Cluster Computing (2019) 22:S15387–S15408
https://doi.org/10.1007/s10586-018-2604-2

A unified algorithm to automatic semantic composition using multilevel workflow orchestration

U. Arul[1] · S. Prakash[2]

Received: 11 February 2018 / Revised: 13 March 2018 / Accepted: 17 March 2018 / Published online: 29 March 2018

Abstract
As a result of state-of-the-art development in service oriented architecture, we need a composition framework and comprehensive algorithm to discover and compose the web services from different environments. In this paper, we present a unified semantic-oriented framework with corresponding algorithm for automatic web service composition that integrates the comprehensive process of modified multistage composition and rigor of web semantics. Our proposed unified algorithm introduces the novel features through modified five stage composition such as transformation of non-functional properties of user requirements to all stages, optimization and semantic validation of abstract workflows using workflow automata, annotating WSDL files with additional ontologies using ontology based service repository, adopting dynamic change of user requirements for discovering candidate services, and selecting most optimal services for concrete composition using non-functional properties are effectively represented. Feasible composition solution obtained for user complex requirements through semantic web service discovery mechanism for discovering and selecting the most suitable service candidates. Furthermore, our unified algorithm can provide a composition solution through wider acceptance of semantics-oriented documents such as web ontology language for services and web service modeling ontology. We evaluate the proposed unified algorithm for automatic generation of composition using our motivating scenario, namely, home loan approval inference process. We also evaluate algorithm for automated and dynamic composition on service repositories of various sizes in increasing the levels of nesting and present the performance results.

Keywords SOA · Automatic semantic composition · Workflow orchestration · Non-functional properties · Levels of nesting · SWSD

1 Introduction

The enhancement of Internet technologies has improved many cutting-edge technologies such as Webservices. The Web services provide a solution to the deployment of a complex enterprise software system in the collaborative and interoperable environment of the service-oriented architecture (SOA) [1, 2]. The main advantage of SOA is that it supports the Web service composition (WSC) which deploys a complex workflow for enterprise applications integration using web services in platform independently. In spite of the benefits of the SOA to the WSC, it has many issues need to be addressed during the development process of existing workflow systems such as, (i) interaction between the services is fixed statically and hardcoded, (ii) modifications of services in terms of inputs, outputs, or behavior can cause the entire system to fail, (iii) static binding of services avoids the reusability the new generated services, and, (iv) difficult to handle the failures of services during runtime of composition process. The major challenge to solve these issues is to build the workflow for complex enterprise systems without human intervention during runtime in automatically and satisfy dynamic change of user requests [3, 4].

✉ U. Arul
arulmee08@gmail.com

S. Prakash
prakash.sav4@gmail.com

[1] Department of Computer Science and Engineering, Dhanalakshmi College of Engineering, Chennai, India

[2] Department of Electronics and Communication Engineering, Jerusalem College of Engineering, Chennai, India

In order to build the AWSC for dynamic change of user requests, we need a semantic oriented approach that allows composition applications can realize about service's functionalities, its required inputs and expected outputs to a level of detail during runtime in machine understandable manner. Furthermost, all existing web services are appeared by having the pure syntactic descriptions rather than semantic. Hence, it is necessary to add semantic capabilities with syntactic service descriptions to realize automatic composition is more feasible.

With regard to automatic service composition, the existing composition process has been decomposed into four phases, such as: (i) Planning—generating an abstract workflow based on the specification collected from user, (ii) Discovery—identifying the services that are matching with plan i.e., the generated workflow, (iii) Selection—choosing the best matched candidate services for deployment, and, (iv) Execution—executing the composition of candidate services. In our literature survey most research efforts has focused on one or more of these four phases.

Our main objective of research in this paper is to provide a comprehensive algorithm to automatic web service composition (AWSC) for satisfying the dynamic change of user requests in automatically and generating a scalable composition process. Our unified composition algorithm additionally proposes a new phase, called as Validation and Optimization that appears after the planning stage. It performs both structural and semantic validation on generated workflow to generate an optimized abstract workflow that satisfies the user requests with most suitable tasks. We provide a more contemporary algorithm for automatic service composition, which addresses the solutions to composition issues by adding the following features:

1.1 Multilevel orchestration for workflow generation

In order to provide a scalable composition solution, our proposed algorithm considers the functional goals of user requests in more comprehensive manner and it generates the abstract workflow in nested multi-levels. In addition, our algorithm can also include dynamically evolving workflows to meet the ever changing user goals at higher level.

1.2 Annotating additional metadata to web services

From our literature survey, we understood that the existing composition approaches used additional metadata (only I/O parameters) for providing semantics to web services. We consider the I/O parameters are inadequate to signify semantics to services. So, our proposed algorithm can as well state the pre-/post- conditions to services for grasping a realistic composition.

1.3 Transformation of QoS parameters (non-functional properties)

In general, the functional properties are identified from the user requests, but the identification of non-functional properties is difficult because that primarily deliberates the QoSparameters. Our proposed algorithm can perform the transformation of NFP's after identified them in user requests by using QoS aware service composition.

Additionally, we propose a framework for automatic and scalable service composition that provides solutions to reduce the complications during the composition process. Our proposed framework considers the aforementioned features for providing automatic service composition in the case of dynamically changing the user requests.

In the rest of the paper, Sect. 2 presents the existing work related to our approach. Section 3 gives the overview of methodology of multilevel workflow orchestration, the enhanced five phase composition, problem formalization and motivating scenario for our automated semantic composition. Section 4 describes our proposed framework for automatic semantic composition using multilevel workflow orchestration. Section 5 elaborates the unified algorithm for multilevel workflow orchestration which adopts the modified five stage automatic composition. Section 6 explores the implementation details of modified five stage composition as well as experimental results and its feasibility analysis. In Sect. 7, we conclude the paper and present our future work.

2 Related work

Web service composition approaches broadly categorized into manual, semi-automated and automated [1, 3, 5]. In manual composition, the user intervention is mandatory in each step of the process, where composition is implemented through standard tools, for example, the most prevalent is Business Process Execution Language for Web Services (BPEL4WS) [6]. Semi-automated composition approaches commonly automate only parts of the complete composition process, for example, they concern service selection of suitable atomic services, where the other composition stages implemented in manually during the composition scheme [7]. The fully automated composition approaches can automate almost entire composition process that range from planning to the selection of atomic web services, where user intervention is restricted for specifying user requirements [1]. The important advantages of full automation of composition comprise scalability and

dynamic handling. Scalability of composition concerns continuous increase of services in repositories over time and it can able to manage effectively the massive amount of services. The dynamic handling of automated composition concerns dynamic change of composition environments, for example, handling of exceptions when service failure and unavailability of service [8].

Many researches presented algorithms to solve automatic web service composition in two or more phases. The approaches described in [9–11] attempts to solve the two phases of composition namely planning and discovery. They are capturing the semantics of the web service using description logic. The composition problem is decomposed as number of sub-services and each constitute an atomic action during the planning stage. The composition service is represents the goal that achieved by combining the some atomic actions. These approaches must require an explicit goal definition, however, such explicit goal definitions are usually not represented.

To our best of knowledge, most of the composition algorithms proposed methodologies in two or more phases to composition, for example, planning and discovery, rather than a comprehensive and unified algorithm for all possible stages of composition. In this paper, we present a unified algorithm that provides a comprehensive composition solution through our modified five stages of composition.

Recently, automated approaches leveraged using AI techniques to represent the composition problem in well-defined and obtain composition solutions in optimal manner. In addition, these intelligent techniques provide methods for representation of knowledge in semantics to facilitating the more enhanced intelligent automated composition [12]. An initial and heuristic based Hierarchical Task Network (HTN) planning was introduced in SHOP2 [13] by accommodating AI planning for service composition. SHOP2 supported for encoding OWL-S processes, when encoding the planning problem in automatic web service composition. The main drawback of this approach is that it applies decomposition rules in planning stage for constructing HTN network and requires a prior knowledge for encoding the rules with the help of ontologies from DAML-S (DARPA agent Markup Language for Services) processes.

Another approach is tried through planning for implementing the process model in automated service composition and adopting the modification in the MBP process [14]. BPEL4WS provides the web services in abstract descriptions as input to MBP and additionally the goal state also given to it. It produces the composite service in descriptions of BPEL4WS as output. But, MBP process is not consumed semantic information for composition and scalability is considered as a main drawback of approach.

The approach represented in [15] attempts to dynamically adjust the service composition by modifying the GOLOG standards and using intelligent agents. Intelligent agents used to reason on automatic service discovery and composition. Also, situation calculus used for specifying the user requirements and constraints. However, the process of encoding and translation are considered as very complex, and level of interoperability measures represented in existing systems is also decreased.

The SWORD framework [16] attempts to find the automatic composition using entity—relationship models and Horn rules. However, the SWORD required the user intervention in generating the final composition plan through rule-based expert system. OWLS-XPlan system [17] produced a planning module called XPlan for generating composite services by using semantic descriptions of services from OWL-S. This system uses XML to derive compliant between XPlan and PDDL (Planning Domain Definition Language). But, this system not utilized the semantic information from domain ontologies and therefore, exact matching is not performed between service input and outputs in planning module.

Another active area of research is QoS aware composition [18–20]. Researchers apply SLA to compositions, but they do not considered dynamic composition. They used the existing composition languages to implement the composite service using predefined template which is used to select the set of suitable services during the composition. In addition, they applied a QoS model and non-functional attributes on composition solutions to ensure that they adhere with pre-defined attributes. Finally, they filtered the produced solutions that compliance with SLA agreement.

From our literature survey, we identified that most of the service composition approaches considered a single stage or two stages composition implemented on partial algorithms. In general, most of the composition approaches designed in different environments, goals and viewpoints. Our composition approach for AWSC purely focused on multilevel workflow orchestration with semantic discovery and semantic selection through a unified and comprehensive algorithm.

3 Multilevel workflow orchestration for automated semantic composition

The automated semantic composition can be realized through two methods: Top-down and Bottom-up composition. We used Top-down composition using multilevel workflow orchestration to achieve the automatic composition. In this section, we describe our methodology for automatic semantic composition that produces a workflow in different sub-levels to adopt the dynamic change of user requests.

3.1 Methodology of multilevel workflow orchestration

Our approach is based on a multilevel orchestration of the list of candidate services according to the functional and non-functional properties of user requirements. The primary feature of our approach is that it invokes the suitable composite services in recursive method until the user requirements are satisfied. Apart from, it can compose together the suitable dynamic and static services in each sub-level. The dynamic services are nested by the other static and dynamic services in recursively for each level of orchestration to accomplish the composition goals. The multilevel (hierarchical) workflow generation and adaptation of user requirements during the composition process is shown in Fig. 1.

3.2 The modified five phase composition

We proposed the modified five phase composition by examining of different composition approaches and reviewed at current solutions in the literature survey [3, 4, 20]. Our contemporary automatic service composition process has been decomposed into five phases such as (i) Planning, (ii) Validation & Optimization, (iii) Discovery, (iv) Selection, and, (v) Execution. The first phase involves generating an abstract workflow that contains set of tasks and the order in which they need to be composed. The second phase performs structural and semantic validation on generated workflow to ensure the connectivity of tasks structurally and semantically correct. The discovery phase involves discovering the services matching with tasks of the workflow. We have enhanced this discovery phase that also performs a mapping from the syntactic to semantic definition and standardization of semantic definition by adding the ontologies for the discovered services. The selection phase involves producing an optimal concrete workflow that contains physically composed services based on methodology of multilevel workflow orchestration and non-functional properties. The execution phase involves executing the services as per the concrete workflow and an alternative abstract workflow has to be chosen when any of services is not available. Figure 2 depicts the modified five phase composition and it augmented with the composition features mentioned in Sect. 1.

3.3 Problem formalization

Our proposed automatic composition algorithm must have to select the appropriate services based on functional properties, on-functional properties and other user constraints for generating executable compositions. To this objective, we introduce the primary definitions that we used to perform the multilevel workflow orchestration for automatic web service composition (AWSC) problem.

Definition 1 (*User problem space*) User Problem Space specifies a set of user's request (U_R) consists of functional properties (FPs), non-functional properties (NFPs) and constraints (U_C). It is described in structural statement over ontology O. Ontology O represents the real world things consists of a set of elements O_E. O_E is a 3-tuple $< C, S, P >$, where C represents a set of unary predicates over O_E and denoting concepts, S represents a set of binary

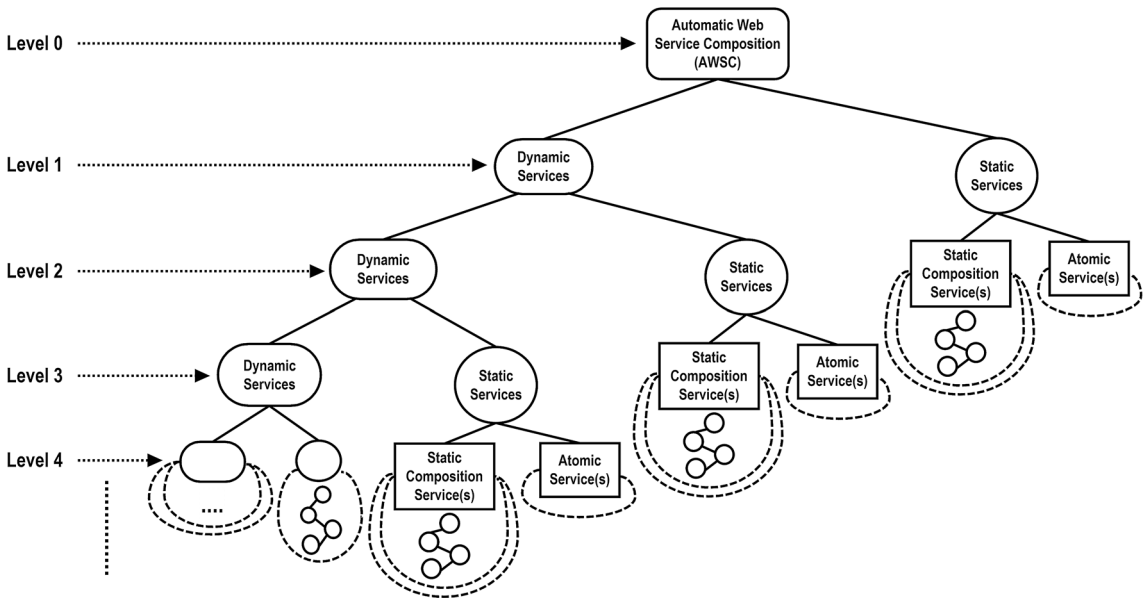

Fig. 1 The proposed methodology of multilevel workflow orchestration

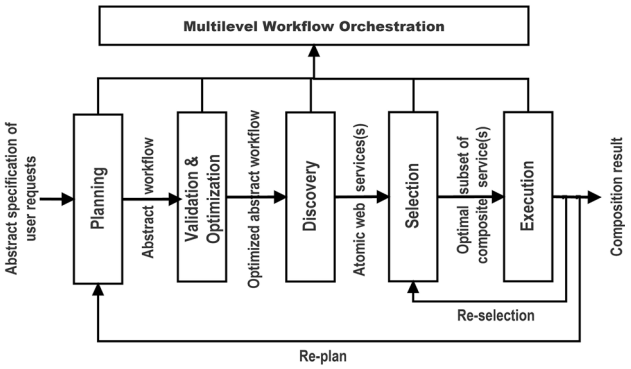

Fig. 2 The modified five phase composition

predicates over C denoting relationship between two concepts, P represents a set of binary predicates over O_E, denoting properties. Every structural statement created as a composition of predicates from C U P or statements composed using logical operators such as $\wedge$, $\vee$, $\neg$ for representing user's request.

In our proposed algorithm, the user problem space providing functional and non-functional properties of user requirements to the planning, discovery and selection phases for constructing an abstract workflow, finding the services from the service repository and selecting the most suitable service instances respectively. The construction of abstract workflow is the *Workflow construction problem* and the planning phase has to identify the set of suitable tasks, conditions and control/data flow for the workflow from the user functional requirements. Next, discovering the services from the repository that matches with user's functional requirements is the *Service discovery problem*. The services are discovered, those has to satisfy the post-conditions with output parameters and the preconditions with input parameters. Finally, selecting the most appropriate service instances that are satisfying the non-functional properties is the *Service selection problem.* In general, the non-functional properties are understandable by the requested users only, but not by service selector, because the non-functional properties are usually expressed in an abstract form that is understandable for the user's semantics. Also, the non-functional properties are generally represented in abstract form that cannot be recognized during the service selection phase. In order to make the service selector to identify the most suitable services for automatic composition, it is mandatory that the abstract non-functional properties must be converted into concrete form. Therefore, before the service selection phase, our algorithm performs a transformation of abstract non-functional properties into concrete non-functional properties. Also, the services are present in the selection phase, among them the most suitable services must be selected for the execution stage, which requires the necessary information such as service identification, operators, and other important parameters. During this transformation, it identifies the terms in the abstract non-functional properties and links the identified terms with ontology in order to representing the meaning for each term. Then it composes the each individual term into concrete non-functional properties that are able to understand by service selection phase.

Definition 2 (*User constraint and constraint expression*) The User Constraint (U_C) refers a collection of relative predicates which must have to meet for confirm the semantics of user request (U_R), in addition, to arrange appropriate ordering of abstract tasks in an abstract workflow (A_W).

A_W can be defined as $A_W = \{t_1, t_2, t_3 \ldots t_n\}$, where t_i indicates abstract tasks in workflow.

The Constraint Expression (C_E) is a formal representation of user constraints is given in the form as following

Constraint : = Constraint Term Relation Instance Data, where

(1) Constraint Term represents a constraint attribute
(2) Relation represents operators such as $=$, $\neq$, $<$, $\leq$, $>$, $\geq$, $\in$, $\subseteq$, $\supseteq$, is, not, in and not in
(3) Instance Data represents data set, including numerical data, set of ontological concepts, other data set etc.

Definition 3 (*Structural validation on abstract workflow*) The structural validation applied on abstract workflow (A_W) to validate the representation of behavioral specification and control flow analysis. We use workflow automata to validate the structural behavior on abstract workflow.

Definition 4 (*Workflow Automata* (*WA*)) Workflow Automata (WA) is defined as a tuple $< T, \Sigma, C, \delta, I, F, V >$ for given abstract workflow (A_W) and a set of constraint attributes.

where,

(1) $T = < T_{Pre}, T_{Eff} >$ is a pair which represents the precondition and effect of each task in workflow (A_W). T_{Eff} denotes the transformation of one task to another with matching the precondition T_{Pre}.
(2) $\Sigma = A_1, A_2, \ldots, A_n$ is finite set of actions. Each task t_i of workflow can perform an action A_i to produce change of task.
(3) C is finite set of states that represents constraints of C_E i.e. precondition T_{Pre} of task t_i. The constraints of C_E are partitioned into three non-overlapping sets as $< C_V, C_L, C_P >$

C_V: set of constraint variables that can provide and store explicit values in task t_i

C_L: set of link constraints that are used to specify control flow for executing actions on task t_i

C_P: set of predicate constraints that represents conditional expression appeared in concurrently executing actions on different tasks of workflow

(4) δ is the transition function, that is, $C \times \Sigma \times (C_V \cup C_L \cup C_P) \rightarrow C$
(5) I is finite set of initial states and $I \subseteq C$
(6) F is finite set of final states and $F \subseteq C$
(7) V is an assignment function, such that, $V: T_{Eff} \rightarrow C$

Definition 5 (*Semantic type definition*) Let T_D be a type definition for the tasks $t_1, t_2,\ldots,t_n$ in the abstract workflow. A type $TYPE_i$ associated with tasks t_i by T_D.

Definition 6 (*Semantic validation on abstract workflow*) The *Semantic Validation* checks the data compatibility between the precondition (T_{Pre}) and effect (T_{Eff}) parameters of the connected tasks that presents the data types in meaningful manner in the abstract workflow.

Our proposed algorithm supports the detection of type incompatibility between the connected tasks and also it assists for automatic type conversions using type transformations. Figure 3 shows semantic data type compatibility between the tasks in the abstract workflow. For example, the datatype of effect (T_{Eff}) of task T_1 is semantically bound to the datatype of precondition (T_{Pre}) of task T_2. Our semantic compatibility validator compares the type compatibility of task T_1 output (source data type) with task T_2 input (target data type). The semantic incompatibility is detected by the compatibility validator when types of T_{Eff} and T_{Pre} may be compatible at syntactic level, but its semantics is actually incompatible. Suppose, T_{Eff} and T_{Pre} are belongs to floating point data type, however T_{Eff} represents the value of acceleration in metres/second and T_{Pre} represents temperature in degrees of Celsius.

Definition 7 (*Quality of service* (*QoS*)) A quality of service can be defined as an attribute QoS^i that representing the value of the non-functional property of web service.

In general, the QoS attributes that can be classified into two categories such as static and dynamic attributes. The static QoS attributes such as scalability, capacity, accuracy, security, price etc., are not changed during the execution of services and usually defined by service providers. The dynamic QoS attributes such as availability, response time, throughput, reputation, stability etc., can be changed during the execution time and these are provided in abstract form of user's requests. Our *QoS* model comprises of dynamic QoS attributes that are proposed as non-functional properties. The selection of web service from the optimal subset of all the discovered services is based on these non-functional properties. We considered the QoS (non-functional) attributes of web services employed a primary affecting factor in the *Service selection problem,* because it is unavoidable due to the multiple services with same functionality for accomplishing the similar task.

Definition 8 (*Semantic service*) A Semantic Web Service SWS can be defined as a tuple, $SWS = < I, O, Pre_{Con}, Post_{Con}, QoS_i >$, where

(1) I is a list of inputs that are reference the set of semantic concepts of the service
(2) O is a list of outputs that are reference the set of semantic concepts of the service
(3) Pre_{Con} represents a condition that must hold before execution of service and must meet to ensure correct service execution
(4) $Post_{Con}$ represents a condition that must hold after execution of service
(5) QoS_irepresents an n-tuple $< QoS_i^1, QoS_i^2,\ldots..QoS_{i\,n} >$, where each QoS_i denotes the corresponding value of a QoS attribute of the tuple $< QoS^1, QoS^2,\ldots..QoS^n >$

Definition 9 (*Registering services in service repository*) Service Repository R can be represented as a set of web services that can be used during the service composition.

Definition 10 (*Service discovery using user request* (U_R)) Service Discovery can be represented by Query Q which is a tuple $Q = < I', O', U_R >$, where

(1) I' is a list of provided inputs that are reference set of concepts in the ontology
(2) O' is a list of provided outputs that are reference set of concepts in the ontology
(3) U_R is a user requests as specified in definition 1, which contains functional properties (FPs), non-functional properties (NFPs) and user constraints (U_C)

We extract functional properties (FPs) and user constraints (U_C) during this discovery stage. We describe the service discovery is a function of choosing candidate

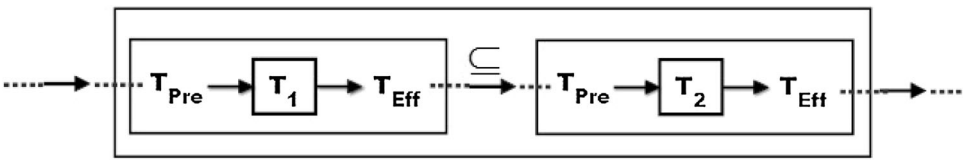

Fig. 3 Task T_1 outputs T_{Eff} is semantic type compatible to the input T_{Pre} of task T_2

services from the service repository for matching with abstract workflow (A_W). The generated candidate services can be represented as

Candidate Services: $A_W X$ FP $\rightarrow$ SWS, where

(1) A_W is a set of abstract tasks produced during the planning stage
(2) FP represents the attributes of functional properties of user request (U_R)
(3) SWS is a set of service candidates produced that ensure the attributes of FP

Definition 11 (*Service selection*) Service Selection can be represented as a tuple S = < SWS, O, NFP >, where

(1) SWS is similar as in definition 8
(2) O is a set of service instances from ontology based service repository which are matched with input or output of the real semantic web service of < IOPE > in OWL-S
(3) NFP is a set of QoS attributes that are semantically represented in the form of concrete representation

Definition 12 (*Web service composition* (*WSC*)) Web Service Composition can be defined as a finite set of service instances that are selected during service selection by user requests and constraints. These selected service instances can be represented in the form of directed acyclic graph G = (V, E) describing the control and dataflow of execution of selected services that used in service composition. The nodes represent service instances those are selected from service repository R, that are need to be executed during service composition, i.e. $\forall O \in V : WS \in R$. The edges represent the control and data flow.

Our approach models an automatic web service composition as a multilevel workflow process which implements an optimized workflow. The optimized workflow represents the space of all candidate composition solutions which are generated by discovering the suitable semantic services for a user request U_R. Our objective is to generate the optimized composition solution, so we used QoS model with respect to satisfying the non-functional parameters.

3.4 Motivating scenario

We present a real time motivating scenario for a home loan approval process which elaborates the automatic composition of a set of web services using multilevel workflow orchestration. In this scenario, the orchestration (main) service called as *HomeLoanApproval* Service that integrates the most suitable services in different sublevels to accomplish the user's requirements. We need to discover several services that are belong to either static or dynamic services and the dynamic services can invoke other most suitable services in recursively in different sublevels.

In order to approve the user's home loan request (the top level goal), there must be a multiple levels of orchestration using sub level processes such as, (1) storing the user requests in the user problem space that is repository of requests, (2) estimating and validating the credit-worthiness of the user or customer, (3) confirming the customer's assets is worthiness for sanctioning the home loan, and, (4) reiterating the home loan approval process for all customers who have applied for a home loan. In our scenario, the *HomeLoanApproval* service i.e., orchestrator service is responsible for retrieving the customer requests from user problem space and collecting the customer's details such as loan information, bank account information, personal information, home address and other details. In addition, the orchestrator service has to schedule the home loan approval plan and ensure whether to approve or reject the customer's loan request. Figure 4 shows the motivating scenario for the home loan approval process using multilevel orchestration.

Firstly, the *HomeLoanApproval* service calls *CalculateCredit* service that calculates the customer's credit worthiness value and sends it back to orchestrator service. Then, the orchestrator service invokes *AssessCreditWorthiness* service that ensures whether the customer is credit worthiness or not. Afterwards, the orchestrator service invokes the *PropertyAppraisal* service only if the customer is credit worthiness, otherwise loan not granted to the particular customer and the loan status information can send to the customer. The *PropertyAppraisal* service ensures the assets and mortgage property of customer is equivalentto the quantity of the loan. If customer property is worth for amount of loan, then *PropertyAppraisal* service sends the output message as loan granted, otherwise it sends the output message as loan not granted to the orchestrator service. Finally, the orchestrator service logs the message in its local database and sends the status of loan to the customer. In some cases, the *HomeLoanApproval* service get an error message since the any of these dynamic services is unavailable. When an error or unavailability services is occurred, then our orchestrator service can return to the service selection stage to choose the next matched alternative service and precede the composition process. The orchestrator service calls these dynamic services in different sublevels in recursively in order to process the loan applications for the group of customers.

4 Automatic web service composition framework

Using the problem formation described in Sect. 3.3, we propose a framework for automatic web service composition based on modified five phase of orchestration. Our

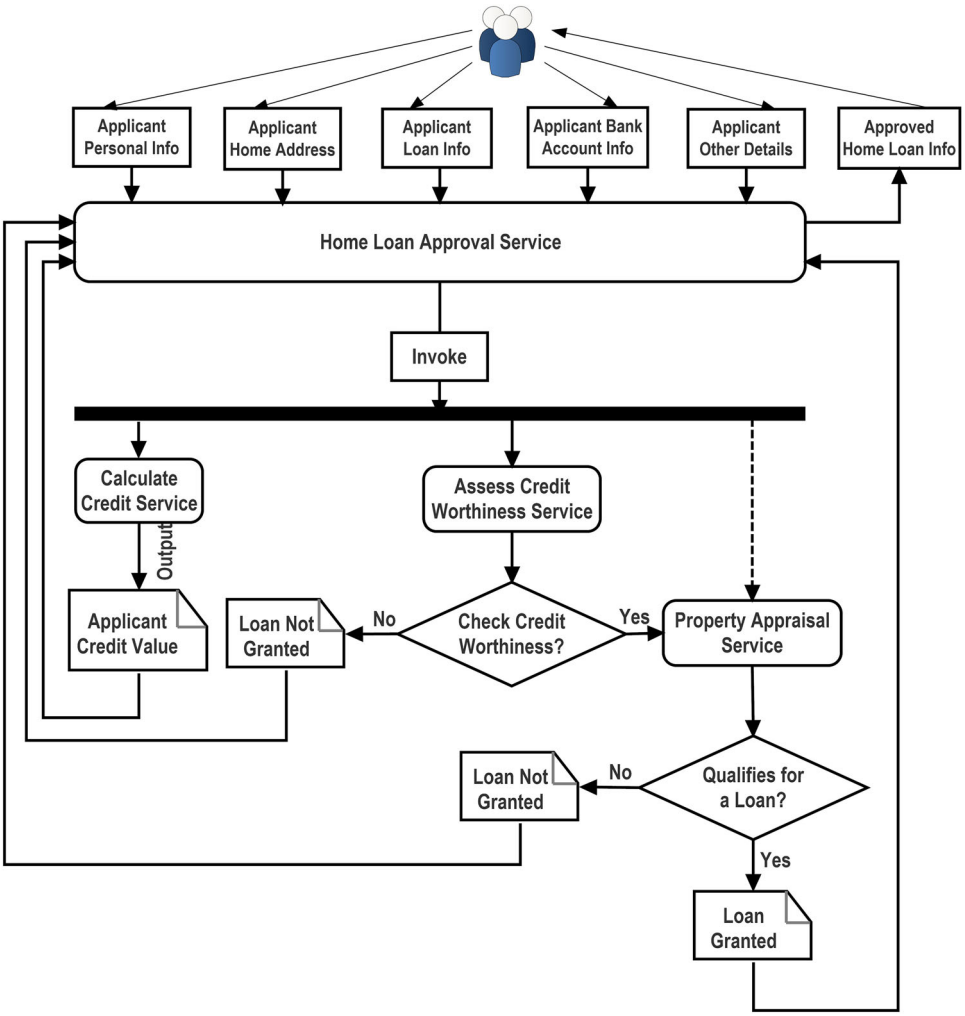

Fig. 4 The motivating scenario for home loan approval process

proposed framework is depicted in Fig. 5 and it consists of five phases for composition. The first phase contains a planner that generates an abstract workflow based on abstract specification retrieved from user problem space. The abstract specification of user requirements includes the functional specifications, non-functional properties and other user constraints. The second phase concentrates on validating the generated abstract workflow in both structurally and semantically and generates the optimized workflow to the web service discovery phase. The third phase called as semantic web service discovery which includes mainly semantic markup for WSDL based web service descriptions and semantic discovery for identifying the semantic candidate services according to each task specification existing in the optimized abstract workflow. The fourth phase concentrates on generating an executable composition from optimized abstract workflow by selecting the most appropriate service instances in fulfilling the non-functional (QoS) properties [18]. The final phase includes an execution engine that executes executable composition (concrete workflow) represented in workflow language. In the following, we describe the basic components of the proposed framework.

4.1 Goal specification through user problem space

The user problem space includes repository storage that enables end users to specify their requirements. A requirement analyzer is integrated into the user problem space to segregate the functional specifications from non-functional properties. The functional requirement of user goal specification includes control flow, data flow, conditions, and other high level goal for abstract representation of workflow. The non-functional properties include both static and dynamic QoS parameters to select the optimal subset of service instances for executing concrete workflow.

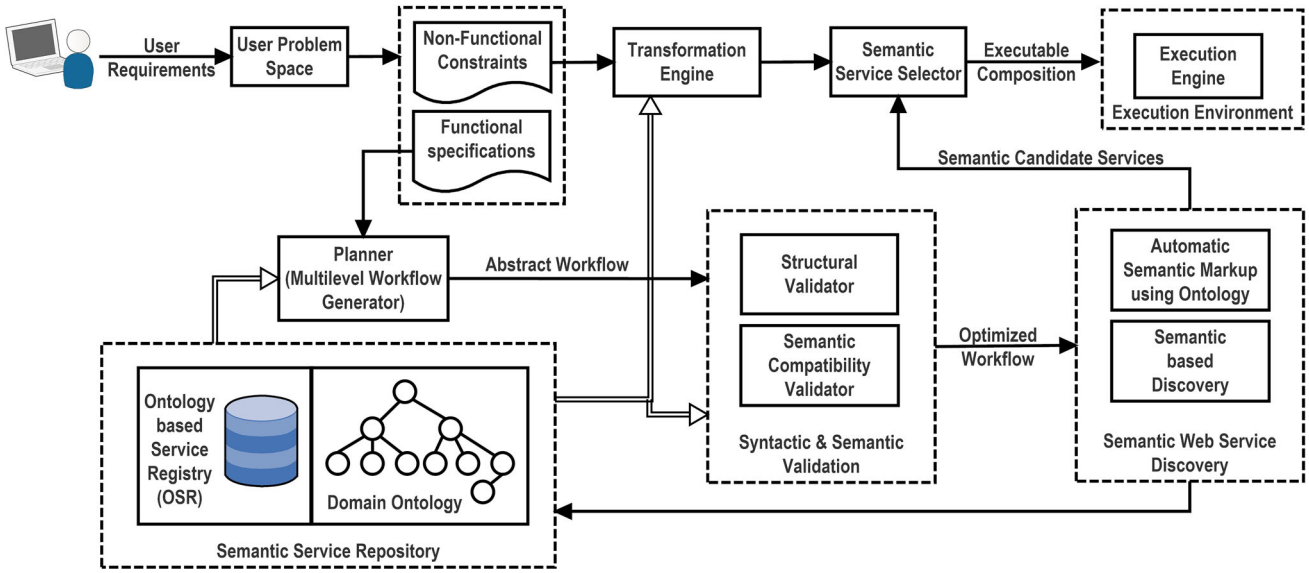

Fig. 5 The framework for automatic web service composition

4.2 Processing non-functional properties through transformation engine

We suggest a novel solution to identify the non-functional properties and transform them into the concrete form through the transformation engine. Commonly, the user requests represented in abstract form that may not be understand by the service selector for selecting the most suitable service instances. So that, the abstract non-functional properties must be transformed to concrete form before making the service instance selection, because of abstract form represents informal representation of user requests. In our research, a new methodology called as transformation engine that identifies the non-functional properties of user requests and it converts them into constraint expression (concrete form of NFP's). The constraint expression has a set of terms that can linked into concepts of domain ontology i.e. the ontology repository and each term has binding information to identify specific task in concrete workflow. Therefore, the constraint expression can facilitate to identify service instances for service selection and concrete composition.

The ontology repository contains a set of classes with hierarchical relationships which allows the constraint expression for performing logical reasoning and getting information about the services. Our transformation engine can select the exactly matched classes from ontology domain as links for connecting terms in constraint expression. The transformation engine can use both ontology based service registry (service domain) and ontology domain to perform the transformation.

Our proposed methodology of transforming the abstract representation of non-functional properties into concrete representation is represented in procedure 1.

Procedure 1 Processing non-functional properties

(1) Take a user request from problem space to transform it into constraint expression.
(2) Divide the abstract form of user request into a set of tokens i.e. terms.
(3a) If the token is a constant (constraint value), then add it into constraint expression.
(3b) Else if the token is an index, then find context and ontology class from domain ontology, after that bind it into suitable class of domain of ontology and add it into theconstraint expression.
(4a) Else if the token has no operator, in this case, it involves a single class alone, then token is converted to constraint variable. Here, the identification of an index of token from domain ontology is done for matching it to a service in service domain and then added it into the constraint expression.
(4b) Else, if the token has an operator (i.e., more than one class involved), then it collects the related (contextual) information in order to find the index and domain for all classes related to the operator from the domain ontology and map the identified classes with corresponding services of service domain. Then add it into the constraint expression.
(5) Repeat the steps from 3 to 4 until each identified tokens processed.
(6) Return a generated constraint expression to semantic service selector.

5 Multilevel workflow orchestration algorithm

The proposed Automatic Web Services Composition (AWSC) based on top-down orchestration approach can be an extension of the conventional service composition approaches. We modified the discovery and the execution stages in the algorithm, by analyzing the characteristics of dynamic services. The main characteristic of dynamic services is it can be able to adapt more than one static service. In the AWSC described here, several sequences of abstract workflows can be generated in the planning stage, and then abstract workflows can be validated using both structural and semantic validation that traversing all abstract workflows and the most optimized workflow can be identified in the optimization stage. The service instances that have required operators are identified from the service registry for each abstract task of optimized workflow during the invocation of discovery stage. We also extended the discovery stage by adding an additional phase that can used to facilitate for adding new services in the service registry. At the end, the AWSC outputs the composite services by combining the operators of the selected services. The following algorithm shows the top-down approach for multilevel composition, where the necessary methods and parameters are presented to accomplish a preferred composition.

Algorithm : Multilevel workflow for Automatic Web Service Composition

Input :*A set of available ontology based services in (Service Repository)* $R = (WS_1, WS_2, \ldots WS_n)$*, and a user request* $U_R = (FPs, NFPs, U_C)$
Output :*A feasible composite workflow that satisfy user request* U_R

1: *Let* ***AbstractTask*** *be a set of tasks for a workflow*
2: *Let* ***Workflow*** *be a set of abstract workflows*
3: *Let* $\boldsymbol{Aw_{new}}$ *be an initialization of abstract workflows*
4: *Let* ***OptimizedWorkflow*** *be a set of structurally validated abstract workflows*
5: *Let* $\boldsymbol{C_L}$ *be a set of link constraint variables for specifying control flow and executing action in task* t_i
6: *Let* $\boldsymbol{TYPE_i}$ *be a data type associated with tasks* t_i *in abstract workflow*
7: *Let* ***ServiceProfile*** *be a set of service profiles of all registered semantic services in service repository*
8: *Let* ***ServiceModel*** *be service types for an optimized workflow*
9: *Let* ***ServiceExample*** *be service instances for a service model*
10: *Let* ***ConcreteWorkflow*** *be a workflow constituted of set of service models*
11: *Let* ***ExecutableWorkflow*** *be a workflow consisting of set of selected services*
// Planning Stage
12: ***Workflow*** ←*generateWorkflow(* U_C*, FPs)*
// Initialization of abstract workflow
13: $\boldsymbol{Aw_{new}}$ ←*InitializeOAW(*A_W*,* U_C*)*
14: *Add* N_S *to* A_W
15: *Add* N_T *to* A_W
16: ***for*** *each node* $N \in A_W$ ***do***
17: *Label N as "unMatched"*
18: ***end for***

// Optimization Stage
19: ***OptimizedWorkflow*** ←*generateOptimizedWorkflow(**Aw**$_{new}$, WA)*
20: *traverseAllAbstractWorkflows(**Aw**$_{new}$,*N_S, N_T, *WA)*
21: ***for****each workflow* Aw_i *of* **Aw**$_{new}$***do***
22:*path.length =* Aw_i*.pathlength*
23: *visited = path.length*
24:***if*** *visited is an* $\mathbf{C_L}$ *set* ***then***
25:*Label the node visited as "matched"*
26: *semanticTypeValidate(*$TYPE_i \rightarrow T_{Eff}$, $TYPE_j \rightarrow T_{Pre}$*)* *// Semantic Type Validation*
27: ***if***$TYPE_i \neq TYPE_j$***then***
28: *Identify the semantic type* $TYPE_i of T_{Eff}$ *associated with the task* t_i
29: *Identify the semantic type* $TYPE_j of T_{Pre}$ *associated with the task* t_j
30: *Convert* $TYPE_j of T_{Pre}$ *to match with* $TYPE_i of\ T_{Eff}$
31: ***else***
32:*Move to next node*
33:***end if***
34:***if*** *visited ==* N_T***AND****all visited's parents are labeled as "matched"* ***then***
35:*storeOptimizedWorkflow(*Aw_{opt}*)*
36: ***else***
37:***for*** *each node* N_i *of visited* ***do***
38: ***if*** *node* N_i *is not an* $\mathbf{C_L}$ *and labeled as "unmatched"* ***then***
39:*traverseAllAbstractWorkflows (*Aw_i,N_i, N_T, *WA)*
40:***else***
41: ***if*** *all*N_i*'s parents are labeled as "matched"* ***then***
42: *traverseAllAbstractWorkflows (*Aw_i,N_i, N_T, *WA)*
43: ***end if***
44: ***end if***
45: ***end for***
46: ***end if***
47:*Traverse next workflow of* **Aw**$_{new}$
48: ***end for***
// Discovery Stage
// Phase 1: Service Registration
49: ***ServiceProfile*** ←*registerSemanticService(ServiceProvider, ServiceName, ServiceDescription)*
50: ***if****ServiceDescription == WSDL* ***then***
51:*Read the abstract definition of WSDL*
52:***if****Types == PrimitiveXSD****then***
53:*Convert PrimitiveXSD to OWL ontology*
54: ***else if****Types == ComplexXSD****then***
55: *Create temporary ontology with a concept and properties*
56: *Search ontology into ontology repository*
57: ***if****Search Ontology not produced result* ***then***
58:*Insert temporary ontology into ontology repository*
59:***else***

```
60:          Select the most suitable ontology from resulted ontologies
61:end if
62:    end if
63: end if
64:Extract Service Name and its description
65:   Extract inputs and outputs of the service
66:   Define preconditions and effects in ServiceProfileof the service
67:Add non-functional properties to ServiceProfile
68: end if
// Phase 2: Discovery Process
69: for i=0 to OptimizedWorkflow.lengthdo
70:ServiceModel ←OptimizedWorkflow[i].getServiceModels( )
71:forj=0 toServiceModel.lengthdo
72:ServiceExample ←searchServices(ServiceModel[j])
73:      ifServiceExample.length is 0 orServiceModel[j]calls a dynamic service then
74:      nestedPlan ←generatePlanning(ServiceModel[j])
75:      ServiceExample ← (ServiceExample) generateWorkflow (U_C, nestedPlan)
76:      end if
77:      ServiceModel[j].setServiceExamples(ServiceModel)
78:end for
79:OptimizedWorkflow[i].setServiceModels(ServiceExample )
80: end for
// Selection Stage
81: ConcreteWorkflow ←performCompositePropertySelection( U_C, OptimizedWorkflow)
// Execution Stage
82: ExecutableWorkflow ←produceExecutableServices (ConcreteWorkflow)
83: fori = 0 to ExecutableWorkflow.lengthdo
84:ExecutableWorkflow[i].Publish( )
85: end for
86: if Workflow is in nestedPlanthen
87:    return ExecutableWorkflow
88: else
89:    return invocation outputs of ExecutableWorkflow
90: end if
```

The algorithm for multilevel workflow for AWSC takes input as a set of ontology based web services from the service repository and user problem space. The user problem space specifies a set of user requests that includes functional properties, non-functional properties and user constraints.

The flow of the algorithm is as follows: First, the algorithm invokes the *generateWorkflow* function which takes the user constraints and functional properties as depicted in line 12 for the planning stage. The user constraints specify a set of context conditions that must be met to ensure the semantics of user request and correct order of abstract tasks in an abstract workflow. The functional properties are derived from the user's requirement and it is varied from the user constraints. The *Workflow* is used for storing the state of the results derived from planning stage. Then, the generated set of workflows is initialized to facilitate the search for optimization stage by invoking function *InitializeOAW* with parameters such as user constraints and functional properties (line 13). In this function, the starting node N_S and terminal node N_T are added to connect all the generated workflows (lines 14–15). All nodes are initialized as "unmatched"in connected workflows to perform the workflow based search(lines 16–18).

Secondly, the algorithm invokes the optimization stage by calling the *generateOptimizedWorkflow* function that takes two arguments such as the initialized workflow

AW_{new} and the workflow automata *WA* (line 19). The *generateOptimizedWorkflow* function calls *traverseAllAbstractWorkflows* to identify the optimized workflow among all connected workflows by performing traversing in workflows using automata (line 20). In *traverseAllAbstractWorkflows*, we first record the tasks (nodes) currently being visited and that are stored into the variable *visited* (lines 21–23). The visited tasks (nodes) matching with link constraint C_L that specifies control flow and executes an action in the task of workflow, then nodes are marked as "matched" (lines 24–25). In addition, our algorithm ensures the type compatibility between the connected tasks and it assists for automatic type conversions to produce semantically validated abstract workflow (lines 26–30). Then, we verify whether node in workflow has been accessed or not. If yes, and all visited nodes are identified as non-terminal nodes, also all of its parents has been marked as "matched". This means that we have found a path from the initial node to the terminal node in workflow A_{Wi}, that represents an optimized workflow Aw_{opt} and we stored the optimal workflow using the function *storeOptimizedWorkflow* (lines 34–35). If not, we call *traverseAllAbstractWorkflows* itself for all the workflows A_{Wi} of the current node, and then complete the traverse for connected workflows in a recursive manner (lines 36–48). Each recursive call of *traverseAllAbstractWorkflows* function that brings next unvisited workflow, Aw_{new}, from connected workflows. We continue this traversing process till all workflows A_{Wi} of connected workflows have visited and identified the optimized workflow for discovery stage.

We segregated the discovery stage into two phases such as (i) Service Registration and (ii) Discovery Process. In service registration, we perform mapping from WSDL to OWL-S before the web service registered into Ontology based Service Repository (OSR). OSR is an ontology based web service registry designed for storing semantic web services and its related ontologies, concepts, properties and relationships. The reason for mapping WSDL to OWL-S, the OWL-S focuses on the description of functional and non-functional properties of semantic web service. WSDL descriptions usually represented in syntactic manner therefore WSDL must be converted into ontology. We use *registerSemanticService* function to store semantic web service profile into OSR (line 49). This function performs parsing of WSDL service, extracts service description details, and fill the required properties that must be needed to construct the service profile of OWL-S. However, WSDL does not contain all required descriptions to build OWL-S particularly non-functional descriptions. The necessary non-functional properties needed to build service profile of OWL-S can be automatically added through extracting from user constraints U_C. We consider XSD types conversion of WSDL is the most important in service registration process of algorithm. Typically, XSD has two types: primitive XSD types and complex XSD types. The primitive XSD is converted to OWL-S in directly. However, the complex XSD is converted into OWL by searching most related ontologies that already exist in OSR. If most suitable ontology is not able to be searched from OSR, then we ask service provider to insert most suitable ontology into OSR. Suppose the ontology inserted by service provider that is also not matching with complex XSD type, then our algorithm can search the suitable ontology from public ontology repositories. Upon the completion of XSD types to ontological concepts, this algorithm automatically register the *ServiceProfile* of new semantic service into OSR (lines 50–68).

The discovery process (phase 2) of discovery stage for multi-level top-down AWSC is shown lines between 69 and 80. This process discovers the suitable *ServiceExample* (service instances) for each *ServiceModel* (service type) of an *OptimizedWorkflow*. If the *ServiceExample* set cannot be discovered for a *ServiceModel* from the local service repository OSR, or the *ServiceModel* is considered as a dynamic service, then internal AWSC is invoked in recursive manner to generate *ServiceExample* set for the *ServiceModel*. Following the call of inner AWSC or dynamic service invocation, a variable *nestedPlan* is generated through the *generatePlanning* method. The inner AWSC can search semantic web services that satisfy the functional goal using the generated nested plan and the matched semantic web services for nested plan can embedded in *OptimizedWorkflow*. The proceduressetServiceExamples and setServiceModels are used to write the attained parameters to the resultant variables.

The method *performCompositePropertySelection* is invoked by passing arguments user constraint U_C and *Optimizedworkflow* to generate *Concreteworkflow* (line 81). The *performCompositePropertySelection*method selects the optimal *ServiceExample* from the discovered *ServiceExamples* for each *ServiceModel* by using the non-functional properties and user constraint. The result of selection stage is *ConcreteWorkflow* that has optimal service examples for the service models.

The execution stage of this algorithm starts in line 82, the *produceExecutableServices* method gets input as *ConcreteWorkflow* from selection stage and generates a set of executable workflows. The generated executable workflows by the execution stage that are published temporarily as web services, these can be accessed by any client application. If the workflow has nested plan, it returns the inner AWSC process in the form of *ExecutableWorkflow* as denoted in line 87. In the invocation of inner AWSC by parent AWSC, the parent AWSC can obtain the *ServiceExample*set from the inner AWSC shown in line 75. The invocation outputs of *ExecutableWorkflow* are retuned

in line 89. This multilevel process for AWSC is invoked when prevalent services located on the multilevel nested structure are generated. Subsequently, the inner AWSC can be invoked recursively and the nested automatic dynamic composition structure can be generated.

6 Implementation and experimental feasibility analysis

This section describes implementation details of the unified composition algorithm through motivating scenario. We also analyze the feasibility of the performance and illustration of experimental results.

6.1 Lower-level implementation

The lower-level implementation of unified composition algorithm is based on proposed framework for the motivating scenario, since the framework presents the abstract model of multilevel workflow orchestration. As we described the problem formalization in Sect. 3.3, our implementation mainly focus on three parts, the workflow generation (planning) phase, the discovery phase and the selection phase. In addition, it focuses on two new parts namely transformation engine and multilevel orchestration.

6.1.1 Workflow generation through planning

The workflow generation for dynamic service composition is generally achieved using AI planning techniques under semantic aware services. Our multilevel workflow generator can generate workflow in automatically using Hierarchical Task Network (HTN) planner. The multilevel workflow generator can encode any format of representation of user specifications and also OWL-S process. We adapted the graph based planning with user constraints for defining the characteristics of the HTN planning. The graph based planning technique generates the abstract plan using task decomposition which involves decomposing the goal task into several subtasks and continuing to apply the decomposition in recursively until the resultant subtasks can be achieved. The multilevel workflow generator can produce an abstract task workflow with a set of constraints for the service discovery phase which attempts to find a composition solution that satisfies the associated constraints and identifying a set candidate services with respect to the given task workflow.

For instance, as per the definition 2 in Sect. 3.3, a set of tasks generated for abstract workflow using decomposition of our motivating scenario is represented in Table 1. In this workflow, the goal task *homeLoanApproval* is decomposed into several subtasks in iteratively until the resultant subtasks are produced by the planner.

6.1.2 Semantic web service discovery

We divided the semantic web service discovery (SWSD) into two separate parts such as (i) mapping of WSDL into a semantic web service during service registration, (ii) semantic service discovery. The primary reason for dividing the SWSD into two separate parts, we find the SWSD is a complex process and in order to reduce the complexity of whole discovery process.

6.1.2.1 Mapping of WSDL into semantic service during service registration Our implementation supports the mapping of WSDL into semantic service descriptions during the service registration. Our proposed framework contains one of the primary components called semantic service repository that constitutes of two parts of databases such as (i) Ontology based Service Registry (OSR), and, (ii) Domain Ontology. The OSR is a repository of services which stores semantic service description document for each service. The semantic service description of a service can itself used to specify the capabilities of the service which is searched during semantic service discovery (SSD). The existing web service descriptions are described in syntactic in nature, so that, the syntactic description of web service i.e. WSDL document must be mapped into semantic service description before it stored in OSR. In order to convert the WSDL document to semantic description of service, we used a domain ontology which is an ontology repository and contains common real world concepts. Semantic service repository also uses WordNet to provide comprehensive ontological concepts for domain ontology. We implemented a conversion function as specified in the proposed algorithm, which initially performs parsing the WSDL of web service, then extracts the details from service description, and then fill the needed properties in OWL-S service profile. But, WSDL file does not include the needed properties to construct OWL-S. So, the required non-functional properties can be extracted from user constraints U_C. This conversion process considered the conversion of XSD types such as primitive and complex types. The primitive type of XSD is converted directly into OWL-S. But, the conversion of complex type of XSD to OWL-S is performed by searching the most suitable related ontologies stored in OSR. In case, the suitable ontology is not found in OSR, then the conversion function searches in WordNet and otherwise it searches the needed ontology from public ontology repositories. After completion of conversion of XSD types into ontological concepts, our conversion function can register the new semantic service into semantic service repository. Also, the

Table 1 Abstract workflow generation through planner

Tasks generation for abstract workflow

homeLoanApproval (getLoanRequest (getApplicantLoanInfo (....
(findCreditWorthiness (salary, assets, liabilities),
assessCreditWorthiness (loan_amount, credit_worthiness_value),
loanApprovalStatus (estimatePropertyEvaluation (....
sendResponse (loan_granted, loan_not_granted))))))

Constraint associated with workflow

$$\begin{aligned}&\textit{Constraint} \equiv (\textit{rasisedBy}\ (\textit{user} \cap \forall\, \textit{hasName}\ .\ (\textit{Rahul}\ )\ ) \cap\\ &\exists\, \textit{homeLoan}\ .\ (\textit{loanAmt} \geq 2500000\ (\textit{hasIndianRupees})\ ) \cap\\ &(\textit{loanProcessed} \leq 2\ (\textit{hasDurationWeeks})\ ) \cap\\ &(\textit{interestRate} \leq 7.8\ (\textit{hasInterestValue})\ ) \cap\\ &(\textit{loanApplied} = 20170502\ (\textit{date}) \cap \textit{bankLocation}\ .\ \textit{Chennai} \cap\\ &\exists\, \textit{bankName} = \text{'SBI'}\ (\textit{hasBankName})\ )\ )\end{aligned}$$

input, output and other necessary parameters of syntactic service is mapped to semantically using the concepts in WordNet and makes the syntactic-to-semantic mapped services directly searchable. Additionally, we used theory of service substitution [4], which describes the semantic relations between ontological concepts of WordNet to develop the semantic relations among the web services.

6.1.2.2 Semantic service discovery The semantic service discovery (SSD) process is primarily used to identify the suitable service instances that must be matched with tasks of the optimized workflow. We consider G_i^T and G_j^T be the graph based nodes (tasks) of optimized workflow for matching with the semantic service instances R and S respectively. The degree of match between two tasks of optimized workflow such as $R \in V(G_i^T)$ and $S \in V(G_j^T)$ is calculated based on the precondition (T_{Pre}) and effect (T_{Eff}) of tasks T_i and T_j matching with the corresponding service instances R and S respectively. For service instances matching with tasks, we used a condition which impose S matches a request of R, only if, (a) effect (T_{Eff}) i.e., output offered by S match effect (T_{Eff}) requested by R, and (b) precondition (T_{Pre}) i.e., input provided by R match precondition (T_{Pre}) required by S. To provide a solution for discovery problem, as we taken the preconditions of R and S, the degree of match between $V(G_i^T)$ represented by R and $V(G_j^T)$ represented by S, the whole formulation is denoted by $dom_{PRE}(R, S)$, is calculated as follows.

$$\mathrm{dom}_{\mathrm{PRE}}(R,S) = \frac{\sum_{u \in \mathrm{PRE_S}} \max_{v \in \mathrm{PRE_R}} \{\mathrm{sim}\ (u, v)\}}{|\mathrm{PRE_s}|} \quad (1)$$

such that $PRE_R$and PRE_S represent the set of preconditions of R and S, respectively. Similarly, as considered the effects (T_{Eff}) of R and S, is denoted as follows.

$$\mathrm{dom}_{\mathrm{EFF}}(R,S) = \frac{\sum_{u \in \mathrm{EFF_R}} \max_{v \in \mathrm{EFF_S}} \{\mathrm{sim}\ (u, v)\}}{|\mathrm{EEF_R}|} \quad (2)$$

We implemented the Semantic Service Discovery (SSD) processor by adopting the above formulas (1) and (2) to find the match between the tasks of optimized workflow and semantic service instances. This discovery processor compares the user constraints associated with optimized workflow with all the service instances in the ontology based service registry (OSR).

6.1.3 Service selection using non-functional properties

The service selection is mandatory for concrete composition that can identify a set of service instances for execution engine. The semantic service selector must select the most suitable service instances that can satisfy the non-functional (QoS) properties [14]. In our implementation, we used dynamic non-functional parameters that are changed during the composition. We considered the dynamic non-functional parameters such as availability, response time, throughput, reputation, stability etc., for service instance selection. In general, non-functional parameters are represented in abstract form that can be

converted into concrete form in order to make understand by the service selector. We used the transformation engine which can identify the non-functional parameters from user requests and also it converts them into the concrete form for service selection. In the Sect. 4.2, we presented our methodology for transforming the non-functional parameters using the transformation engine.

6.2 Experimental results and feasibility analysis

Our comprehensive algorithm provides an optimal solution for AWSC problem using modified five stage of composition, even it does augmented with multilevel workflow generation to handle the dynamic change and scalability during runtime of composition. So, it is considered a complicated task to provide a most suitable solution for AWSC problem, because of the difficulties such as providing an implementation solution for all stages of composition, generating an optimal workflow for planning, discovering and identifying the suitable candidate services from large search space, and finding an optimal composition solution. Even though some of the research implementations [18, 21, 22] concentrated on identifying a solution for optimal composite service, our algorithm combines all the required solutions applied in modified five stage of composition.

The experimental setup for implementation of our unified composition algorithm was conducted in the environment as follows: Intel core i5 CPU with speed of 2.50 GHz, running under Windows 7 OS. In order to analysis the performance of the proposed algorithm, we used two predominant standards such as OWL-S and WSMO for creating the semantic web services. Also, we created a set of semantically annotated web services from their corresponding WSDL file using OWL-S [23] and WSMO [24]. We implemented a set of WSML files using WSMO tool which provided goals and mediators. We looked at goals to specify user's functional requirements that used for discovering and mapping with candidate services. We used mediators in order to perform transformation between different kinds of data and different processing levels. We implemented a semantic service repository that stored semantic descriptions of thousands of web services. In addition, we used the Web Service Challenge (WSC) 2009 dataset that contains more than 40,000 concepts and 4000 service instances.

6.2.1 Analysis of automatic workflow generation

To conduct the experiments of automatic workflow generation for our comprehensive algorithm, we used multilevel workflow generator. Our multilevel workflow generator is implemented by combining the benefits of an efficient planner Xplan and HTN planer. In order to use Xplan with HTN planner for generating workflow in AWSC, we used a conversion tool called as OWLS2PDDL that converts OWLS service descriptions to corresponding planning descriptions for PDDL language. The generated planning descriptions are taken as input by Xplan to produce a workflow plan for service composition that matches a given functional requirement of user.

Our multilevel workflow generator initially pre-processes the user query and divides the given query into atomic terms and then matches all terms into ontological terms. The semantic relations of all terms are also processes and stored as ontological relational terms in ontology service registry (OSR). After the pre-processing is completed, then construction of workflow started against all these ontological relational terms by provided them as inputs to Xplan and HTN planners.

We evaluated our approach of automated workflow generation on varying the number of semantic web services (size of repository) as well as varying levels of nesting for each repository size. We tabulated pre-processing time, number of tasks generated for constructing the workflow and number of optimal workflow selected. We observed that there was a significant increase in pre-processing time from initial to subsequent runs on the same size of repository. Table 2 presents the performance results of our unified composition algorithm on automatic workflow generation for the user query.

The experiments were conducted by varying the levels of nesting on different size of repository. The performance of workflow generation is graphically presented in Fig. 6 where pre-processing time, number of tasks implemented and number of optimal workflow generated respectively, on varying the levels of nesting. The graphs exhibit incremental behavior of pre-processing time, number of tasks generated and number of optimal workflow generated by varying levels of nesting on fixed repository sizes of 1000, 2500 and 4000.

6.2.2 Analysis of semantic web service discovery

In order to evaluate the performance of service discovery of our unified algorithm, we used the repository of services and service instances of WSC 2009 dataset. We executed a various user queries based on our motivating scenario and presented a sample user query as listed in Table 3. The process of service discovery can automatically finding the service instances from the repository that matches the requested output parameters which should satisfy the post-conditions and requested input parameters which should satisfy the pre-conditions and the same process is described in Sect. 6.1.2.2.

Table 2 Performance on workflow generation

Number of web services (repository size)	Levels of nesting	Pre-processing time (ms)	Number of tasks implemented	Optimal workflow generation
1000	2–6	37	332	25
1000	7–12	46	596	34
1000	13–18	58	830	47
2500	2–6	48	489	28
2500	7–12	59	846	45
2500	13–18	72	995	57
4000	2–6	57	667	42
4000	7–12	77	980	60
4000	13–18	88	1347	85

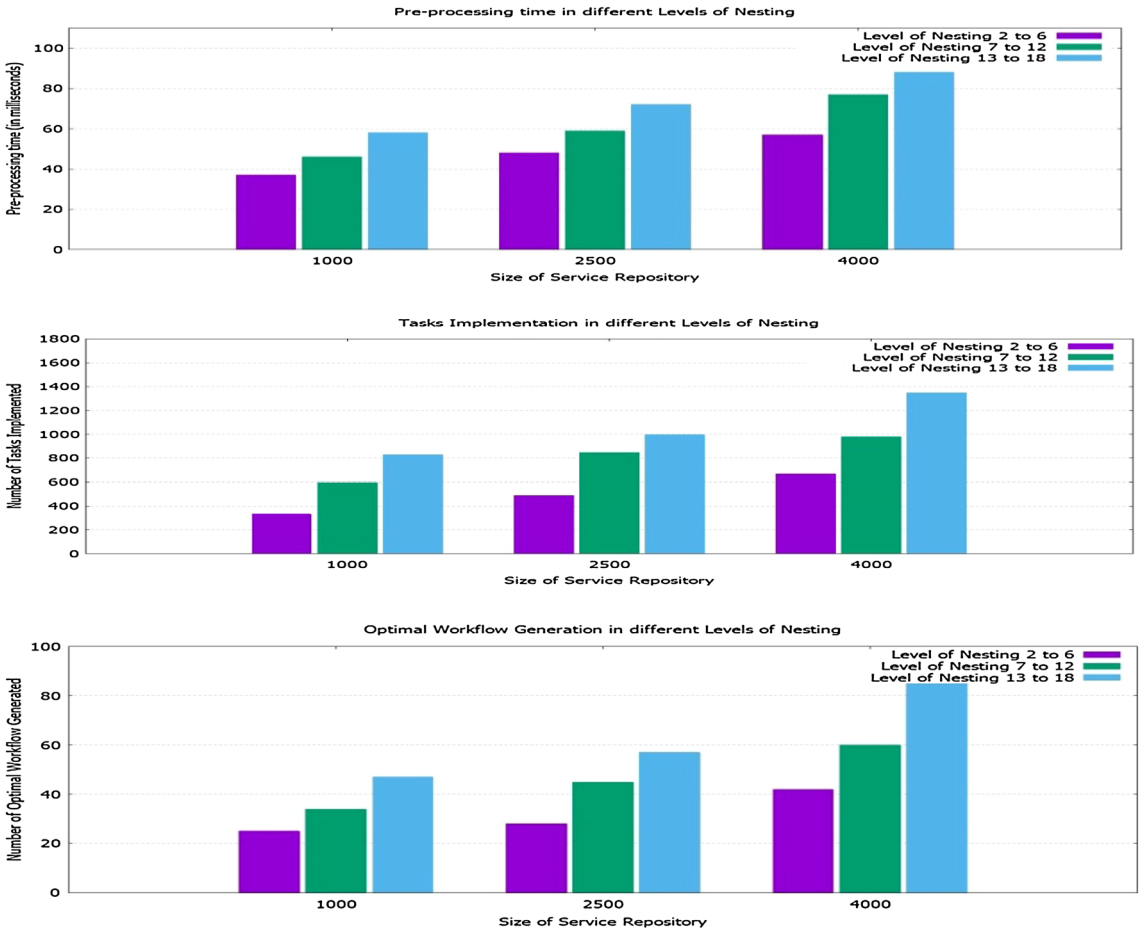

Fig. 6 Performance on workflow generation

We ran different types of discovery based matchmakers along with our unified algorithm (by holding a threshold = 0.5) on the repository of services and WSC 2009 dataset. We compared our discovery technique of unified composition algorithm against the different discovery based matchmakers such as Description Logics (DL), Keyword-based and Hybrid-based. Our discovery based matchmaker can combine concept level matching (concept and concept type) and graph based matching (graphpath, interior nodes and leaves). We adopted this combination of discovery techniques that can compensate the deficiencies of specific types of matchmakers. We describe the

Table 3 Semantic web service discovery—a sample query using motivating scenario

Semantic service	Input parameters	Pre-conditions	Output parameters	Post-conditions
User query	ApplicantName, ApplicantPersonalInfo, ApplicantLoanInfo, ApplicantBankAccInfo, ApplicantPanNum, ApplicationSubmitDate		LoanApprovedInfo, LoanRejectedInfo, LoanProcessingNum, LoanAccountNum	
CalculateCreditValue	ApplicantName, ApplicantLoanInfo, ApplicantBankAccInfo, ApplicantPanNum		CreditScore	
AssessCreditWorthiness	ApplicantName, ApplicantLoanInfo, ApplicantBankAccInfo, ApplicantPanNum	CreditScore $\leq$ 750	LoanRejectedInfo	LoanCancel-MesgToApplicant
		CreditScore $\geq$ 750	LoanProcessingNum	
PropertyAppraisal	ApplicantName, LoanProcessingNum	LoanProcessingNum Generated	LoanRejectedInfo	LoanCancel-MesgToApplicant
			LoanApprovalInfo	LoanApproval-MesgToApplicant
RejectLoan	ApplicantName	LoanRejectedInfo	LoanCancel-MesgToApplicant	
ApproveLoan	ApplicantName	LoanApprovalInfo	LoanAccountNum, LoanApproval-MesgToApplicant	

configuration of other types of matchmakers as follows: (a) Description Logic based matchmaker: uses an approximate matching filters for user requests and produces a ranking of service candidates with different degrees of match that is similar to SAWSDL-MX1. (b) Keyword-based matchmaker: uses a non-logic based similarity function for performing matching on a weighted value of synonyms and it is same as to Jaccard matching algorithm. (c) Hybrid-based matchmaker: uses the logic and non-logic based matching techniques which is similar as OWLS-MX.

6.2.3 Evaluation service discovery results

In order to evaluate the performance of different types of matchmakers and discovery technique of our unified algorithm, we used the evaluation metrics of information retrieval such as precision and recall. We calculated mean average of precision (MAP) [25, 26] that relates to the precision values of discovered semantic services returned by the matchmaker algorithm for all user queries at different recall points. We considered the evaluation of a single query is not sufficient toarrive a significant conclusion, so we calculated MAP over all user queries. We made a comparison of our unified composition algorithm with Description logic-based, Keyword-based and Hybrid-based as depicted in Table 4 and the same graphically represented in Fig. 7. We observed from the evaluation of various discovery-based matchmakers, our algorithm has produced the best Mean Average Precision of 0.75.

6.2.4 Performance analysis of composition process

To analysis the performance of our composition algorithm, we selected five measurable factors that influenced during entire composition process. The measurable factors such as (i) Transformation of NFPs, (ii) Generation of workflow, (iii) Workflow optimization, (iv) Discovery of services, and, (v) Selection of suitable services. These factors comprise composition process to provide an execution of workflow through services in a concrete environment. Also, we considered an important variable called level of nesting that impacts the entire performance of service composition. Apart from the variable level of nesting, other four variables (refer Table 2 in Sect. 6.2.1) that are explicitly affecting the composition process such as: preprocessing time for user query, number of tasks implemented for constructing the workflow, number of optimal workflow generated, and, the size of web service repository. From the analysis of experimental results of our composition algorithm, we plotted a graph (Fig. 8) and its corresponding table representation (Table 5). Figure 8 represents the execution time in milliseconds (msec) for the measurable factors that were increased proportionally while progressive increment in level of nesting. From our experimental analysis of composition process, we found the primary factor of increased in execution time during the service selection. We identified the service selection which consumed at maximum time (51 percent) of total composition time during the execution of our motivating scenario.

Table 4 Mean averaged precision/recall of different discovery algorithms

Precision	Recall			
	Description logic based	Keyword based	Hybrid based	Our unified composition algorithm
0	1.00	0.92	0.93	0.97
0.1	0.91	0.84	0.91	0.94
0.2	0.87	0.81	0.86	0.92
0.3	0.69	0.79	0.78	0.86
0.4	0.58	0.70	0.66	0.78
0.5	0.32	0.64	0.62	0.76
0.6	0.28	0.60	0.59	0.68
0.7	0.12	0.51	0.57	0.63
0.8	0.00	0.27	0.35	0.60
0.9	0.00	0.10	0.16	0.49
1	0.00	0.06	0.08	0.32

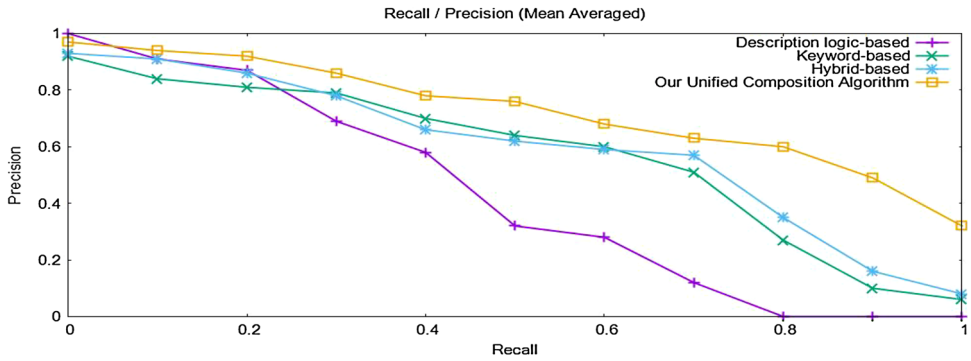

Fig. 7 Mean averaged precision/recall graph of different discovery algorithms

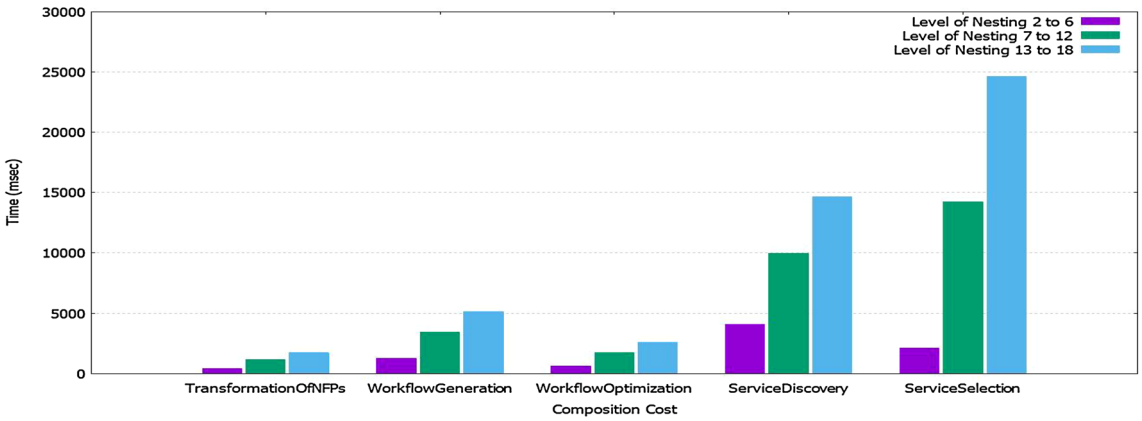

Fig. 8 Analysis of composition cost of AWSC

6.2.5 Discussion

We evaluated our composition algorithm through the measurable factors such as transformation of NFPs, workflow generation, workflow optimization, service discovery and service selection that are considered as affecting factors of the whole process of composition. The service selection is considered as primary affecting factor, since, it has to select more number of service instances and in-turn the selected services must matched with a task of generated abstract workflow. Also, we identified when increasing in level of nesting during workflow generation

Table 5 Analysis of composition cost of AWSC

Level of nesting	Transformation of NFPs	Workflow generation	Workflow optimization	Service discovery	Service selection
2–6	428	1284	642	4063	2128
7–12	1154	3461	1731	9957	14,235
13–18	1710	5130	2565	14,671	24,642

that proportionally increase the time in service selection. The composition cost of other measurable factors such as transformation of NFPs, workflow generation and optimization are consumed very small time during planning phase. The composition cost of service discovery is increased progressively to corresponding increment of number of tasks in generated workflow, however, its progressiveness is lies between workflow generation and service selection at most of the time. We observed the composition time of transformation of NFPs is linearly increased depends on complexity of its associated user constraints.

7 Conclusion

Due to the revolutionary advancement of web service technologies and increasing enormous amount of web services, we require dynamic and automatic service composition to reuse the prevalent services in efficiently. It is also mandatory that the web service composition solutions should be obtained in more optimal way. In this paper, our comprehensive algorithm uses the modified five stage composition for discovering and orchestrating web services based on semantic web technologies. With the use of semantic description of web services and ontological classes, our proposed algorithm can find the composite services matching for the user request. The entire composition process is done automatically without intervention of any manual process. Our algorithm finds the composition solution for any user requests, whether the given user request is simple or complex, and also it automatically generates the service composition in OWL-S descriptions. An optimization on composition is considered as another important affecting factor for AWSC. We implemented additional features with our algorithm to provide optimal composition solution, these features includes, transforming the non-functional properties of user requests to all stages during automatic composition and optimization on the resultant abstract workflows from planning stage using workflow automata.In order to provide seamless automatic composition, the non-functional properties from user requests were identified and converted into constraint expression to select most optimal semantic services. The workflow optimization is implemented as an additional phase in composition process, called as validation and optimization, it performs both structural and semantic validation on workflows and produced the optimized workflows. The discovery process is modified in our algorithm which is intended for identifying the suitable web services that matching with tasks in abstract workflow and additionally it performs transformation of syntactic description of web service (WSDL) into a semantic description using the standards OWL-S and WSMO. The process of transformation of WSDL to semantic description of web services is also done by annotating WSDL files with additional ontologies stored in ontology based service repository (OSR).

Our comprehensive algorithm provides a practical solution to enterprise service orchestration developers. The experimental evaluation and feasibility analysis for our motivating scenario proves the relationship among functional factors and also scalability of our algorithm for large scale of services and ontologies. We performed various experiments to analysis the performance of our unified algorithm and its results were also observed. The experiments conducted during automatic workflow generation stage that shows incremental behavior of pre-processing time, number of tasks and optimal workflow generated in efficiently when varying levels of nesting on different repository sizes. We conducted experiments to evaluate the performance of different types of matchmakers and discovery technique of our unified algorithm using the evaluation metrics of information retrieval. We observed from the performance of various discovery-based matchmakers, our algorithm has produced the best Mean Average Precision of 0.75. In addition, we analyzed the composition cost of our composition algorithm through the measurable factors such as transformation of NFPs, workflow generation, workflow optimization, service discovery and service selection and its experimental analysis shows that the method of generating composite service is seamless and scalable to handle largest collection of web services while increasing the levels of nesting.

Our feature research includes introducing fault-tolerant composition [27, 28] and investigating the different context

aware composition approaches to adapt more interactive process and robustness in composition solution. Furthermore, analyzing the choice of bio-inspired algorithms such as Ant Colony Optimization, Evolutionary Algorithm and Particle Swarm Optimization and other recent composition algorithms are also part of our future work. We are also discovering technologies and domain-specific languages in automated web service composition to develop a comprehensive framework for problem solving in software engineering.

References

1. Rao, J., Su, X.: A survey of automated web service composition methods. Semant. Web Serv. Web Process Compos. **3387**, 43–54 (2005)
2. Kona, S., Bansal, A., Blake, M.B., Gupta, G.: Generalized semantics-based service composition. In: IEEE International Conference on Web Services (ICWS), pp. 219–227 (2008)
3. Srivastava, B., Koehler, J.: Web service composition: current solutions and open problems. In: Proceedings of Workshop Planning for Web Services (2003)
4. Kona, S., Bansal, A., Simon, L., Mallya, A., Gupta, G.: USDL: aservice-semantics description language for automatic service discoveryand composition. Int. J. Web Serv. Res. **6**(1), 20–48 (2009)
5. Milanovic, N., Malek, M.: Current solutions for web service composition. IEEE Internet Comput. **8**(6), 51–59 (2004)
6. Thatte, S: BPEL4WS (Version 1.1). http://www.ibm.com/developerworks/library/specification/ws-bpel (2003)
7. Casati, F. et al.: Adaptive and Dynamic Service Composition in eFlow. In: Proceeding of 12th International Conference on Advanced Information Systems Engineering(CAiSE'00) (2000)
8. Pistore, M., Marconi, A., Bertoli, P., Traverso, P.: Automated composition of web services by planning at the knowledge level. In: Proceeding of 19th International Joint Conference on Artificial Intelligence (IJCAI'05) (2005)
9. Rao, J., Dimitrov, D., Hofmann, P., Sadeh, N.: A mixed initiative approach to semantic web service discovery and composition: SAP's guided procedures framework. In: International Conference on Web Services. ICWS'06, 2006, pp. 401–410 (2006)
10. McIlraith, S., Son, T.C.: Adapting golog for composition of semantic web services. KR **2**, 482–493 (2002)
11. Pistore, M., Roberti, P., Traverso, P.: Process-level composition of executable web services: on-the-fly versus once-for-all composition. In: Gomez-Perez, A., Euzenat, J. (eds.) The Semantic Web: Research and Applications, pp. 62–77. Springer, Berlin (2005)
12. Zhan, R., Arpinar, B., Aleman-Meza, B.: Automatic compositionof semantic web services. In: Proceeding of International Conference on Web Services(ICWS'03) (2003)
13. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using Shop2. J. Web Semant. **1**(4), 377–396 (2004)
14. Feng, Y., Ngan, L.D., Kanagasabai, R.: Dynamic service composition with service-dependent QoS attributes. In: 2013 IEEE 20th international conference on web services (ICWS), pp. 10–17 (2013)
15. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proceeding of Eighth International Conference on Knowledge Representation and Reasoning, pp. 482–493 (2002)
16. Ponnekanti, S.R., Fox, A.: SWORD: a developer toolkit for web service composition. In: Proceeding of 11th International WWW Conference (WWW'02), pp. 83–107 (2002)
17. Klusch, M., Gerber, A., Schmidt, M.: Semantic web service composition planning with OWLS-XPlan. In: Proceedings of AAAI Fall Symposium onSemantic Web and Agents (2005)
18. Dong, W., Jiao, L.; QoS-awareWeb service composition based on SLA. In: Fourth International Conference on Natural Computation (ICNC), vol. 5, pp. 247–251 (2008)
19. Yan, J., Kowalczyk, R., Lin, J., Chhetri, M.B., Goh, S.K., Zhang, J.: Autonomous service level agreement negotiation for service composition provision. Future GenerComputSyst **23**(6), 748–759 (2007)
20. Wada, H., Champrasert, P., Suzuki, J., Oba, K.: Multiobjective optimization of sla-aware service composition. In: IEEE Congress on Services-Part I, 2008, pp. 368–375. IEEE (2008)
21. Claro, D.B., Albers, P., Hao, J.K.: Selecting web services for optimal composition. In: ICWS International Workshop on Semantic and Dynamic Web Processes, Orlando-USA (2005)
22. Ramanathan, R., Latha, B.: Towards optimal resource provisioning for Hadoop-MapReduce jobs using scale-out strategy and its performance analysis in private cloud environment. Clust. Comput. (2018). https://doi.org/10.1007/s10586-018-2234-8
23. OWL-S 1.1: http://www.daml.org/services/owl-s/1.1 (2012)
24. WSMO:. http://www.w3.org/2005/04/FSWS/Submissions/1/wsmo_position_paper.html (2005)
25. Bellahsene, Z., Bonifati, A., Rahm, E.: Schema Matching and Mapping. Springer, New York (2011)
26. Algergawy, A., Nayak, R., Siegmund, N., Ḱoppen, V., Saake, G.: Combining schema and level-based matching for web service discovery. In: ICWE (2010), pp 114–128 (2010)
27. Vijayakumar, K., Arun, C.: Automated risk identification using NLP in cloud based development environments. J. Ambient Intell. Hum. Comput. (2017). https://doi.org/10.1007/s12652-017-0503-7
28. Vijayakumar, K., Arun, C.: Continuous security assessment of cloud based applications using distributed hashing algorithm in SDLC. Clust. Comput. (2017). https://doi.org/10.1007/s10586-017-1176-x

U. Arul has received his Diploma in Computer Technology from Directorate of Technical Education in 1994. He has received his B.E. degree in Computer Science and Engineering from Madras University in 1998. He received his M.E. degree in Computer Science and Engineering from Anna University in 2007. He is at present a Ph.D. Research Scholar in Computer Science and Engineering pursuing from Anna University. Further, currently he is working as Associate Professor in the Department of Computer Science and Engineering at Dhanalakshmi College of Engineering, Chennai, India. His research areas are Internet Technology, Semantic Web, Ontology Learning and Cloud Computing.

S. Prakash has received M.E. in the Department of Electronics and Communication Engineering at B.I.T., Mesra, India, in 1992 and received Ph.D. in the Department of Instrumentation at I.I.Sc., Bangalore, India, specializing in the area of amorphous memory devices in 1997. Then, he worked as a research fellow in the Department of Electrical and Computer Engineering, at NUS, Singapore on Ohmic contacts on GaN materials (September 1997–February 2000). Further, he continued his research as post-doctoral fellow on TFT's for imaging and display application in the Department of Electrical and Computer Engineering at University of Waterloo, Waterloo (January 2000–May 2002). At present, he is working as a professor in the Department of ECE at Jerusalem College of Engineering, Chennai, India. He has more than 20 years of research cum teaching experience in Indian and abroad universities. He has authored and co-authored in more than 20 research papers at several International journals and conferences. Under his guidance three have received their Ph.D. from Anna University and 13 research scholars from different universities are pursuing their research. His areas of interest are fabrication and characterization of amorphous devices, simulation and emulating memristor, computer networks and VLSI design.