Providing efficient quality web services as per the consumer requirement by composing web service on demand

Raghu Rajalingam¹, R Prabhu Doss²

¹PG Scholar, Dept of Computer Science and Engineering, Prist University, Tanjore, Tamilnadu, INDIA.
²Asst Professor, Dept of Computer Science and Engineering, Prist University, Tanjore, Tamilnadu, INDIA

Abstract: In today’s quick technology and customer need changes everyone wants a better quality of IT services wish to consume. Web services here referred as IT services provided by 3rd parties known as service provider. Present days no organization wants to invest heavily on IT services or IT infrastructure. As it is becoming more expensive to maintain it. Every one cost efficient and quick to integrate their IT requirement moving towards Web service enabled IT solutions. Everyone started focusing on integrating the existing IT solution with in organization or externally available IT solution. In this space web services has dominating the software industry. Every one focusing converting the existing solution to some form web services and offering the services to customer. The existing technology of web services are extended to completely customize the way services are composed as per customer requirement so that it will add or provide value as well as the maximum benefit to the customer. Dynamically composing the web services based on the user constraint and providing services to customer always challenging. The proposed paper focus on to addressing how to compose web services based on the user constraint dynamically. The proposal is win-win for both service provider as well as the service consumer.

Keywords: Service oriented computing; composition of services; dynamic composition; UDDI registry; web services.

I. Introduction

In today’s fast moving technology world everyone wants to have better quality of service whatever they want to use with their constraints addressed. Here user constraints vary from user to user and it cannot be same for everyone. User wants to use a better reliable service or user wants to use less expensive services or with multiple constraints such as less expensive with more reliable service constraints. Currently lots of research is focused on in this area how to compose web services dynamically to address user constraints. Composition of web services plays a critical role in case of B2B and integration of enterprise application. Here some time composition of web service may be a single service or it could be more than one dependent different web service to complete the user request. Most of the time service composition consumes lots of development time as well as to create new application. Now there is another parameter is added called user constraint and this will add more time to the existing problem. The problem with current scenario is Service Consumer (SC) and Service Provider (SP) get in to legal agreement for consuming and providing service for a pre-determined duration such as 1 years or 2 year time frame. SC has to use the services offered by the SP with whom he has a legal agreement for that duration whether the services offered by him is good or bad. Imagine if services offered by the SP are really bad or not meeting the expectation of the SC, then he has to terminate the agreement to find better SP or give feed back to the SP that he is not happy with his service. Here SC’s are forced to use the available services and SP dominate with their services without any improvement. SC doesn’t have a much choice as there are few SP’s and there is no much competition to improve the quality of the services. How about provide a multiple option’s to SC’s so that the overall service delivery quality will improve. SC’s will choose the service only meet their explicitly expressed constraints. Here SC’s are not tied to any SP’s and they can choose any service as per their requirement as well as their constraints. This will create a healthy competition among the SP’s to provide better quality of service. Following are some of issues need to be taken care.

- Web services are dynamically created and updated so the decision should be taken at execution time and based on recent information.
- How to access easily the different web services provided by the different service provider with different conceptual data model without any technical difficulties.
- How to ensure the access of the web services allowed only for the authorized user. It should not be the case where everyone accesses all web services.

Web services can be categorized in two ways on the basis of their functionality.

- Semantic annotation describes the functionality of web service and
- Functional annotation describes how it performs its functionality.
WSDL (Web Services Description Language) is used for specification of messages that are used for communication between service providers and consumer. Web services can be composed in two ways. One is static web service composition and the other is automated/dynamic web service composition. In static web service composition, web services are manually composed, i.e., each web service is executed sequentially in order one by one to fulfill the service consumer request. Static web service composition is a time-consuming and tedious task. This requires lots of developer effort to ensure services are properly composed. In case of automated web service composition, a generic framework with intelligence is used to compose web services which may internally have many web services to fulfill the service consumer request. From the service consumer point of view, it is considered as a single service. Web services are composed either centralized data flow or decentralized data flow approaches. In case of dynamic web services composition, both have advantages and some limitations. The limitation of centralized data flow is that all component services must pass through a composite service.

II. Preliminaries

Below section provides some insight about the web services composition, automated web services composition and actors involved in dynamic web services composition.

A. Web Services Composition

Web services are distributed applications. The main advantage of web services over other techniques is that web services can be dynamically discovered and invoked on demand, unlike other applications in which static binding is required before execution and discovery. Semantic and ontological concepts have been introduced to dynamically compose web services in which clients invoke web services by dynamically composing it without any prior knowledge of web services. Semantic and ontological techniques are used to dynamically discover and compose at the same time (run time).

B. Automated Web services composition

The automated web service composition methods generate the request/response automatically. Most of these methods are based on AI planning. First request goes to Translator which performs translation from external form to a form used by system, and then the services are selected from repositories that meet user criteria. Now the Process Generator composes these services. If there is more than one composite service that meets user criteria then Evaluator evaluates them and returns the best selected service to Execution Engine. The results are returned to clients (Requester). There should be well-defined methods and interfaces through which clients interact with the system and get the response.

III. Proposed Solution

Here we are proposing a framework to address composing the web service dynamically based on the user context as well as with user constraints.

1. **User Constraint Query Builder** – User constraint query builder will construct the user query based on the user-specific constraint for searching the web services from WS Meta Data server.
2. **Service Requester** – Constructs the user request object and submits the request to WS Meta data server.
3. **WS Meta data server** – All the Meta data related information is indexed for the faster retrieval of the search data.
4. **WS – Composition** – Constructs the WS – Client dynamically based on the user chosen web service.
5. **WS- Instance cache** – Is an in memory db to persist the instance of the frequently accessed WS instance.
6. **Service Repository** – Master DB of all the WS registered by the service provider.
7. **Execution engine** – A common framework to execute the WS calls.
8. **Service Rating engine** – Automatically compute the service Rating based on request/response time, QOS etc.
9. **Service Recommendation engine** – Based on the user query service recommendation engine will recommend the service for his query.
10. **Response** – A common response object for the WS execution request.

### IV. Proposed Technique

#### A. Methodology
The methodology of proposed model is given as:

1. Web services Meta data are indexed for the faster retrieval during the search operation.
2. The web services are registered in registries.
3. Service requester send request for service.
4. Translator converts the query into a form used by internal system.
5. The request arrives at composition module. Matching Engine checks for the requested service from WSDBs. If it finds the desired interface base service composition then it sends results to Evaluator.
6. Evaluator evaluates these selected web services in two steps. In first step it evaluates the web services on the basis of interface based search, whereas in second step it performs the evaluation on basis of functionality based rule. After evaluation it sends selected services to composer. The purpose of composer is to compose these component web services. Multiple WSDBs are introduced, so that if one goes down then we can make use of other databases. A timestamp (aging) is maintained with each URI in WSDB. If the request arrives before the expiration of that time then it looks for the service in WSDB.
7. If Matching Engine does not find requested service composition from web services database then it start searching from web.
8. The web services are searched from multiple registries and results are returned to Evaluator. Matching Engine also saves their references in WSDB with aging factor. The purpose of aging is that it maintains the updated information about web services as the contents are refreshed each time when aging time expires.
9. Evaluator evaluates these web services based on interface based and functionality based rules.
10. Composer composes these evaluated resultant services and sends result to Execution Engine. Execution Engine executes these web services and through translator results are sent back to requester.
11. After the successful/Failure transaction the service rating will be computed and persisted in the DB.

Later user can use rating as one of the constraint for his web service search.

#### B. Pseudo Code

**Algorithm:** Dynamic Web services composition based on user constraints.

**Input:** Request for web service

**Output:** Composed service

Web services with their constraints data are ingested in WS-Meta data server;
User enters request for web service along with their constraints;
WS-Query builder translates the web service request;
Web Service request will be submitted to the WS-Meta data server;
WS-Meta data server search services for the submitted user query;
WS-Meta data server returns the matching services as per the user ws query;
User selects and submits web service matching his requirement to WS-Composition Engine.
WS-Composition search for the submitted web service from WS-Instance cache;
If selected WS instance is available from WS-Instance cache then
Return instance to Execution engine.
If selected WS instance is not available from WS-Instance cache then
Create a new instance and add to the WS-Instance cache.
Return instance to Execution engine.
Execution engine will execute the selected web service and return the result;

Here we have query builder using which user can specify user constraints is. Based on the query a search request will be submitted to WS-Meta data server. WS-Meta data server will return the search result based on
the user constraint. If any record match with the user query result will be returned to the user. User with the returned list of web service matching his constraint can choose one and complete his operation. When user chooses a specific service matches the constraint then a search is done in WS-Instance cache is there already an instance is available. If no instance is available then a query will be sent to service repository to ensure that service still registered and available. Sometime services are registered for a short time and later removed from the service registre. To avoid such problem it is always good to make check before composing or executing the web service. Core component of this framework is WS-Meta data server. The Meta data server contains only the Meta data of the web services along with use full information such as pricing, availability, reliability, user rating, is partner etc. As these information may not available in service registry. Service registry may contain service name, service provider and name space etc. We are creating an index server which very fast in retrieving the required data based on the user query and it contains the Meta data of the services. The web services information along with the constraints are ingested in the Meta data and frequently kept updated as a when data changes. Sometime the data contained in Meta data server will obsolete as this is not the master database for the web services. This is a secondary server for fast searching data. Service registry will be the master copy of web services. WS-Instance cache is another important component as it contains the frequently used web services to speed up the web service composition and execution. There will be few services which will be used frequently by the user. We cache such service instances in WS-Instance cache so that in future if the same web service requested then it will retrieve from the cache instead of creating new one. If referred web service instance is not available in WS-Instance cache then retrieve the instance and execute the web service call. If it is not available then create the instance and add to the WS-Instance cache. This approach used to increase the speed of web service execution.

V. Conclusion

With dynamic composition of the web services based on the user constraints always ensures better quality of the services rendered to the service consumer. By this approach it will create a healthy eco system among the service provider to provide better quality service.

References

[1]. A Constraint-Based Approach to Horizontal WebService Composition, Ahlem Ben Hassine, Shigee Matsubara and Toru Ishida.3


