A Context-Aware Reflective-State Framework to Reconfigure Service-Based Applications

Yousef Rastegari\textsuperscript{a,}\textsuperscript{*}, Fereidoon Shams\textsuperscript{a}

\textsuperscript{a}Department of Computer Engineering and Science, Shahid Beheshti University, Tehran, Iran.

\textbf{ARTICLE INFO.}

\textit{Article history:}
Received: 25 August 2015
Revised: 13 November 2015
Accepted: 02 March 2016
Published Online: 26 August 2016

\textbf{Keywords:}
Context-Aware Adaptation, Reflective-State Framework, Service-Based Application

\textbf{ABSTRACT}

Service-based applications (SBA) offer flexible functionalities in wide range of environments. Therefore, they should be able to dynamically adapt to different quality concerns such as security, performance, etc. For example, we may provide secure services for the specific partners, or change SBA configuration based on context information. In this paper, we present context models and fuzzy logic controllers to reconfigure SBA at runtime. To separate the control aspects of SBA from its functionalities and maintain the states of SBA, we used Reflective-state pattern. Whenever context information changes, the violated SBA instances enter the adaptation process and the control unit takes care of analyzing and selecting possible variants in the corresponding points. We used Jmeter load simulator and Zabbix monitoring system to measure the performance improvement in our framework. Additionally we characterized our work in comparison with related studies, according to S-CUBE adaptation taxonomy.

\section{1 Introduction}

Service-oriented computing is increasingly being adopted as a paradigm for building loosely coupled, distributed and adaptive software applications, called service-based applications (SBA). SBAs are composed of software services, and those services may be owned by the developing organization or third parties \[1\]. SBA offers flexible functionalities in wide range of environments. Therefore it should be able to dynamically adapt to different quality concerns such as security, performance, etc.

The adaptation mechanisms are classified into Adaptive, Corrective, Preventive, and Extending according to S-CUBE adaptation taxonomy. Most of the existing approaches focus on Adaptive mechanisms \[2-4\], which modify the SBA in response to changes affecting its environment like contextual changes or the needs of a particular user. Corrective mechanisms \[5-7\] replace a faulty service with a new version that provides the same functionality and quality. Preventive mechanisms \[8\] use prediction techniques to detect the probable failures or SLA violations and also assess the accuracy of prediction. Extending mechanisms \[9\] aim to extend the SBA by adding new required functionalities.

Local knowledge is used in context-aware approaches to allow runtime re-configuration of SBA according to changes in context. In this paper, we propose an adaptive framework which uses fuzzy logic controllers (FLC) to analyze changes in context and tune the configuration parameters of SBA. Also the adaptive framework allows realizing suitable adaptation strategies like re-negotiation and re-selection.
to achieve adaptation requirements. Our work is classified as an Adaptive mechanism which consists of dynamic elements (i.e., meta states and state-based services) and management rules (i.e., fuzzy rules) to govern their behaviors [10]. When certain violations are detected, re-configuration unit takes care of analyzing and selecting possible variants in the corresponding points, such that all the variation constraints are met and, moreover, the QoS optimality is ensured.

We used itinerary purchase case study for performance analysis. In Adaptable SBA, fuzzy logic controllers determine the required memory, and the adaptation strategies are realized to request the service providers for provisioning the required memory. So the users’ requests are executed 52% faster and CPU waiting queue is decreased by 10%. We only measured the performance improvement of adaptable SBA, but it could be extended to other quality of services like security and availability by simply adding corresponding guarantee terms in service level agreement offers. Therefore we can claim that using our framework for developing SBA would guarantee better satisfaction for end-users.

The rest of this paper is organized as follows. Section 2 provides an introduction to Reflective-state design pattern. The architecture of proposed framework and its elements are described in Section 3. Section 4 describes the adaptation aspects of the framework. It starts with modeling context information and continues with presenting the structure of fuzzy logic controllers. In Section 5, we present experimental results. The framework characteristics are compared with related works in Section 6. Finally, the paper is concluded in Section 7.

2 Reflective-state Design Pattern

A framework should be established based on known design patterns. A design pattern is inspired from the best practices for a software design challenge within a particular context, and is like a recipe to prevent occurrence of design problems again. There are a few design patterns that are applicable in adaptation such as Proxy, Visitor, Reflection, and Reflective-state; each one designed to solve some aspects of adaptation challenges. Proxy is used for realizing service re-selection or dynamic binding. Visitor is applicable for implementing an abstract service in different quality concerns, for example, consider an abstract service which is implemented in both cost-effective and secure modes, and users’ requests are delegated to them.

Reflective-state [11] pattern is a refinement of State design pattern based on Reflection [12] architectural pattern. It inherits Meta level and Base level from Reflection to separate control aspects from functional aspects (i.e., separation of concerns). It also inherits State objects and state-based services from State pattern. At Meta level, application states and transitions are created by Meta state and Meta transition, respectively. Each Meta state is related to a Concrete state in Base level. Concrete states could provide several services. At Meta level, a central controller, which handles incoming events, holds the current active Meta state.

A user’s request can represent a service that has a state-specific implementation and an event that causes a transition to the next state. The Controller gets and intercepts all service requests. Then the intercepted messages are delegated to the current Meta state. The current Meta state instantiates corresponding Concrete state and passes the service request to it. The Concrete state performs the service implementation and returns the result. Finally the Controller either returns the result to the user, or induces state transition.

3 Proposed Framework

Figure 1 shows the architecture of our proposed context-aware reflective-state framework. In our framework, an application is split into two parts to separate the control aspects related to states and their transitions from the functional aspects related to state-based services. Changes to information kept in the Meta level affect subsequent Base level behavior.

Meta Level: Figure 2 illustrates the Meta level elements. The Controller instantiates and initializes Meta states and Meta transitions. It maintains and changes the reference to the current Meta state during transitions. The Controller delegates service requests to the current Meta state. Meta states implement a specific state-chart for an SBA. Meta states and Meta transitions provide information about selected SBA properties and make the SBA self-aware. As a result, the control aspects do not complicate the SBA design, and additionally we can modify the Meta states at runtime and reflect the changes on the running SBA instances.

Base Level: Figure 3 illustrates Base level elements. Base level covers the functional aspects. Its implementation builds on Meta level. It includes Concrete states and state-based services. Each Concrete state includes the services that are permitted to be consumed for the users in that state. The Context element covers all types of context information based on context conceptual model which will be described in Section B

The benefits of applying Reflective-state pattern in
Our framework includes:

- The adaptation strategies like re-configuration and re-selection are easily realized by modifying the Meta states and Meta transitions. For example by adding / removing / merging / replacing states and transitions.
- We can suspend the failed instance of an SBA for adaptation, and then, resume the instance to continue execution.
- We can separately develop the control mechanisms of SBA, and reflect the effects of new/modified rules on the running SBA instances.

4 Adaptation

In this section, first we identify the relationship between configuration parameters and context information, and then analyze the suitability of adaptation strategies to react to context changes.

4.1 Context Modeling

To provide adequate service for users, applications should be aware of their contexts and automatically adapt to their changing contexts, known as context-awareness [13]. Service-based applications should be equipped with the required mechanisms to adapt quickly to changes in the context, particularly at runtime [14]. When context changes, it might affect the processes and/or quality metrics. Therefore, SBA requires identifying and codifying the relevant context information such that it can be monitored and exploited to trigger adaptation requirements.

Context conceptual model is illustrated in Figure 4. Context information could be divided into Business, Computing, User, Infrastructure, and Ambient categories [15]. For example, Business context includes stakeholders, regulations, business trends, and User context includes end-user preferences as well as user activities. Each context category (e.g. user) consists of several context types (e.g. role), and each context type consists of several context atoms (e.g. normal, loyal, VIP) [16]. Context atoms present the status of monitored data in a higher abstraction view [17].

Sensors monitor context data, and send their measurements to persistent database. Then the monitored data is processed and refined to be represented in con-
text models. Neither monitored data nor context models are error-free and completely trusted. Also, many of inconsistent pieces of information could affect the quality indicator that is called consistency problem in [18]. Therefore, fuzzy decision maker seems to be a good solution for this problem. We use fuzzy logic controllers to adjust SBA configuration parameters based on context information. As shown in Figure 5, fuzzy dimensions are context types and context atoms are the values of each dimension. Each fuzzy logic controller adjusts one parameter of SBA configuration.

4.2 Realizing Adaptation

Dynamic re-configuration entails adaptation strategies for the replacement, migration, creation and removal of SBA entities at runtime. In our framework, when a configuration value is determined by FLC, we check the current service level agreements to find out whether it is possible to obtain the new value or not. If the value was not obtained, the Controller (at Meta level) suspends the SBA instance and applies suitable adaptation strategies. Table 1 was inspired from the work of Bucchiarone et al. [19]. As shown in Table 1, the common adaptation strategies are corresponded with the changes in context and the way each strategy could be realized in our framework. For example, if customer’s request should be processed faster, then the Controller prepares new agreement offers and negotiates with service providers for provisioning more available memory. The selection of the adaptation strategy may be characterized by a simple rule or may require advanced reasoning, based on some ontologies, or may involve user decisions. In this paper, we suggest an incremental selection mechanism. First we use the mapping in Table 1 to discover suitable strategies for current context models, and then we select and try to realize the strategy which has the lowest adaptation cost. If adaptation requirement was not achieved with the first strategy, then we try the next one with the lowest adaptation cost and continue until satisfying the requirement or failing the request.

5 Experimental Results

In this section we use a typical ‘itinerary purchase’ scenario [20, 21] to prototype our framework for developing adaptable service-based applications and to measure the performance improvement in our framework.

5.1 Case Study

The itinerary purchase process is handled by the following independent and collaborating parties: Customer, Travel agency, Airline, Hotel, and Payment system. The itinerary purchase scenario is as follows. (1) First, the customer must be authenticated by the travel agency to view the available itineraries. (2) Next the customer selects the desired itinerary and requests the travel agency for reservation. (3) The travel agency starts two parallel interactions with the hotel and the airline parties and waits until the reservation responses arrive. If both of the reservations are confirmed, then the travel agency calculates the total cost of itinerary. (4) After the total cost is determined, an interaction between the travel agency and the payment system will be started. Again, the travel agency waits until the payment is confirmed by the payment system. (5) Finally, an interaction will be started to notify the customer about the purchase status and its details. As shown in Figure 6, itinerary purchase meta level includes common Meta states and adaptation Meta states.

The common Meta states of itinerary purchase are
Table 1. Suitability of Adaptation Strategies to React to Context Changes

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Ambient</th>
<th>User</th>
<th>Computing</th>
<th>Business</th>
<th>Steps of Realizing Strategy</th>
<th>Adaptation Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Re-execution</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>1. terminate SBA instance&lt;br&gt;2. execute SBA instance from last safe point</td>
<td>very low</td>
</tr>
<tr>
<td>Re-negotiation</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>1. suspend SBA instance&lt;br&gt;2. prepare new agreement offer&lt;br&gt;3. negotiate for agreement&lt;br&gt;4. resume/re-execute SBA instance</td>
<td>low</td>
</tr>
<tr>
<td>Re-selection</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>1. suspend SBA instance&lt;br&gt;2. discover candidate services&lt;br&gt;3. select the best service&lt;br&gt;4. prepare agreement offer&lt;br&gt;5. negotiate for agreement&lt;br&gt;6. bind to new service&lt;br&gt;7. resume/re-execute SBA instance</td>
<td>medium</td>
</tr>
<tr>
<td>Re-composition</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>1. suspend SBA instance&lt;br&gt;2. make alternative composition models&lt;br&gt;3. select the best composite model&lt;br&gt;4. select services for all nodes of composite model&lt;br&gt;5. negotiate with new service providers&lt;br&gt;6. bind to new services&lt;br&gt;7. resume/re-execute SBA instance</td>
<td>hi</td>
</tr>
</tbody>
</table>

Figure 6. Itinerary Purchase Meta Level

described below:

- **Unavailable**: In our scenario, customer must have a valid session to start the purchase process. Before that, no itinerary plan is available for selection. This state includes login and register web services.
- **Wait for customer selection**: In this state, the valid customer can view itineraries and request an itinerary plan for reservation. This state includes view itineraries and select itineraries web services.
- **Reservation**: In this state, travel agency requests hotel and airline parties for temporary reservation. This state includes reserve hotel and reserve airline web services.
- **Checking out**: After getting reservation responses, travel agency asks payment system for processing customer credit card. This state includes pay online and pay by wallet web services.
- **Notifying**: After getting payment confirmation, travel agency registers the reservation permanently and sends relevant information to customer’s email. This state includes finalize purchase and mail web services.

The adaptation Meta states of itinerary purchase are described below (refer to adaptation process in Figure 6):

- **Suspending**: SBA instance will be suspended whenever an adaptation (re-configuration) event enters the Controller.
- **Adapting**: In this state, SBA instance is modified to obtain the configuration value determined by fuzzy logic controllers. In our experiments, we use re-negotiation and re-selection adaptation strategies which changes service level agreements (SLA) and substitutes service providers, respectively.
- **Resuming**: The suspended instance is resumed to continue the execution based on the new SBA structure.

In this scenario, we consider Traffic (the number of concurrent customers) and Customer priority as the context information; also we select Available memory as the configuration parameter. Figure 7 illustrates fuzzy surface. Since the services are provided by third
Parties, the travel agency (i.e. the owner of SBA) should negotiate with service providers or substitute providers to fulfill the required available memory. For example, when the number of customers increases and the priority of customer is high, the travel agency asks the service providers to offer more available memory while processing the request. Traffic is the type of computing context and Customer priority is the type of user context, therefore we only consider re-negotiation and re-selection strategies according to Table 1. We set the following ranges for Traffic, Customer priority, and Available memory:

- Traffic = [0-50] concurrent users. A high number of concurrent users will stress the SBA server, so we limited the test to our available computing resources.
- User priority = [0-10]. This parameter is defined based on the role of customer. We set Apache Jmeter to generate customers with random roles.
- Available memory = [0-100] % of 1 Gigabyte. This parameter is defined in SLA offers. Travel agency negotiates with service providers for SLA offers. This parameter identifies the required memory that the provider must provision while processing the travel agency’s requests.

5.2 Performance Evaluation

The evaluation structure is illustrated in Figure 7. We developed the web services of itinerary purchase scenario as mentioned in previous section and deployed them on virtual servers. Each virtual server hosts one or more service providers and their corresponding web services. Each web service has at least two other alternatives on different virtual servers. In this scenario, service providers offer compute resources to the travel agency (i.e. the owner of SBA). The service providers offer available compute resources via an agreement template. Once the travel agency has filled in all its requirements, it sends the offer to the service provider. The provider then checks whether the requested service can be provisioned. In case the service can be provided, it sends back a completed counter offer to the travel agency that in turn can now choose to create a negotiated agreement based on the offer. In case the service provider is not able to fulfill all the requirements stated by the travel agency, it can also send back a counter offer indicating a service it is able to provide instead. At a later point in time, the travel agency may recognize that it requires more or less resources to efficiently complete its computation. In that case it may start a re-negotiation of the agreement in order to scale the resources up or down, according to its requirements. However, if the provider was not able to fulfill travel agency’s requirements, the travel agency could use re-selection strategy to substitute the provider with a new service provider which accepts the agreement offer.

We installed Zabbix Server on Server #1 to store monitored data and provide graphical reports. Zabbix includes Server part and Agent part that communicate with each other. Zabbix Agent was installed on Server #2 to monitor the quality characteristics of SBA, and send their measurements to Zabbix Server. We used Apache Jmeter to generate virtual customers with different priorities. One hundred requests were simulated by 50 concurrent customers with a ramp up time of 300 seconds. Since Zabbix Server and Apache Jmeter consume most of resources (e.g. CPU and RAM), we deployed SBA and FLCs on a separated server (Server #2). Server #2 was an Intel Celeron CPU 2.2 GHz PC with 1GB of RAM running Linux Ubuntu 12.04LTS and JDK 1.7.0-17.

We evaluated the SBA for itinerary purchase process in the following two scenarios with similar Traffic load and Customer priorities. Performance improvement was measured based on the metrics that are related to customer (end-user) and SBA server. End-user metrics include average response time and throughput. SBA server metrics include average CPU load and average execution time.
(1) Scenario one (non-adaptable SBA): SBA is only working in common Meta states (refer to Figure 6). The available memory was set to its default value (100 Megabytes) in each virtual server and we did not apply adaptation strategies at runtime. It means that service providers supply at most 100 Megabytes of memory for processing travel agency’s requests. Figures 9 and 10 show the performance graphs. Without adaptation, it takes 24 minutes to process all customers’ requests with average response time of 424 seconds and average throughput of 4.2 requests per minute.

(2) Scenario two (adaptable SBA): SBA is working in both common Meta states and adaptation Meta states (refer to Figure 6). The available memory was determined by fuzzy logic controllers, and then re-negotiation and re-selection strategies were applied to fulfill the required memory. Figures 11 and 12 show the performance graphs. With adaptation, it takes 12 minutes to process all customers’ requests with average response time of 158 seconds and average throughput of 9 requests per minute.

The performance improvement of adaptable SBA is depicted in Table 2. In Adaptable SBA, fuzzy logic controllers determine the required available memory, and the adaptation strategies are realized to request
the service providers for provisioning the required memory. So the users’ requests are executed 52% faster and CPU waiting queue is decreased by 10%.

As shown in Table 3 in adaptable SBA, customers with higher priority experienced better response time. Now consider

$$\text{satisfaction} = \sum_{i=1}^{n} w_i QoS_i$$  \hspace{1cm} (1)

in which, n indicates the number of quality parameters that customer satisfaction depends on, $w_i$ indicates the weight (priority) of ith quality of service, and $QoS_i$ indicates the value of ith quality of service. In itinerary purchase scenario, we measured the performance improvement of adaptable SBA, but it could be extended to other quality of services like security and availability by simply adding respective guarantee terms in SLA offers. Therefore we can claim that the use of our proposed framework for developing SBA would guarantee better satisfaction for end-users.

6 Related Work

To classify our work, we defined its characteristics using S-CUBE adaptation taxonomy [22]. The adaptation taxonomy distinguishes approaches by the following three questions: 1) why is adaptation needed? 2) What are the adaptation subject, aspect, and scope?, and 3) How does software adaptation take place? As shown in Figure 13 our proposed framework is an adaptive adaptation approach which accommodates to the changes in the SBA context. It covers quality aspects like performance, security, etc. The adaptation subject is SBA instance with a temporary scope (i.e. the adaptation effects hold only in a particular context). The adaptation implementation is interactive in which human involvement is required for defining and developing fuzzy rules. The adaptation facilities are completely separated and independent from the subject of adaptation in a reflective manner. We classified the related studies in Table 4.

MOSES [2] 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 architected as a composite service. RuCAS [3] 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. Some other works like [16] and [3] applied fuzzy controllers to improve SBA functionalities based on context information (e.g. user, environment and computational contexts). For example, Beggas et al. [4] proposed a middleware that calculates ideal QoS model using a fuzzy control system to context information and user preferences. Then, the middleware selects the best service among all variants having the nearest QoS value to the ideal. 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.

VicCure [23] is a corrective adaptation method which extracts monitored misbehaviors to diagnose
Table 2. Performance Results

<table>
<thead>
<tr>
<th>Scenario</th>
<th>End-User Metrics</th>
<th>SBA Server Metrics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AVG. Res. Time (s)*</td>
<td>Throughput</td>
</tr>
<tr>
<td>Scenario 1</td>
<td>424</td>
<td>4.2</td>
</tr>
<tr>
<td>Scenario 2</td>
<td>158</td>
<td>9</td>
</tr>
<tr>
<td>Improve.</td>
<td>62%</td>
<td>53%</td>
</tr>
</tbody>
</table>

* It shows the average length of time taken from sending HTTP request until receiving HTTP response. Network latencies and web server (apache and tomcat) latencies cause the gap between average response time and average execution time.

** Number of processes in the CPU queue waiting for computation. CPU load of less than 2 is acceptable; 3-4 is slightly high and 4 plus is bad.

*** It shows only the average processing time after receiving HTTP request and just before sending HTTP response.

Table 3. Average Execution Time of SBA for Each Customer Role

<table>
<thead>
<tr>
<th>Priority</th>
<th>Role</th>
<th>AVG. Execution Time of Adaptable SBA (sec.)</th>
<th>AVG. Execution Time of Common SBA (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hi</td>
<td>VIP</td>
<td>25</td>
<td>70</td>
</tr>
<tr>
<td>Medium</td>
<td>Loyal</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Low</td>
<td>Normal</td>
<td>41</td>
<td></td>
</tr>
</tbody>
</table>

Figure 13. Grey Boxes Identify the Characteristics of Our Research, According to S-CUBE Adaptation Taxonomy

them with self-healing algorithms and then repair them in a non-intrusive manner. Since VieCure uses recovery mechanisms to avoid degraded or stalled systems, it is also considered a preventive approach. Ismail et al. [5] 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. [6] 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 computed the distances between query and service specifications. They used complex queries expressed in an XML-based query
Table 4. Classification of Adaptation Approaches

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive</td>
<td></td>
</tr>
<tr>
<td>Our work</td>
<td>QoS, Contextual changes</td>
</tr>
<tr>
<td>Cardellini et al. [2]</td>
<td>New/modified non-functional requirements</td>
</tr>
<tr>
<td>Takatsuka et al. [3]</td>
<td>Contextual changes</td>
</tr>
<tr>
<td>Beggas et al. [4]</td>
<td>QoS, User contextual changes</td>
</tr>
<tr>
<td>Psaier et al. [23]</td>
<td>QoS, Misbehaviors</td>
</tr>
<tr>
<td>Ismail et al. [5]</td>
<td>SLA violations</td>
</tr>
<tr>
<td>Zisman et al. [6]</td>
<td>QoS</td>
</tr>
<tr>
<td>Mezghani and Ben Halima [7]</td>
<td>Non-functional requirements changes</td>
</tr>
<tr>
<td>Preventive</td>
<td></td>
</tr>
<tr>
<td>Wang and Pazat [8]</td>
<td>QoS, Prevent unnecessary adaptation</td>
</tr>
<tr>
<td>Extending</td>
<td></td>
</tr>
<tr>
<td>Oriol et al. [9]</td>
<td>QoS</td>
</tr>
</tbody>
</table>

language SerDiQueL. DRF4SOA [11] is built on service component architecture (SCA) to model the 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.

Wang and Pazat [8] 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 going 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 trustworthiness level of the suspicion in order to decide whether to accept or to neglect the suspicion.

SALMon [9] is a monitoring framework that supports different adaptation strategies in the SBS 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), MAESoS, PROSA, PROTEUS, and CASE (for adaptation purposes whenever malfunctions in the system occur).

7 Conclusion

In this paper, we used Reflective-state pattern to present a framework for runtime adaptation of service-based applications. Reflective-state pattern separates the control aspects of SBA from its functionalities and maintains the states of SBA. Whenever an adaptation requirement is triggered, the control unit can suspend the running instances of SBA for adaptation. Also, the adaptation designer can easily manage re-configuration rules at Meta level and reflect their effects on the running SBA instances at Base level.

The basic guideline we followed in the framework is to utilize fuzzy logic controllers that reconfigure SBA to cope with changes in the application context. Fuzzy logic controllers tune the configuration parameters of SBA to ensure QoS optimality. To obtain the new configuration values, we applied re-negotiation and re-selection adaptation strategies. Selecting a suitable adaptation strategy is a complex problem and depends on many factors. So we used a mapping table to correspond adaptation strategies with context changes. Then we discover suitable strategies and select the strategy with the lowest adaptation cost. If the adaptation requirement was not achieved by the selected strategy we choose the next strategy with the lowest adaptation cost.
In this paper, two service-based applications for itinerary purchase scenario were developed. Experimental results showed that the adaptable SBA that was developed based on our proposed framework provides better performance than the traditional (non-adaptable) SBA. Realizing adaptation strategies like re-negotiation and re-selection decrease the gap between required quality levels and existing quality levels. Therefore, we can claim that developing SBA based on our adaptive framework would guarantee better end-user satisfaction too.

References


[21] AnisCharfi and MiraMezini. Hybrid web service


Yousef Rastegari is PhD candidate at 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).

Fereidoon Shams has received his PhD in Software Engineering from the Department of Computer Science, Manchester University, UK, in 1996 and his M.S. from Sharif University of Technology, Tehran, Iran, in 1990. His major interests are Software Architecture, Enterprise Architecture, Service Oriented Architecture, Agile Methodologies, Ultra-Large-Scale (ULS) Systems and Ontological Engineering. He is currently an Associate Professor of Software Engineering Department, Shahid Beheshti University of Iran. Also, he is heading two research groups namely ASER (Automated Software Engineering Research) (aser.sbu.ac.ir) and ISA (Information Systems Architecture) (isa.sbu.ac.ir) at Shahid Beheshti University.