Dynamic Web Service Composition Based on Operation Flow Semantics

Demian Antony D'Mello
Department of Computer Science & engineering
St. Joseph Engineering College Mangalore
Karnataka, INDIA – 575 028

Ananthanarayana V S
Department of Information Technology
National Institute of Technology Karnataka
INDIA – 575 025

ABSTRACT
Dynamic Web service composition is a process of building a new value added service using available services to satisfy the requester's complex functional need. In this paper we propose the broker based architecture for dynamic Web service composition. The broker plays a major role in effective and efficient discovery of Web services for the individual tasks of the complex need. The broker maintains flow knowledge for the composition, which stores the dependency among the Web service operations and their input, output parameters. For the given complex requirements, the broker first generates the abstract composition plan and discovers the possible candidate Web services to each task of the abstract composition plan. The abstract composition plan is further refined based on the Message Exchange Patterns (MEP), input and output parameters of the candidate Web services to produce refined composition plan involving Web service operations with preferable execution flow. The refined composition plan is then transferred to generic service provider to generate executable composition plan based on the requester's input/output requirements & preferences. The proposed effective Web service discovery and composition mechanism is defined based on the concept of functional semantics and flow semantics of Web service operations.

General Terms
Distributed Computing, Web Technology, E-Commerce

Keywords
Web Services, Compositions, Flow Semantics, Discovery

1. INTRODUCTION
The success of Web service technology lies in the effective & efficient dynamic discovery and compositions of advertised Web services. A Web service is an interface, which describes a collection of operations that are network accessible through standardized XML messaging [1]. At present, the Web service architecture is based on the interactions between three roles i.e. service provider, service registry and service requester. The interactions among them involve publish, find and bind operations [1]. Web service discovery is the mechanism, which facilitates the requester, to gain an access to Web service descriptions that satisfy his functional needs. The dynamic composition process assembles the available services to build new service to satisfy the requester's complex demand. UDDI [2] is the early initiative towards discovery, which facilitates both keyword and category based matching. The main drawback of such mechanism is that, it is too syntactic and provides no support for dynamic composition of Web services. There is a need to describe the Web services in a natural way to improve the effectiveness of the discovery and composition mechanism. The conceptual Web service architecture [1] involving service registry (UDDI) does not provide infrastructure or the mechanism for effective & efficient dynamic Web service discovery and composition. To enable effective and efficient dynamic Web service discovery and composition, the existing architecture has to be augmented by introducing new roles and new operations.

1.1 Literature Survey and Brief Review
In literature, different architectures are proposed for dynamic Web service discovery and composition. We classify the architectures based on the storage of Web service information and processing component of discovery and composition as follows: *agent based architectures*, *broker based architectures*, *peer-to-peer architectures* and *hybrid architectures*. In agent based Web service architectures [4], the service agents are used to initiate the request, terminate the request and to process the messages. In the broker based architectures [5] [6], the broker is used for the optimal selection of Web services for the composition plan towards dynamic integration of QoS-aware Web services with end-to-end QoS constraints. The peer-to-peer composition architecture is an orchestration model which is defined based on the peer-to-peer interactions between software components hosted by the providers participating in the composition [7]. Such architectures are also capable of composing Web services across wide area networks with the service composition based on the interface idea integrated with Peer to Peer technologies [8]. In hybrid architectures, along with service registry, other roles (for example third party provider in [9] and composition engine in [10]) are defined for the abstract composition plan generation and execution.

A variety of techniques have been proposed in literature which integrates existing services based on several pieces of information. Most of the composition strategies are defined based on the output and input matching of available Web services [11]. Such composition mechanisms use chain [12], graph (tree) [13], vector [14] data structures for the dynamic composition of concrete services. The main problem with this approach is that, the repository search time is quite more for the matching of output parameters with the inputs. Also domain ontology has to be used for effective matchmaking. The requester's constraints are useful to build composition involving concrete Web services [15]. The atomic or composite Web services are composed to satisfy the complex demand based on business rules or policy information [16]. The context

(process/user) or view also plays a major role in effective Web service composition [17]. The goal [18], service behavior [19], user satisfaction and interaction patterns [20] guide the effective dynamic Web service composition.

The Web service composition can be modeled in different ways. The Petri nets [21], Labeled behavior diagrams (LBD) [22], Mathematical model [23], UML Activity [24] and state chart diagrams [25], Workflow Model [26] and Finite automata [27] are the major modeling methods used to represent the composite Web service. In this paper, we propose a methodology to build an abstract composition plan which is defined based on the flow graph concept. The abstract composition plan (graph) is then refined by selecting suitable candidate Web services and their operations. The paper also proposes the broker based architecture for the dynamic Web service composition which facilitates the requester to discover the suitable Web service(s) for his simple or complex functional need.

The rest of the paper is organized as follows. The next subsection gives definitions for the terminology used throughout the paper. In section 2, we present a model to describe the Web service operation functionality and flow. Section 3 presents the broker based architecture for the discovery and composition. Section 4 presents the Web service discovery and composition mechanism. In section 5, we discuss prototype implementation and experiment results. Section 6 draws the conclusions.

1.2 Terminology Used in the Paper

Here the authors present definitions of terms used throughout the discussion.

Simple Request. A simple request for the Web service contains a single operation or functionality to be executed by the service provider. For example, reserve train ticket is a simple request.

Complex Request. A complex request for the Web service contains a set of related or unrelated multiple operations (functionality) to be executed by the service provider. For example, arrange tour with operations like reserve flight ticket, reserve hotel room and book taxi is a complex request.

Atomic or Primitive Web Service. The atomic Web service is a well defined network accessible application interface which is a collection of related operations implemented by single provider.

Composite Web Service. A composite Web service is collection of operations/activities where each activity is offered or implemented by different service providers. The composite service provider may offer number of activities/operations by reusing the existing services available over the Web.

Core Operation. A core operation of Web service is an important operation of Web service. A Web service may contain any number of core operations. For example, the travel service may contain core operations like reserve train ticket and cancel train ticket.

Supplementary Operation. A supplementary operation is a Web service operation which provides support to the core operation. The supplementary operations indirectly add the value to the core services i.e. these operations support the execution of core operations. For example, the operation check train ticket availability is a supplementary operation.

Abstract Operation. The functionality of an operation described in WSDL document of Web service during service advertisement. This represents a set of concrete operations supported by the advertised Web service.

Virtual Operation. A single, compact and complete description of functionally similar abstract operations is referred to as virtual operation.

Web Service Composition Problem. Given a set of available Web services (atomic or composite), providing single or multiple operations (activities), create a new Web service by combining the available services (service operations) that realizes the complex service request of the requester.

2. THE SEMANTIC MODEL FOR WEB SERVICE DESCRIPTION, DISCOVERY AND COMPOSITIONS

The effective Web service discovery and composition enforces the service providers to follow the functional semantics and flow semantics during service advertisements. The service requesters also need to use the functional semantics while describing the service request. In this section, we briefly explain the concept of functional semantics and flow semantics for Web services.

2.1 Functional Semantics for Web Service Operations

The functional semantics approach [28] adopts the natural way of expressing the functionality of Web service operations i.e. abstract operations of Web services are expressed in terms of actions, objects, qualifiers and nouns. Thus functionality (OP_{Desc}) of an abstract operation can be described in the following *three* formats.

(i) OP_{Desc} = {(Generic Action) (Qualifier)* (Domain Object)$^+$ (Domain Noun)}

(ii) OP_{Desc}= {(Specific Action) (Qualifier)* (Domain Object)$^+$}

(iii) OP_{Desc} = {(Qualifier)* (Domain Object)$^+$ Action Noun}

All abstract operation descriptions are preprocessed before mapping them to virtual operations [28]. The following rules guide the preprocessing of abstract operation description.

Rule 1. If the action noun is present along with the generic action then the generic action is replaced by specific action which is related to the action noun and the action noun is eliminated from the description.

Rule 2. If the action noun is found in the operation description with no generic or specific action then the specific action of the action noun is considered eliminating the action noun.

As an illustration, consider the abstract operation description "prime number generation". This description is transformed into "generate prime number" which can be considered as a virtual operation.

2.2 Flow Semantics for Web Service Operations

The Web service can be viewed as collection of interdependent or independent operations. For example, the travel Web service may offer its services though the following three interdependent

operations namely (i) check train ticket availability (2) make train ticket reservation (iii) cancel train ticket. We define a graph structure called *Operation Dependency Graph (ODG)* which represents the possible order of execution of operations of a Web service.

Operation Dependency Graph (ODG). Operation dependency graph is a directed acyclic graph with finite vertices which represent the number of operations of a Web service. A directed edge (u v) between any two vertices u and v indicate the possible order of execution such that, the activity v is executed after successful execution of activity u i.e. the activity v is dependent on activity u.

The authors define *two* forms of activity dependencies called *weak dependency* and *strong dependency*. The weak dependency is found between any two supplementary operations or between supplementary and core operation. The strong dependency is found between any two core operations or between core and supplementary operation. As an illustration, consider the Web service "Tour Arrangement Service" involving *five* core operations. The possible order of execution of activities can be modeled using ODG as shown in Figure 1. The dotted lines in ODG represent the weak dependency and thick circles represent the core operations of Web service.

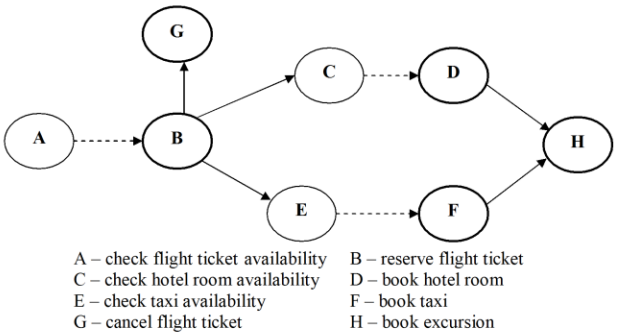

Fig 1: Operation Dependency Graph (ODG) of Web Service

The ODG of all published Web services are represented using flow knowledge as follows.

Operation Predecessor List (OPL). Operation predecessor list of an operation O_P is a sorted list containing operations for which O_P is dependent on them i.e. list of operations which are predecessors of O_P in the ODG.

Operation Dependency List (ODL). Operation dependency list is a sorted list with finite elements where each element contains *two* fields namely *operation identifier* and *opl-link*; where, opl-link is a pointer to OPL of an operation.

Figure 3 shows the snapshot of ODL after advertisement of Web services. The operations Op_1 to Op_{10} of *four* Web services are found in the ODG (Figure 2) of published Web services.

2.3 Extension of WSDL 2.0 Document

We extend the WSDL 2.0 [3] structure to publish the Web services with functional semantics and flow semantics as follows. We select the *documentation* element of the WSDL to insert the information which is necessary for effective service discovery and composition. We define a new tag called *operationDesc* to insert the functional semantics of all abstract operations present in the Web service. We also define *flowDesc* element to insert operation dependencies. The extended WSDL document with functional and flow semantics for the "train reservation service" is depicted in Figure 4.

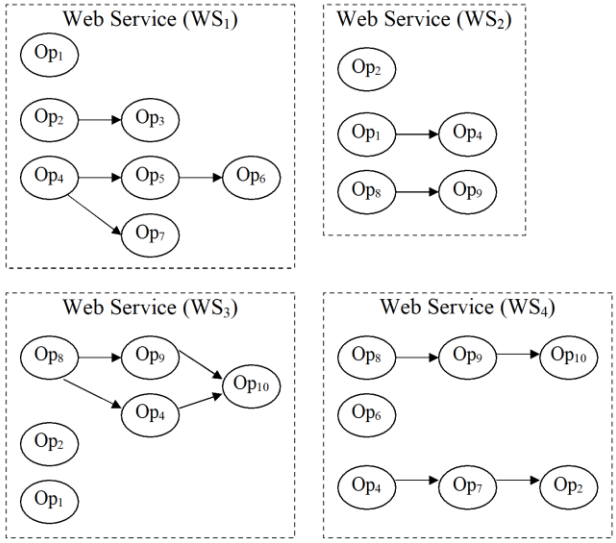

Fig 2: Operation Flow Structure of Published Web Services

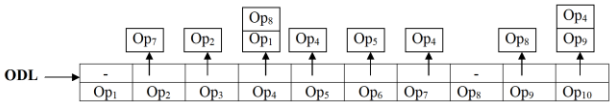

Fig 3: Operation Dependency List (ODL) of Web Services

3. THE ARCHITECTURE FOR DYNAMIC WEB SERVICE COMPOSITIONS

The authors propose broker based architecture for dynamic Web service discovery and composition by introducing *two* new roles to the conceptual architecture [1] with a few new operations between different architectural roles. The architecture involves a total of *five* roles. They are *Service provider*, *Service requester*, *Service composer (generic service provider)*, *Service registry* and the *Broker*.

The *register* operation is defined between the provider and broker. The provider registers service specific information including WSDL to the broker for Web service publishing. The *publish* operation is defined between the broker and service registry which saves the service binding and WSDL details into service registry. The *find service* operation is defined between the service requester and broker to obtain candidate Web services for a service request. The *execute composition* operation is defined between the broker and service composer in which the broker sends an abstract composition plan involving Web services and their operations. The *monitor functional knowledge* operation is defined between the generic service provider and broker in which the domain analyst monitors the functional knowledge updated by the service registrations. A variety of update operations are defined between internal

components of the broker. Figure 5 presents the broker based architecture depicting various architectural roles and operations.

```
<?xml version="1.0" encoding="utf-8" ?>
<description>
<documentation>
<operationDesc xmlns:xf = "http://www.w3.org/2001/XMLSchema-Instance"
               xf:schemaLocation= "http://www.nitk.ac.in/itdept/descSchema/desc.xsd"
               xmlns= "http://www.nitk.ac.in/itdept/descSchema">
        <operationList>
                <operation>
                        <operationName>Check-Train-Ticket</operationName>
                        <operationType>S</operationType>
                        <semantics>
                                <action>check</action><object>train</object>
                                <object>ticket</object><noun>availability</noun>
                        </semantics>
                        <input>date</input><input>number</input><input>class</input>
                        <output>date</output><output>time</output><output>class</output>
                        <output>status</output>
                </operation>
                <operation>
                        <operationName>Reserve-Train-Ticket</operationName>
                        <operationType>C</operationType>
                        <semantics>
                                <action>reserve</action><object>train</object>
                                <object>ticket</object>
                        </semantics>
                        <input>date</input><input>number</input><input>class</input>
                        <output>date</output><output>time</output><output>class</output>
                        <output>number</output>
                </operation>
                <operation>
                        <operationName>Cancel-Train-Ticket</operationName>
                        <operationType>C</operationType>
                        <semantics>
                                <action>cancel</action><object>train</object>
                                <object>ticket</object>
                        </semantics>
                        <input>number</input><input>date</input>
                        <output>status</output>
                </operation>
        </operationList>
        <information>
                <action name="reserve"><related>book</related></action>
                <object name="ticket"><related>seat</related></object>
        </information>
</operationDesc>
<flowDesc xmlns:xf = "http://www.w3.org/2001/XMLSchema-Instance"
          xf:schemaLocation= "http://www.nitk.ac.in/itdept/descSchema/flow.xsd"
          xmlns= "http://www.nitk.ac.in/itdept/flowSchema">
        <succ name="Reserve-Train-Ticket">
                <pred>Check-Train-Ticket</pred>
        </succ>
        <succ name="Cancel-Train-Ticket">
                <pred>Reserve-Train-Ticket</pred>
        </succ>
</flowDesc>
</documentation>
<types>…</types>
<interface>…</interface>
<binding url="www.indianrailways.com">…</binding>
<service name="RailwayReservation "></service>
</description>
```

Fig 4: Extended WSDL Document of Train Reservation Service

3.1 Architectural Roles and Operations

Service Provider. Service provider is a responsible and authentic business organization which registers its services with the broker. A provider is allowed to register multiple services into the service registry through the broker. On successful service registration, the broker returns the service key to the provider.

Service Requester. Service requester is either a business organization or a person who intends to utilize the services published by the provider. The service requester has to submit the service request to the broker for the service discovery.

Service registry. Service registry (e.g. UDDI) is a repository which stores the service deceptions including business details, service details and binding information. The service registry provides access to service information through interface operations. The service registry also saves the WSDL link of the published Web service.

Generic Service Provider (Service Composer). Service composer is a business organization which executes the requester's complex requirements if they are not satisfied by the available atomic or composite Web services. The service composer first generates the executable composition plan by refining the operation dependencies of abstract composition plan based on parameter constraints and then executes a set of Web service operations as defined by the composition flow.

Broker. Broker is a middleware which is responsible for service registration, publishing, discovery and composition plan generation. The broker is designed with major *five* architectural components. They are *Service publisher*, *Service discovery & composition plan generator*, *Service knowledge*, *functional knowledge* and *flow knowledge*.

3.2 Broker Components and their Functions

Service knowledge component of the broker is interlinked data structure which stores the abstract details of service i.e. operations supported by the Web service and their input/output details (discussed in next sub-section). The functional knowledge is the structure which represents actions, action nouns, qualifiers and domain objects of service domains. The detailed functional knowledge structure is found in [28]. The flow knowledge is the structure which represents the dependency among various operations supported by numerous Web services (refer section 2.2).

Service publisher component reads the Web service description from the provider and updates the functional knowledge, flow knowledge and service knowledge accordingly. The service discovery and composition plan generator uses the functional knowledge to map the requested abstract operations into virtual operations present in service knowledge. If all the requested operations are found in the single Web service then the Web service key is returned to the requester. If there exists no single Web service which fulfills the requested operations then the composition plan generator uses the flow knowledge to build abstract composition plan consisting of requested operations, supplementary operations and candidate Web services. The refined composition plan is then transferred to the generic service provider for the execution.

3.3 Extended Service Knowledge Structure

For the effective Web service composition, we define additional data structures called Input List (IL), Output List (OL) and Parameter List (PL) as follows.

1. Input List (IL). Input list is a dynamic sorted array with finite elements. Each element has two fields namely ws-id and op-id where, ws-id is Web service identifier and op-id is operation identifier. This list is sorted based on the Web service identifier.

2. Output List (OL). Output list is a dynamic sorted array with finite elements. Each element has two fields namely ws-id and op-id where, ws-id is Web service identifier and op-id is operation identifier.

3. Parameter List (PL). Parameter list is a dynamic sorted array with finite elements. Each element has four fields namely para-id, para-name, input-link and output-link. para-id is

unique identifier generated by the broker, para-name refers to operation parameter name, input-link refers to pointer to IL (Web service operations for which the parameter is input parameter) and output-link is a pointer to OL (Web service operations for which the parameter is input parameter).

Apart from these three interlinked structures, the service knowledge also consists of two more interlinked structures called Web Service List (WSL) and Service Operation Tree (SOT). The definitions of these structures are presented in [29].

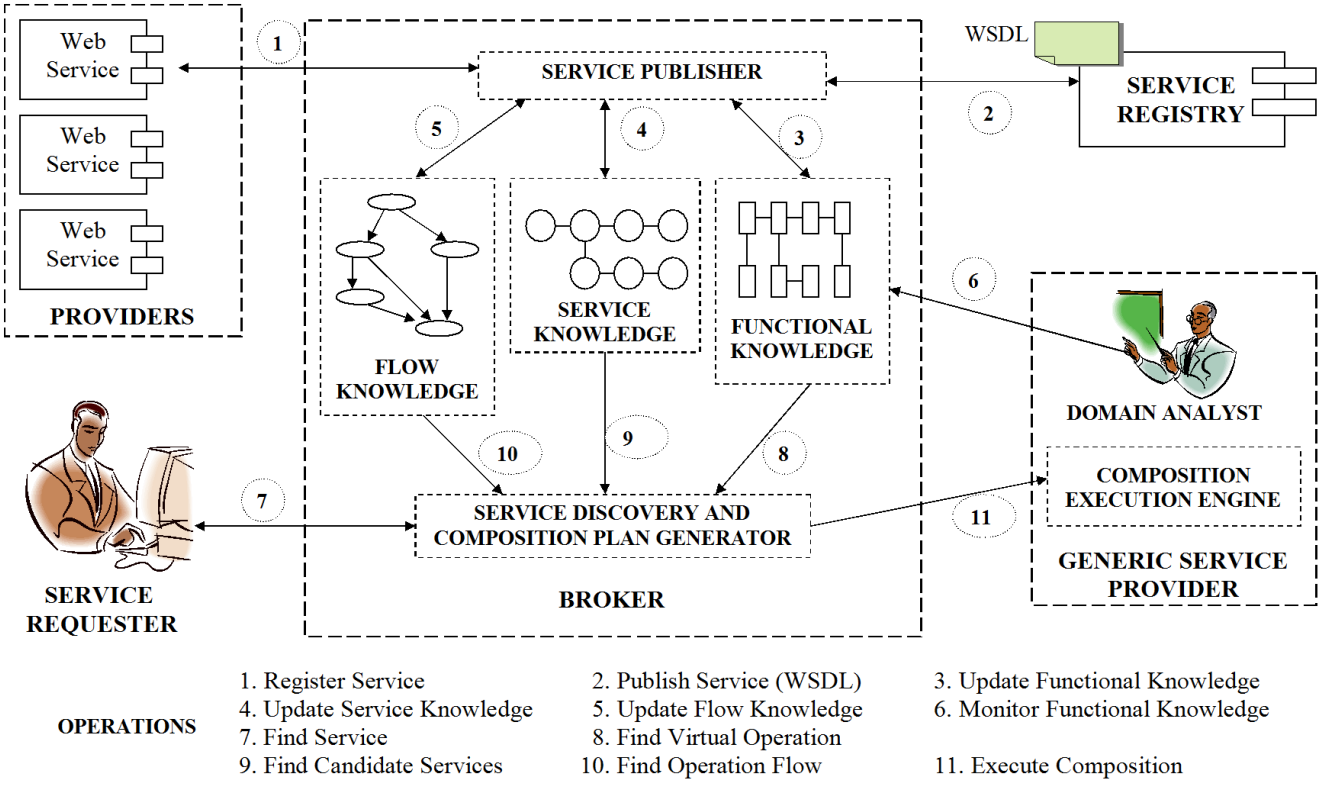

Fig 5: The Broker based Architecture for Dynamic Web Service Discovery and Composition

3.4 Assumptions

The proposed broker based architecture for efficient & effective Web service discovery and composition is designed based on the following assumptions.

1. The composite Web service provider has to advertise the individual activities of composite Web service as operations.
2. The provider of the Web service has to browse the functional knowledge before the service registration in order to use the existing functional knowledge or augment the functional knowledge with additional related action, object, noun and qualifier words.
3. Service provider has to use the formats of functional semantics [28] to describe the Web service operations.
4. While publishing operations on new domain object, the provider has to identify the object type.
5. The provider of the Web service has to supply related words for objects, actions, qualifiers, and nouns during Web service registrations to improve the effectiveness of discovery mechanism.
6. While describing the operation description the provider should provide the complete description (major domain objects along with sub- objects) of the functionality.
7. The provider of the Web service has to precisely identify the core and supplementary operations.
8. The provider and requesters have to avoid the use of plurals of domain object and action.
9. The provider should describe the input and output parameters of operations with generic parameter concepts.
10. The requester has to understand the request format for fruitful discovery results.

4. WEB SERVICE DISCOVRY AND COMPOSITION MECHANISM

In this section, the authors describe the Web service discovery and composition mechanism designed for the broker based Web service architecture.

4.1 Effective Web Service Discovery

The Web service discovery process for the simple or complex service request is described below.

1. The service request is preprocessed according to functional semantic rules to retrieve the functional requirement(s) of service request.
2. The action list, qualifier list (if required), object list and noun list (if required) of the functional knowledge are searched to get the corresponding identifiers. The non-availability of any identifier results in discovery failure.
3. After obtaining required identifier(s) from the functional knowledge, the operation patterns for each task of the service request are formed. After building the operation patterns, the patterns are searched in virtual operation list (VOL). If the pattern is found then the corresponding operation identifier is retrieved from the VOL otherwise, discovery failure is reported.
4. The abstract Web service information of all published Web services is searched by traversing SOT (Figure 6) for the requested operation identifier(s). The discovered Web services are stored against the requested operations.
5. The Web services found common in all requested operation (s) are selected as candidate Web services for the service discovery and are returned to the requester.

The absence of a common Web service for all requested operations triggers the composition mechanism which is presented in the next sub-section.

4.2 Web Service Composition Mechanism

The composition mechanism to generate abstract composition plan involving Web service operations for the complex service request is described as follows.

1. Initialize the ODG (adjacency list) as empty graph
2. For each requested operation (op-id) do the following.
   - Insert the op-id into ODG as a new node.
   - Search in ODL for the presence of op-id.

     If the op-id (say p_i) is found with empty OPL then the operation becomes the independent operation. If some operations (p_j) are found in OPL then do Step-3
3. For every operation (p_j) in OPL of p_i do the following
   - If p_j is present in request then, include the edge (p_j->p_i) in the resulting ODG (adjacency list) of composition plan. Repeat step 3 for p_j until p_i is reached or empty OPL is found or already visited operation is encountered.
   - If the p_j is not present in request and p_j is supplementary operation then insert new node (p_j) to ODG and insert the edge (p_j->p_i). Repeat step 3 for p_j until p_i is reached or empty OPL is found or already visited operation is encountered.
   - If p_j is not present in request and p_j is not supplementary operation and there exist predecessors p_k and p_m of p_j, such that, p_k is in request and p_k->p_m then, insert the edge (p_k->p_i).
4. The candidate Web services for all supplementary operations are now obtained by traversing SOT.

```
Algorithm WebServiceDiscovery
Input: A list of requested service operations
Output: Web service keys or failure report
Initialize Cand-List [] = NULL // Global list
Begin
   For each requested operation, obtain the operation identifier (op-id) from VOL
   If op-id is not found then
      Report "Discovery Failure" and Exit
   Else
      Sort the operations based on op-id
      Let Opr1, Opr2, ..OprM be the sorted list of operations
      For each Opri do
        Let T be the root of SOT, Node=child-link (T)
        Find-Candidates (SOT, Node, Opri)
        If (Cand-List=Φ) then
            Report "Discovery Failure" and Exit
        End If
   End If
End

Procedure Find-Candidates (node, opr)
Begin
   If (Node = NULL) then
      Return
   End If
   If (op-id (Node) = opr) then
      Traverse_Inorder(node, opr);
   Else
      Find_Cand(child-link(node), opr);
      Find_Cand(sibling-link(node), opr);
   End If
End

Procedure Traverse-Inorder(node, opr)
Begin
  Traverse the child branch until ws-link =NULL or sibling-link!=NULL
  Let X be the such node
  If (ws-link (X) !=NULL) then
    Traverse the WSL through ws-link of X
    Record the candidate Web services from WSL
  End If
  If (sibling-link (X) !=NULL) then
    Traverse-Inorder(sibling-link(X), opr);
  Traverse-Inorder(X, opr);
End
```

Fig 6: Discovering Web services from SOT & WSL

As an illustration, consider the service request involving *five* operations {Op_2, Op_4, Op_8, Op_9, Op_{10}} and *four* advertised Web services as in Figure 2. Table 1 shows the discovered Web services for the requested operations Op_2, Op_4, Op_8, Op_9 and Op_{10} through the Web service discovery algorithm (Figure 6).

Table 1. Operations and discovered Web Services

Operation	Discovered Web services
Op_2	WS_1, WS_2, WS_3, WS_4
Op_4	WS_1, WS_2, WS_3, WS_4
Op_8	WS_2, WS_4
Op_9	WS_2, WS_3, WS_4
Op_{10}	WS_3, WS_4

Observed that, no single Web service is found common to all requested operations. Thus, the broker generates the abstract composition plan which is shown in Figure 7.

4.3 Refining Abstract Composition Plan

The abstract composition plan is now refined in sequence based on the message exchange pattern (MEP) of Web service operations, core and supplementary operations, Input and Output parameters and quality of service (QoS) like reliability.

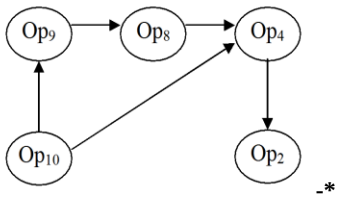

Fig 7: Abstract Composition Plan

Plan Refinement based on the Message Exchange Pattern

Let M+K be the nodes of abstract composition plan (M= number of requested core operations and K= number of supplementary operations explored), Let G be the ODG of abstract composition plan.

Step-1. For each node (S) in G, perform Step-2.

Step-2. For each directed edge from node S to node D (S→D) perform Step-3.

Step-3. Let C_S be the Web services selected at S and C_D be the Web services selected at D.

- Eliminate all the Web services at S having MEP other than {Out-Only, Out-In, In-Out and Robust-Out-Only}.
- Eliminate all the Web services at D having MEP other than {In-Only, In-Out, In-Optional-Out, Robust-In-Only and Out-In}.

Plan Refinement based on the Supplementary and Core operation Relationship

For every pair of nodes in ODG say, S and D with a dependency relation (S→D) or (D→S) where S represents supplementary operation and D is core operation, eliminate the Web services which are not common in S and D.

Plan Refinement based on the Input and Output Parameters

The parameter list, input list and output list of service knowledge are used for the plan refinement. Let M_O be the nodes of ODG with zero out-degree and M_I be the nodes with zero in-degree.

Step-1. For every node pair (X, Y) such that, X Є M_O and Y Є M_I perform step-2.

Step-2. Let C_X be the Web services selected at X and C_Y be the Web services selected at Y.
If there exists one pair of Web services (C_{Xi}, C_{Yj}) such that, any one output parameter of C_{Xi} is input parameter of C_{Yj} then, insert an edge between node X and node Y.

Plan Refinement based on the Quality of Service (QoS)

Let W be the number of Web services attached to any node of composition plan. The Web services with highest QoS score are retained for the individual activities of composition plan. Now the composition plan consists of nodes each having Web service operation in it.

After refinement of the abstract composition plan it is transferred to the generic service provider with plan identifier and requester identifier. The service composer transforms the refined abstract composition into executable composition plan involving concrete service operations. This transformation is performed based on the functional requirements defined on the input and output parameters of the abstract operations.

5. EXPERIMENTATION

The prototype of the proposed broker based Web service discovery and composition mechanism is implemented on the Windows XP platform using Microsoft Visual Studio .NET development environment and Microsoft visual C# as a programming language. The broker is designed and implemented as a standalone visual program which interacts with the provider and requester through different interface forms. The service repository is implemented as a Web service which in turn communicates with the SQL server 2000 database. The database table is created to store the information (including WSDL link) of all published Web services.

The requester of a Web service can submit the simple request as well as complex request through different interface forms. The requester can also browse the functional knowledge for the successful discovery of requested functionality. The requester can browse the service knowledge i.e. operations supported by the specific Web service. The prototype also displays the refined composition plan to the requester along with the generic provider key and composition plan key. The authorized provider of Web services is allowed to browse and augment the functional knowledge in order to improve the effectiveness of Web service discovery. The interface form is created for the provider to publish the Web service along with the WSDL. The service publisher component of the broker processes the WSDL to extract functional and flow information of operations after publishing the Web service into service registry.

The authors have conducted several experiments involving simple and complex service requests. We use a collection of 35 Web services having total of 60 distinct operations from XMethods service portal (http://www.xmethods.com) [30] and divide them into SIX categories. The Web service operations are published to the broker using functional semantics. The simple Web service requests are created according to the functional semantic rules. The Recall and Precision of discovery process are recorded. The recall of discovery is less than 100% as some Web service descriptions take multiple functional semantic representations. The precision is 100% provided both the provider and requester precisely follow the functional semantic rules. The service operation tree of published Web services also take less memory which results in 30% compactness (compression ratio). The complex service requests are also created to test the effectiveness of Web service composition in travel domain. We define few Web services in travel domain and register them into broker with functional semantics, flow semantics, input parameter concepts and output parameter concepts of operations. The empirical results proved the correctness of proposed composition mechanism defined on functional and flow semantics of web service operations.

6. EXPERIMENTATION

The dynamic Web service discovery enables the requester to consume the desired Web services. The service composition satisfies the requester's complex need by integrating available Web services. The functional semantics and flow semantics of

Web service operations enable the effective Web service discovery and composition mechanism. The service knowledge component of broker represents the abstract Web service information which facilitates the quick Web service discovery. The proposed broker based architecture for Web services facilitates effective & efficient Web service discovery and composition through functional and flow semantics of Web service operations. The paper also suggests the augmented WSDL 2.0 to enable semantics based discovery and composition. The empirical results prove the correctness of concepts proposed for the Web service discovery and composition.

7. REFERENCES

[1] H. Kreger, "Web Services Conceptual Architecture (WSCA 1.0)", Published May 2001, [online] Available: www.ibm.com/software/solutions/webservices/pdf/wsca.pdf, [visit: April 2007].

[2] Riegen, C.V. (Ed), 2002. *UDDI Version 2.03 Data Structure Reference* [online]. OASIS Open 2002-2003. Available from: http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm [Accessed 8 November 2007].

[3] David Booth and Canyang Kevin Liu, "Web Services Description Language (WSDL)" Version 2.0 Part 0: Primer, W3C Recommendation, Published: 26 June 2007, [online] Available: http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626, [visit: December 2008].

[4] Paul A. Buhler, Dominic Greenwood and George Weichhart, "A Multi-agent Web Service Composition Engine, Revisited", In Proceedings of the 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (CEC-EEE 2007), IEEE 2007.

[5] Tao Yu and Kwei-Jay Lin, "A Broker-Based Framework for QoS-Aware Web Service Composition", In Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service, 2005. EEE '05, pp. 22- 29, IEEE 2005.

[6] Liangzhao Zeng, Boualem Benatallah, Anne H.H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang, "QoS-Aware Middleware for Web Services Composition", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 30, NO. 5, pp. 311-327, MAY 2004, IEEE.

[7] Boualem Benatallah, Quan Z. Sheng and Marlon Dumas, "The Self-Serv Environment for Web Services Composition", IEEE INTERNET COMPUTING, JANUARY • FEBRUARY 2003, pp. 40-48, IEEE.

[8] LIU AnFeng, CHEN ZhiGang, HE Hui and GUI WeiHua, "Treenet:A Web Services Composition Model Based on Spanning tree", In Proceedings of the 2nd International Conference on Pervasive Computing and Applications, 2007 (ICPCA 2007), IEEE 2007.

[9] Rajesh Karunamurthy, Ferhat Khendek and Roch H.Glitho, "A Novel Business Model for Web Service Composition", In Proceedings of the IEEE International Conference on Services Computing (SCC'06), IEEE 2006.

[10] Maja Vukoví´c, Evangelos Kotsovinos and Peter Robinson, "An architecture for rapid, on-demand service composition", Journal of Service Oriented Computing and Applications -SOCA (2007) 1:pp. 197–212, Springer-Verlag London Limited 2007.

[11] Lukasz Juszczyk, Anton Michlmayr, Christian Platzer, Florian Rosenberg, Alexander Urbanec and Schahram Dustdar, "Large Scale Web Service Discovery and Composition using High Performance In-Memory Indexing", In Proceedings of the 9th IEEE International Conference on E-Commerce Technology and The 4th IEEE International Conferenceon Enterprise Computing, E-Commerce and E-Services(CEC-EEE 2007), IEEE 2007.

[12] Lian Li Ma Jun Chen ZhuMin and Song Ling, "An Efficient Algorithm for Web Services Composition with a Chain Data Structure", In Proceedings of the 2006 IEEE Asia-Pacific Conference on Services Computing (APSCC'06), IEEE 2006.

[13] Dong-Hoon Shin and Kyong-Ho Lee, "An Automated Composition of Information Web Services based on Functional Semantics", In Proceedings of the 2007 IEEE Congress on Services (SERVICES 2007), IEEE 2007.

[14] Buhwan Jeong, Hyunbo Cho, Boonserm Kulvatunyou and Albert Jones, "A Multi-Criteria Web Services Composition Problem", In Proceedings of the IEEE International Conference on Information Reuse and Integration, 2007 (IRI 2007), pp. 379-384, IEEE 2007.

[15] Haiyan Zhao and Hongxia Tong, "A Dynamic Service Composition Model Based on Constraints", In Proceedings of the Sixth International Conference on Grid and Cooperative Computing(GCC 2007), IEEE 2007.

[16] Bart Orri¨ens, Jian Yang and Mike. P. Papazoglou, "A Framework for Business Rule Driven Service Composition", LNCS, vol. 2814, pp. 52-64, Springer 2003.

[17] Zakaria Maamar, Soraya Kouadri Moste´ faoui, and Hamdi Yahyaoui, "Toward an Agent-Based and Context-Oriented Approach for Web Services Composition", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 17, NO. 5, MAY 2005, pp. 686-697, IEEE.

[18] Maja Vukoví´c, Evangelos Kotsovinos and Peter Robinson, "An architecture for rapid, on-demand service composition", Journal of Service Oriented Computing and Applications -SOCA (2007) 1:pp. 197–212, Springer-Verlag London Limited 2007.

[19] Daniela Berardi, Diego Calvanese and Giuseppe De Giacomo, "Automatic Composition of *e*-Services",Technical Report", [online]: http://www.dis.uniroma1.it/~mecella/publications/eService/BCDLM_techRport_22_2003.pdf, [visit]: April 2009.

[20] Shuchao Wan, Jun Wei, Jingyu Song and Hua Zhong, "A Satisfaction Driven Approach for the Composition of Interactive Web Services", In Proceedings of the 31st Annual International Computer Software and Applications Conference(COMPSAC 2007), IEEE 2007.

[21] Rachid Hamadi and Boualem Benatallah, "A Petri Net-based Model for Web Service Composition", In Proceedings of the 14th Australasian database conference, Pages: 191 - 200, ACM, 2003.

[22] Gunter Preuner and Michael Schrefl, "Requester-centered composition of business processes from internal and external services", Data & Knowledge Engineering 52 (2005) 121–155, ScienceDirect-Elsevier.

[23] Bixin Li, Yu Zhou, Ying Zhou and Xufang Gong, "A Formal Model for Web Service Composition and Its Application Analysis", In Proceedings of the 2007 IEEE Asia-Pacific Services Computing Conference, IEEE 2007.

[24] Yang Xu and Youwei Xu, "Towards Aspect Oriented Web Service Composition with UML", In Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), IEEE 2007.

[25] Youssef Gamha, Nacéra Bennacer, Lotfi Ben Romdhane, Guy Vidal-Naquet and Bechir Ayeb, "A Statechart-Based Model for the Semantic Composition of Web Services", In Proceedings of the 2007 IEEE Congress on Services (SERVICES 2007), IEEE 2007.

[26] Eric Bouillet, Mark Feblowitz, Hanhua Feng, Zhen Liu, Anand Ranganathan and Anton Riabov, "A Folksonomy-Based Model ofWeb Services for Discovery and Automatic Composition", In Proceedings of the 2008 IEEE International Conference on Services Computing, IEEE 2008.

[27] Philippe Balbiani, Fahima Cheikh and Guillaume Feuillade, "Composition of interactive Web services based on controller synthesis", In Proceedings of the 2008 IEEE Congress on Services 2008 - Part I, IEEE 2008.

[28] Demian Antony D'Mello and V. S. Ananthanarayana, "Effective Web Service Discovery Based on Functional Semantics", Communicated to First International Conference on Advanced Computing (ICAC 2009), Chennai, December 2009.

[29] Demian Antony D'Mello and V. S. Ananthanarayana, "A Tree Structure for Efficient Web Service Discovery", Communicated to Second International Conference on Emerging Trends in Engineering & Technology (ICETET-09), Nagpur, December 2009.

[30] XMethods, [online] Available: http:\\www.xmethods.com, [visit: February 2009].