
Received: 15 September 2016
Revised: 10 May 2018
Accepted: 31 July 2018
Published Online: 27 July 2019

Keywords:
Web Service Selection, Quality of Service, Hidden Markov Model, Service-Based Applications, Software Adaptation.

Abstract

Service-based application (SBA) is composed of software services, and these services may be owned by the developing organization or third parties. To provide functionalities based on the user’s preferences, SBA’s constitute services should be selected dynamically at runtime. For each distinct user’s request, we aim at finding a sequence of services which mostly satisfies user’s preferences. Furthermore, we aim at reporting in a systematic manner the list of relevant contributions similar to our work focusing on the adaptation mechanisms. We applied Hidden Markov Model (HMM) to propose a QoS-based service selection method. The method is presented in three steps: Modelling, Learning, and QoS-based Selection. We used real-world QoS dataset to investigate the fitness and the execution time of the method. We compared this work with GSA-based and PSO-based service selection methods. We built and trained an HMM for selecting services for a given sequence of tasks in which the selected services are mostly aligned with user’s preferences. Experimental results showed that our method achieves the maximum fitness in a reasonable time. Since service-oriented environments are ever changing, unsupervised learning approaches like Maximum Likelihood Estimation or Viterbi Training should be used to modify the elements and the probabilities of HMM.

1 Introduction

Service-oriented computing is increasingly adopted as a paradigm for building loosely coupled, distributed and adaptive software applications, called service-based applications (SBA). SBA is composed of software services (i.e. constitute services), and those services may be owned by the developing organization or third parties 1. SBA adaptation is required to overcome the runtime changes in functionalities and quality objectives. Therefore, it is desirable to modify SBA’s constitute services through (semi) automatic adaptation mechanisms.

Adaptation mechanisms are the techniques and facilities provided by SBA that enable adaptation strategies like service re-composition, service re-selection, or service re-negotiation 2. The realization of adaptation mechanisms may be done automatically or may require user involvement; that is, human-in-the-loop adaptation. The adaptation mechanisms are classified into Adaptive, Corrective, Preventive, and Extend according to S-CUBE adaptation taxonomy.
Most of the existing approaches focus on Adaptive mechanisms [4–7] which modify the SBA in response to changes affecting its environment like contextual changes or the needs of a particular user. Corrective mechanisms [8–15] replace a faulty service with a new version that provides the same functionality and quality. Preventive mechanisms [16–18] use prediction techniques to detect the probable failures or SLA violations and also assess the accuracy of prediction. There are few approaches targeting Extending mechanisms [19–22] which aim to extend the SBA by adding new required functionalities.

In this paper we focus on SBA customization based on user’s preferences, which is a subset of Adaptive mechanisms. Suppose itinerary purchase scenario in which the travel agency is the owner of SBA. Travel agency orchestrates the available services to fulfil the customers’ requests. Each customer has specific preferences that are expected to be satisfied by travel agency. A customer may prefer cost-effective flight and normal hotel, while the other customer may prefer high-quality flight and luxury hotel. These quality concerns could be achieved by dynamically selecting web services based on user’s preferences. As shown in Figure 1, dynamic service selection includes the following steps [23]:

1. Converting a user’s request to a machine understandable model
2. Discovering candidate services for each task of a given process
3. Selecting the best set of services among candidate services based on QoS constraints and user’s preferences
4. Executing the solution (made in step 3) with Business Process Execution Language (BPEL) engine and producing the results

Finding the best services through the ordinary methods leads to an NP-hard problem. Therefore several heuristic and dynamic programming approaches were proposed to model service selection as an optimization problem. In this paper we present a dynamic service selection method which has a strong mathematical basis. The proposed method is based on Hidden Markov Model (HMM), i.e., a mathematical model inspired from Markov Chain. The method considers user’s preferences while selecting appropriate services for a given set of tasks. The process of applying HMM in the service selection problem includes following steps: Modelling, Learning, and QoS-based Selection. In modelling step, HMM is applied in the service selection problem. The output HMM from the previous step is initialized in learning step by supervised or unsupervised learning methods. The Viterbi algorithm is used in QoS-based selection step to select the most appropriate services in a reasonable time.

Execution time and fitness are the critical factors in comparing and proposing service selection methods. Execution time shows the length of time taken for selecting services and producing a composition model in common business process languages like WS-BPEL. Fitness shows how much the output model satisfies user’s preferences. Fitness has direct relationship with user satisfaction. In comparison with GSA-based, PSO-based, and GA-based service selection methods, our method achieves the maximum fitness in a reasonable time.

The rest of this paper is organized as follows. The characteristics of this work are compared with related studies in Section 2. Section 3 describes HMM briefly. The proposed service selection method based on HMM is described in Section 4. Section 5 presents the experimental results. Finally the paper is concluded in Section 6.

2 State of the Art

To develop the related work, we have followed the principles and guidelines of Systematic Literature Reviews (SLRs) as defined by Kitchenham [24]. Nevertheless, the goal in this paper is not to develop an exhaustive SLR with all the work available in the literature, but to report in a systematic manner the list of relevant contributions similar to our work focusing on the quality of service adaptation mechanisms in service-based applications. We have performed a manual search with the term “adaptation” AND “service based application” AND “quality of service” on top ranked journals and conferences from 2010 to 2015. The terms have been applied to title, abstract and keywords. By applying this search protocol, we found 145 papers covering the search criteria. 80 papers were discarded by title, 38 by abstract, and 8 papers were discarded after a fast reading, leading to a total of 19 papers that present different approaches. We classified them in four following classes based on the usage of the adaptation process: Adaptive, Corrective, Pre-
ventive and Extending.

2.1 Adaptive Adaptation

MOSES [4] is a QoS-based adaptation framework based on MAPE components. It is classified as an adaptive adaptation method. MOSES uses abstract composition to create new processes and also service selection to dynamically bind the processes to different concrete web services. MOSES is applicable where a service-oriented system is architectured as a composite service. RuCAS [5] is a rule-based service platform, which helps clients to manage their own context-aware web services via Web-API or GUI-based interface. RuCAS together with an autonomic manager could shape a self-managing ecosystem. Beggas, et al. [6] proposed a middleware that calculates ideal QoS model using a fuzzy control system to fit context information and user preferences. Then, the middleware selects the best service among all variants having the nearest QoS value to the ideal. Chouinef, et al. [25] proposed a fuzzy framework for service selection. These types of approaches are classified as context-aware or perfective adaptation in which the quality characteristics of SBA are optimized, or the application is customized or personalized according to the needs and requirements of particular users. CHAMELEON [7] is an adaptive adaptation framework which personalize/customize the application according to the device and network contexts in B3G mobile networks. They enriched the standard Java syntax to specify adaptable classes, adaptable methods and adaptation alternatives that specify how one or more adaptable methods can actually be adapted. In [26], two hidden markov models (HMM1 and HMM2) are used for context-aware service selection. HMM1 is used for modelling context information and HMM2 is used for modelling invoked services. Building HMM is different from this work, since they did not consider quality of services in selection phase. Also the efficiency and the scalability of the model are not evaluated. Since service providers do not expose the details of functional and quality of web services, it is hard for consumers to make an efficient service contract. Wang, et al. [27] proposed incentive contract to offer qualities based on consumer preferences. Canton-Puerto, et al. [28] used Baum-Welch algorithm to train HMM. They considered QoS parameters like cost, performance etc. Unlike we make relation between web services (hidden states) and tasks (observed states), they mapped web services to different qualities.

2.2 Corrective Adaptation

VieCure [8] is a corrective adaptation method which extracts monitored misbehaviours to diagnoses them with self-healing algorithms and then repairs them in non-intrusive manner. Since VieCure uses recovery mechanisms to avoid degraded or stalled systems, it is also a preventive approach. Psai, et al. [9] proposed a corrective adaptation architecture which reconfigure local interactions among service oriented collaborators or substitute collaborators to maintain system functionalities. The adaptation mechanisms operate based on similarity and socially inspired trust mirroring and trust teleportation. The authors integrate VieCure with GENESIS2 [29] (i.e. an SOA-based testbed generator framework) to realize control-feedback loop and simulate adaptation scenarios in collaborative service-oriented network. Ismail, et al. [10] proposed SLA violation handling architecture which performs incremental impact analysis for incrementing an impact region with additional information. To determine the impact region candidates, they defined Time inconsistency (direct dependency between services) and Time unsatisfactory (dependency between a service and the entire process) relationships. Then the recovery instance obtains the relevant information to identify the appropriate recovery plan. The proposed strategy would reduce the amount of change. Zisman, et al. [11] proposed a reactive and proactive dynamic service discovery framework. In pull (reactive) mode, it executes queries when a need for finding a replacement service arises. In push (proactive) mode, queries are subscribed to the framework to be executed proactively. They compute the distances between query and service specifications. They used complex queries expressed in an XML-based query language SerDiQueL. In another work by Malbub, et al. [12], PROSDIN framework is proposed which proactively perform SLA negotiation with candidate services. The goal is to reduce the lengthy negotiation process during service discovery and substitution. DRF4SOA [13] is built on service component architecture (SCA) to model program independent from technologies and encapsulate each MAPE phase in SCA Composites which allows exposing their business as a service. DRF4SOA implements substitution and load balancing strategies to tackle non-functional requirements. SEco [14] is a dynamic architecture for service-based mobile applications. It consist SEco agent and SEco manager. SEco agents gather and send quality data of running applications to SEco manager. SEco manager decide on quality improvement and send adaptation actions to SEco agent. To support architectural dynamisms, SEco agent implements dynamic offloading or dynamic service deployment strategies. SAFDIS [15] is an OSGi-based framework which uses short-term and long-term reasons to maintain the SBA quality above a minimum level. SAFDIS considers only the migration of services by registering and unregistering bundle of services.
2.3 Preventive Adaptation

Some works try to prevent service based applications from future faults or SLA violations. Wang et al. [16] make adaptation decisions through two-phase evaluations. In estimation phase, they estimate the QoS attribute (e.g., execution time) in the future and compare the estimated value with the target value defined in the SLA. If a violation is tent to happen, a suspicion of SLA violation is reported to decision phase. In decision phase, they use static and adaptive decision strategies to evaluate the trustworthy level of the suspicion in order to decide whether to accept or to neglect the suspicion.

Unnecessary adaptations can be costly and also faulty even in the proactive case. Metzger et al. [17] propose a preventive approach for augmenting service monitoring with online testing to produce failure predictions with confidence. In a similar work [18], Metzger selected prediction techniques and defined metrics to assess the accuracy of predictions. Jingjing et al. [30] proposed a proactive service selection method to prevent service provider overloads. The proactive method is based on analysing a time series of services received to forecast the overloads through a negotiation process.

2.4 Extending Adaptation

Auxo [19] is an extending adaptation approach which realize adaptation concerns through modifying the runtime software architecture (RSA) model. Auxo proposes an architecture style (interfaces, connectors and components) and runtime infrastructure which maintains an explicit and modifiable RSA model. To fulfill the modification requests, they modify the RSA model, evaluate the architecture constraints, and enact changes to the real system. SALMon [20] is a monitoring framework that supports different adaptation strategies in the SBA lifecycle by providing the knowledge base (accurate and complete QoS) to the following expert systems: WeSSQoS (for service selection based on user requirements), FCM (for service deployment on a cloud federation system), SALMonADA (for identifying and reporting SLA violations), MAE-SoS, PROSA, PROTEUS, and CASE (for adaptation purposes whenever malfunctions in the system occur). Daubert et al. [21] proposed Kevoree, a reflective framework which provides models@runtime approach to design adaptable SBA. Models@runtime considers the reflection layer as a real model that can be uncoupled from the running architecture for reasoning, validation and simulations purposes and later automatically resynchronized with its running instance. CLAM [22] is a cross-layer adaptation manager for SBA. CLAM provides Application, Service and Infrastructure models. Each model element is associated with Analysers, Solvers and Enactors. A cross-layer rule engine governs the coordination of Analysers, Solvers and Enactors. For each adaptation need, CLAM produces a tree of the possible alternative adaptations, identifies the most convenient one, and applies it.

To classify our work, we defined its characteristics using S-CUBE adaptation taxonomy. The adaptation taxonomy distinguishes approaches by three following questions: 1) Why is adaptation needed (adaptation usage)?, 2) What are the adaptation subject and aspect?, and 3) How does adaptation strategy take place?. As shown in Table 1, this research presented an adaptive method which customizes SBA based on user’s preferences and quality constraints. The adaptation subjects are SBA’s constitute services and their composition models. We applied HMM to realize service selection adaptation strategy.

3 Hidden Markov Model

An HMM has an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols [31]. In Figure 2, hidden states are presented by circles and observed symbols are presented by rectangles. HMMs are applicable in speech processing, natural language processing, extracting target information from documents etc.

HMM is formally defined in formula 1, where S is the set of states, and V is the set of possible observations [32].

\[ \lambda = (A, B, \pi) \]  
\[ S = \{s_1, s_2, ..., s_N\} \]  
\[ V = \{v_1, v_2, ..., v_M\} \]
### Table 1. Classification of SBA Adaptation Approaches, According to S-CUBE Adaptation Taxonomy.

<table>
<thead>
<tr>
<th>Usage</th>
<th>Subject</th>
<th>Aspect</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOSES [4]</td>
<td>Adaptive</td>
<td>Constitute services; Composition instance</td>
<td>New/modified non-functional requirements Service selection; Coordination pattern selection</td>
</tr>
<tr>
<td>RuCAS [5]</td>
<td>Adaptive</td>
<td>Web context-aware services</td>
<td>Contextual changes Dynamic binding Calculating ideal QoS values and selecting a service variant having the nearest QoS values to the ideal</td>
</tr>
<tr>
<td>Beggas, et al. [6]</td>
<td>Adaptive</td>
<td>Constitute services</td>
<td>QoS, User contextual changes Switching among adaptation alternatives considered at deployment time</td>
</tr>
<tr>
<td>CHAMELEON [7]</td>
<td>Adaptive</td>
<td>Adaptable service class</td>
<td>QoS; User needs; Contextual changes</td>
</tr>
<tr>
<td>VieCure [8]</td>
<td>Corrective and Preventive</td>
<td>Constitute services</td>
<td>QoS; Misbehaviours Recovery technique</td>
</tr>
<tr>
<td>Psaier, et al. [9]</td>
<td>Corrective</td>
<td>Local interactions</td>
<td>Unexpected low performance Regulation by link modification or substitution of actors based on similarity and trust metrics</td>
</tr>
<tr>
<td>Ismail et al. [10]</td>
<td>Corrective</td>
<td>Process instance; Services</td>
<td>SLA violations Reduce the amount of service that need to be recovered (or changed)</td>
</tr>
<tr>
<td>Zisman et al. [11]</td>
<td>Corrective</td>
<td>Constitute services</td>
<td>QoS Service discovery in pull (reactive) mode and push (proactive) mode</td>
</tr>
<tr>
<td>PROSDIN [12]</td>
<td>Corrective</td>
<td>Constitute services</td>
<td>QoS SLA negotiation; Dynamic discovery and binding</td>
</tr>
<tr>
<td>DRF4SOA [13]</td>
<td>Corrective</td>
<td>Components; Services</td>
<td>Non-functional requirements changes Substitution; Load balancing</td>
</tr>
<tr>
<td>SEco [14]</td>
<td>Corrective</td>
<td>Constitute portable services</td>
<td>QoS; Manageability Dynamic deployment; Dynamic offloading</td>
</tr>
<tr>
<td>SAFDIS [15]</td>
<td>Corrective</td>
<td>Constitute services</td>
<td>QoS Registering and unregistering services (bundle of services)</td>
</tr>
<tr>
<td>Wang [16]</td>
<td>Preventive</td>
<td>SBA instance; Constitue services</td>
<td>QoS Prevent unnecessary adaptation Making adaptation decisions through two-phase evaluations (estimation and decision)</td>
</tr>
<tr>
<td>Metzger [17]</td>
<td>Preventive</td>
<td>Constitue services</td>
<td>QoS, Prevent unnecessary adaptation Augmenting service monitoring with online testing</td>
</tr>
<tr>
<td>Metzger [18]</td>
<td>Preventive</td>
<td>Constitue services; Third-party services</td>
<td>QoS, Failure prediction Applying prediction techniques</td>
</tr>
<tr>
<td>Auxo [19]</td>
<td>Extending</td>
<td>Component; Connector; Interface</td>
<td>Unexpected environments Modifying runtime software architecture models</td>
</tr>
<tr>
<td>SALMon [20]</td>
<td>Extending</td>
<td>Constitue services</td>
<td>QoS Model-based and Invocation-based configuration of SALMon; Re-selection; Redeployment</td>
</tr>
<tr>
<td>Kevoree [21]</td>
<td>Extending</td>
<td>Business process; Composition and coordination; Infrastructure</td>
<td>QoS-based cross-layer adaptation Using reflection and models@runtime techniques</td>
</tr>
<tr>
<td>CLAM [22]</td>
<td>Extending</td>
<td>Whole SBA model</td>
<td>cross-layer adaptation Different strategies like: add/remove service, mismatch solving, parallelize process activities, etc.</td>
</tr>
<tr>
<td>Current research</td>
<td>Adaptive (customization)</td>
<td>Constitute services, Composition model</td>
<td>QoS changes; User’s preferences Applying HMM for QoS-based service selection</td>
</tr>
</tbody>
</table>
$Q$ is a sequence of hidden states with length $T$ and $O$ is a sequence of observations. Each observation in $O$ is emitted by a hidden state in $Q$. As presented in formula 6 and formula 7, $A$ is the transition array, storing the time-independent probability of state $j$ following state $i$, and $B$ is the observation array, storing the probability of observation $m$ being produced from the state $i$, independent of $t$.

$$Q = (q_1, q_2, ..., q_T) \quad (4)$$

$$O = (o_1, o_2, ..., o_T) \quad (5)$$

$$A = [a_{ij}], a_{ij} = P(q_t = s_j | q_{t-1} = s_i) \quad (6)$$

$$B = [b_j(m)], b_j(m) = P(x_t = v_k | q_t = s_i) \quad (7)$$

In formula 8, $\pi$ is defined as the initial probability array:

$$\pi = [\pi_i], \pi_i = P(q_1 = s_i) \quad (8)$$

The model makes two assumptions including the Markov assumption and the independence assumption, presented in formula 9 and formula 10. The Markov assumption, states that the current state is dependent only on the previous state. The independence assumption, states that the output observation at time $t$ is independent only on the current state; it is independent of the previous observations and states:

$$P(q_t | q_{t-1}^T) = P(q_t | q_{t-1}) \quad (9)$$

$$P(o_t | o_{t-1}^T) = P(o_t | q_t) \quad (10)$$

Given a HMM $\lambda$ and a sequence of observations $O$, we would like to compute $P(O | \lambda)$, i.e., the probability of observing sequence $O$.

The probability of the observations $O$ for a specific state sequence $Q$ and the probability of the state sequence are shown in formula 11 and formula 12 respectively:

$$P(O | Q, \lambda) = \prod_{t=1}^{T} P(o_t | q_t, \lambda) \quad (11)$$

$$P(Q | \lambda) = \pi_{1} \times a_{1q_{1}} \times a_{q_{2}}q_{2} \times ... \times a_{q(T-1)q_{T}} \quad (12)$$

So we can calculate the probability of the observations given the model as:

$$P(O | \lambda) = \sum_{Q} P(O | Q, \lambda) \times P(Q | \lambda) \quad (13)$$

The result shows the probability of observing sequence $O$ by considering state sequence $Q$.

Figure 3. HMM Is Applied for the Service Selection Problem.

4 Service Selection Based on HMM

In this section, we applied HMM for the service selection problem in three following steps: Modelling, Learning, and QoS-based Selection.

4.1 Modelling

The formal definition of service selection problem is as follows:

$$\lambda = (VisitRate, SelRate, \pi) \quad (14)$$

$AWS$ is the set of available web services, and $AT$ is the set of tasks of all processes.

$$AWS = (ws_1, ws_2, ..., ws_N) \quad (15)$$

$$AT = (t_1, t_2, ..., t_M) \quad (16)$$

We define $WS$ as a sequence of web services with length $T$. We also define $T$ as a sequence of tasks. Each task in $T$ consumes a corresponding web service in $WS$.

$$WS = (ws_1, ws_2, ..., ws_T) \quad (17)$$

$$T = (t_1, t_2, ..., t_T) \quad (18)$$

As defined in Eq. 2 and Eq. 3, HMM includes sequence of hidden states and sequence of possible observations. As defined in Eq. 7, hidden state $s_i$ has the probability $b_{i}(m)$ to produce the observation $v_m$. Considering $S_i$ as the $ith$ candidate web service, and $v_m$ as the $mth$ task of a process, we could say $b_{i}(m)$ indicates the probability of selecting $i$th web service for $m$th task. Figure 3 represent this assumption in graphical mode. As you can see, web services are modeled with hidden states and process’s tasks are modeled with observations.

We defined $VisitRate_{ij}$ to indicate the probability of visiting $j$th web service, just after the $i$th web service is visited. Visit rate is similar to Eq. 6 which defines the transition probabilities between hidden
states. Visit rates are initialized in learning step and their values could be updates by the log records of the services repository server (i.e. UDDI server).

\[
VisitRate_{ij} = \frac{\text{no. of times } WS_j \text{ is visited after } WS_i}{\text{no. of times } WS_i \text{ is visited}}
\] (19)

We need to present rational definitions for the state transition probabilities (Eq. 6), and the output probabilities (Eq. 7). We defined \(SelRate_{i}(m)\) to indicate the probability of selecting \(i^{th}\) web service for \(m^{th}\) task. Selection rate is similar to Eq. 7 which defines the output probabilities. Selection rates are initialized in learning step and their values could be updates by the logs of BPEL engine.

\[
SelRate_{i}(m) = \frac{\text{no. of times } WS_i \text{ is selected for } T_m}{\text{no. of times } WS_i \text{ is selected}}
\] (20)

### 4.2 Learning

After modelling the service selection problem by HMM, the output model should be trained by supervised learning methods or unsupervised learning methods. The unknown parameters of an HMM are the transition probabilities and the output (observation) probabilities.

In supervised learning, we use a database of sample HMM behaviours and estimate the transition probabilities and the output probabilities. The Visit rate array (Eq. 19) could be estimated either based on the log records of services repository or based on the log records of BPEL engine. The Selection rate array (Eq. 20) could be estimated based on the history of service invocations. The selected web services are described in WS-BPEL format. BPEL engine executes given WS-BPEL process and invokes the selected web services. Therefore, the log of service invocations is kept by BPEL engine.

In unsupervised learning, HMM parameters have to be estimated from the observed sequences and the parameters are updated based on new samples. If database of samples are not available, the standard unsupervised approaches like Maximum Likelihood Estimation or Viterbi Training could be applied [33].

### 4.3 QoS-Based Selection

It is necessary to consider user’s quality preferences while selecting the best services for each task. So we define the fitness function based on user’s preferences and the QoS parameters of services. Some quality parameters like execution-time and cost have inverse relationship with their measurements (i.e., a higher value shows a lower degree of quality), whereas some quality parameters like reliability and availability are in direct relationship with their measurements (i.e., a higher value shows a higher degree of quality). Since we need a fitness function composed of the above measures, with a value in range of \([0, 1]\), we use Eq. (21) for the quality parameters with inverse relationship and Eq. (22) for the quality parameters with direct relationship.

\[
V(Q^K_{ij}) = \begin{cases} 
\frac{Q^K_{ij} - \min_v(Q^K_{ij})}{\max_v(Q^K_{ij}) - \min_v(Q^K_{ij})}, & \text{if } \max_v(Q^K_{ij}) \neq \min_v(Q^K_{ij}) \\
1, & \text{if } \max_v(Q^K_{ij}) = \min_v(Q^K_{ij}) 
\end{cases}
\] (21)

\[
V(Q^K_{ij}) = \begin{cases} 
\frac{\max_v(Q^K_{ij}) - Q^K_{ij}}{\max_v(Q^K_{ij}) - \min_v(Q^K_{ij})}, & \text{if } \max_v(Q^K_{ij}) \neq \min_v(Q^K_{ij}) \\
1, & \text{if } \max_v(Q^K_{ij}) = \min_v(Q^K_{ij}) 
\end{cases}
\] (22)

Fitness function is defined in Eq. (23):

\[
F_{ij} = \sum_{k} V^K_{ij} W_k, 0 \leq W_k \leq 1, \sum W_k = 1
\] (23)

Where, \(W_k\) is the weight of \(k^{th}\) quality parameter identified in user’s preferences, \(V^K_{ij}\) is the standardized value of the \(k^{th}\) QoS parameter of the \(j^{th}\) candidate web service for the \(i^{th}\) task, and \(P^K_{ij}\) is the standardized fitness value of the \(j^{th}\) candidate web service for the \(i^{th}\) task.

In order to consider the effects of user’s preferences in selecting the best services, Eq. 11 is modified to Eq. 24. The fitness function (Eq. 23) adjusts the output probabilities based on user’s preferences.

\[
P(T|WS) = \prod_{t=1}^{T} P(T_t|WS_t) \times F_{tt}
\] (24)

Finally, the Viterbi algorithm is used to find the best services for a given sequence of tasks. The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states that produces a sequence of observations. The time complexity of the Viterbi algorithm is \(O(T \times |S|^2)\), where \(|S|\) indicates the number of hidden states (i.e. web services) and \(T\) indicates the length of output (i.e. tasks).

### 4.4 Pseudo Code

The pseudo code of the proposed method is shown in Algorithm 1. The algorithm starts with checking whether the HMM structure exists or not. If not, the structure is defined using VisitRate to produce state
Algorithm 1 QoS-Based Service Selection

input:
all available web services (SERVICES_REPOSITORY),
all processes’ tasks (PROCESSES),
user’s preferences (QoS_WEIGHT),
requested process (REQUESTED_PROCESS),
log of service invocations (LOG)

output:
a sequence including the best web services for realizing REQUESTED_PROCESS (SOLUTION)

1: begin
2: if (HMM does not exist) then //build and train HMM
3: for each service i in SERVICES_REPOSITORY do
4:   AWS_i ← service i // Eq. 15
5:   for each task i in PROCESSES do
6:     AT_i ← task i // Eq. 16
7:     //Use LOG records and Eq. 19 to produce state transition (probabilities)
8:     //Use LOG records and Eq. 20 to produce output probabilities
9:   end for
10: end for
11: end if
12: for each task i in REQUESTED_PROCESS do
13:   T_i task i //Eq. 18
14:   for each task i in REQUESTED_PROCESS do
15:     for each service j in SERVICES_REPOSITORY do
16:       set F_ij using QoS_WEIGHT // Eq. 23
17:     end for
18:   end for
19: end for
20: end for
21: //Use Viterbi algorithm to produce sequence WS (Eq. 17) as a SOLUTION which includes the most appropriate web services for the given sequence of tasks T (Eq. 18)
22: return SOLUTION
23: end

transition probabilities (refer to Eq. 19) and SelRate to produce output probabilities (refer to Eq. 20). Next, vector T is defined as a sequence of tasks in requested process. Each task in T consumes a corresponding web service. Then, two nested loops make Fitness matrix (refer to Eq. 23) which is used to adjust the output probabilities based on user’s preferences. Finally, the Viterbi algorithm is used to produce sequence WS, which includes the most appropriate web services for the given sequence of tasks T.

5 Experimental Results

5.1 Hypothesis

Prior to defining the experimental hypotheses, we utilized the “Goal/Question/Metric” (GQM) template [34] to explicitly define the experimentation goal G1 as follows and its regarding evaluation questions and metrics.

Goal G1: “To analyse the efficiency of the proposed method for the purpose of selecting the most appropriate services among all candidate services based on user’s preferences”.

Question Q1: Does the method show any improvement in selecting the services that are mostly aligned with user’s preferences and improves user satisfaction?

- Metric M1.1: Fitness. Fitness shows how much the output model satisfies user’s preferences. Fitness has direct relationship with user satisfaction.

Question Q2: Does the method show any improvement in scalability?

- Metric M2.1: Execution time. Execution time shows the length of time taken for selecting services and producing a composition model in common business process languages like WS-BPEL. As show in Figure 4, we measure the execution time changes with increasing the number of tasks from 10 to 100 and with increasing the number of candidate services from 5 to 50.
As shown in Table 2, we evaluated the efficiency and scalability of the proposed method for QoS-based service selection. Particularly, we aimed at evaluating the **fitness** and the **execution time** of the method.

This work was compared with GSA-based [23] (i.e. our previous work) and PSO-based service selection methods. We did not consider genetic algorithm (GA) in our comparison, since the PSO algorithm is better in finding the optimized selection with higher fitness than the genetic algorithm [35]. The measurements have been conducted on an Intel Celeron CPU 2.2 GHz PC with 1GB of RAM running Ubuntu 12.4LTS and JDK 1.7.0-17. As shown in Figure 4, the number of tasks changes from 10 to 100, and the number of candidate services changes from 5 to 50 depending on the experiment.

We used web service QoS dataset released by Al-Masri et al. [36] to evaluate service selection methods in performing users’ requests with different preferences. This dataset includes 5,000 web services with the measurements of their quality of services. The selected QoS parameters that are used in our experiments are listed below:

- **Response time (ms):** Time taken to send a request and receive a response
- **Availability (%):** The ratio of the number of successful invocations to the number of total invocations
- **Reliability (%):** The ratio of the number of error messages to the number of total messages

The quality parameters are classified in three levels including Bronze (qos_weight: 0.2), Silver (qos_weight: 0.3) and Gold (qos_weight: 0.5). In our experiments, we generated weights of quality parameters randomly.

### 5.2 User Satisfaction

Since heuristic algorithms (e.g. GSA, PSO, GA) depend on the initial population and the number of iterations, we measured fitness value in below scenarios:

#### A) Fitness changes, with increasing the number of iterations from 10 to 100

First scenario was performed 10 times with different user’s preferences. We considered 10 tasks and 5 candidate services for each task. As shown in Figure 5, HMM produced the most optimized sequence of web services that resulted in highest fitness value in each experiment. Since the Viterbi algorithm is a dynamic programming technique, the fitness value does not depend on the number of iterations. GSA moves much more quickly towards the convergence point (i.e. finding the fitter composite web service) in comparison with the PSO algorithm.

#### B) Fitness changes, with increasing the number of candidate services from 5 to 50

In second scenario, we measured the changes of fitness value with increasing the number of candidate services. The results are shown in Figure 6. The number of candidate services increases the population of candidate solutions and affects the final result. This scenario was also measured in 10 experiments with different user’s preferences. In each experiment, the GSA
Table 2. The GQM Metrics for Evaluation.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Question</th>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>Q1: User satisfaction</td>
<td>M1.1: Fitness</td>
<td>Fitness changes, with increasing the number of iterations from 10 to 100. (No. of tasks=10; No. of candidate services=5) FITNESS CHANGES, WITH INCREASING THE NUMBER OF CANDIDATE SERVICES FROM 5 TO 50. (No. of tasks=10; Iterations=100)</td>
</tr>
<tr>
<td></td>
<td>Q2: Scalability</td>
<td>M2.1: Execution time</td>
<td>Execution time changes, with increasing the number of tasks from 10 to 100. (Iterations=100; No. of candidate services=5) EXECUTION TIME CHANGES, WITH INCREASING THE NUMBER OF CANDIDATE SERVICES FROM 5 TO 50. (No. of tasks=10; Iterations=100)</td>
</tr>
</tbody>
</table>

5.3 Scalability

Figure 6. Fitness Changes, With Increasing the Number of Candidate Services (C.S.). (No. of Tasks=10; Iterations=100; No. of Candidate Services: From 5 to 50).

Figure 7. Execution Time Changes, With Increasing the Number of Tasks. (Iterations=100; No. of Candidate Services=5; No. of Tasks: From 10 to 100).

Figure 8. Execution Time Changes, With Increasing the Number of Candidate Services. (No. of Tasks=10; Iterations=100; No. of Candidate Service: From 5 to 50).

algorithm and the PSO algorithm were performed in 100 iterations. Although increasing the number of candidate services improves the fitness in both GSA and PSO algorithms, HMM is still more effective in selecting web services and producing composite models which are more aligned with user’s preferences.

In our proposed method, web services are modelled with hidden states and tasks are modelled with observations. So, for each task, candidate services are the hidden states that have an emission probability to the target task. Since the time complexity of the Viterbi algorithm has direct relationship with the square of the number of hidden states, our proposed method is a negligible value of 0.15 second gap between performing HMM and performing the least time consuming heuristic algorithm, i.e., PSO. Furthermore, most of the business processes have less than 100 tasks. Therefore we could claim that our proposed method is still applicable.

In our proposed method, web services are modelled with hidden states and tasks are modelled with observations. So, for each task, candidate services are the hidden states that have an emission probability to the target task. Since the time complexity of the Viterbi algorithm has direct relationship with the square of the number of hidden states, our proposed method is
mostly efficient in cases that there is less than $\sim 20$
candidate services for each task (see Figure 8).

In GSA and PSO algorithms, the number of candidate
services increases the population of candidate solutions and affects the execution time. As shown in
Figure 8, these types of algorithms are able to consider hundreds of candidate services in performing user’s requests.

This section could be concluded as follows. Our method achieves the maximum fitness in each experiment. Although our method is a little more time-consuming than the heuristic methods (e.g. GA, PSO, and GSA), it selects most appropriate services in a reasonable time even when the number of web services increases. The comparison of HMM with heuristic algorithms in service selection filed shows that HMM is a useful method, which improves the lacks of heuristic algorithms like fitness in a reasonable time.

6 Conclusion

In this paper, we applied Hidden Markov Model for the QoS-based service selection problem. We presented the method in following steps: Modelling, Learning, QoS-based selection. In modelling step, the HMM definition of service selection problem was described. The output HMM from the modelling step is initialized in learning step by supervised or unsupervised learning methods. The Viterbi algorithm is used in QoS-based selection step to find the most appropriate services in reasonable time.

We compared this work with GSA-based service selection method and PSO-based service selection method. Our method achieves the maximum fitness in each experiment. Although our method is a little more time-consuming than the heuristic methods (e.g. GA, PSO, and GSA), it selects most appropriate services in a reasonable time even when the number of web services increases.

In future, we would use unsupervised approaches like Maximum Likelihood Estimation or Viterbi Training to overcome continues modifications in HMM including: available web services, existing tasks, transition probabilities, and output probabilities.

References


Yousef Rastegari received his PhD from Department of Computer Engineering and Science, Shahid Beheshti University. He is member of two research groups namely ASER (Automated Software Engineering Research) (aser.sbu.ac.ir) and ISA (Information Systems Architecture) (isa.sbu.ac.ir).

Afshin Salajegheh has received his BS from Tehran University and MS in Artificial Intelligence and PhD in software engineering form Islamic Azad university science and research branch. He is faculty member of software engineering and computer science at Islamic Azad University South Tehran Branch since 1998. His major interests are software engineering, software architecture, Data mining and Data base. He also has worked as Senior IT Consultant, system analyst and Designer, Programmer, Project Manager and Data scientist for more than 23 years.