Thirumaran and Brendha *Hum. Cent. Comput. Inf. Sci.* (2016) 6:13
DOI 10.1186/s13673-016-0067-0

Human-centric Computing and Information Sciences

RESEARCH Open Access

Incremental stages of a semantic framework for automating the changes on long term composed services

M. Thirumaran and G. Gayathry Brendha*

*Correspondence:
gayathry.brendha@gmail.com
Department of Computer Science and Engineering, Pondicherry Engineering College, Puducherry 605014, India

Abstract

Enterprises enhance their business on the web with the help of web services. This enhancement is achieved by composing the pre-existing services, so that they will be able to provide solutions for the problems on the web. Due to rapid development in technology, the need for making changes in the composed services by the respective analysts becomes an essential task. Thus, the change management process becomes a challenging area in web service. Although, the existing solutions use ontology for change management, they have been designed majorly for IT developers rather than analyst. Therefore, we concentrate on providing a change management framework that will make use of an enriched ontology set and semantic reasoner for implementing the changes by the analyst itself. The semantic reasoner component parses the change request from the analysts and determines the possibility for making the change. The framework represents the implemented changes in the form of a S-BPEL notation (Semantic BPEL) which is then converted to their corresponding BPEL notations by a BPEL constructor so that they can be deployed in the run time environment for composing the services.

Keywords: Web services, Change management framework, Semantic reasoner, Ontology, S-BPEL, BPEL

Background

Problem

Service oriented computing helps in outsourcing the business functionalities from web service providers, so that these functionalities can be composed by service composition to create solutions for problems occurring over the web. A composed Web service is therefore an on-demand and dynamic collaboration between autonomous Web services that collectively provide a value added service [1]. The composed services are of two types: Short term and Long term composed services (LCS) [2]. The collaboration among the LCS will be for a longer duration, on the other hand the collaboration among the services in short term composed services are dissolved when the requirements are met [3]. Changes in web services usually occur due to various factors like rapid technological development, new requirements from the end-users [4], complaints, to improve the enterprise business standards so that it compete with other business enterprises etc.

Therefore, providing a framework that will help the analyst to make the changes is of prime importance. These changes are classified into two types: top- down and bottom-up [1]. The changes arising from the enterprise owners refer to top-down changes and those arising from the outsourced providers refer to bottom-up changes. Our framework is designed to work out the problems that may arise in top-down changes.

Motivation

Service oriented applications may undergo frequent changes due to the regularly changing [5]. As a result, it is important to make changes in the member services and also in the manner in which the services are co-operated with each other. In general, Change management refers to the task of addressing changes in a timely, planned and systematic manner [6]. But, this task is difficult because the XML representation of Web services guarantees syntactic interoperability but it is unable to semantically describe services [7, 8], in addition, during such change requirements, the analyst must also ensure that various factors like the relationship between the existing services after the change, the correctness of the corresponding services involved in the change etc. are also preserved. Secondly, to implement the changes IT professionals were required as the analyst could not perform the changes, this increased the cost and time for the enterprises because the existing frameworks were designed for IT people. Further in the existing systems, there was no guarantee on the success of the requested change before its implementation. i.e. the result of the change (success or failure) and also the impact that will be caused on the value added services will be revealed only after the changes has been implemented in the service level. Due to this, establishing the consistency within a service and between the services will introduce high cost to the enterprises.

Thus, all the above stated challenges motivated us to create a framework that will assist the analyst to govern the incoming change requests successfully.

Solution

This research provides a solution for the analysts to perform the changes by himself without any assistance from the IT professionals. To achieve this goal, our framework uses an enriched ontology set which contains information regarding the relationship between the services, properties concerned with each service etc. It resolves ambiguous description of service functionality and external interface. It also reduces human intervention while integrating services in service-oriented architecture (SOA) [9]. Since, the ontology furnishes the framework with sufficient information based on the change request from the analyst; the process of deciding on implementing the changes becomes an easier task. The framework is designed with special components like: Semantic reasoner and Semantic analyser. The semantic analyser will parse the change expression and determine the possibility for making the change. The possibility for the change is decided with the help of the semantic reasoner which uses the semantic rules to provide the appropriate information to the analyser during the decision making process. Our framework is supported with number of important components like:

Enriched ontology set—our framework leverages the web services to a higher level by representing them using ontology like OWL [10]. Unlike the existing frameworks that use ontology only for retrieving domain information, our framework represents

the services in terms of its relationship between services etc. As a result, the reasoning capacity of the framework is increased which increases the success rate of implementing the changes.

Semantic analyser—we use an analyser to determine the possibility of executing the changes before they are implemented at the service level. The information for this determination is obtained from the semantic reasoner component.

Semantic reasoner—this component is the heart of our framework, it contains the semantic rules that will be used by the semantic analyser to gather information for making decision on the feasibility of making the changes. The reasoner is also powered by the change impact analysis, which has the capability to forecast the condition of the SOE when the requested change is performed.

Semantic schema generator—once the change is found to be plausible, we verify the changes in a semantic schema generator rather than implementing the changes directly in the service level. The component makes use of push down automata (PDA) and finite state machine (FSM). If the changes implemented at semantic schema level are found to be satisfactory to the analyst, he can proceed for implementing at the service level.

Scenario for explaining the framework

Our semantic change management framework is analysed by considering three domains: travel agency, banking and telecom domain. The travel agency has three operations: Booking a flight ticket, booking a cab and then a hotel.

This section provides a detail description about the semantic change management framework [11] by considering the travel agency scenario. Let us consider an analyst maintaining a travel agency which consists of sequence of services like book airline, book hotel. Each of these services has their corresponding rules and policies that are indicated as R11, R12, p1 and p2.

A person who needs to avail these services has to first login by providing his credentials like personal information, travel details such as departure date etc. All these information are expected to be in a secure form for which the agency provides a policy for encryption. After providing the details he is directed to book his air ticket and also his corresponding hotel services. The details of his hotel booking are sent back to the customer in hashed form. The operations getting the personal information, booking flight and cab should be performed sequentially only, i.e. these operations can be performed only one after another, whereas the other operations such as hotel booking and registering for weather services can be performed in a parallel manner.

Related work

The change management framework [12, 13] discussed by Akram et al. combines the ordinary and re-configurable petrinets to deal with the incoming bottom up change requests. Algorithms called change detection, change management and change reaction algorithms are executed on these petrinets to maintain the workflow of the services. The changes implemented are not verified and further the construction of petrinets becomes challenging when the number of services to be included increases. The same author in the architecture described in [1] uses request brokers for performing the requested change. The ontology used is a domain ontology which is used by the request brokers

to decompose the request, select the appropriate services, invoke the selected services, perform the changes and finally to provide the results to the user. Allocating an instance of the broker to each user becomes an overhead in this architecture. The change management framework by Xumin Liu [2] deals with top down changes. The change to be made is first observed in the schema graph, a graph derived from the ontological representation. After successful verification the change is implemented at the instance level. Although semantics has been considered, it deals only with single requests at a time and the essence of ontology used for the purpose of change management is also very limited. Xumin Liu along with Bouguettaya try to automate the process of making top down changes by proposing a change management framework which has two components called change model and change reaction. The work of change model is to specify the requested top-down changes, while the goal of change reaction is to enact the changes. Apostolou et al. [6] present an ontology-based approach where systematic response of e-Government systems is obtained to by applying formal methods. They have claimed that such a synthesis of systematic response to changes with knowledge to deal with them has a positive impact on the change management process.

Aissi et al. [14] has used ontologies for developing the SLOAP recommendation system (SLOAP) to enhance the exploration in spatial data warehouse (SLOAP) by means of querying the warehouse. The dimension, measure, attributes of the data warehouse are represented as concepts in the ontology. The similarity between the queries is determined by minimum number of edges [15] which separate the query references in the ontology which is called the Rada distance. The queries are then analysed and candidate queries are determined by ranking method which are recommended to other similar users who will approach the warehouse in the future. Rafiei and Kardan [16] help to find experts in online communities based on content analysis and social network analysis. The content analysis uses concept maps a kind of knowledge representation to find the semantic similarities between two words by taking into consideration the meaning of the words. Though this approach was found to be very useful for our change management framework, this cannot be used as concept map creation requires high expertise about the domain. The Table 1 discusses the advantages of our framework over the existing change management framework. The above change management works require IT professionals to perform the requested change. Due to this the cost and time spent for these people increases. Our work aims in creating an environment to make the changes by the analyst itself without the involvement of the IT people. Further, the possibility of making a successful change is not verified before it is being implemented at the service level. Therefore, our framework uses a parsing technique to determine the possibility of making the change so that the burden at the analyst side is reduced. Finally, the observed change management frameworks use petrinets which has many disadvantages like petrinets can be represented only as tasks and are useful for change management scenario where the changes are made by the IT developers. But in our framework we use FSM which are designed in terms of states and actions, as result all the services are represented in terms of states so that the exact state in which the requested change has to be incorporated can be easily known by the analyst. Since petrinets are complex structures they require higher memory capacity for its storage due to which there is high degree for reaching undesirable states.

Table 1 Comparison of existing change management frameworks

Feature	Existing frameworks	Semantic change management framework	Reason/component contributing
Memory usage	High	Low	
Prior determination for implementing changes	No	Yes	Semantic parsing
Automatic verification and validation	No	Yes	Finite state machine
Probability to reach correct states	Low	High	Push down automata
Professionals involved in enacting the changes	Developers	Analyst, developers	Semantic information from ontology
Information provided in the ontology	Domain	Functionality of the services	
Information about the relationship between the functionality of the services	No	Yes	Representing ontology in terms of functionality of the services
Rule/policy level information about the services	No	Yes	Representing ontology in terms of functionality of the services
Prior determination on the impact caused by the requested change	No	Yes	Change impact analyser component
Capacity of the framework to provide reasons in case of failures	No	Yes	Semantic reasoner component
Possibility to determine the exact position for successful implementation of change	No	Yes	Push down automata

Semantic web service change management framework

Figure 1 represents the stages and the components of our framework. The framework is powered by three main components: Enriched ontology set, semantic analyser and semantic reasoner. We will discuss in detail about these components in the forth coming sections.

Tentatives

The framework requires some initial processes to be performed on the change request, so that it can be processed by the framework. These are achieved by the following components:

Request analyser

Initially, the change request from the analyst is screened to determine the domain and also the context of the change. This information is retrieved from the service repository where the concrete web services are stored.

LCS grammar

The service repository contains information about individual services and also about the value added services. To make the process of parsing easier, we represent each LCS in terms of grammar symbols. An LCS in general is represented as:

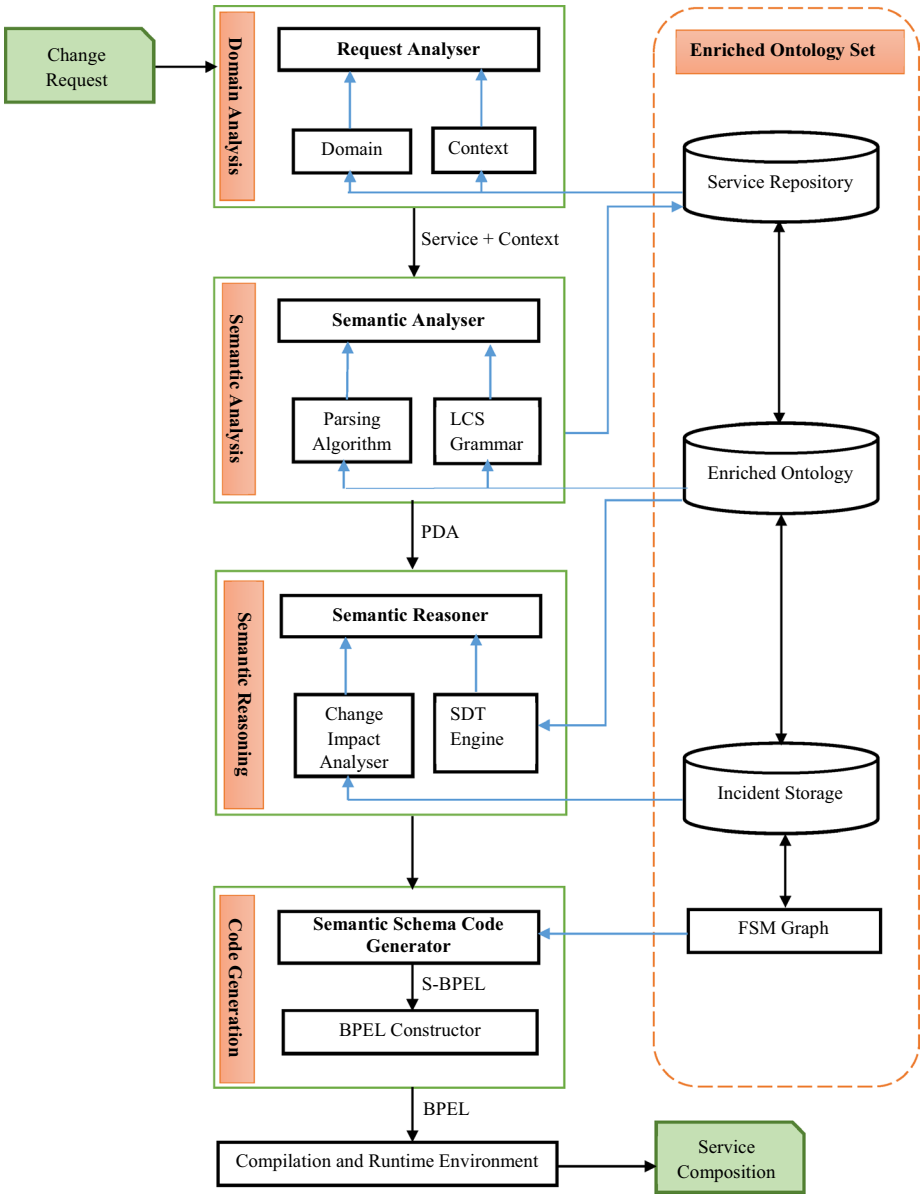

Fig. 1 Stages of a semantic change management framework

$$L \rightarrow \{S_1, O, S2, O \ldots . S_n\}, \tag{1}$$

$$S_1 \rightarrow \{op_1[R/P], O, op_2[R/P], \ldots \ldots \ldots op_n[R/P] \tag{2}$$

$$S_n \rightarrow \{op_1[R/P], O, op_2[R/P], \ldots \ldots \ldots op_n[R/P] \tag{3}$$

where S_1,.....S_n represent the services present in the LCS, op_1.....op_n are the operations that make up the individual services, [R/P] is the rules and policies that have to be followed while considering the operations and O represents the pattern (Sequential or parallel) in which they are co-operated.

Enriched ontology set

To increase the reasoning capacity of our framework, we build an enriched ontology set. The other frameworks use only domain information for making the changes, whereas our framework represents the functionality of the services. As a result, the exact position [rule, policy or operational level] where the change has to be implemented can be easily identified by the analyst.

Semantic analyser

The semantic analyser helps to determine the feasibility of the requested change. This feasibility is achieved by parsing the change request with the help of the LCS grammar and the SDT engine of the semantic reasoner which will be elaborated later in this section. The parsing can be: Top- down parsing, Bottom-up parsing or workflow verification. If the change request is to form a new LCS or a new set of value added service or to add a new service in the existing composition, then we use top-down parsing. In top-down parsing the existing individual services for composing a new LCS are selected based on the information provided by the LCS grammar and enriched ontology set. The selected services are then parsed to determine the rules and policies that has to be included while composing the services. If the request is for making changes in the existing composition like adding a new service, removing a service or rule or policy from the composition, then we use bottom-up parsing. This parser makes use of the SDT engine where the semantic rules about the LCS grammar are stored.

If the request if for adding a new individual service to the already existing services, then the analyser will parse the change request and forecast the services that has to be preceded and succeeded after the new service that has to be added. Similarly, if the request is for removing a service or rule or policy, the parser based on the information provided by the SDT will forecast the services, rules or policies that will be affected due to the removal. Based on this forecasted result, the analyst can decide on whether to remove the service or not. For e.g. Consider an enterprise for travel LCS has four services as: get_userdetails, bk_fight, bk_cab and sms_initimation, if the analyst attempts to delete the get_userdetails, then the parser with the help of the information provided in the SDT will forecast and suggest that the service must not be deleted. For this purpose, the analyser will first generate the list of item sets, this item set will retrieve the semantic information for each grammar symbol from the already existing SDT. After the completion of item set generation, a parsing table is computed which tells the semantic references for each grammar symbol. Finally, parsing is performed, which uses the information from the item set generation and parsing table and determines the feasibility for implementing the change.

If the change request is to verify its workflow, then the composition based operator precedence parser is used. Based on the context and change operators appearing in the change request, the services that can be executed in sequential, parallel, conditional etc. are determined. The determined set of services is compared with the change request, if the services in the determined set and the request are found to appear in the same order, then we determine that the workflow for the request is valid. For instance consider our running example where the change request is: l + e + c* s [+: sequential operator, *: parallel operator]. Here, "l" refers to get_userdet, "e" refers to an airline service,

"t" to an hotel service, "s" to the sms service. According to the scenario, first an airline service has to be executed, then a cab service, the third in the workflow can be a hotel or sms service or both, this implies that the airline and cab service has to be executed one after other. i.e. in a sequence, whereas the hotel and sms services can be executed in parallel. Although there are two sequential operators in the request we have considered, the get_userdet have to be executed first, this order in which they have to be executed will be dictated by the parser. For this purpose based on the context of the request, the appropriate information from the ontology is used to determines the precedence for each operator, following this precedence table is constructed which is referred by the parser to verify the workflow.

Semantic reasoner

The semantic reasoner is built up of two components: semantic description engine and the change impact analyser.

Semantic description engine The semantic reasoner with the fetch the semantic information from the enriched ontology set to create rules for each existing LCS. These rules will be represented in terms of grammar notations discussed in "LCS grammar" section. The rules provide information about the rules, policies, relationship with the other services for each symbol of the change request. Since the semantic information are written in the form of rules, for each change request, instead of searching the entire the ontology for the relevant information, the information for the appropriate LCS is alone searched, thereby reducing the search space for the change request. The rules in the SDT are used by the parsers viz. top-down, bottom-up and operator precedence parsers for dictating about the feasibility of the change request.

Change impact analyser Before any change is to be executed, the change impact analyser will check the incident storage database to determine whether the requested change has been recorded previously, if a similar change has already been executed, then the impact caused by those changes are studied before the analyst proceeds to implement the changes. For the current request, the threshold estimation, similar change requests pattern, impact value estimation are calculated and then compared with the estimations that were made on pre-occurred services. The purpose of this comparison is to assure risk free implementation of the change request. For e.g. if the enterprise wants a change to be made immediately, then the analyst cannot wait till the IT team completes the change, therefore, there arises a need to implement the changes by himself. In this case, the analyst without knowing the impact that would be caused on the LCS by the change cannot proceed to execute the changes directly on the service logic, because executing an incorrect change on the LCS would make the changed LCS to behave in an undesired manner. Therefore, it is necessary that the analyst gains detail knowledge about the impact so that he can assure a risk-free incorporation of the changes in the LCS. If there are no pre-occurred services, then that new incident is stored in the database.

Semantic schema generator For each LCS a corresponding PDA is constructed, a PDA's schema in general will consist of the input state, a stack and the output state. In our

case, the input state will be a service/rule/policy, the stack symbol will consists of the hidden factors and the output state consists of the service/rule/policy which we want to be changed. When any changes are made in this schema, it will be reflected in the PDA structure of the LCS. Therefore, the changes that have been decided for execution can be verified using this PDA before they are being implemented at the service level. In our running example, consider the analyst wants to replace the existing airline service to another service due to QoS constraints [17, 18], then the PDA schema can be represented as" e, [response time], j", where, "e" is the emirates service and "j" is the jet services. When the change has found to be feasible by the parser, then it is implemented at the schema of the PDA. If the change is found be satisfactory and found that the correct behaviour of the LCS is maintained, then it can be proceeded for executing at the service level.

The changes in the service level are implemented with the help of the Finite State Machine (FSM), the FSM will maintain the event and the state information of each corresponding services involved in the change management process until the entire change enactment process is completed. The state information for managing the changes is obtained from the ontology. The event in our case refers to the action that is involved in invoking a particular service. Since the services are represented in terms of states. i.e. rule, policy, the exact position where the change has to be implemented is easily identified by the analyst. Thus from this FSM, a schema code is generated for each change and stored. The schema code at this stage will be in the form of S-BPEL since it also contains the semantic information. Since the state information is maintained at each stage, if the analyst wants to refer the process that was involved for a particular change, then the FSM states that were involved in the corresponding change process can be referred.

BPEL constructor

Since, the outcome of the Schema code generator is in the form of a S-BPEL notations, it cannot be directly executed on an run time [19] environment directly as the system cannot interpret the semantic symbols directly. Therefore, we use a BPEL constructor which converts the S-BPEL notations to the BPEL symbols.

After the conversion, these constructors can be used to directly deploy them in an run time environment or they can also be used to combine the changed LCS with other services and enable them to share data.

Experiment methodology

This section provides a detailed discussion on the implementation of PDA and the parsers.

Push down automata

Our change management framework uses the PDA for verifying the possibility of making the changes. The PDA implementation for our running example is indicated in Fig. 2. The Table 2 represents the LCS grammar for the travel scenario discussed in "Solution" section.

The services involved in the change enactment process are taken as states. The input symbol will be the actual service that is in composition, the constraint factor is the condition that has to be satisfied for choosing the alternate service instead of the actual

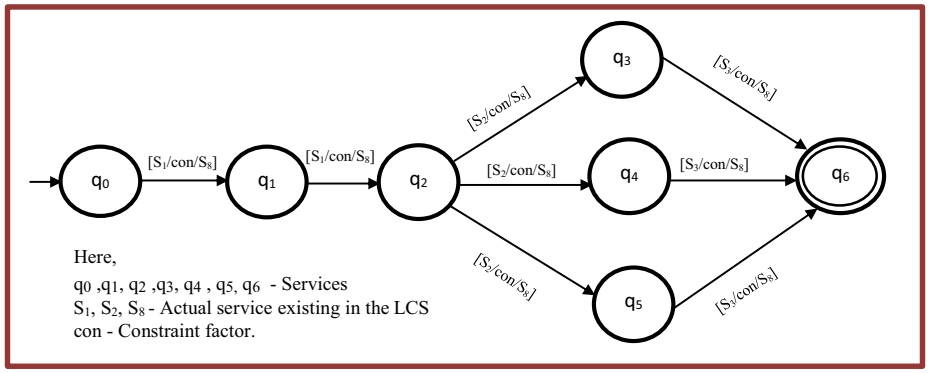

Fig. 2 PDA implementation example

Table 2 LCS grammar for travel agency scenario

Grammar	Explanation	Current composition	Change request
L → (S1.S2). ((S3.S4)*S5)	S1 = login		(op1/op2)/((op3/op4)*op6)
S1 → OP1	S2 = book flight ticket		
S2 → OP2	S3 = book cab		
S3 → OP3	S4 = book hotel		
S4 → OP4	S5 = weather services	(op1/op2)/((op3/op4)*op5)	
S5 → OP5	OP1 = get_credentials		
S6 → OP6	OP2 = get_cust details		
	OP3 = get_flightdetails		
	OP4 = get_returndate		
	OP5 = get_temperature		

service. This alternate service is selected from the list of reference services. The service that is found to satisfy the given constraint is chosen based on the information provided from the ontology for the given constraint.

For our travel scenario, if the analyst decides to replace the existing hotel service say Oberoi with Taj due some QoS constraints, then according to the PDA the actual service will be the Oberoi and the reference service will be the other hotel services. The services that are found to satisfy the given QoS (say availability 99.9 %) will be decided from the information stored in the ontology. The PDA schema for this case will be:

$$< Ob, \text{avail}\,(99\,\%), \text{Ta} > \rightarrow < O, \text{avail}\,(99\,\%), [\text{Ta, Ro, Ac, Ch, Ob, Me}\,] > \quad (4)$$

$$< \text{Ob, avail}\,(99\,\%), \text{Ta} > \rightarrow < \text{O, avail}\,(99\,\%), [\text{Ta, Ro, Ac}\,] > \quad (5)$$

$$< \text{Ob, avail}\,(99\,\%), \text{Ta} > \rightarrow < \text{O, avail}\,(99\,\%), [\text{Ta}] > \quad (6)$$

In the above schema, the input state is the Ob [Oberoi service], the constraint factor is the condition, i.e. the replacing service should provide its service always, the output service is the required alternate service that has to be replaced.

Initially, the stack consists of all the available hotel services from the reference set, next time when the push down automata reads the constraint, it refers the ontology and fetches only those services that satisfy the given required constraint, this set is then checked to determine if the output service requested by the analyst is present, if it is present then it actual service is replaced by the existing service.

Workflow parser for the scenario

This section discusses in detail about the rules that have to be followed for constructing a dynamic precedence table followed by the operator precedence parsing algorithm that is being used for determining the flow in which the request has to be performed.

Rules to construct the precedence table

First, the terminals and non-terminals of the extracted grammar as well as the change request expression are identified.

Non-terminals are the services considered while the terminals are the operations that form a service of the grammar. The composition operators specify the way in which the services have to be performed. Some of the composition operators are listed below in Table 3.

After the terminals and non-terminals are identified, the precedence table is constructed. Consider the symbols of the LCS grammar as α, β, the following are the rules to construct the precedence table:

1. If the symbols encountered are non-terminals, then the relationship between the symbols α and β is determined with the help of ontology and the non-terminal which is independent is given higher precedence. Eg: if β has higher precedence then the rule is written as $\beta > \alpha$.
2. If the considered non-terminals (α, β) are dependent on another set of non-terminals say (γ,δ) then both the pairs of non-terminals are given equal precedence ($\alpha = \beta$).
3. If the encountered symbols (α, β) are both terminals, then their corresponding non-terminal (based on the context) is determined. Then the rule 1 is applied.
4. If α is a non-terminal and β is a terminal then, corresponding non-terminal of β is determined and then rule 1 is applied. If the non-terminal of β is found to be α, then α is given the higher precedence.
5. If α and β are both composition operators then, the context is retrieved from the ontology based on *is follows* relationship.
6. If α is a composition operator and β is a non-terminal, then a check is made to determine if β is dependent on any of the other terminals (or) not. If β is independent,

Table 3 Composition operators and their purpose

Operator	Purpose
/	Sequential execution
*	Parallel execution
||	Concurrent execution
<>	Conditional execution

then it is given the higher precedence else the composition operator is given a higher precedence.

7. If α is a composition operator and β is a terminal, then the non-terminal related to β and based on the context, is determined, say γ. A check is made to determine if γ is dependent on any of the other terminals (or) not. If γ is independent, then it is given the higher precedence else the composition operator is given a higher precedence.

Algorithm for composition based operator precedence parsing

After the construction of a dynamic precedence table (M), we then make use of operator precedence parsing to determine the flow in which the requested change has to be performed (Fig. 3).

The change request in the form of an expression is provided as an input to the parser by the analyst. Based on the dynamic operator precedence table, the precedence of the operators involved in the change expression are verified. If the first symbol is found to have an higher precedence, then it is pushed on the stack, otherwise a reduce operation is performed with the help of the productions in the LCS grammar. Figure 4 depicts the algorithm in the flow chart representation.

Precedence table formulation for travel scenario

The precedence table (Table 4) for the above grammar is formulated dynamically with the help of the rules provided in the above section. From the Table 4, initially the encountered symbols i.e. are both non-terminals and are also same, therefore no composition is possible. Next, the encountered symbols are again non-terminals, as a result Rule 1 is applied and we find that S1 holds higher preference when compared to S2. When symbols S1 (α) and OP1 (β) are encountered, based on rule 4 we find the corresponding non-terminal of OP1 which is S1, then from rule 1 we determine that S1 has precedence, therefore its corresponding terminal OP1 will get higher preference when compared to S2. For cases where both are composition operators (/,*), from rule 6, the relationship "is

Algorithm Composition based Operator Precedence Parsing (ce, M)
Input: A change expression (ce) and a precedence table (M).
Output: order for executing ce.
Begin
set input (i) to point to the first symbol of ce$
if $ is present as the stack top and the symbol pointed by i is also $
Return
else begin
let α be the symbol present on the stack top and β be the symbol pointed by i
if α <. β or α = β**then**
push β onto the stack and advance i to the next symbol in ce
end if
else if α .> β **then**
perform reduce operation by replacing the symbol on the stack with its corresponding terminal production
Repeat
pop the stack
until
there are no more elements in the stack
End

Fig. 3 Algorithm for composition based operator precedence parsing

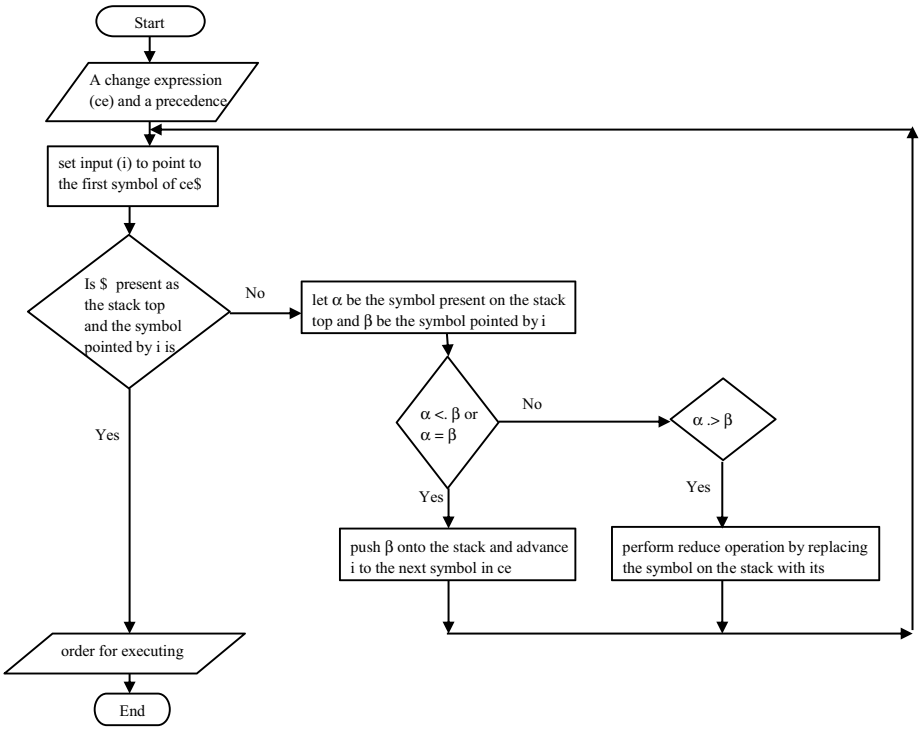

Fig. 4 Algorithm for composition based operator precedence parsing

Table 4 Precedence table for the change request in the travel scenario

Services	Services					Operators					Composition operators	
	S1	S2	S3	S4	S5	OP1	OP2	OP3	OP4	OP5	/	*
S1	–	.>	.>	.>	.>	.>	.>	.>	.>	.>	.>	.>
S2	<.	–	.>	.>	.>	<.	.>	.>	.>	.>	.>	.>
S3	<.	<.	–	.>	=	<.	<.	.>	=	.>	.>	.>
S4	<.	<.	<.	–	.>	.>	<.	=	.>	.>	.>	.>
S5	<.	<.	=	.>	–	<.	<.	<.	<.	.>	.>	.>
/	<.	.>	.>	.>	<.	<.	.>	.>	.>	.>	=	.>
*	<.	<.	<.	<.	<.	<.	<.	<.	<.	.>	<.	=

follows" from the ontology is retrieved which states that sequential operator (/) must be given the higher priority. Similarly, by applying the appropriate rules discussed in previous section, precedence for the encountered grammar symbols is determined.

Composition based operator precedence parsing for travel scenario

After the precedence table has been formulated, parsing is performed which will be done by the composition precedence evaluator component of our framework by following the algorithm in Fig. 5. The parsing for the change request of our scenario is indicated in Table 5.

Initially a $ symbol is added in the stack and at the end of the input i.e. the change expression. The algorithm in Fig. 5 is applied where first the $ and the first symbol of the change expression (op1) is compared, as a result op1 is given a higher preference. Based on the obtained results, a service composition tree as shown in Fig. 6 is composed whose leaves consist of services and the nodes consist of the service composition operator.

Therefore it is move to the stack, according to the algorithm in Fig. 5 if the element on the stack top is a non-terminal it is reduced to its corresponding terminal S1, now the top element on the stack which is $ and the next input composition operator are compared, thus op2 is given the higher importance and pushed to the stack which is followed by the reduce operation to S2.

Similarly, we parse all the symbols of the input till the $ symbol is reached. Finally, all the stack elements are popped from the stack, to obtain the order in which the services have to be executed.

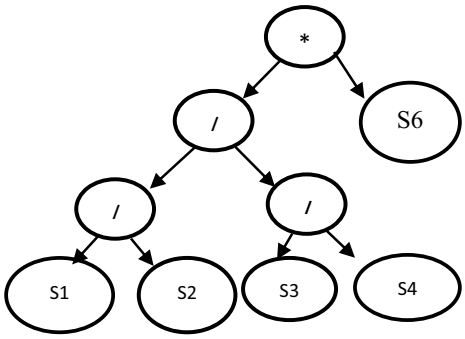

Fig. 5 Service composition tree for travel scenario

Table 5 Composition based operator precedence parsing

Stack	Precedence	Input
$	<.	op1.op2.op3.op4*op6$
$op1	.>	op2.op3.op4*op6$
$S1	<.	op2.op3.op4*op6$
$S1	.>	op2.op3.op4*op6$
$S1.op2	<.	op3.op4*op6$
$S1.S2	<.	op3.op4*op6$
$S1.S2.	.>	op3.op4*op6$
$S1.S2.op3	.>	op4*op6$
$S1.S2.S3	<.	op4*op6$
$S1.S2.S3.op4	.>	*op6$
$S1.S2.S3.S4	<.	*op6$
$S1.S2.S3.S4*	.>	op6$
$S1.S2.S3.S4*op6	.>	$
$S1.S2.S3.S4*S6	.>	$
Final composition expression	*$S1.S2.S3.S4*S6$*	

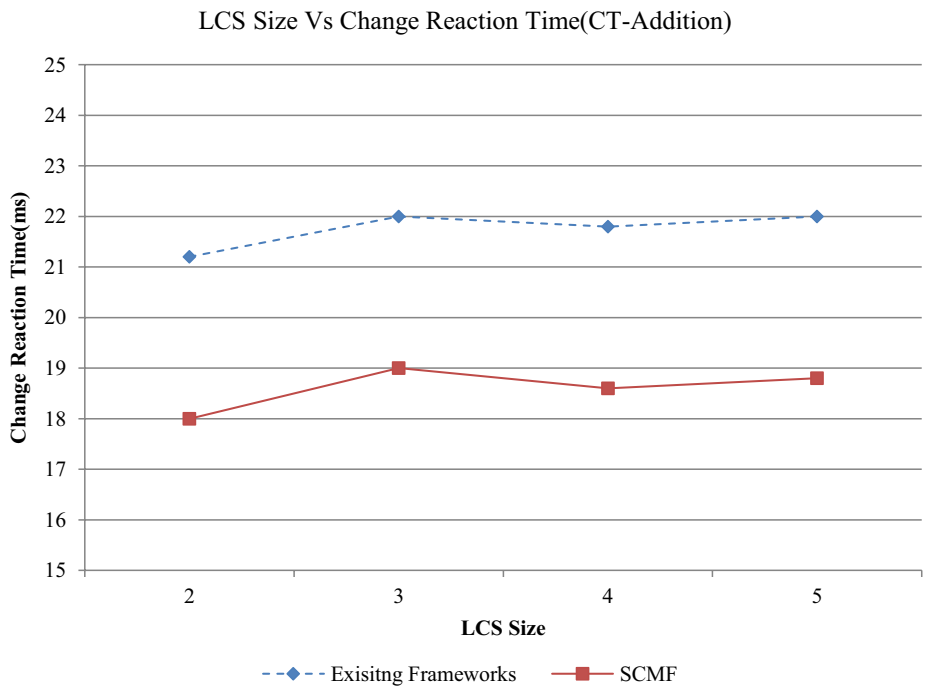

Fig. 6 LCS size vs change reaction time for addition operation for existing frameworks and SCMF

Metrics

Our Change Management Framework is developed with help of software development environments like Netbeans and a tool called Protégé. Protégé is an ontology tool which has a graphical interface helping us to create ontology. Once the ontology has been created, a corresponding owl file for the created ontology will be generated, which is used for inferring information from the ontology. Similarly, Netbeans is an IDE that provides an environment to develop and deploy the services required for our LCS. The information from the ontology can be inferred with the Jena API, which is a Java library that can be added in the Netbeans environment to perform the information retrieval from the ontology.

The proposed change management framework is designed in such a way that it provides higher accuracy, level of knowledge gained, high precision and degree of automation. The change management framework also has reduced risk and change reaction time. Accuracy of our framework is a measure of the extent to which the decisions made by the framework have been made successfully and cautiously. Level of knowledge is the number of nodes traversed by the system in the ontology to obtain the necessary information for implementing the requested change.

$$Accuracy = \sum_{i=1}^{n} \frac{\left(SC_i \cap \left(\frac{O'}{O}\right)\right)}{TC} \tag{7}$$

where SC_i = Number of successful changes, TC_i = Total number of change requests arrived, O' = Number of entities that are referred from ontology to satisfy the change, O = Total number of entities present in the ontology.

$$Level\ of\ Knowledge\ gained = \sum_{SC_{i=1}}^{n} \{2 \times (L_n)\} - 1 \tag{8}$$

where L_n = Number of leaf nodes visited, Sc_i = Number of successful changes.

Degree of automation is the extent to which the framework is capable of making the decisions regarding the requested change without the manual interruption. This is calculated with the help of the PDA and the ontology, i.e. the number of scenarios that use ontology among the total number of scenarios for performing the requested change will be used to determine the degree of automation. The correctness of the system is the measure of the extent to which the change is performed exactly as per the analyst needs. This is measured with the number of valid and invalid nodes in the PDA.

$$Degree\ of\ Automation = \sum_{i=1}^{n} \left\{ \left[Sc \cap \left(\frac{O'}{O} \right) \right] \right\} \tag{9}$$

where SC = Number of successful changes, O′ = Number of entities that are referred from ontology to satisfy the change, O = Total number of entities present in the ontology.

$$Correctness = \left[\sum_{i=1}^{n} \{ \neg (bp + ir + nr) + df + cf + con \} \right] \times 100 \tag{10}$$

where, ¬ = Negation, bp = Break point, Ir = Invalid reference, Nr = Null reference, Df = Data flow, Con = Constraint.

Precision and recall is calculated with the help of the ontological information. Precision is the number of ontological functions that have been returned by the system to implement the requested change from the total number of ontological functions present in the ontology. Recall is the number of ontological functions that are found to be relevant for implementing the change but has not been returned by the system. Although, all the information stored in the ontology is targeted to provide clear information about the services, sometimes there are chances that some information may lead the analyst to ambiguity.

$$Precision = \frac{g(x)}{f(x)} \tag{11}$$

where, f(x) = Total number of functions that were returned (both relevant and irrelevant), g(x) = Number of relevant functions from f(x).

$$Recall = \frac{g(x)}{g'(x) + f(x)} \tag{12}$$

where f(x) = Total number of functions that were returned (both relevant and irrelevant), g(x) = Number of relevant functions from f(x), g'(x) = Number of relevant functions but not returned by the system.

The risk involved in implementing the requested change for our system is calculated by the ontology and the parser, i.e. the number of ontological functions returned by the system that may be lead to ambiguity in incorporating the requested change when considered. The Table 4 discusses the metrics and their corresponding formulae.

$$Risk = \sum_{i=1}^{n} 1 - \left\{ SC_i \cap \left(\frac{O'}{O} \right) \right\} \tag{13}$$

where SC_i = Number of successful changes, O′ = Number of entities that are referred from ontology to satisfy the change, O = Total number of entities present in the ontology.

We also calculate the time taken for processing the entire change request with the help of ontology and the parser. The formula for the change reaction time is given below,

$$Change\ Reaction\ Time = \sum_{i=1}^{n} QP + OF \tag{14}$$

where QP = Time taken for query processing, OF = Time taken to refer the ontology.

Experimental results

The analytical results of the change reaction time with respect to LCS size and change size are depicted with the following graphs. LCS size is the number of member services that are involved in the composition. For e.g.: in our travel scenario the LCS size is three (air, cab, hotel or weather services). The comparison of the LCS size and change reaction time for the change type addition and deletion is shown in Figs. 7 and 8.

From these graphs it is evident that the time taken to implement a change request at the service level is found to lesser in our framework for both the cases (CT-Addition and CT-Deletion) than the existing framework. The change reaction time is also compared against the change size; change size is the number of member services in which changes

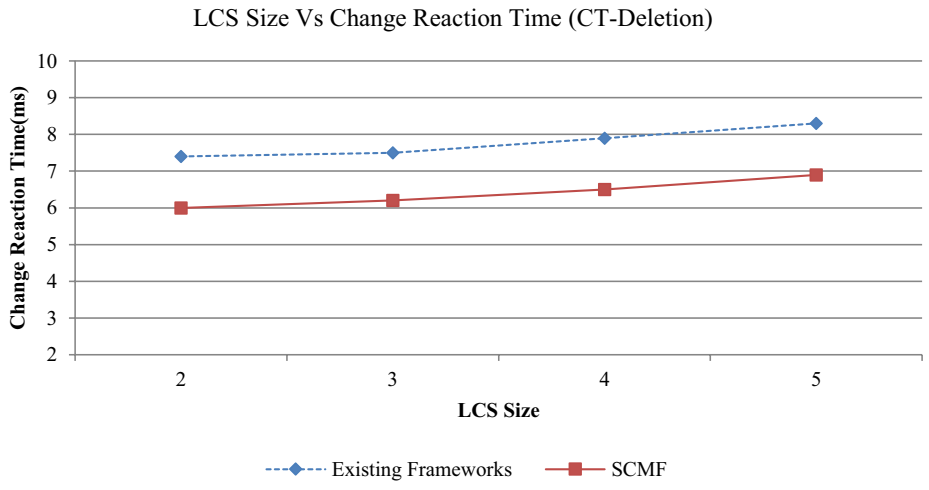

Fig. 7 LCS size vs change reaction time for deletion operation for existing frameworks and SCMF

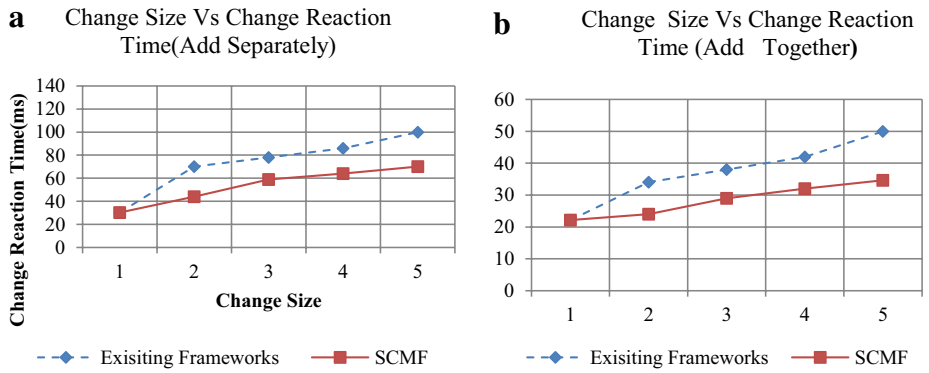

Fig. 8 Change size vs change reaction time for adding **a** Services separately and **b** Services together

have to be made to obtain the requested change. The changes can be separate or as a group mode.

In group, the changes are performed together, whereas in group mode the changes are made one after another separately. These are represented graphically in Figs. 9 and 10 respectively.

The correctness of our semantic change management framework is calibrated with the help of the PDA for both successful and failure cases and is indicated in Table 6.

Table 6 consists of the number nodes in the PDA, number of valid nodes among them and number of invalid nodes among them. The lesser the number of invalid nodes greater are the chances for having a successful change request. The Accuracy of the system is depicted in Fig. 10, where the accuracy is compared with the LCS size, the number of change requests.

The degree of automation for our framework is elaborated in Table 7. Scenario in the figure refers to each activity that contributes to a change management event. For e.g.: if the analyst wants to replace an existing service in a LCS by another service, then first the analyst has to get the related services from ontology, choose one among the extracted service, verify them logically and along with the change operators with the PDA and workflow parser, and then finally commit the change if the results of the semantic

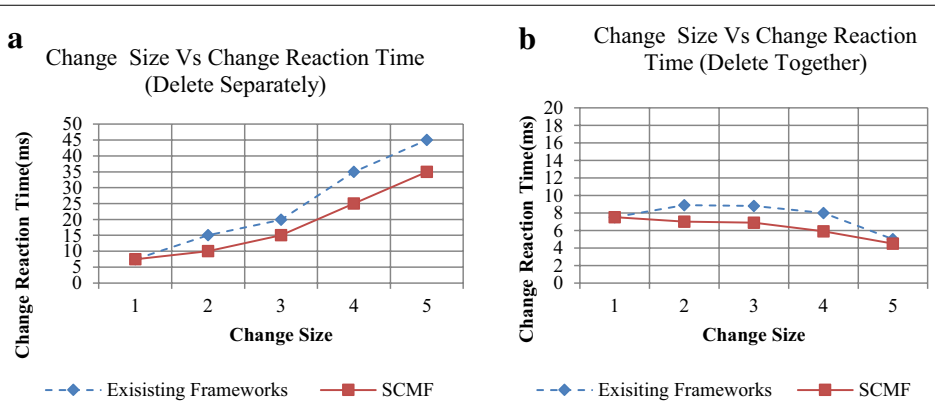

Fig. 9 Change size vs change reaction time for deletion of **a** Services separately and **b** Services together

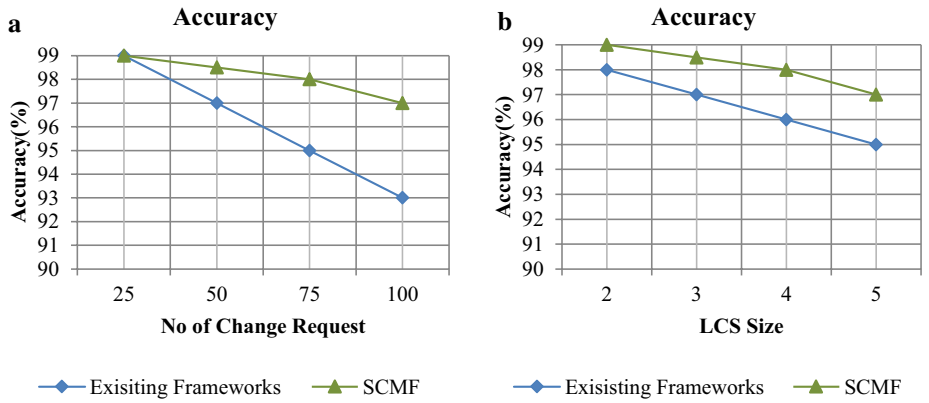

Fig. 10 Comparison of accuracy with **a** Number of change requests and **b** LCS size between existing frameworks and SCMF

Table 6 Correctness of the semantic framework for successful and failure cases

C-ID	C-Type (CT)	Constraint (con)	Status (ST)	No. of nodes	No. of valid nodes	No. of invalid nodes	Reaching final state
1	New	Qos	Success (S)	5	5	0	1
2	New	Qos	Success (F)	7	5	0	1
3	New	Functionality	Failure	5	3	2	0
4	New	Functionality	Success	7	7	0	1
5	New	Functionality	Success	5	5	0	1
6	New	Qos	Failure	5	4	1	0
7	New	Functionality	Failure	5	3	2	0
8	Change existing	Rule	Success	7	7	0	1
9	Change existing	Rule	Success	5	5	0	1
10	Change existing	Rule	Failure	5	3	2	0
11	Change existing	Rule	Failure	3	3	0	1
12	Change existing	Policy	Failure	4	3	2	0
13	Change existing	Policy	Success	7	7	0	1
14	Change existing	Policy	Success	5	5	0	1
15	Change existing	Policy	Success	3	3	0	1
16	Change existing	Policy	Success	4	4	0	1
17	Removal	Policy	Success	4	4	0	1
18	Removal	Policy	Failure	7	5	2	0
19	Removal	Rule	Success	5	5	0	1
20	Removal	Rule	Failure	7	6	1	0
21	Removal	Policy	Failure	5	3	2	0
22	Removal	Policy	Success	5	5	0	1

Table 7 Degree of automation for the semantic framework for successful and failure cases

C-ID	C-Type (CT)	Constraint (con)	Status (ST)	No. of scenarios	No. of scenarios using ontology	No. of ontological entities triggered	Reaching final state
1	New	Qos	Success (S)	8	6	5	1
2	New	Qos	Success (F)	7	5	5	1
3	New	Functionality	Failure	8	6	4	0
4	New	Functionality	Success	8	6	6	1
5	New	Functionality	Success	8	6	5	1
6	New	Qos	Failure	8	6	2	0
7	New	Functionality	Failure	8	6	1	0
8	Change existing	Rule	Success	6	4	3	1
9	Change existing	Rule	Success	6	4	2	1
10	Change existing	Rule	Failure	6	4	1	0
11	Change existing	Rule	Failure	6	3	3	1
12	Change existing	Policy	Failure	6	3	2	0
13	Change existing	Policy	Success	6	5	4	1
14	Change existing	Policy	Success	6	5	3	1
15	Change existing	Policy	Success	6	4	4	1
16	Change existing	Policy	Success	6	5	4	1
17	Removal	Policy	Success	4	3	3	1
18	Removal	Policy	Failure	4	3	1	0
19	Removal	Rule	Success	4	3	2	1
20	Removal	Rule	Failure	4	3	1	0
21	Removal	Policy	Failure	4	3	1	1
22	Removal	Policy	Success	4	3	3	1

analyser are positive. In this, each step is called as scenarios as each of them play a significant role in the process of implementing the incoming change requests. Since our framework is semantically driven, the number of scenarios that extract information from the ontology should be high and should be retrieved automatically by the framework without analyst's interruption.

All the ontological entities triggered by the system may not be useful for implementing the requested change. Therefore, the Level of Knowledge gained is the number of entities that were used for achieving the requested change is depicted in Table 8. For e.g. The C-ID 1 may have a total of five ontological entities triggered, but it is not necessary that all the five will provide the required information. Thus, the number of entities that provided the useful information is called the level of ontology.

Precision and recall for the successful and failure cases is shown by the graphs depicted in Fig. 11. From the graphs, it can be concluded that the precision for the existing framework is lower when compared to the precision of the semantic framework, this implies

Table 8 Level of knowledge gained by the semantic framework for successful and failure cases

C-ID	C-Type (CT)	Constraint (con)	Status (ST)	No. of scenarios	No. of scenarios using ontology	No. of ontological entities triggered	No. of required ontological entities	Reaching final state
1	New	Qos	Success (S)	8	6	5	5	1
2	New	Qos	Success (F)	7	5	5	5	1
3	New	Functionality	Failure	8	6	4	1	0
4	New	Functionality	Success	8	6	6	5	1
5	New	Functionality	Success	8	6	5	5	1
6	New	Qos	Failure	8	6	2	1	0
7	New	Functionality	Failure	8	6	1	1	0
8	Change existing	Rule	Success	6	4	3	3	1
9	Change existing	Rule	Success	6	4	2	2	1
10	Change existing	Rule	Failure	6	4	1	1	0
11	Change existing	Rule	Failure	6	3	3	3	1
12	Change existing	Policy	Failure	6	3	2	1	0
13	Change existing	Policy	Success	6	5	4	4	1
14	Change existing	Policy	Success	6	5	3	3	1
15	Change existing	Policy	Success	6	4	4	4	1
16	Change existing	Policy	Success	6	5	4	4	1
17	Removal	Policy	Success	4	3	3	3	1
18	Removal	Policy	Failure	4	3	1	0	0
19	Removal	Rule	Success	4	3	2	2	1
20	Removal	Rule	Failure	4	3	1	1	0
21	Removal	Policy	Failure	4	3	1	0	1
22	Removal	Policy	Success	4	3	3	3	1

that the number of ontological functions found useful for a change was higher than the human computed functions (Figs. 12, 13).

The difference between the proposed framework and existing frameworks

In the existing work Fig. 14, initially the change request is implemented at the schema level and then at the instance level. Schema level verification is performed on a schema graph [20] which consists of abstract services as nodes and relationship between each of these services as edges. Since the schema level consists only of abstract services, any errors or misleading activities which will lead to incompleteness of the schema graph can be identified. To perform the changes, the existing system uses two supporting components called, ontology providers and web service providers. The ontology provider is

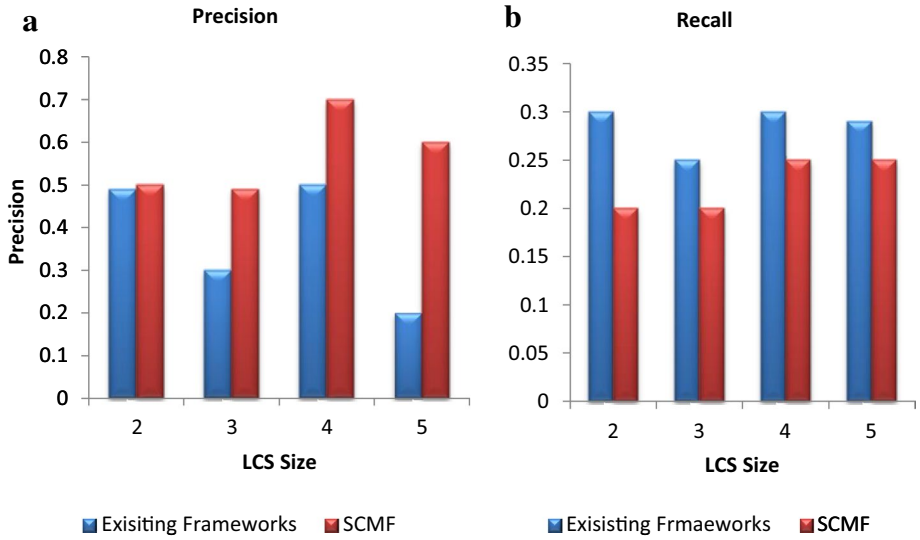

Fig. 11 Comparison of precision (**a**) and recall (**b**) between existing frameworks and SCMF

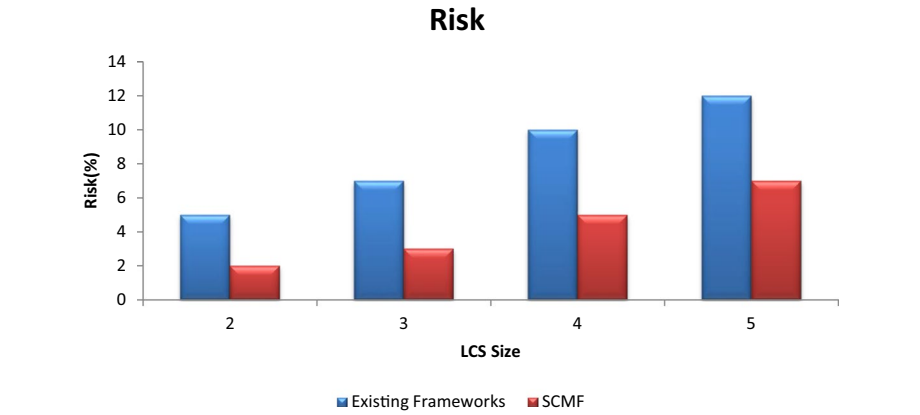

Fig. 12 Comparison of risk between existing frameworks and SCMF

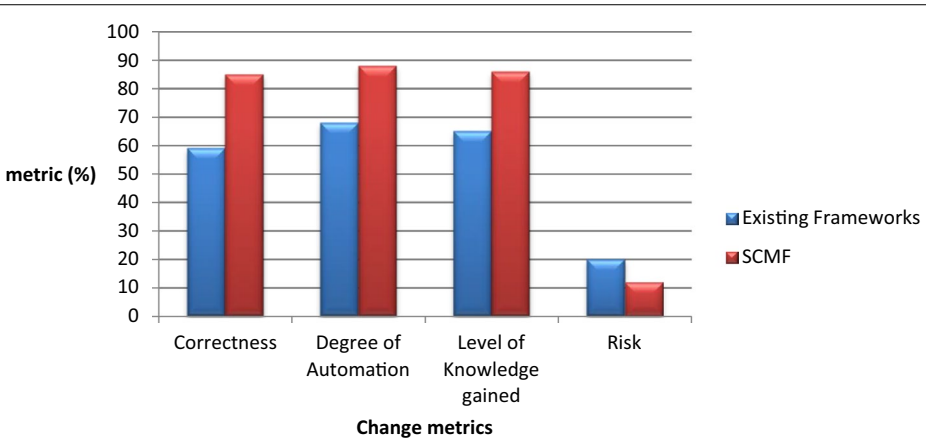

Fig. 13 Overall comparison of the metrics between existing frameworks and SCMF

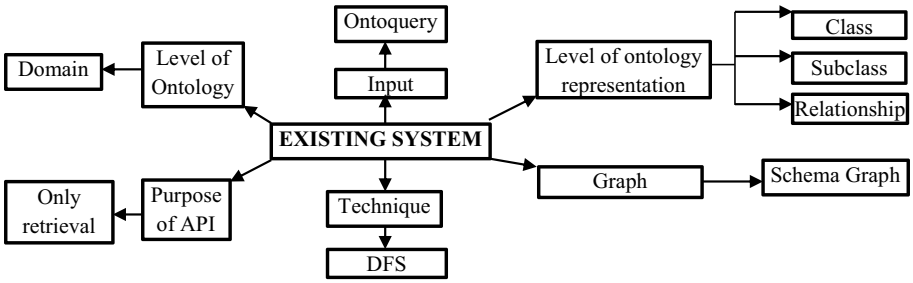

Fig. 14 System block diagram for existing frameworks

a component that provides and maintains the set of ontologies for each of the web services. The web service provider is a component that provides services (business functionalities), from these providers certain numbers of services are selected to form an LCS instance so that they can in composition provide solutions to various web based applications.

In our proposed work Fig. 15, we aim to develop a Framework, which gives importance to create an enriched ontology. The ontology that been created will be able to represent the services in terms of class, subclass, instances, relationship and also the functionality of the services. The ontology is used to store information about the functionality of the services, the rules used in the services and relationship between the individual and among services. The functionality of the services represent the services at functional and object level. The API apart from extracting the ontological information is also capable of inferring new knowledge with the help of the axioms and rules in the ontology. In the existing framework, the ontology is used only for retrieving domain knowledge [21], therefore only the class and sub-class features of the ontology have been used, whereas in our proposed framework, we use class, sub-class, relationship, object property, data property and instances features of the ontology have been used to represent the functional and non-functional aspect of our framework. To perform automatic verification

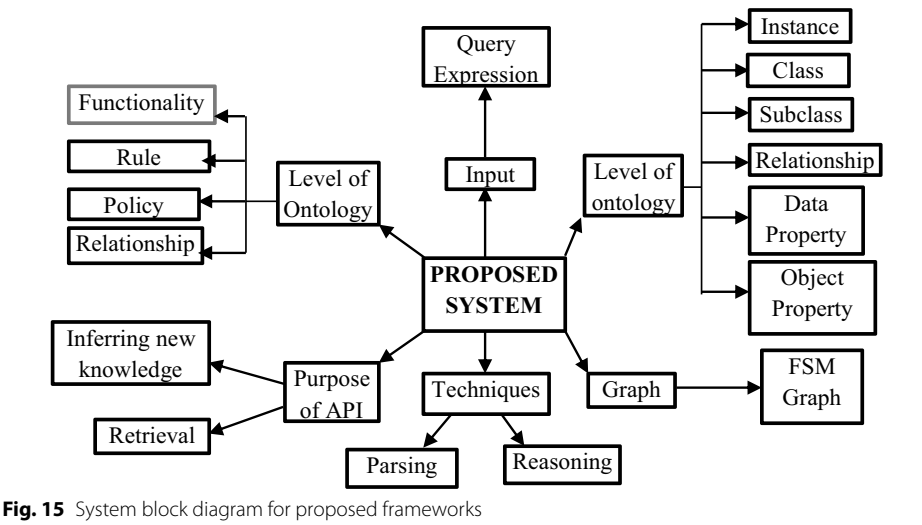

Fig. 15 System block diagram for proposed frameworks

and validation of the changes before its implementation at the service level, the semantic framework uses the PDA graph and the parsers to obtain the necessary information.

Discussion

This section deals with the conclusion and the future enhancements that can be added to make the process of change management more flexible to the analyst.

In this paper we concentrated on providing a change management framework that would make use of an enriched ontology set and semantic reasoner for implementing the changes by the analyst itself. The semantic reasoner component parses the change request from the analysts and determines the possibility for making the change. The framework represents the implemented changes in the form of a S-BPEL notation (Semantic BPEL) which is then converted to their corresponding BPEL notations by a BPEL constructor so that they can be deployed in the run time environment for composing the services.

Thus one prime future goal can be to extend the framework so that it can handle multiple change requests at a time. In our framework, implementing the requested changes in the service level is done by the analyst directly on the schema; instead the framework can be enhanced in such a way that it provides a GUI interface for making the changes at the schema level so that the burden at the analyst side will be reduced further.

Conclusions

The Semantic Reasoner based change management framework contributes the following points and thus manages the evolving changes in the LCS. The research has the following salient features:

- Presents a parsing methodology to determine the possibility of implementing the requested change before it is being implemented at the service level with the help of the workflow parsing mechanism.
- Maintains the state information of the services involved in change enactment by the PDA methodology. This methodology is also responsible for automatic validation of the requested change.
- Presents a framework where the functionality of the services is also included in the ontology so that the more information about the services is provided to the analyst during the change enactment.
- The framework not only makes it possible for the analyst to perform the requested change, but also facilitates the analyst to compose services from the scratch without any ambiguity with the help of the information provided in the ontology, PDA and parser.
- Presents an environment for the analysts to implement the changes so that the cost and time spent on the IT professionals for making the changes is reduced.

In the proposed framework, semantic information about the services like relationship among the services, operations and rules present in a service etc. has been incorporated with the help of ontology so that it can be used by the analyst during the change management process. Correct workflow among the services during composition is verified

by using composition based operator precedence parser. Initially, the parser will generate a precedence table dynamically and then this table is used to ensure that the correct workflow is maintained among the services when the change is applied. Therefore, the main goal of our work is to (i) to ensure correct flow is maintained during the composition of services during the change management process. (ii) to reduce the burden at the analyst side by creating a framework that is semantically driven by an enriched ontology so that the changes are performed by the analyst itself without depending on the IT developers. Due to the enriched ontology, the number of bugs that may arise during the implementation of the change can be greatly reduced; it also brings clarity in the workflow and subsequently brings process clarity.

In the semantic framework, as the size of the LCS increases, the accuracy of the semantic framework is found to be increased. Similarly, the correctness of the system in terms of extracting the most relevant information for incorporating the changes is also found to be high. In the proposed work, the retrieval of the required information is done automatically by the system thereby the degree of automation is also increased subsequently. The number of ontological functions (Precision) for implementing the changes is more than the number of relevant functions left by the system (Recall).

Future enhancement

Change management is a wide area of research that requires more novel techniques and approaches to meet the arising business needs. Although, the proposed change management framework is capable of deciding the feasibility of the incoming request beforehand with the help of PDA and parser, the framework can handle only one request at a time. Thus one prime future goal can be to extend the framework so that it can handle multiple change requests at a time. In our framework, implementing the requested changes in the service level is done by the analyst directly on the schema; instead the framework can be enhanced in such a way that it provides a GUI interface for making the changes at the schema level so that the burden at the analyst side will be reduced further. Further our framework is designed focusing on the analyst; it can also be extended to include the involvement of the end users.

Authors' contributions
MT is the brain of this manuscript who has provided the idea for this work. BG was responsible for drafting this manuscript technically. Both authors read and approved the final manuscript.

Acknowledgements
I would express my heart filled gratitude to my family for their constant support to finish this paper, I would be failing in my duty if I don't acknowledge my uncle who provided me with reference papers that made the literature extract of my paper.

Competing interests
The authors declare that they have no competing interests.

Received: 27 July 2015 Accepted: 27 May 2016
Published online: 10 August 2016

References

1. Liu X, Bouguettaya A, Wu J, Zhou L (2010) Ev-LCS: a system for the evolution of long-term composed services. IEEE Transact Serv Comput 5(2):102–115
2. Liu X, Bouguettaya A (2007) Managing top-down changes in service-oriented enterprises. InICWS 1072–1079

3. Kanimozhi M (2014) Enhancing change management in long term composed services. Int J Innov Res Sci Eng Technol 3(3):10698–10704
4. Mastroianni C, Papuzzo G (2014) A self-organizing P2P framework for collective service discovery. J Netw Comput Appl 39:214–222
5. Han SN, Lee GM, Crespi N (2013) Semantic context-aware service composition for building automation system. IEEE Transact Ind Inf 5(2):752–761
6. Apostolou D, Mentzas G, Stojanovic L, Thoenssen B, Lobo TP (2010) A collaborative decision framework for managing changes in e-Government services. J Gov Inf Q 28(1):101–116
7. Paliwal AV, Shafiq B, Vaidya J, Xiong H, Adam N (2012) Semantics-based automated service discovery. IEEE Transact Serv Comput 5(2):260–275
8. Meditskos G, Bassiliades N (2010) Structural and role-oriented web service discovery with taxonomies in OWL-S. IEEE Transact Serv Comput 5(2):278–290
9. Albukhitan S, Alnazer A, Helmy T (2016) Semanitc annotation of arabic web resources using semanitc web services. Porc Comput Sci 83:504–511
10. Hatzi O, Vrakas D, Nikolaidou M, Bassiliades N, Anagnostopoulos D, Vlahavas I (2012) An integrated approach to automated semantic web service composition through planning. IEEE Transact Serv Comput 5(3):319–332
11. Liu X, Akram S, Bouguettaya A (2011) Semantic support for change management. In: Liu X, Akram S, Bouguettaya A (eds) Change management for semantic web services, 1st edn. Springer, New York, pp 19–30
12. Cheng J, Liu C, Zhou M, Zeng Q, Ylä-Jääski A (2014) Automatic composition of semantic web services based on fuzzy predicate petri nets. IEEE Transact Automation Sci Eng 12(2):680–689
13. Akram MS, Medjahed B, Bouguettaya A (2003) Supporting dynamic changes to in web service environments. Springer, Berlin, pp 319–334. ISBN 978-3-540-20681-1
14. Aissi S, Gouider MS, Sboui T, Said LB (2015) A spatial data warehouse recommendation approach: conceptual framework and experimental evaluation. Hum Cent Comput Inf Sci 5(1):1–8
15. Farrag TA, Saleh AI, Ali HA (2013) Semantic web services matchmaking: semantic distance-based approach. J Comput Electr Eng 39(2):497–511
16. Rafiei M, Kardan AA (2015) A novel method for expert finding in online communities based on concept map and PageRank. Hum Cent Comput Inf Sci 5(1):1–18
17. Lin SY, Lai CH, Wu CH, Lo CC (2014) A trustworthy QoS based collaborative filtering approach for web service discovery. J Syst Softw 93:217–228
18. Dasgupta S, Aroor A, Shen F, Lee Y (2014) SMARTSPACE: multiagent based distributed, platform for semantic service discovery. IEEE Transact Syst Man Cybern Syst 44(7):805–821
19. Parejo JA, Segura S, Fernandez P, Ruiz-Cortés A (2014) QoS-aware web services composition using GRASP with path relinking. J Expert Syst Appl 41(9):4211–4223
20. Akram S, Bouguettaya A, Liu X, Haller A, Rosenberg F (2010) A change management framework for service oriented enterprises. Int J Next Gener Comput 1(1):1
21. Ding LY, Zhong BT, Wu S, Luo HB (2016) Construction risk knowledge management in BIM using ontology and semanitc web technology. Saf Sci 87:202–213