Pattern Based Adaptation for Service Oriented Applications

Vallidevi Krishnamurthy and Chitra Babu
Department of Computer Science and Engineering
Sri Sivasubramaniya Nadar College of Engineering
Anna University - Chennai
Chennai, India
email: (vallidevik, chitra) @ssn.edu.in

ABSTRACT
Service Oriented Architecture (SOA) facilitates developing applications that are inherently dynamic in nature since the service binding happens at runtime by matching the functional as well as Quality of Service (QoS) requirements of the user. Further, the running application can be dynamically reconfigured by monitoring the application for possible violations in the agreed QoS requirements. This paper advocates the use of various patterns to facilitate such dynamic reconfiguration in the various layers of the SOA Reference Architecture. Towards this objective, Family of Adapters pattern has been used in the service component layer to achieve dynamic switching between different versions of the same service without human intervention. In addition, an SOA design pattern has been employed in the business process layer to enhance the efficiency of the application. This pattern based approach has been tested by applying it to a sample SOA based e-Shopping application case study.

Categories and Subject Descriptors
D.1 [Programming Techniques]: Distributed Programming;
D.2.7 [Software Engineering]: Distribution, Maintenance and Enhancement

General Terms
Performance

Keywords
Dynamic Reconfiguration, Adaptation, Patterns, Service Oriented Architecture, Software Architecture

1. INTRODUCTION
Reconfiguration is a generic approach for building adaptive software systems. The software application architecture can be changed beyond simple replacement of components: Components might be added, removed, and connected differently by rewiring them. This process is known as architectural reconfiguration. Service Oriented Architecture (SOA) represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable and potentially reusable services, implemented as web services[7]. These services reside in an open and ever changing environment based on the Internet. Since the service binding happens at runtime, SOA based applications are inherently dynamic in nature. Appropriate services are selected according to user requirements dynamically and are composed to realize a business process functionality. Service oriented applications should be agile enough to adapt to the changes in environment and user needs, to be competitive in the market.

Dynamic reconfiguration [3] in SOA is typically achieved by:

1. Monitoring the system
2. Evaluating the QoS
3. Reconfiguring and structuring the services

The QoS parameters such as response time and reliability should be regularly monitored for the possible violations in the Service Level Agreement (SLA). In order to avoid potential SLA violations, faltering services need to be appropriately replaced. The service oriented application can be structured according to architectural patterns that solve problems specific to the areas of concern. Architectural patterns and SOA patterns are reused that have the benefit of providing strong support for adapting systems built using these patterns.

(D,R,FN,SN)[16] architecture is followed in the current work, which supports dynamic re-composition. The application builder should specify how the application will be changed in case of system service failures, performance degradation, and system evolution. The rules can be specified as reconfiguration policies. In this way, after the application is deployed, the data collection service can continuously monitor the behavior of the participating services and can collect data at runtime. The data collected is periodically analyzed to decide whether it is necessary to trigger a reconfiguration. If a reconfiguration is triggered, the application specification including the workflows and participating services will be dynamically updated according to the reconfiguration policy. The entire control process is then repeated as new data is collected via monitoring.

The rest of the paper is organized as follows. Section 2 explains the proposed approach for the dynamic reconfiguration using patterns. Section 3 describes the application case study of an e-Shopping scenario which illustrates the proposed approach. Section 4 explains the related work. Section 5 concludes and provides the future directions.

2. PATTERN ORIENTED DYNAMIC RECONFIGURATION
Service Orientation is said to have its roots in a software engineering theory known as “separation of concerns”. Consequently, SOA Reference Architecture (SOA RA)[2] has been chosen as the underlying architecture. This architecture comprises of 9 layers in which 5 layers are horizontal and 4 are vertical. The horizontal layers relate to the overall functionality of the SOA solution. The vertical layers are non functional in nature.
that cut across all the functional layers. An SOA based e-Shopping application has been taken as a case study where the application is dynamically configured with the various versions of the same service whenever needed. The proposed approach involves monitoring of QoS parameters such as load, availability and response time. Whenever a possible violation in the agreed upon QoS values is foreseen, suitable reconfiguration of the application is carried out through appropriate patterns in different layers of the SOA RA. To successfully maintain the different versions for a particular application, a family of adapters pattern is applied in the service component layer. Publish subscriber pattern is also applied in the same layer to improve the business by increased availability of various products through timely restocking. Capability recomposition[8] which is a service composition pattern is applied in the business process layer as the various services involved in the e-Shopping application are composed in that layer. The application of this pattern helps in reducing the response time of a particular service. The consequences of using the pattern are represented as a part of the pattern itself. Hence, patterns are used as a potentially important vehicle for incorporating the significant design decisions.

2.1 Family of Adapters Pattern

Family of adapters pattern was derived from the idea of chain of adapters technique[14]. Family of adapters pattern is implemented by inheriting the implementation of the basic functionalities from the older version. The class diagram for the family of adapters pattern is shown in Figure 1. The basic version 1 interface is connected to the database which has the basic implementation of discount calculation and security services in it. The V1 to V2 adapter inherits the V1 implementation in it and thus the V2 interface is obtained. Similarly, V2 to V3 adapter which inherits the functionalities of V2 from the previous adapter leads to the interface V3. The adapter pattern provides compatibility between the various versions of interfaces.

Family of adapters pattern, when applied by the service developers, has an advantage of providing an ability to dynamically switch between different versions of the service easily. The basic version interface is always exposed to the service consumer. The higher end version of that same service is composed only when the administrator of the service consumer requests for it. Hence, the service consumers can optimize their expenditure by paying the additional fee for the higher end version of the service only for a limited calendar period. A service provider who designs the service with such flexibility will naturally be a preferred provider among the consumers.

3. APPLICATION CASE STUDY

The e-Shopping application is represented in the SOA RA as shown in Figure 2. The lower layer is the operational layer in which all the databases are represented. Above this layer, are the components or the implementation of the services, that constitutes the service component layer. Services layer gives the interfaces of the implementation provided by the service component layer. Business process layer provides the flow of the service implementation. The consumer layer captures the corresponding user interface design.

The workflow of the e-Shopping application is illustrated in Figure 3. Initially, the customer registers and logs in. During login process itself, the shipping address should be provided. The shipping address is sent to the broker which checks the corresponding warehouse inventory whenever an item is added to the cart. Once the customer logs in, he can browse through the catalog and can select the items needed. Once all the required items are added to the cart, he can pay the bill. Once the payment process is completed successfully, a notification is sent to the broker. At this point, the broker checks the corresponding warehouse details available in the broker database and sends the expected dispatch date and the shipping details to the end user. Further, the broker notifies the respective shipping company which in turn approaches the corresponding warehouse and collects the goods and delivers them to the end users.

3.1 Application Evolution

There are three versions of the e-Shopping application. The first version provides only the basic functionalities such as security and discount calculation. This version of the login service provides an attractive multimedia view. Security is achieved using the pipe and filter pattern[5] where filter 1 checks the purchase limit and the filter 2 performs the Personal Account Number(PAN) card validation if needed depending upon the purchase limit. In future, the security can be strengthened, by adding another filter without disturbing the other parts of the application. This is the advantage of using pipe and filter pattern in this scenario. The discount mechanism is implemented using the master slave pattern where one of the slave fetches the product cost from the database and the other slave calculates the discount. The master calculates the total cost of the product.

The second version of the application is useful when the load is heavy i.e., only during certain calendar period of time. This second version of login service is having the advanced feature of checking the load for the login service. The web server stress tool[1] simulates the load for the login service. The load monitor proactively checks for any possible violations in the assumed load. When the load exceeds a predefined threshold, it triggers the dynamic switching from the multimedia view to the normal view. In this case, the change is realized by the end user. The administrator can select the calendar period according to his need so that he enjoys the benefit of not paying for the advanced version throughout the year. This is achieved by dynamically selecting the services according to user needs. The second version being an advanced one includes both the basic and advanced features.

The third version of the login service is used both by the

Figure 1: Family of Adapters Pattern Class Diagram

Figure 3. Initially, the customer registers and logs in. During login process itself, the shipping address should be provided. The shipping address is sent to the broker which checks the corresponding warehouse inventory whenever an item is added to the cart. Once the customer logs in, he can browse through the catalog and can select the items needed. Once all the required items are added to the cart, he can pay the bill. Once the payment process is completed successfully, a notification is sent to the broker. At this point, the broker checks the corresponding warehouse details available in the broker database and sends the expected dispatch date and the shipping details to the end user. Further, the broker notifies the respective shipping company which in turn approaches the corresponding warehouse and collects the goods and delivers them to the end users.

3.1 Application Evolution

There are three versions of the e-Shopping application. The first version provides only the basic functionalities such as security and discount calculation. This version of the login service provides an attractive multimedia view. Security is achieved using the pipe and filter pattern[5] where filter 1 checks the purchase limit and the filter 2 performs the Personal Account Number(PAN) card validation if needed depending upon the purchase limit. In future, the security can be strengthened, by adding another filter without disturbing the other parts of the application. This is the advantage of using pipe and filter pattern in this scenario. The discount mechanism is implemented using the master slave pattern where one of the slave fetches the product cost from the database and the other slave calculates the discount. The master calculates the total cost of the product.

The second version of the application is useful when the load is heavy i.e., only during certain calendar period of time. This second version of login service is having the advanced feature of checking the load for the login service. The web server stress tool[1] simulates the load for the login service. The load monitor proactively checks for any possible violations in the assumed load. When the load exceeds a predefined threshold, it triggers the dynamic switching from the multimedia view to the normal view. In this case, the change is realized by the end user. The administrator can select the calendar period according to his need so that he enjoys the benefit of not paying for the advanced version throughout the year. This is achieved by dynamically selecting the services according to user needs. The second version being an advanced one includes both the basic and advanced features.

The third version of the login service is used both by the

Figure 1: Family of Adapters Pattern Class Diagram
administrator and end users. During the discount period, many users are expected to login to the site. Monitoring is done on the availability of all the products. An alarm message is sent to the administrator whenever the availability reaches its minimum threshold for any of the product. Due to this, the administrator gains knowledge regarding the list of items that needs to be reordered. This prevents the scenario of end users getting the message that “the particular product is out of stock”. This version helps improving the sales productivity. Publish subscriber pattern is used in this context to monitor the availability of the products and to notify the administrator regarding the re-ordering list. Family of Adapters pattern is used by the service provider to dynamically switch between these three versions of the application based on the user requirements.

3.2 Dynamic Service Substitution
Dynamic service substitution in payment process is implemented by monitoring the load simulated by the web server stress tool. This tool simulates the load for the payment gateway. If gateway 1 is expected to have a violation in the response time due to heavy load, the alternative route of gateway 2 is selected well in advance. Compositions are performed using Business Process Execution Language (BPEL) for both the gateway services. Nevertheless, it is not known prior to the execution, which gateway will be handling the payment. It is finalized on the fly by the web server stress tool with load as the deciding factor. This method of dynamic service substitution, improves the performance of the service by guarantying the best possible response time at all times.

3.3 Efficiency Improvement
An SOA design pattern describes standard solutions to recurring problems during composition of SOA based applications. These SOA design patterns are independent of a particular application domain. Capability recomposition is an SOA design pattern applied with a view of increasing the efficiency of execution of composed services whose capabilities find use in different contexts. Using the agnostic service logic to solve a single problem alone is wasteful and does not leverage the reuse potential of the logic. Agnostic service capabilities can be designed for repeated invocation in support of multiple compositions that solve various problems. The Capability recomposition pattern for the e-Shopping application is illustrated in Figure 4. Through that pattern, the individual capabilities of the original services can be repeatedly aggregated together with additional capabilities into different composition configurations. This enables capabilities to collectively solve the large problem for which they were originally designed in addition to several other problems.

The customer authentication service is composed twice, once when the customer logs in and then when the customer checks his bank
account in the e-shopping application (Online Shopping Mall) bank. The execution time is found to decrease when the web service is composed for the second time. The readings for 5 trials are as shown in Table 1.

<table>
<thead>
<tr>
<th>Attempt</th>
<th>Execution time on 1st Composition (ms)</th>
<th>Execution time on Recomposition (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>34</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>37</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>31</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>27</td>
</tr>
<tr>
<td>5</td>
<td>38</td>
<td>27</td>
</tr>
</tbody>
</table>

Thus, the capability recomposition pattern was applied and it reduced the execution time in the e-shopping application.

### 4. IMPLEMENTATION

This work was implemented using Java/J2EE and Microsoft .NET technologies. Open source technologies such as Java/J2EE, openESB, openUDDI, Sun Java Application server 9.0 (Glassfish) and MySQL v5.1 are used. The IDE NetBeans 6.5 has been used. The Inquired WSDL endpoints from the UDDI registry are used to create BPEL workflow by using Netbeans BPEL designer tool. The designer tool provides a simple drag and drop interface to generate the required BPEL. The application was tested on different operating systems such as Linux and Windows and different browsers such as Internet Explorer, Mozilla Firefox and Google Chrome.

### 5. RELATED WORK

This section gives a brief summary of various works related to how patterns are applied to achieve reconfiguration in many distributed applications and the different methods of reconfiguring the service oriented applications.

#### 5.1 Design Patterns

Ramirez [15] had proposed twelve adaptation-oriented design patterns which use a template similar to that of Gamma design patterns [10]. These adaptive design patterns help in overcoming the challenges faced in monitoring, decision making, and reconfiguration. Since the adaptive logic is separated from the functional logic, it allows the solutions to be reused across multiple applications and domains. These adaptation oriented patterns are represented in UML which provides structural and behavioral information. More particularly, structural information is represented in UML class diagram and the behavioral information is represented in UML state chart diagram. Among the twelve adaptation oriented design patterns, three design patterns support the monitoring mechanism, five design patterns help to overcome the challenges in decision making and four design patterns support reconfiguration.

Gomaa [12] had proposed four software reconfiguration patterns for a software product line and these can be reused in other software product lines that require reconfiguration. They are master/slave pattern, centralized control pattern, client/server pattern and decentralized control pattern. In the master/slave reconfiguration pattern, the master receives the client request for services and divides the work among identical slave components. Once the responses are obtained from the slaves, the master merges and summarizes these responses. The slave components exist only at the behest of the master component. The centralized control reconfiguration pattern is used in real time control applications. In this context, the central controller component controls the removal or replacement of the components. In the client/server reconfiguration pattern, the clients are added or removed only after completing the current transaction, if it is a sequential server. If it is a concurrent server, it can be removed or replaced only after completing the current set of transactions and queuing up new transactions. The
5.2 Reconfiguration in Service Oriented Architecture

Dynamic reconfiguration is currently achieved in workflow specifications employing web services using techniques that modify endpoint bindings and control structures. Dynamic reconfiguration method for web services based on policy is implemented using the two phases namely the semantic analysis phase and reconfiguration policy specification phase [6]. A middleware framework to provide the self-healing properties for QoS management in web service-based distributed interactive applications is proposed by Halima [13]. SIROCO middleware [9] platform deals with the reconfiguration of service orchestrations for unavailable services. Dynamic Reconfigurable ESB Service Routing (DRESR) [4] approach allows the abstract routing table to be changed at runtime in which the service provider for each node, service composition logic and service integration topology can be changed. The dynamic reconfiguration using template based web service composition [11] approach, overcomes the limitations of WS-BPEL (Web Services Business Process Execution Language) such as lack of modularity and flexibility.

6. CONCLUSIONS

A pattern based approach was proposed in the different layers of the SOA reference architecture to achieve dynamic reconfiguration of service oriented applications. The e-Shopping application has been taken as a case study. It was shown how this application adapts to the environment and the user requirements by switching between three different versions of the login service using the family of adapters pattern. The first and the third version of the application employs architectural patterns such as pipe and filter, master-slave and publish-subscribe. These patterns were applied in the service component layer. In SOA RA, as the view part is totally separated from the business logic, the second version of the login service was easier to achieve. These three versions of login service are provided by the same service provider. The next level of reconfiguration is dealt with switching of services from different service providers. This was tested with the different payment gateway providers. These two methods of dynamic reconfiguration are triggered whenever the assumed maximum threshold for load is foreseen to exceed its limit. To improve the efficiency of the application by reducing the time of execution of the a particular service, one of the SOA design patterns capability recomposition pattern was applied. Exploring dynamic reconfiguration of SOA based applications through the aspect oriented design patterns is being pursued currently.

7. REFERENCES

