CHAPTER 5

QoS AWARE SERVICE COMPOSITION

5.1. PREAMBLE

The service registry provides information of elementary services and composite services that are registered by a service provider. Number of reusable elementary services are combined together to form a composite service. This process is called service composition.

Service composition constructs composite service, which contains \( n \) tasks in sequence ( \( i^{\text{th}} \) level task \([l_i]\) will be executed after \( i-1^{\text{th}} \) level task \([l_{i-1}]\) ) or in parallel ( all tasks in same level will be executed simultaneously ) or in switch ( either \( i^{\text{th}} \) or \( j^{\text{th}} \) node of same level task will be executed ) or loop (the \( i^{\text{th}} \) level task will be executed \( n \) times) depending on execution flow.

The service composition can be realized using two approaches (i) Orchestration and (ii) Choreography (Sebastien et al 2008). The key difference between these two is that the orchestration approach assumes a single central point of control over the entire scope of the process execution, while the choreography approach assumes that execution control is shared, potentially across multiple business processes.

Service composition can be classified as follows,

i. Static (or design time) service composition.

ii. Dynamic (or run time) service composition.

iii. Model driven service composition.

v. Declarative service composition.

vi. Automated service composition.


Service composition requires an effective method to facilitate quick and simple composition of services and remains as a key challenge to realize the full potential of services.

Existing service composition methodologies use one or possible combinations of the above said methodologies for service composition (Pires et al 2002, Sudhir 2007, Aniss et al 2008, Shuying et al 2009, Xiaohui et al 2010 and Incheon et al 2014). Similarly QoS aware service compositions for constructing efficient composite service using non functional requirements are also available in literature (Jaeger et al 2005, Berbner et al 2006, Nguyen & Kowalczyk 2006, Diego & Maria 2008, Guobing et al 2014). These methodologies use Business Process Execution Language (BPEL) description for service composition which consumes more time and space to allocate the best service. These methodologies do not store the dynamic composition sequences and frequently used services for future references. For service selection, only a subset of QoS is considered. It is also noted that all structural activities are not accounted for computing QoS aware composite service.

A dynamic orchestration based QoS aware service composition is proposed over extended three tier Service Oriented Business Intelligence (SOBI) architecture. This dynamic service composition process selects the best service using service discovery process discussed in Chapter 4 and allocates it to respective level of composite service. Apart from functional attribute and QoS attribute verification for service discovery, the service signatures or interfaces of participating services are verified for constructing a composite service.
5.2. DYNAMIC QoS AWARE SERVICE COMPOSITION

The EXWSA infrastructure is improved by providing two repositories namely, Composition Sequence Repository (CSR) and Frequent Service Sequence Pattern Repository (FSSPR) for efficient QoS aware service composition. The extended SOBI architecture, Quality Aware Service Integration Architecture (QASIA) is illustrated in Figure 5.1.

![Diagram of Quality Aware Service Integration Architecture (QASIA)](image_url)

**Figure 5.1 Quality Aware Service Integration Architecture (QASIA)**
5.2.1 Quality Aware Service Integration Architecture (QASIA)

The proposed architecture extracts functional attributes and QoS from WSDL description of web services. This extracted information is divided into three classes namely, functional attributes, interface attributes and QoS attributes. These attribute values are stored in Functional Attribute Tree (FAT), Service Interface Registry (SIR) and Indexed Sequential Array (ISA) respectively.

The FAT contains name of the service, description of service, location of service with protocol, service category. ISA contains QoS parameters like response time and availability. SIR contains data such as number of input and number of output parameter with their types.

The service discovery process searches the FAT and extracts all matching services. The parameter information of a discovered service is obtained from SIR and is verified with its predecessor service parameters that are available in a composite service. QoS values for all matching services are extracted from the ISA for service selection. The modules of QASIA and its description is given below

i. Composition tree construction – Constructs composition tree from BPEL description.

ii. Conversion and Accumulation of Composition Sequence – Converts composition tree to composition sequence and stores composition sequence in CSR.

iii. Frequent Service Sequence Pattern Extraction – Extracts FSSPs from CSR.

iv. Accumulation of FSSPs – Stores FSSPs in FSSP Repository.
v. Service Ranking – Ranks the discovered services using multi-utility ranking function.

vi. User Feedback Collection – Obtains user perceived QoS values from the user for ranking.

vii. Alternative Service Selection – Selects an alternative service if the best service fails to bind during dynamic composition.

viii. Server Benefit Function – Calculates the server load before binding.

ix. Metadata – Stores information of data sources.

The data tier contains data sources to be analyzed using BI services. These data sources are heterogeneous and distributed. Along with the data sources the data tier contains integrated data store, FAT, SIR, ISA, CSR and FSSPR. This proposed architecture permits both static and dynamic service composition.

5.2.1.1 Static and Dynamic Service Composition

In static service composition, the sequences of execution for all compliant BI scenarios are provided as a separate service definition. The proposed architecture selects the required service using service discovery for different requirements. The selected services are allocated during design of composite service. The allocated service is not modified during execution of composite service. The pseudo code for an orchestration based service composition is shown in Figure 5.2. This contains number of services, which are invoked in sequence and they bind to particular service for composite service execution.
invokeAuthoservice  
if user is identification is valid  
  if user type is admin  
    invoke service1  
    invoke service2  
    invoke service3  
    .....  
    invokeserviceN  
  end if  
end if  

Figure 5.2 Pseudo code for orchestration (Static composition)

New static composite service is also created by replacing the allocated services of composite service with new compliant service from registry using discovery process. Most of the existing service discovery systems use either string matching approach or ontological approach (Aniss et al 2008) for searching, but the proposed system uses both approaches to search services. The string matching approach uses standard shared vocabulary or business messages for ensuring interoperability between different services. While the ontological approach uses the relationship between the vocabularies. To automate the service allocation process and to allocate best service during runtime dynamic composition is preferred.

Dynamic composition is implemented to invoke appropriate services based on user request during runtime or binding. The pseudo code for orchestration based dynamic service composition is shown in Figure 5.3. This pseudo code calls the service discovery module for selecting a service from service repository. The selected service is dynamically bound during execution.
invokeAuthoservice
if user is identification is valid
    if user type is admin
        call Service Discovery for service1
    call Service Discovery for service2
    call Service Discovery for service3
    .......
    call Service Discovery for serviceN
end if
end if

Figure 5.3 Pseudo code for orchestration (Dynamic composition)

Co-ordination among services involved in composition is ensured using coordinator service module. This service verifies the sequence of execution of services involved in composite service.

The conversation service passes the output parameter of predecessor to input parameter of current service using SOAP envelops. The Execution monitoring service works in synchronous with co-ordination service and registers all information in log file. During failure, the log information is used to restore to the previous safe state. To ensure reusability and to improve the effectiveness of frequently used services, the service repository is further extended to store the composition sequences in a separate registry.

The proposed QASIA has the following advantages,

- Allows static and dynamic service composition.
- Uses both vertical and ontological approach for service composition.
- Increases usability of services by providing accurate information of services in respective sub registries.
- Reduces service discovery time.
- Ensures consistency – during service composition the system will not allow services to communicate if they are from different domain (Category / Sub category), even if they have same signature.

### 5.2.2 QoS Service Composition using Composition Sequence

This section proposes a dynamic QoS aware service composition methodology over extended web service architecture Quality Aware Service Integration Architecture (QASIA). This system scans the BPEL description of composite service and converts into a composition tree.

#### 5.2.2.1 Composition Tree

The composition process constructs composition tree using Business Process Execution Language (BPEL) description of composite service (Bing et al 2009). The construction of composition tree is illustrated in Figure 5.4a and 5.4b.

![Diagram](attachment:composition_tree.png)

Figure 5.4 Composite Service for a business process
The business process flow contains activity nodes and nodes ‘A’ to ‘J’ represent services in Figure 5.4a. The tree representation of composite service is shown in Figure 5.4b.

5.2.2.2 QoS of Composite Service

The service registry contains number of services with same functionality. These services may differ in non functional or QoS attributes like response time, latency and cost. To improve the efficiency of composite service, the candidate services are ranked and the best service is selected for each leaf node in composition tree according to the constraints specified by a consumer. Subset of QoS attributes used for service composition is shown in Table 5.1. This table contains six QoS attributes, where response time and latency are to be minimized and the other four are to be maximized during service selection (Zibin et al 2014).

### Table 5.1 QoS attributes

<table>
<thead>
<tr>
<th>S.No.</th>
<th>QoS Attribute Name</th>
<th>Description</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Response Time</td>
<td>Maximum or Average Time taken to send a request and receive a response.</td>
<td>ms</td>
</tr>
<tr>
<td>2.</td>
<td>Availability</td>
<td>Ratio of number of successful invocations to the total invocations.</td>
<td>%</td>
</tr>
<tr>
<td>3.</td>
<td>Throughput</td>
<td>Total Number of invocations for a given period of time.</td>
<td>Invokes/second</td>
</tr>
<tr>
<td>4.</td>
<td>Successability</td>
<td>Ratio of number of response to the number of request messages.</td>
<td>%</td>
</tr>
<tr>
<td>5.</td>
<td>Reliability</td>
<td>Ratio of the number of error messages to the total messages.</td>
<td>%</td>
</tr>
<tr>
<td>6.</td>
<td>Latency</td>
<td>Time taken for the server to process a given request.</td>
<td>ms</td>
</tr>
</tbody>
</table>
5.2.2.3 Composite tree construction

The tree construction process scans the composite service Business Process Execution Language (BPEL) description and searches for BPEL tags like sequence, flow or parallel, switch or select, while and pick that depicts the sequence of execution of services. For such tags, activity nodes are created and added to the composition tree. Within these activity tags, invoke tags are provided to call required services. For each invoke tag, a service node is added as a child to the respective activity node. A node of the composition tree contains the following information:

i. Node type: Activity or Service

ii. Level : 0,1,2…n

iii. Value: Activity type / Service ID

The node type value is assigned using tag information. A level starts with 0 and is incremented by 1 for every subsequent level. The level is used to differentiate is same service is used in different levels. The value attribute may have type activity node or the service ID of a discovered service.

The service discovery process reads the functional requirement of a service from the BPEL description and searches the FAT. For every satisfying service, the signature information is obtained from SIR and verified with the signature information of its predecessor. QoS values of matching services are obtained for service selection.

BPEL description of a composite service with six activities and 11 services at different levels and corresponding composition tree is illustrated in Figure 5.5.
<table>
<thead>
<tr>
<th>BPEL description Composition for business process</th>
<th>Composition tree</th>
</tr>
</thead>
</table>
| `<SEQUENCE>` <invoke>sA< 
| `<invoke>`sB< 
| `<PARALLEL>` < <SEQUENCE> 
| `<invoke>`sC< 
| `<invoke>`sE< 
| `</SEQUENCE>` < <SEQUENCE> 
| `<invoke>`sF< 
| `<SELECT>` < <invoke> <sG< 
| `<invoke>`sH< 
| `</SELECT>` < <invoke> <sK< 
| `</SEQUENCE>` < <LOOP> 
| `<invoke>`sI< 
| `<invoke>`sJ< 
| `</LOOP>` < `</PARALLEL>` 
| `<invoke>`sL< |

| Figure 5.5 Composition tree construction |

The signature of service B is verified with signature of service A during service discovery and allocation. Similarly the service discovery process verifies signature of B to select service C, F and I since they are executed in parallel. The sequence of execution of services for the constructed composition starts with service A which is followed by B, parallel activity node (to run sub process) and finally L are executed in sequence. Parallel activity node of composition tree starts with services C, F and I simultaneously in different binding servers and these services take the result of service B.
The constructed composition tree is converted into symbols and is stored in Composition Sequence Registry (CSR). The CSR stores the composition sequence as symbolic notations. A symbolic notation contains information like level detail of a node, activity node type, service ID of binding service. The composition sequence conversion module scans the composition tree from root level activity node and converts the node information into symbolic notation. For instance, consider a business intelligence process for data federation containing four services to be executed in sequence as shown in Figure 5.6.

![Composition Tree and Symbolic Notation](image)

**Figure 5.6 Symbolic notation conversion**

The conversion process scans the tree from the root node and places a symbol ‘\( r \)' in symbolic notation and appends the label ‘SE’ to indicate sequence activity. This is appended with service IDs that are to be executed in sequence delimited by colons. Symbolic notation conversion for the above instance is shown in Figure 5.6. The composition sequence for the composition tree constructed in Figure 5.5 is shown in Table 5.2. The table also has the symbols used for conversion and its description.
Table 5.2 Contents of Composition sequence registry

<table>
<thead>
<tr>
<th>Composition sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSE(l1sA:l1sB:l1PL(l2SE(l3sC:l3sE):l2SE(l3sF:l3SL(l4sG:l4sH) :l3sK):l2LP(l3sI:l3sJ)):l1sL)</td>
</tr>
</tbody>
</table>

Where:
- **s** - Service
- **r** - Root level node
- **l**$_i$ - Level of node (Where i=1…n)
- **SE** - Sequence activity node (all services is to be executed linearly)
- **PL** - Parallel activity node (all services is to be executed simultaneously)
- **SL** - Select activity node (Subject to condition any one service is to be executed)
- **LP** - Loop activity node (all services is to be executed repeatedly)
- **:** - To concatenate sibling nodes
- **( )** - Enclose siblings

### 5.2.2.4 Service selection using multi utility ranking function

The service nodes of composition tree are analyzed and the required services are extracted from service registry. The selected services along with the existing services are ranked via ranking function.

The proposed architecture classifies service requests into three categories namely, (1) service request without QoS constraints, (2) service request with QoS constraints and (3) service request with QoS constraints and weight of QoS attributes as shown in Figure 5.7.
Figure 5.7 Classification of service request and respective ranking methodology

The QoS attributes are categorized into maximization attribute and minimization attributes. These are normalized for ranking the services. Table 5.3, shows the normalization followed by proposed service ranking module for respective service request. The service request does not contains any QoS requirement uses, linear or vector or Min-Max normalization as shown in equation 5.1a, 5.1b, 5.2, 5.3a and 5.3b respectively.

The user request with QoS is classified into two categories (i) user request with QoS and (ii) user request with QoS and Weight. It is ranked using distance vector normalization as shown in equation 5.4, and the later is ranked using the hybrid or multi-utility normalization equation 5.5. The multi-utility equation normalize the QoS attribute values and calculate the score of individual attributes using frequency of recurrence of service, advertised QoS value, perceived QoS value and weight of attribute.
Table 5.3 Normalization Methodologies for respective user request

<table>
<thead>
<tr>
<th>User Request</th>
<th>Methodology</th>
<th>Formulas Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without QoS</td>
<td>Linear Normalization</td>
<td>( Q_{\text{new}}(A_i) = \frac{A_{i\text{min}}}{A_i} ) (5.1a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Q_{\text{new}}(A_i) = \frac{A_{i\text{max}}}{A_i} ) (5.1b)</td>
</tr>
<tr>
<td></td>
<td>Vector Normalization</td>
<td>( Q_{\text{new}}(A_i) = \frac{1}{\sqrt{\sum_{i=1}^{n} A_i^2}} ) (5.2)</td>
</tr>
<tr>
<td></td>
<td>Max-Min Normalization</td>
<td>( Q_{\text{new}}(A_i) = \frac{A_{i\text{max}} - A_i}{qA_{i\text{max}} - A_{i\text{min}}} ) (5.3a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( Q_{\text{new}}(A_i) = \frac{A_i - A_{i\text{min}}}{A_{i\text{max}} - A_{i\text{min}}} ) (5.3b)</td>
</tr>
<tr>
<td>With QoS</td>
<td>Distance Vector normalization</td>
<td>( Q_{\text{new}}(A_i) = \sqrt{(A_i - th_i)^2} ) (5.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( A_i ) - ( i )th attribute of web service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( A_{i\text{min}} ) - minimum value of ( i )th attribute of web service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( A_{i\text{max}} ) - maximum value of ( i )th attribute of web service</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( th_i ) - threshold value of ( i )th attribute of web service</td>
</tr>
<tr>
<td>With QoS and weight</td>
<td>Multi-Utility Normalization</td>
<td>( Q_{\text{new}}(A_i) = \frac{1}{fre} \times fn(P(A_i)) + (w_i \times fn(R(A_i))) ) (5.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( fre ) - Number of times the service is used in particular duration (Eg. Per day).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( P(A_i) ) - Quality rating value of attribute provided by the service provider.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( R(A_i) ) - Average quality rating value of attribute perceived by the previous service instances measured using benchmark tools.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( w_i ) - Weight of the QoS attribute given by current service user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( fn ) - Any normalization function or distance vector function.</td>
</tr>
</tbody>
</table>
The ranking module uses sum of individual score of normalized QoS attributes to rank the discovered services. The QoS score for selected services is calculated using Equation 5.6.

\[ TQ(s_j) = \sum_{i=1}^{n} Q_{new}(A_i) \]  

(5.6)

The QoS attributes like average response time, latency are to be minimized and attributes like availability, throughput are to be maximized. The overall rating for service is calculated using multi utility equation - Equation 5.7 and Equation 5.8. The minimization attributes are converted into maximization attributes as discussed in section 4.2.6.

\[ MTQ(s_{j_{max}}) = \sum_{i=1}^{n} Q_{new}(A_{i_{max}}) \]  

(5.7)

\[ MTQ(s_{j_{min}}) = \sum_{i=1}^{n} Q_{new}(A_{i_{min}}) \]  

(5.8)

A service which, is ranked first is allocated to its respective leaf node. A runner service is stored temporarily for replacement of the allocated service. This is used in a situation of high load on the binding server that requires reselection of services and replanning of the service bindings (Rose 2014).

5.2.2.5 QoS Computing for Activity Nodes of Composition Tree

The activity node of a composition tree is classified into four categories namely, sequence, parallel, select and loop. Table 5.4 describes the QoS parameters for each type of activity (Debdoot et al 2008). The overall QoS of composite service is computed by summing of QoS values of all activity nodes involved in composition tree (Fabien et al 2008).
Table 5.4 QoS formula for activity nodes

<table>
<thead>
<tr>
<th>QoS Property</th>
<th>Sequence</th>
<th>Parallel</th>
<th>Select</th>
<th>Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Service Cost</strong></td>
<td>$\sum_{i=1}^{n} \text{sc}(sl_i)$</td>
<td>$\sum_{i=1}^{n} \text{sc}(sl_i)$</td>
<td>$n \times \text{sc}(sl_i) \times r(sl_i)$</td>
<td>$n \times \text{sc}(sl_i)$</td>
</tr>
<tr>
<td><strong>Average Response Time</strong></td>
<td>$\sum_{i=1}^{n} \text{rt}(sl_i)$</td>
<td>$\text{max}_{i=1}^{n} \text{rt}(sl_i)$</td>
<td>$\text{max}_{i=1}^{n} \text{rt}(sl_i)$</td>
<td>$n \times \text{rt}(sl_i)$</td>
</tr>
<tr>
<td><strong>Throughput</strong></td>
<td>$\text{min}_{i=1}^{n} \text{tpt}(sl_i)$</td>
<td>$\sum_{i=1}^{n} \text{tpt}(sl_i)$</td>
<td>$\sum_{i=1}^{n} \text{tpt}(sl_i)$</td>
<td>$\text{tpt}(sl_i)$</td>
</tr>
<tr>
<td><strong>Successability</strong></td>
<td>$\prod_{i=1}^{n} \text{suc}(sl_i)$</td>
<td>$\prod_{i=1}^{n} \text{suc}(sl_i)$</td>
<td>$\sum_{i=1}^{n} \text{suc}(sl_i) \times r(sl_i)$</td>
<td>$\text{suc}(sl_i)$</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td>$\prod_{i=1}^{n} \text{rbt}(sl_i)$</td>
<td>$\prod_{i=1}^{n} \text{rbt}(sl_i)$</td>
<td>$\sum_{i=1}^{n} \text{rbt}(sl_i) \times r(sl_i)$</td>
<td>$\text{rbt}(sl_i)$</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>$\sum_{i=1}^{n} \text{lat}(sl_i)$</td>
<td>$\text{max}_{i=1}^{n} \text{lat}(sl_i)$</td>
<td>$\sum_{i=1}^{n} \text{lat}(sl_i) \times r(sl_i)$</td>
<td>$n \times \text{lat}(sl_i)$</td>
</tr>
</tbody>
</table>

where,

$sl_i$: Service at level i.

$\text{sc}(sl_i)$: Service cost of level i service.

$\text{rt}(sl_i)$: Average response time of level i service.

$\text{tpt}(sl_i)$: Throughput of level i service.

$\text{rbt}(sl_i)$: Reliability of level i service.

$\text{suc}(sl_i)$: Successability of level i service.

$\text{lat}(sl_i)$: Latency of level i service.
\( n: \) Number of leaf nodes of activity node.

\( r(s_l): \) Rate at which the level \( l \) service is selected.

### 5.2.2.6 Alternative Service Selection

The services that are selected for composition are executed in respective server where its definition is available. A service binding is decided by the benefit function of binding server. The benefit function \( Bfn \) is calculated based on the current load on server as shown in equation 5.9. The existing service is replaced with the second best service in ranking process, if \( Bfn \) is greater than a threshold and is less than the time taken for replacement.

\[
Bfn = \frac{C_{max}(s) - C_{curr}(s)}{C_{max}(s)} \quad (5.9)
\]

where:

\( C_{max}(S) \) – Maximum load on server (100)

\( C_{curr}(S) \) – Current load on server (in PHP \texttt{sys_getloadavg()} )

Composition sequences available in composition sequence repository are further analyzed to extract interesting patterns to aid service composition.

### 5.2.3 An Efficient Service Composition using Frequent Service Sequence Patterns over Quality Aware Service Integration Architecture

The methodology of service composition proposed in this section initially builds a composition tree as discussed in section 5.2.2.4 and allocates
a service for service nodes using service discovery and ranking. The reusability is ensured by converting the composition sequences into symbolic notation and is stored in CSR. These symbolic notations are analyzed and FSSPs are extracted using FP growth algorithm explained in Section 5.2.3.1. The service selection for composite service begins with searching FSSP repository for allocating service to respective service nodes of composition tree.

5.2.3.1 Frequent service sequence pattern extraction (FSSP)

The stored symbolic notations are analyzed and FSSPs extraction is accomplished using Frequent Pattern (FP) growth algorithm (Haixun & Chang-Shing 2002, Jiawei & Micheline 2006). Consider the Table 5.5 that contains symbolic notation of composition sequences of various composite services for the business intelligence process data federation.

**Table 5.5 FSSPs extraction using FP growth for Data Federation**

<table>
<thead>
<tr>
<th>Symbolic notation of various Composite Service Sequence</th>
<th>Extracted &amp; Rearranged Service Sequence (based on frequency of occurrence of service)</th>
<th>After truncation (Minimum support = 3)</th>
<th>FSSPs (Minimum support = 3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>rSE(l1s1: l1s2: l1s5)</td>
<td>l1s2, l1s1, l1s5</td>
<td>l1s2, l1s1</td>
<td>(l1s1, l1s2)</td>
</tr>
<tr>
<td>rSE(l1s2: l1s4)</td>
<td>l1s2, l1s4</td>
<td>l1s2</td>
<td>(l1s2, l1s3)</td>
</tr>
<tr>
<td>rSE(l1s2: l1s3)</td>
<td>l1s2, l1s3</td>
<td>l1s2, l1s3</td>
<td></td>
</tr>
<tr>
<td>rSE(l1s1: l1s2: l1s4)</td>
<td>l1s2, l1s1, l1s4</td>
<td>l1s2, l1s1</td>
<td></td>
</tr>
<tr>
<td>rSE(l1s1: l1s3)</td>
<td>l1s1, l1s3</td>
<td>l1s1, l1s3</td>
<td></td>
</tr>
<tr>
<td>rSE(l1s2: l1s3)</td>
<td>l1s2, l1s3</td>
<td>l1s2, l1s3</td>
<td></td>
</tr>
<tr>
<td>rSE(l1s1: l1s3)</td>
<td>l1s1, l1s3</td>
<td>l1s1, l1s3</td>
<td></td>
</tr>
<tr>
<td>rSE(l1s1: l1s2: l1s3: l1s5)</td>
<td>l1s2,l1s1,l1s3, l1s5</td>
<td>l1s2,l1s1,l1s3</td>
<td></td>
</tr>
<tr>
<td>rSE(l1s1: l1s2: l1s3)</td>
<td>l1s2,l1s1, l1s3</td>
<td>l1s2,l1s1,l1s3</td>
<td></td>
</tr>
</tbody>
</table>

s1 - Service performs User Registration.
s2 - Service performs data extraction.
s3 - Service perform data transformation.
s4 and s5 – Services perform loading.
The first column shows the service sequence from CSR with the user request data federation. The service sequences from different composite service sequence are extracted. The extraction process scans each activity node and collection of service retrieved to create separate sequence. Here service sequence $l_1s1$, $l_1s2$, $l_1s5$ is extracted from composition sequence $rSE(l1s1: l1s2:l1S5)$. The extracted services are rearranged based on their frequency of occurrence to construct FP growth tree. In the above instance, the frequency of $l1s2$ is relatively higher than the other services and so $l1s2$ is placed first. The services that do not satisfy the minimum threshold are pruned from the list. From FP growth tree, the FSSPs with Minimum Support as 3 are extracted. The extracted patterns are stored in FSSP repository along with user request as shown in Table 5.6. In FSSPs, the services are arranged based on original composite service sequence in order to improve the service search and allocation process.

Table 5.6 Contents of FSSP repository

<table>
<thead>
<tr>
<th>Input Query</th>
<th>FSSPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Federation</td>
<td>$l1s1,l1s2$</td>
</tr>
<tr>
<td>Data Federation with transformation</td>
<td>$l1s2,l1s3$</td>
</tr>
</tbody>
</table>

Consider a new request for data federation. The service nodes of composition tree are allocated with the two frequently used services $l1s1$, $l1s2$ for respective nodes using FSSPs. For the remaining nodes the required services are selected from service repository. Similarly for the composition sequence $rSE(l1sA:l1sB:l1PL(l2SE(l3sC:l3sE):l2SE(l3sF :l3SL(l4sG: l4sH): l3sK): l2LP(l3sI: l3sJ)):l1sL)$ of composition tree shown in Figure 5.4b, the service nodes A, C, L are allocated using FSSP $l3sU1024$, $l3sU0047$ and $l1sU0569$ (where $U1024$, $U0047$ and $U0569$ are service ID assigned by proposed architecture during FAT construction) respectively.
The symbolic notation and patterns contained in SSR and FSSPR respectively are replicated in various servers located at different geographical locations to ensure availability. The contents of SSR and FSSPR are hashed using Secured Hash Algorithm (SHA-2) and stored to check consistency. The SHA-2 generates 512 bits message digest.

The stored contents are replicated in multiple locations to permit more users to access the contents for service composition. The consistency of replicated contents is verified using this message digest. The updated contents are pushed to other replicated locations frequently.

5.2.3.2 Service selection and allocation for composite service

The services for a service node of composition tree are selected and allocated using FSSP extracted from FSSP repository that satisfies a user request. This process completely or partially allocates service for service nodes of the composition tree. The unallocated service nodes of partially filled composition tree are allocated using services discovery. The steps involved in proposed service selection process are shown in Figure 5.8. The service allocation begins with extraction and allocation of frequently used services from FSSPs. The services for unallocated service nodes are allocated using service discovery process and multi-utility ranking with FAT, SIR and ISA.
Figure 5.8 Service selection and allocation

5.3. RESULTS AND DISCUSSION

The proposed QASIA is experimented by implementing service as shown in Table 5.7. Experimental data is taken from Quality of Web Services [QWS] Dataset with 2500 services (Mohammad & Thomass 2009). This QWS data set contains simple service with 9 QoS attribute values.
Table 5.7 Services implemented for experimentation

<table>
<thead>
<tr>
<th>Name of Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Composition Tree Construction service</td>
<td>Constructs Composition tree using BPEL description.</td>
</tr>
<tr>
<td>FP Growth service</td>
<td>Extracts the FSSP from service sequence repository.</td>
</tr>
<tr>
<td>Service selection and allocation service</td>
<td>Two service selection and allocation phases are performed.</td>
</tr>
<tr>
<td>Control Service</td>
<td>Controls sequence of execution of all mutually exclusive services available in orchestrated code.</td>
</tr>
<tr>
<td>Ranking Service (QoS Computing)</td>
<td>Ranks the selected services using Multi-Utility Ranking</td>
</tr>
</tbody>
</table>

For experimentation, synthetic BI composition plan is created that contains all four activities and 15 services. This BI plan performs various operations of data preprocessing, data integration and association rules extraction. The composition tree construction process constructs the tree and allocates services for service nodes using FSSP contents and service discovery.

The synthetic FSSP repository is constructed to store frequent patterns. This is populated with FSSP which contains 5 frequently used service nodes for the above BI composition plan. The remaining service nodes are allocated using service discovery and selection process.

The service selection process is initiated with 5 compliant services with varying QoS attributes in each class of 15 nodes. The time taken to allocate service for all nodes of composition tree using the proposed system is compared with the conventional service allocation system without FSSP. The above experiment is repeated by increasing the compliant service for each
node by 5 till each class contains maximum of 25 compliant services. The linked chart of conventional service selection without FSSP and proposed service allocation methodology without alternative service replacement and with alternative service replacement are shown in Figure 5.9. The service allocation time of the proposed system is 32.7% less than that of the conventional service composition system that discovers service from service repository.

![Service allocation time chart](chart.png)

**Figure 5.9 Service allocation time**

The experimental results show that the proposed QoS aware service composition using symbolic notation methodology is significantly better than the conventional methodology since the frequently used services are allocated from FSSP.

### 5.4. BENEFITS OF SERVICE COMPOSITION USING FSSPS

- This proposed dynamic service composition methodology allocates better quality and frequently used service for respective service nodes of composite service.
• The proposed method reduces the service allocation time by 32.7%.

• As the frequently used services increases, more number of services are allocated from FSSPs for recurring request, reducing service discovery and allocation time.

• Replicated FSSPs ensure availability, swift response and reusability

5.5 SUMMARY

The proposed QASIA extends the infrastructure of conventional web service architecture to ensure reusability of composition information. The symbolic representation of composition tree is efficient in terms of space utilization compared to XML representation. The FSSPs extracted from symbolic notations are used for service allocation. The replicated contents of CSR and FSSPR ensure reusability and availability.