A Dynamic Architecture for Runtime Adaptation of Service-based Applications

Yousef Rastegari and Fereidoon Shams
Faculty of Computer Science and Engineering, Shahid Beheshti University, Iran

Abstract: *Service-Based Applications (SBA) offer flexible functionalities in wide range of environments. Therefore they should dynamically adapt to different quality concerns such as security, performance, etc. For example, we may add particular delivery service for the golden customers, or provide secure services for the specific partners, or change service invocation based on context information. Unlike other adaptation methods which substitute a faulty service or negotiate for service level objectives, we modify the architecture of SBA, that is, the underlying services structure and the runtime services implementation. In this regard, we propose a reflective architecture which holds business and adaptation knowledge in the Meta level and implements service behaviours in the Base level. The knowledge is modelled in the form of Meta states and Meta transitions. We benefit from Reflective Visitor pattern to materialize an abstract service in different concrete implementations and manipulate them at runtime. Each service implementation fulfils a specific quality concern, so it is possible to delegate user requests to appropriate implementation instead of reselecting a new service which is a time consuming strategy. We used Jmeter load simulator and real-world Quality of Service (QoS) dataset to measure the architecture efficiency. Also, we characterized our work in comparison with related studies according to the European Software Services and Systems Network (S-CUBE) adaptation taxonomy.*

Keywords: *Software engineering, service-based application, software adaptation, reflection, quality of service.*

Received October 28, 2015; accepted July 4, 2016

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., service structure), and those services may be owned by the developing organization or third parties [14]. SBA adaptation is required to overcome the runtime changes in functionalities and quality objectives. Therefore, it is desirable to modify services orchestration and services implementation through (semi) automatic adaptation mechanisms. The former refers to a centralized logic that describes the order in which the various services are called and the way their parameters are formed and used, while the latter refers to materializing abstract services in different concrete forms regarding different quality concerns.

Adaptation mechanisms are the techniques and facilities provided by the underlying SBA that enable adaptation strategies like service reconfiguration, service reselection, or service renegotiation [11]. 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 Extending according to the European Software Services and Systems Network (S-CUBE) adaptation taxonomy. Most of the existing approaches focus on Adaptive mechanisms [1, 3, 24] which modify the SBA in response to changes affecting its environment like contextual changes or the needs of a particular user. Corrective mechanisms [7, 9, 13, 16, 20, 22, 23, 30] replace a faulty service with a new version that provides the same functionality and quality. Preventive mechanisms [18, 19, 25] use prediction techniques to detect the probable failures or Service Level Agreement (SLA) violations and also assess the accuracy of prediction. There are few approaches targeting Extending mechanisms [4, 21, 26, 27] which aim to extend the SBA by adding new required functionalities.

In this paper, we propose a dynamic architecture which supports the development of adaptive service based applications. We believe that dynamic architecture is an essential engineering technique in enabling truly adaptation and evolution of SBA in both quality and functional aspects. To practically realize architectural dynamism, we apply known design patterns to model the constitute services of SBA and propose key adaptation strategies on how to enable the architecture to be modified at runtime.

Indeed, we use reflective techniques to separate the control aspects of SBA (at the Meta level) from its implementation (at the Base level). When an adaptation requirement is triggered, the adaptation unit which is located at the Meta level analyses the situation and prepares an adaptation plan. Then the adaptation plan is realized by modifying the meta-objects and their relevant implementations at the Base

level. The Meta level makes the SBA self-aware and builds the implementation of the Base level. Although our architecture could be classified in the adaptive adaptation and the extending adaptation, it could be used as an underlying architecture for the corrective and preventive mechanisms that are presented in the related studies (refer to section 5).

Section 2 describes the proposed architecture in detail. Section 3 describes dynamic delegation strategy and compares it with common service reselection strategy. In section 4, we present experimental results. The architecture characteristics are compared with related works in section 5. Finally, the paper is concluded in section 6.

2. Proposed Architecture

In this section, we describe the architecture elements and the collaboration among them.

2.1. Overview

Here we present an overview of the architecture. To build an adaptive architecture, we take advantage of following design patterns:

- The composite pattern [6] is used for modelling service structure. It lets clients treat individual objects and compositions of objects uniformly. Moreover, we use service connectors like pipe and selector, to create composite services and manipulate service granularity.
- The reflective visitor pattern [17] is used for materializing an abstract service in various concrete forms. The designer can deal with new quality concerns by simply defining new Visitor subclasses in the Visitor hierarchy.
- The reflective state pattern [5] is used for maintaining the states of SBA and providing state-dependent services to users. It uses delegation mechanism to pass user requests to meta-objects, which in turn find and consume state-dependent services. The Reflective State pattern applies the Reflection architectural pattern to implement a finite state machine in the Meta level, by means of meta-objects that represent state and transitions, and use the interception and materialization mechanisms for implementing the control aspects in a transparent manner.

As shown in Figure 1, the Meta level holds behavioural and adaptation knowledge in form of meta-objects, including Service constructive meta-object, State meta-objects, and Visitor meta-object. The State meta-object expresses the particular condition of SBA during its life-time. When a user requests for a specific service, the State meta-object handles the request as follows. First, the State meta-object gets requested Service object from Service constructor meta-object. The Service constructor meta-object is the entry point to the whole atomic and composite service model. Then, the State meta-object passes the abstract service object to the Visitor meta-object to be materialized in requested concern. Each concern is implemented by corresponding Visitor subclass. These concerns are implemented in reflective visitor pattern. The Visitor meta-object finds and consumes the requested service. The service result is passed to the State meta-object to be presented to the user.

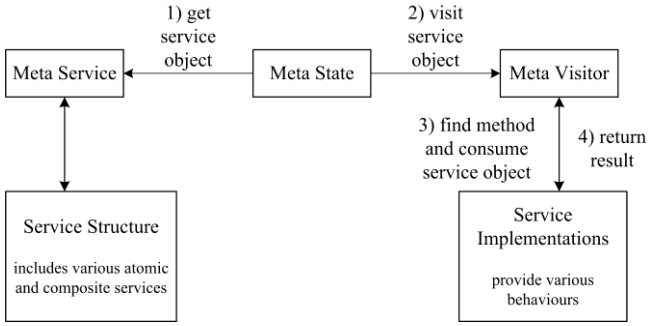


Figure 1. Architecture overview.

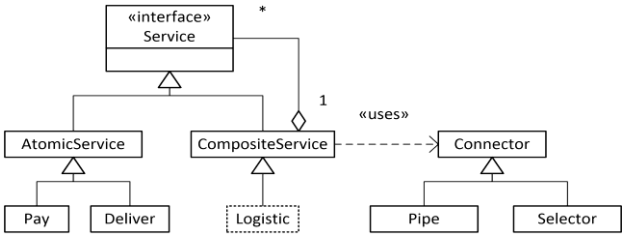
The benefits of using reflective techniques include:

- Since the states of SBA are preserved, we can suspend the failed SBA instances for adaptation and then resume them to continue their execution.
- We can separately develop the control mechanisms of SBA (at the Meta level), and reflect the effects of new/modified rules on the running SBA instances (at the Base level).
- The Meta level makes the SBA self-aware and builds the various implementations of the Base level. Indeed, the reflective techniques support separation of concerns. The concerns like business knowledge, adaptation rules, analysing, planning could be considered at the Meta level and the concerns like implementation logic and technologies could be considered at the Base level.
- The adaptation strategies like reconfiguration and reselection are easily realized by modifying the Meta states and Meta transitions. For example, by adding/removing/merging/replacing states, transitions and their corresponding services.

2.2. Service Structure

Service granularity generally refers to the size of a service and identifies the optimal scope of business functionalities [8]. The granularity is changed during service composition. Service composition is a new way for interweaving atomic services in order to exhibit a new functionality or a new service quality level. We exploited the composition connectors presented in [15] for composing two or more web services. A pipe connector composes web services W_1, …, W_n and call methods in that order. The pipe connector additionally passes the results of calls to

methods in W_i, to those in W_{i+1}. A selector connector that composes web services W_1, …, W_n, can select one web service out of them and call methods in that web service only. After composing two atomic web services W_1 and W_2, we need to devise a method to generate its interface from the standard Web Services Description Language (WSDL) interfaces of W_1 and W_2.




```
//Extend CompositeService class for adding Logistic class
Pipe pipe = new Pipe();
methods = array('Pay', 'Deliver');
ServiceInterface i = pipe.invoke(methods);
//Add service interface i in Logistic class
```

Figure 2. Modelling service structure in composite pattern.

As shown in Figure 2, we used composite design pattern to model abstract service structure. Both atomic service and composite Service classes are implementing the service class. The composite service can contain two or more atomic or composite services. The composite class uses the connector class as composition operator. The connector class contains necessary composition operators such as pipe, selector, etc.

Suppose that we want to add logistic composite service which is a sequence execution of pay and deliver atomic services. First we add logistic class by extending composite service class. Then we use pipe connector to generate logistic service interface. Finally, we add the interface in the logistic class (see Figure 2).

2.3. Service Implementation

We used reflective visitor pattern for implementing abstract services in different crosscutting concerns. The crosscutting concerns may refer to SBA quality model (e.g., secure or high available services), or a set of end-user preferences (e.g., cost-effective or medium level services), or technical constraints (e.g., asynchronous implementation of services).

The reflective visitor pattern structure is shown in Figure 3. The visitor class declares a visit method to be responsible for dynamic dispatch among concrete visitors. The corresponding service can be invoked automatically at runtime through the visit method which is defined as the only interface visible to the outside of the system. Indeed, the client is shielded from any potential changes of the implementation details. The concrete Visitor class defines a set of consume operations; each implements the specific behaviour for the corresponding service class (which is defined in the service structure).

With reflective visitor pattern,

1. The adaptation designer can easily add new crosscutting concerns by simply defining new concrete Visitor classes (e.g., the ConcreteVisitor1 in Figure 3). On the other hand,
2. The adaptation designer can also easily add new services by simply defining new Visitor subclasses (i.e., extended classes) in the Visitor hierarchy (e.g. the ExtendedConcreteVisitor1 in Figure 3). These two adaptation strategies extend the SBA in order to cover both functional and quality concerns at runtime.

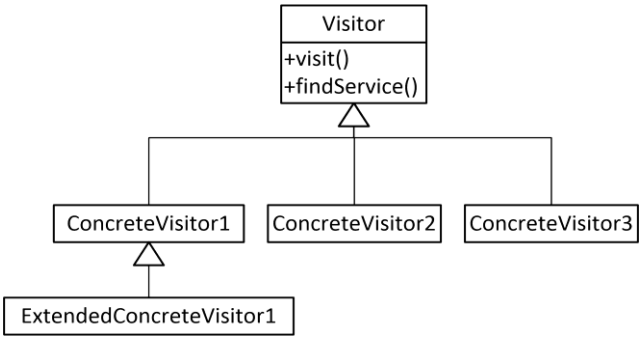


Figure 3. The reflective visitor pattern structure.

We used Shine[1] framework to implement these adaptation strategies.

Suppose that we should perform the pay and deliver services in secure mode for the golden users, and also we should provide the services in asynchronous mode for the specific partners. Therefore, we create secure and asynchronous concrete visitor classes. Then we implement pay and deliver services in secure mode and asynchronous mode. We can extend the visitor class to add new concrete Visitor classes for further concerns like cost-effective services (the dashed box in Figure 4).

In the previous section, the new logistic service was added to the service structure. Here we extend the secure concrete visitor class to provide the secure implementation of the Logistic service. The logistic service could be implemented in asynchronous mode and cost-effective mode in a similar way.

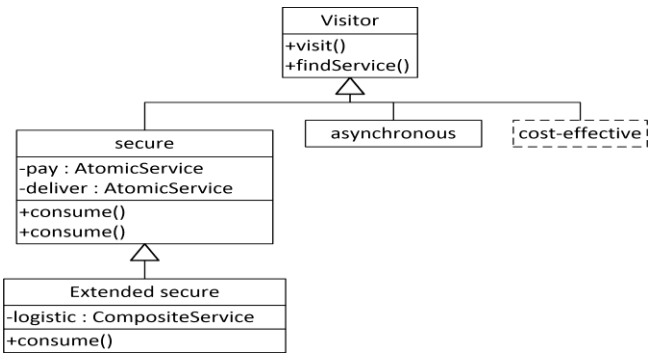


Figure 4. Implementing pay, deliver and logistic services in different crosscutting concerns.

[1]https://java.net/projects/shine-pattern

2.4. Modelling SBA Knowledge

The Reflective state pattern implements the control aspects in the meta level, separating them from the functional aspects that are implemented by the Context object and the concrete visitor objects located at the base level (see Figure 5). As a result, the control aspects do not complicate the SBA design, and additionally we can modify the meta-objects at runtime and reflect the changes on the running SBA instances.

The controller instantiates and initializes the Meta states and the meta transitions. The controller maintains and changes the reference to the current Meta state during transitions. Each Meta state includes the services that are permitted to be consumed for the users in that state.

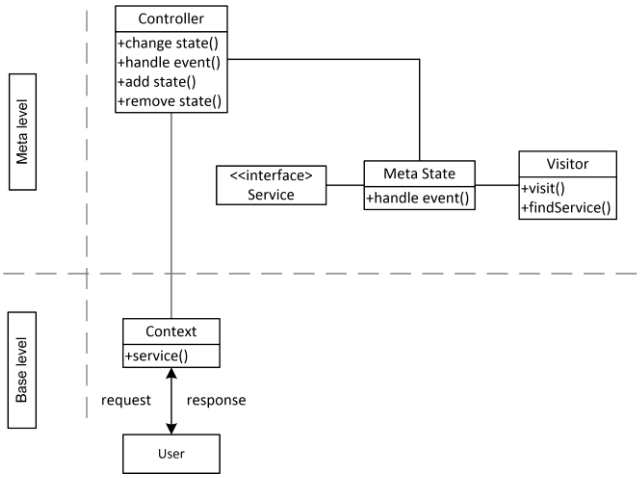


Figure 5. The reflective state pattern is used to handle user requests and provide state-dependent services.

Figure 6 shows the Meta states of an online customer. We associated the available services with each Meta state. For example, the Pay invoice is the only available service for the customers in the purchasing Meta state. The Pay service is implemented in secure, asynchronous and cost-effective modes according to Figure 4. The corresponding Meta state selects appropriate implementation based on business rules (e.g., if the customer is Golden then provide the Secure services) or context information (e.g., if the customer is from Iran then provide the cost-effective services).

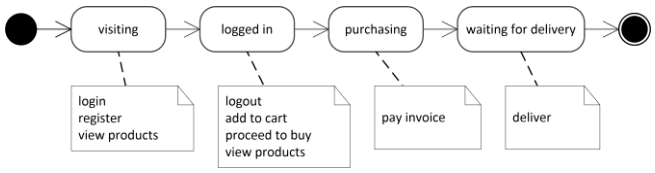


Figure 6. The Meta states of an online customer.

2.5. Collaboration

Figure 7 illustrates the collaboration among architecture classes for intercepting and handling a user request. A request can represent a service that has a state-specific implementation and an event that causes a transition to the next state. The context object gets all service requests and passes them to the controller to be intercepted. Then the intercepted messages are delegated to the current meta state. The current meta state handles the message by instantiating the suitable service object and passing the service object to the visitor object via the visit method. The visitor object finds and consumes the suitable implementation of the requested service object and returns the result. Finally the controller either passes the result to the user, or induces state transition.

3. Dynamic Delegation versus Service Reselection

Reselection is an adaptation strategy that substitutes a faulty service provider with a reliable one. Realizing reselection strategy is a time consuming process, since it discovers candidate services to select the best one and changes the biding from the faulty provider to the new one. Therefore, instead of reselecting a service it could be ideal to duplicate the highly viewed or vital services and spread the load among them to improve the performance. In our proposed architecture, we can easily add new implementations for an abstract service and apply different spreading mechanisms such as Round Robin (RR), First in First out (FIFO) etc., in the Visitor meta-object. Also, it is possible for a faulty service provider to delegate user requests to its replicas; the delegation would continue until the faulty provider gets back to the normal state.

Also, the visitor meta-object could be responsible for context-based requests. It processes the context information and delegates the request to an appropriate service implementation. As a result, instead of reselecting services for different contexts, we can switch among different implementations. In addition, we can easily add new implementations for new contexts by applying the adaptation strategies mentioned in section 2.3.

4. Experimental Results

In this section we evaluate the performance of the proposed architecture. Particularly, we aim at evaluating the overhead and the adaptation efficiency of the architecture. To measure the overhead, we compared the average response time of calling services "through our architecture" with "direct invocation". To quantify the adaptation efficiency, we measured the response time improvement regarding the dynamic delegation strategy which decreases service reselection rate. The measurements have been conducted on an Intel celeron CPU 2.2 GHz PC with 1 gigabytes of memory running Windows 7 and and Java Development Kit (JDK) 1.7.0-17.

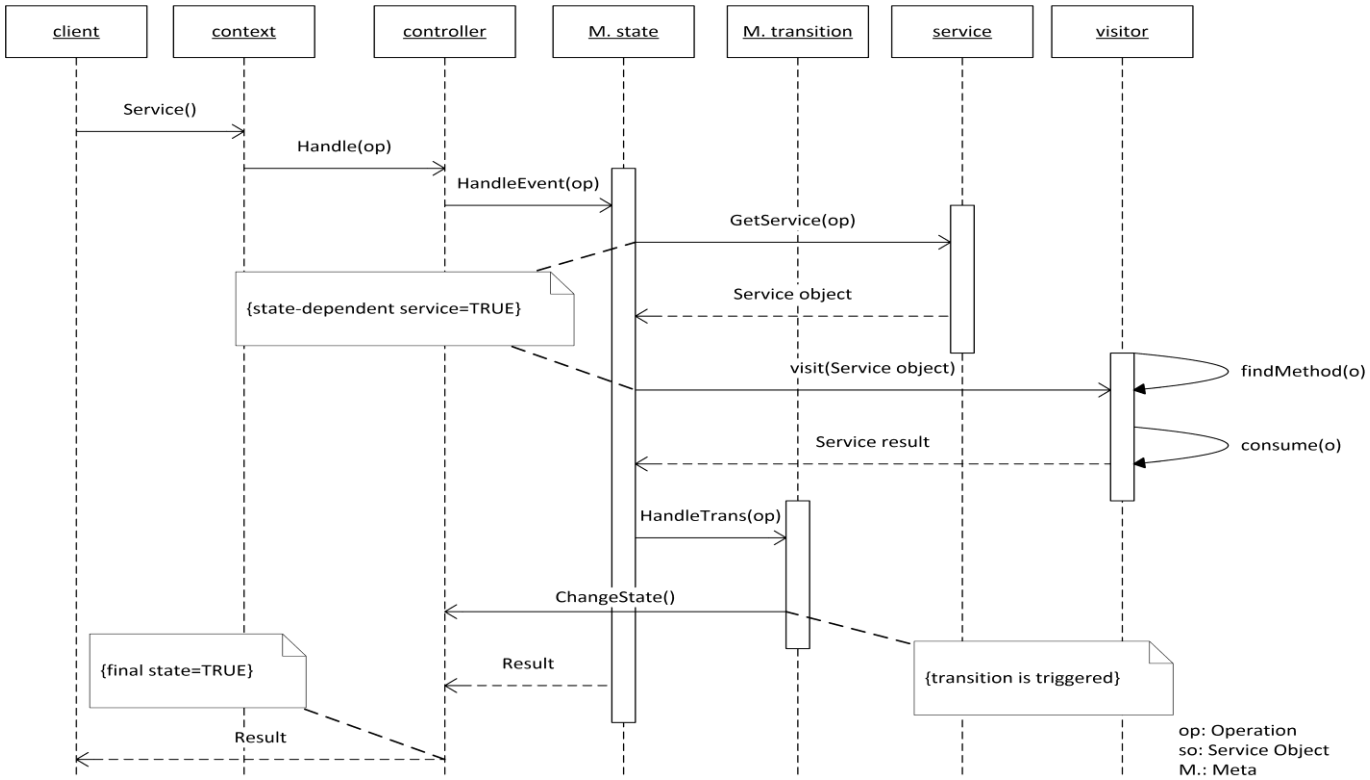


Figure 7. The collaboration view of intercepting and handling user requests.

4.1. Overhead Evaluation

We developed the proposed architecture using Java programming language. The collaboration among the architecture elements was set up according to Figure 7. To simulate a heavy load and analyse the average overhead of the architecture, we used Apache Jmeter and simulated 30 thousand requests by 2 concurrent users. The pay web service was selected in only secure mode for this benchmark. The evaluation has been done in the following cases simultaneously:

1) The pay web service was invoked 15 thousand times directly.
2) The pay web service was invoked 15 thousand times through our architecture.

The evaluation was done under normal operating conditions (i.e., one invocation at a time).

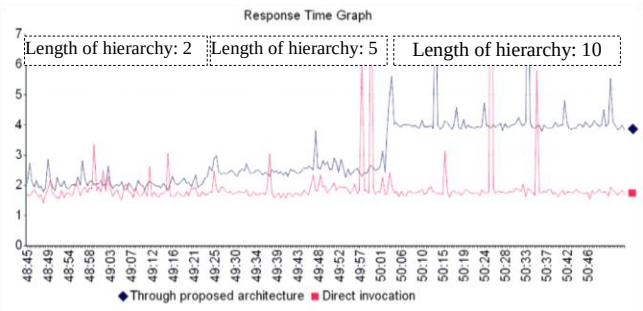


Figure 8. Response time graph.

The most time consuming element in our architecture is the method find method. As shown in the timeline of the visitor object in Figure 7, the find method takes a service object as argument. It queries visitor hierarchy to find the corresponding consume method based on the service object. There is a nested loop statement in the find method that traces up over the visitor hierarchy (inner loop) for the "Service object and all its ancestors" (outer loop) until the corresponding consume method is found. The length of visitor hierarchy and the length of service hierarchy refer to the number of ancestors in the relevant path. Since the length parameter affects the overall overhead, we have changed it after each 5 thousand requests as follows: 2, 5, and 10 ancestors (see Figure 8). We considered 10 as the maximum number of ancestors, because it is unlikely to develop an application with more than this length. We also considered that the consume method is found at the last round of search. Overhead evaluation results are depicted in Table 1. The average response time overhead is negligible (i.e., 1 millisecond).

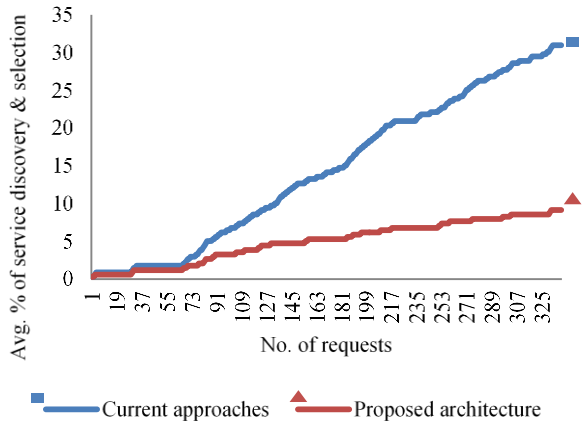
Table 1. Performance (response time) measurement results.

	#Samples	Average	90% line	99% line	Throughput
Through proposed architecture	15000	2	4	6	115.4
Direct invocation	15000	1	2	4	115.4

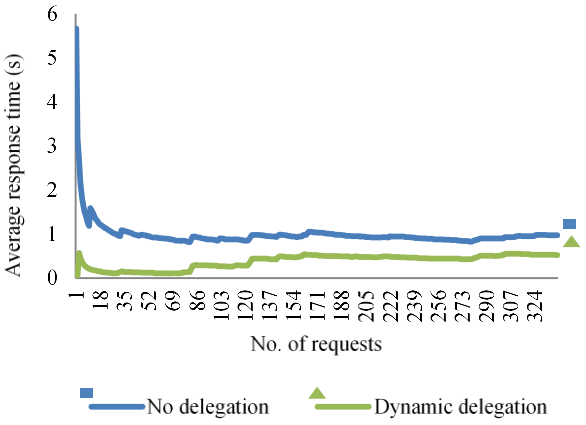
4.2. Adaptation Efficiency

Here we demonstrate that automatically delegating the request to a different service implementation has a better performance than reselecting a new service. To compare the average reselection rate and the response time of our architecture with current approaches, we used real-world Quality of Service (QoS) evaluation

results from 339 users (from 31 countries) on 5,825 Web services [28, 29]. We considered the location of user request as context information (i.e., the country of user) to select the closest service provider and improve the response time. The evaluation results are shown in Figure 9.



a) Average percentage of service discovery and service selection.



b) Average response time (seconds).

Figure 9. Dynamic delegation (in our architecture) decreases the number of reselection and improves the response time.

Figure 9-a shows the average percentage of service discovery and selection. When the location of a user request differs from the previous one usually a reselection occurs. However, in our architecture when the location changes, a new implementation of the requested service are added once, and then the delegation mechanism is used for the upcoming requests from that location. As mentioned in section 3, the delegation mechanism decreases the number of service discovery and selection, and consequently improves the response time (see Figure 9-b)).

In the above experiment, we assumed that there is already an alternative implementation for the situation at hand. So, the architecture is able to select a suitable service implementation because it already exists in the SBA. As mentioned in section 3, in the service reselection strategy, service discovery consumes time, obviously because the service has still to be discovered. Even when the set of services is known (if services are proactively discovered), service negotiation and service selection protocol could introduce an overhead, and hence the proposed architecture would still be better.

5. State of the Art

To develop the related work, we have followed the principles and guidelines of Systematic Literature Reviews (SLR) that is proposed by Kitchenham [12]. SLR is a method for combining the best quality scientific studies on a specific topic or research question. 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, Preventive and Extending.

5.1. Adaptive Adaptation

MOdel-based SElf-adaptation of SOA systems (MOSES) [3] is a QoS-based adaptation framework based on Monitor-Analyze-Plan-Execute (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. Rule-based framework for managing Context-Aware Services (RuCAS) [24] 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.* [2] 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. 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 [1] is an 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.

5.2. Corrective Adaptation

VieCure [23] 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 [23] uses recovery mechanisms to avoid degraded or stalled systems, it is also a preventive approach. Psaier *et al*. [22] proposed a corrective adaptation architecture which reconfigures 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 [23] with Genesis2 [10] (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*. [9] 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*. [30] 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 extensible mark-up language (XML)-based query language SerDiQueL. In another work by Mahbub and Spanoudakis [16], ROactive Service DIscovery and Negotiation (PROSDIN) framework is proposed which proactively performs SLA negotiation with candidate services. The goal is to reduce the lengthy negotiation process during service discovery and substitution. Drf4soa [20] 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. Mezghani and Ben Halima [20] implements substitution and load balancing strategies to tackle non-functional requirements. SEco [13] 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 decides on quality improvement and sends adaptation actions to SEco agent. To support architectural dynamisms, SEco agent implements dynamic offloading or dynamic service deployment strategies.

Self-Adaptation For DIstributed Services (SAFDIS) [7] is a framework based on Open Service Gateway initiative (OSGi), which uses short-term and long-term reasoners to maintain the SBA quality above a minimum level. SAFDIS considers only the migration of services by registering and unregistering bundle of services.

5.3. Preventive Adaptation

Some works try to prevent service based applications from future faults or SLA violations. Wang and Pazat [25] 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*. [19] 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.

5.4. Extending Adaptation

Auxo [26] is an extending adaptation approach which realizes 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 fulfil the modification requests, they modify the RSA model, evaluate the architecture constraints, and enact changes to the real system. SLA Monitoring (SALMon) [21] 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:

1. WeSSQoS (for service selection based on user requirements).
2. Federated Cloud Management (FCM) for service deployment.

3. SLA monitoring and Agreement Document Analysis (SALMonADA) for identifying and reporting SLA violations.
4. Monitoring and Adaptation Environment for Service-oriented Systems (MAESoS).
5. Proactive adaptation of Service-based Applications (PROSA).
6. Continuous Adaptive Requirements Engineering (CARE), for adaptation purposes whenever malfunctions in the system occur.

Daubert *et al*. [4] 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. Cross-Layer Adaptation Manager (CLAM) [27] 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 [11]. The adaptation taxonomy distinguishes approaches by three following questions:

1. Why is adaptation needed (adaptation usage)?.
2. What are the adaptation subject and aspect?.
3. How does adaptation strategy take place?.

As shown in Table 2, our proposed architecture is an adaptive and extending adaptation method. It covers both functional and quality aspects. The adaptation subject is SBA instance with a permanent scope. The adaptation facilities are completely separated and independent from the subject of adaptation in a reflective manner.

Table 2. Classification table.

	Usage	Subject	Aspect	Strategy
MOSES [3]	Adaptive	Constitute services; Composition instance	New/modified non-functional requirements	Service selection; Coordination pattern selection
RuCAS [24]	Adaptive	Web context-aware services	Contextual changes	Dynamic binding
Beggas *et al*. [2]	Adaptive	Constitute services	QoS, User contextual changes	Calculating ideal QoS values and selecting a service variant having the nearest QoS values to the ideal
CHAMELEON [1]	Adaptive	Adaptable service class	QoS; User needs; Contextual changes	Switching among adaptation alternatives considered at deployment time
VieCure [23]	Corrective and Preventive	Constitute services	QoS; Misbehaviors	Recovery technique
Psaier *et al*. [22]	Corrective	Local interactions	Unexpected low performance	Regulation by link modification or substitution of actors based on similarity and trust metrics
Ismail *et al*. [9]	Corrective	Process instance; Services	SLA violations	Reduce the amount of service that need to be recovered (or changed)
Zisman *et al*. [30]	Corrective	Constitute services	QoS	Service discovery in pull (reactive) mode and push (proactive) mode
PROSDIN [16]	Corrective	Constitute services	QoS	SLA negotiation; Dynamic discovery and binding
DRF4SOA [20]	Corrective	Components; Services	Non-functional requirements changes	Substitution; Load balancing
SEco [13]	Corrective	Constitute portable services	QoS; Manageability	Dynamic deployment; Dynamic offloading
SAFDIS [7]	Corrective	Constitute services	QoS	Registering and unregistering services (bundle of services)
Wang [25]	Preventive	SBA instance; Constitute services	QoS; Prevent unnecessary adaptation	Making adaptation decisions through two-phase evaluations (estimation and decision)
Metzger [19]	Preventive	Constitute services	QoS, Prevent unnecessary adaptation	Augmenting service monitoring with online testing
Metzger [18]	Preventive	Constitute services; Third-party services	QoS, Failure prediction	Applying prediction techniques
Auxo [26]	Extending	Component; Connector; Interface	Unexpected environments	Modifying runtime software architecture models
SALMon [21]	Extending	Constitute services	QoS	Model-based and Invocation-based configuration of SALMon; Reselection; Redeployment
Kevoree [4]	Extending	Business process; Composition and coordination; Infrastructure	QoS-based cross-layer adaptation	Using reflection and models@runtime techniques
CLAM [27]	Extending	Whole SBA model	cross-layer adaptation	Different strategies like: add/remove service, mismatch solving, parallelize process activities, etc.
Our proposed architecture	Adaptive and Extending	SBA instance	QoS changes; Functional changes	Add atomic service; Compose existing services to publish a new service; Implement a service in different qualities; Delegate service requests to different implementations dynamically

6. Conclusions

In this paper, we presented a dynamic architecture for runtime adaptation of service-based applications. We proposed adaptation strategies that modify and extend the architecture of SBA to cope with adaptation requirements that may come from different contexts. Adaptation requirements represent the necessity to change the SBA in order to remove the difference between the actual (or predicted) situation and the

expected one. The requirements may include new required functionalities or quality concerns like availability, security, optimality, etc,

To design the dynamic architecture, we employed Reflective state and reflective visitor patterns which provide mechanisms for changing structure and behaviour of applications dynamically. Since the reflective patterns split an application into Meta level and Base level, it is possible to separate the control aspects of the application from its implementation. Therefore we can prepare the adaptation plans in the Meta level, and realize them in the base level.

Reflective State pattern holds adaptation and business knowledge in the meta level. The knowledge is modelled in the form of meta states and meta transitions. Each meta state holds state-dependent services and provides them in different concrete forms like secure services, cost-effective services, asynchronous services, etc., meta states modify service structure and service implementation by interacting with Service meta-object and visitor meta-object respectively. Since meta states preserve the states of SBA, we can suspend the failed instances for adaptation, and then resume them to continue their execution.

With reflective visitor pattern, the adaptation designer can easily add new crosscutting concerns by simply defining new concrete visitor classes. On the other hand, the adaptation designer can also easily add new services by simply defining new Visitor subclasses (i.e., extended classes) in the visitor hierarchy. These two adaptation strategies extend the SBA in order to cover both functional and quality concerns at runtime.

In future, we will develop a software framework based on our proposed architecture. The framework is a partially complete SBA that is intended to be instantiated. It defines the architecture for the family of service based applications and provides the basic building blocks to create them. The framework also defines APIs for performing the adaptation strategies.

References

[1] Autili M., Cortellessa V., and Benedetto P., Inverardi P., "On the Adaptation of Context-Aware Services," *in Proceedings of the International Workshop on Service Oriented Computing*, Vienna, pp. 25-30, 2007.

[2] Beggas M., Médini L., Laforest F., and Laskrid M., "Towards an Ideal Service Qos in Fuzzy Logic-Based Adaptation Planning Middleware," *Journal of Systems and Software*, vol. 92, pp. 71-81, 2014.

[3] Cardellini V., Casalicchio E., Grassi V., Iannucci S., and Presti F., "MOSES: A Framework for Qos Driven Runtime Adaptation of Service-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 38, no. 5, pp. 1138-1159, 2012.

[4] Daubert E., Barais O., Nain G., Sunye G., Jézéquel J., Pazat., and Morin B., "A Models@ Runtime Framework for Designing and Managing Service-Based Applications," *in Proceedings of the 1st International Workshop on European Software Services and Systems Research: Results and Challenges*, Zurich, pp. 10-11, 2012.

[5] Ferreira L. and Rubira C.., "The Reflective State Pattern," *in Proceedings of the 5th Pattern Languages of Programs Conference*, Monticello, 1998.

[6] Gamma E., Helm R., Johnson R., John V., and Booch G., *Design Patterns: Elements of Reusable Object-Oriented Software*, Pearson Education, 1995.

[7] Gauvrit G., Daubert E., and Andre F., "Safdis: A Framework to Bring Self-Adaptability to Service-Based Distributed Applications," *in Proceedings of 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, Lille, pp. 21-218, 2010.

[8] Haesen R., Snoeck M., Lemahieu W., and Poelmans S., "On the Definition of Service Granularity and Its Architectural Impact," *in Proceedings of International Conference on Advanced Information Systems Engineering*, Montpellier, pp. 375-389, 2008.

[9] Ismail A., Yan J., and Shen J., "Incremental Service Level Agreements Violation Handling with Time Impact Analysis," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1530-1544, 2013.

[10] Juszczyk L., Truong H., and Dustdar S., "Genesis-a Framework for Automatic Generation and Steering of Testbeds of Complexweb Services," *in Proceedings of 13th IEEE International Conference on Engineering of Complex Computer Systems*, Belfast, pp. 131-140, 2008.

[11] Kazhamiakin R., Benbernou S., Baresi L., Plebani P., Uhlig M., and Barais O., *Service Research Challenges and Solutions for the Future Internet*, Springer, 2010.

[12] Kitchenham B., "Guidelines for Performing Systematic Literature Reviews in Software Engineering," EBSE Technical Report ver. 2.3, Keele University, 2007.

[13] La H. and Kim S., "Dynamic Architecture for Autonomously Managing Service-Based Applications," *in Proceedings of the IEEE 9th International Conference on Services Computing*, Honolulu, pp. 515-522, 2012.

[14] Lane S., Gu Q., Lago P., and Richardson I., "Towards A Framework for The Development of Adaptable Service-Based Applications," *Service Oriented Computing and Applications*, vol. 8,

no. 3, pp. 239-257, 2014.
[15] Lau K. and Tran C., *Emerging Web Services Technology*, Birkhäuser Basel, 2008.
[16] Mahbub K. and Spanoudakis G., "Proactive Sla Negotiation for Service Based Systems: Initial Implementation and Evaluation Experience," *in Proceedings of IEEE International Conference on Services Computing*, Washington, pp. 16-23, 2011.
[17] Mai Y. and De Champlain M., "Reflective Visitor Pattern," *in Proceedings of the 6th European Conference on Pattern Languages of Programms*, Irsee, pp. 299-316, 2001.
[18] Metzger A., "Towards Accurate Failure Prediction for The Proactive Adaptation of Service-Oriented Systems," *in Proceedings of the 8th Workshop on Assurances for Self-Adaptive Systems*, Szeged, pp. 18-23, 2011.
[19] Metzger A., Sammodi O., Pohl K., and Rzepka M., "Towards Pro-Active Adaptation with Confidence: Augmenting Service Monitoring with Online Testing," *in Proceedings of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Cape Town, pp. 20-28, 2010.
[20] Mezghani E. and Ben Halima R., "DRF4SOA: A Dynamic Reconfigurable Framework for Designing Autonomic Application Based on SOA," *in Proceedings of IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, Hammamet, pp. 95-97, 2012.
[21] Oriol M., Franch X., and Marco J., "Monitoring the Service-Based System Lifecycle with SALMon," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6507-6521, 2015.
[22] Psaier H., Juszczyk L., Skopik F., Schall D., and Dustdar S., "Runtime Behavior Monitoring and Self-Adaptation in Service-Oriented Systems," *in Proceedings of 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Budapest, pp. 164-173, 2010.
[23] Psaier H., Skopik F., Schall D., and Dustdar S., "Behavior Monitoring in Self-Healing Service-Oriented Systems," *in Proceedings of the 34th Annual IEEE Computer Software and Applications Conference*, Seoul, pp. 357-366, 2010.
[24] Takatsuka H., Saiki S., Matsumoto S., and Nakamura M., "Developing Service Platform for Web Context-Aware Services Towards Self-Managing Ecosystem," *in Proceedings of Service-Oriented Computing ICSOC Workshops*, Paris, pp. 270-280, 2014.
[25] Wang C. and Pazat J., "A Two-Phase Online Prediction Approach for Accurate and Timely Adaptation Decision," *in Proceedings of IEEE 9th International Conference on Services Computing*, Honolulu, pp. 218-225, 2012.
[26] Wang H., Ding B., Shi D., Cao J., and Chan A., "Auxo: An Architecture-Centric Framework Supporting The Online Tuning of Software Adaptivity," *Science China Information Sciences*, vol. 58, no. 9, pp. 1-15, 2015.
[27] Zengin A., Marconi A., and Pistore M., "CLAM: Cross-Layer Adaptation Manager for Service-Based Applications," *in Proceedings of the International Workshop on Quality Assurance for Service-Based Applications*, Lugano, pp. 21-27, 2011.
[28] Zhang Y., Zheng Z., and Lyu M., "Exploring Latent Features for Memory-Based Qos Prediction In Cloud Computing," *in Proceedings of the 30th IEEE Symposium on Reliable Distributed Systems*, Madrid, pp. 1-10, 2011.
[29] Zheng Z., Zhang Y., and Lyu M., "Distributed Qos Evaluation for Real-World Web Services," *in Proceedings of the IEEE International Conference on Web Services*, Miami, pp. 83-90, 2010.
[30] Zisman A., Spanoudakis G., Dooley J., and Siveroni I., "Proactive and Reactive Runtime Service Discovery: A Framework and Its Evaluation," *IEEE Transactions on Software Engineering*, vol. 39, no. 7, pp. 954-974, 2013.

Yousef Rastegari is PhD candidate at Department of Computer Engineering and Science, ShahidBeheshti 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, ShahidBeheshti 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 ShahidBeheshti University.