Abstract—These Web services technology provides a platform on which we can develop distributed application by using existing service oriented application. Software product line on the other hand provides a systematic mechanism to manage the reusable components of the product family within the product line. In this work we followed an approach to combine these two approaches to achieve the benefits of both. The common and variants features of software product line can be modeled by using a feature diagram. We first perform a commonality and variability analysis of the requirements of the product line. From such analysis we develop a dependency graph consisting of the features and constraints among them. A walk through the graph can solve the constraints. We draw goal models of various functional features of the product line to get a clear view whether a certain goal can be achieved by the product line model. In order to check the consistency of the feature model we encode the feature by using OWL-DL. Such encoding facilitates to check whether any valid product can be composed from the product family model. We perform our analysis and experiment by using a Travel Agent Web Service.

Index Terms—Semantic web, web services, OWL-DL, feature model, product line, web service composition.

I. INTRODUCTION

Semantic Web Services is an emerging technological concept that combines technologies from Web Services, the Semantic Web, and automatic reasoning. Semantic Web Services emphasizes enabling dynamic, execution-time discovery, composition, and invocation of Web Services [1] allowing automated ad-hoc interaction between web-based applications [2]. It builds on the existing Web Services technology that provides software developers a framework for accurate documentation of services to allow the development-time discovery, composition, and statically coded invocation of services in the Web. The aim of Semantic Web Services technology is to automate these tasks that currently require human software developers.

The predicted advantages of the Semantic Web Services can be achieved from up-to-date, comprehensive, and effective utilization of Web Services. As the decision to use a particular service can be made at the execution time instead of development time of the service requestor software, it is possible to use services that are not available, or even existing, at development time but are only discovered at execution time. This will dramatically improve the usability of Web Services technology in volatile environments, such as in mobile and ubiquitous computing, where the landscape of available services can change rapidly, for example, when a device moves from one space to another. Such benefits can also be made available to non-mobile open environments, such as the Web where the available services evolve over the time and many different kinds of services may be available to satisfy the particular goals.

Semantic Web [3] can be used to represent the description of services in a precisely machine understandable form which could form the basis of automatic and execution time reasoning about the type, structure and properties of services. Other reasoning methods, such as logical reasoning can be combined with semantic web not only to describe services in accurate manner but also to discover promising services and composing them into complex workflows, and to determine the correct way to invoke them.

The Semantic Web and Web Services are two complementary initiatives, building on the notational ground provided by XML. Semantic Web is declarative representation approach to describe domain information in machine processable way, while Web Services provide a mechanism to invoke services across the Web in a way that resembles remote procedure calls.

The central parts of Web Services technology are descriptions of the interfaces of services specified in Web Services Description Language (WSDL) [4] to make it possible to invoke them over the Internet. There are a number of additional specifications dealing with many other aspects of Web Service use, namely, orchestration, choreography, addressing, security, resource states, and so on.

Software product line is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or missions and that are developed from a common set of core assets in a prescribed way [5]. The main idea of software product line is to explicitly identify all the requirements that are common to all members of the family as well as those that vary among products in the family. A particular product is then derived by selecting the required variants and configuring them according to the product requirements.

Software product line and web service composition share some similarities among them. A web service provider has a set of specified product features that have to be fulfilled by all the possible service composition provided to users. On the contrary, some services are not required by all the service composition but included in a few composition based on user requirements.

The objective of this work is to apply software product line techniques to manage the service composition and reuse of services across the composed services. The reuse mechanism of software product line can be successfully applied to web services. In this paper we present a systematic mechanism that allows us to analyze the service composition in the light of software product line. We perform a case study of Travel
Agent web service.

In order to understand the available features that can be reused during various web services composition, we present a commonality and variability analysis (CVA) of the Travel agent web service. Such analysis portrays clearly the common and variant features of the said web service and depicts the dependencies and constraints of the services. A goal model checks whether the user objectives or goals can be accomplished from the composed services. We draw goal model for the common functionalities of travel agent web service. Besides analyzing the service composition, we also verify the feature relations of the web services. We model the features using semantic web technique OWL-DL [6] and verify the feature composition using Protégé [7] and RacerPro [8].

In the rest of the paper, Section II gives a brief description of Travel Agent Web Service. The common and variants features of the web service are illustrated by using feature diagram. We then describe our analysis mechanism of our case study in Section III. The semantic web based verification of feature composition is illustrated in Section IV. Finally we conclude our paper and outline our future plan in Section V.

II. TRAVEL AGENT WEB SERVICES OVERVIEW

A Travel Agent Web Service is a system for booking tickets for train, bus, air, cruise etc. as well as for booking accommodation system. The basic operational scenarios are similar in all the Travel Agent Web Service systems. However, there are many variants on the basic theme, e.g., holiday packages can be provided by some Travel Agent Web services while some others only provide basic ticket services

A. Travel Agent Domain Model Using FODA

![Travel agent web service feature diagram.](image)

Modeling variants is an important process during designing software product line. The feature oriented domain analysis (FODA) [9] focuses on identifying features that characterize a domain. Features are user visible aspects or characteristics of a system and are organized into And/Or graph in order to identify the commonalities and variants of the application domain. Feature modeling is an integral part of the FODA method and the Feature Oriented Domain Reuse Method (FORM) [10]. The commonalities and variants within features are exploited to create a set of models that is used to implement any member product of that family.

In feature tree, features are represented in graphical form as trees. The internal nodes of a tree represent the variants and their leaves represent values of corresponding variants. Graphical symbols are used to indicate the categories of features. The feature diagram depicts the classification of mandatory features and variant features as well as their dependencies. Mandatory features are those which are present in all products in the respective domain. Variant features appear only some members of the domain which differentiate one product from others.

A partial feature tree of Travel Agent Web Service System is shown in Fig. 1. Extensions of feature diagram described in [11] have been used here. Variant features are also classified as Optional, Alternative and Or features. An example of optional feature is ‘Accommodation System’. An alternative feature describes one of many features. An example of alternative feature is ‘Payment System’ which can be either Bank Transfer System or Card System. An or-feature describes any of many features. For example a ‘Ticket Booking’ can be made by multiple services or multiple times or by both. Variants may depend on other variants. Two types of dependencies are illustrated in this paper.

III. ANALYSIS OF SERVICE COMPOSITION

Our approach contains two workflows, one for the web service product developer and another for the user. This approach provides a two-stage process. From the point of view of product line development, the development process is divided into the domain engineering phase and the application engineering phase; whereas from the point of view of web service design, the process is divided into the preparation stage (before implementation) and the customization stage (after implementation). An overview of our approach is illustrated in Fig. 2.

At the top half of Fig. 2, i.e., the domain engineering phase, the developer performs a commonality and variability analysis (CVA) of the system requirements. The results of the CVA include a formalized specification of all the common and variable functional properties. Each variable has some associated parameters to configure, called parameters of variation. Some of these parameters may have dependencies or tradeoffs with other parameters of variation. We model this dependency of variations in a variability dependency graph.

Each commonality and parameter of variation is also associated with a set of service compositions that satisfy them. We call this relation the mapping-relation. The results of the CVA are captured in a product-line decision model. Given the system requirements, the developer can identify the component web services that are relevant to the domain and the system.

For the application engineering phase, the services retrieved from the service broker are composed according to the common functional requirements. The composition set is then pruned to a smaller set in which each composition satisfies all the commonalities. This set serves as the composition search space for the variability, and we call it the commonality composition set.

After implementation of the web services, the user can
access a default web service with basic functionalities. The user can then customize the functional properties by setting all the parameters of variation in the decision model. By interacting with the decision model, the input requirements from the user are always kept consistent through solving the constraints in the variability dependency graph. The consistent specification of the variabilities can be fed into a query algorithm to search valid compositions in the composition search space.

Travel Agent Web Service consists of several different services. The web service can also invoke other third party services. Fig. 3 shows a sample structure for Travel Agent service composition. The services are connected by the dashed lines represent the optional services in a valid composition.

A. Commonality and Variability Analysis (CVA)

The commonalities and variabilities of the Travel Agent service compositions are listed according to their label, type (Functional requirements as “FR”), their description and their service mappings and parameter(s) of variation and any dependencies among these parameters (Fig. 4).

B. Dependency Graph

We model the dependencies of the parameters in the CVA in a dependency graph (Fig. 5). In the dependency graph, each node represents variability and contains the information of all the parameters related to this variability. The edges between the nodes represent and contain the constraints between the two variabilities. The graph may not be fully connected. A graph-walk algorithm can be used to traverse the sub-graphs in a random order or a user-preferred order to solve all the constraints.

Example Commonality:
- C1:
  - FR
  - There must be a Travel Agent Service.
  - Travel Agent Service = any service in Travel Agent Service set \{TA1, TA2, TA3, \ldots\}
- C2:
  - FR
  - There must be one or more Ticket Service.
  - Ticket Service = any service in Air Ticket Service set \{ATS1, ATS2, \ldots\} or Train Ticket Service set \{TTS1, TTS2, \ldots\} or Bus Ticket Service set \{BTS1, BTS2, \ldots\} or Cruise Ticket Service set \{CTS1, CTS2, \ldots\}.

Example Variability:
- V1:
  - FR
  - Existence of Air Ticket service
  - \{True, False\}
  - If V1 is False, (V2 or V3 or V4) is True
- V2:
  - FR
  - Existence of Train Ticket service
  - \{True, False\}
  - If V2 is False, (V1 or V3 or V4) is True

Fig. 2. Overview of analysis approach.

Fig. 3. Composition of travel agent web service.

Fig. 4. CVA analysis.
C. Goal Model

In order to generate all the service compositions that satisfy the commonalities, we need to represent the common functionalities in the goal model. Fig. 6 shows the goal model for the common functionalities. Note that we have preserved the possibility of using third-party services to implement the functionality ThirdPartyService(OtherData) in the goal model. The functional composition process finds appropriate components to implement these abstract methods in the goal model in accordance with the service mapping table. This service mapping table is constructed together with the goal model by looking up the CVA results. For example, there is a mapping: ThirdPartyService = any element from \{N/A, GPS-MAP, TP1, TP2, ...\}.

IV. Semantic Web Based Analysis

By using OWL-DL we model six types of relations, namely mandatory, optional, alternative, or, optional alternative and optional or. Two additional constraints: requires and excludes are also modeled. Modeling feature models using OWL-DL have several advantages, such as facilitating feature model storing, sharing and distributing and assisting cooperative designing.

First, OWL ontology is built for various nodes and edges in the feature model. The ontology is constructed in three steps:

Step 1: We identify the nodes (concepts and features) present in a feature diagram. Each node in the diagram is modeled as an OWL class. Moreover, we assert that these classes are mutually disjoint. In OWL, all classes are assumed to overlap unless it is otherwise stated that they are disjoint with each other using a disjoint axiom.

Step 2: For each node, we create a Rule class. This Rule class has two kinds of conditions: firstly, a necessary and sufficient (NS, EquivalentClass) condition, using an existential restriction to bind the Rule node to the corresponding feature node in the diagram; and secondly, a number of (possibly 0) necessary (N, subClassOf) constraints later, serving two purposes: 1) to specify how each of its child features is related to this node, capturing the various relations between features, 2) to specify how this feature node is constrained by other features, in the form of requires, excludes, Optional Alternative and Optional Or.

Step 3: The root concept and features in a feature diagram are interrelated by various feature relations, represented by different edge types in the diagram. In our OWL model, for each of these edges, we create an object-property. We assert that the range of the property is the respective feature class. The OWL syntax used in this paper is summarized in Table I. For a parent feature A and for each of its child features B₁, B₂, ..., Bₙ the initial modeling produces the following ontology.

\[ A \sqsubseteq \top \quad \text{hasA} \sqsubseteq \text{ObjectProperty} \]
V. Modeling Various Types of Features

A. Mandatory

A mandatory feature is included if its parent feature is included. For each of the mandatory features $B_1$, $B_2$, ..., $B_n$ of a parent feature $A$, we use one $N$ constraints in $ARule$ to model it. It is a someValuesFrom restriction on $hasB_i$, stating that each instance of the rule class must have some instance of $B_i$ class for $hasB_i$. The following ontology fragment shows the modeling of mandatory feature set and parent feature $A$.

$$ARule \equiv \exists hasB_i \cdot B_i \text{ for } 1 \leq i \leq n$$

It can be seen from Fig. 1 that the root node has a mandatory child feature $PaymentService$, which is itself a non-leaf node. We create two new classes (parent and child) for these two non-leaf nodes.

For parent feature:

```xml
TravelAgent ⊑ T
hasTravelAgent ⊑ ObjectProperty
TravelAgentRule ⊑ T ⊑ ∀ hasTravelAgent ⊑ TravelAgent
```

For child feature:

```xml
PaymentService ⊑ T
hasPaymentService ⊑ ObjectProperty
PaymentServiceRule ⊑ T ⊑ ∀ hasPaymentService ⊑ PaymentService
TravelAgentRule ⊑ T ⊑ ∀ hasPaymentService ⊑ PaymentService
```

B. Optional

An optional feature may or may not be included in a diagram, if its parent is included. For each of the optional features $B_1$, $B_2$, ..., $B_n$ of a parent feature $A$, no additional statements are required to model this relationship. In that case they are denoted by:

$$Bi \equiv T \cdot hasBi \equiv ObjectProperty$$

```xml
BiRule ⊑ T
BidRule ⊑ T \equiv ∃ has Bi ⊑ Bi
```

Accommodation service is an optional feature of Travel Agent Web Service and it may, or may not be included in a configuration of Hall Booking System. As it is a non-leaf node, we create one new class $AccomService$ for Accommodation Service Reservation Charge and one object-property $hasAccomService$. The ontology is augmented as follows. Note that no new restriction on $TravelAgent$ is added.

```xml
AccomService ⊑ T
hasAccomService ⊑ ObjectProperty
AccomServiceRule ⊑ T
AccomServiceRule ↦ ∃ hasAccomService · AccomService
```

C. Alternative

One and only one feature from a set of alternative features can be included, if their parent feature is included in a configuration. Hence, for a set of alternative features $Bi$ and $Bj$ are two children of parent feature $A$, we use disjunction of someValuesFrom restrictions over $hasBi$ and $hasBj$ to ensure that some feature will be included. We use the complement of distributed disjunction of the conjunction of two someValuesFrom restrictions to ensure that only one feature can be included. The ‘or’ symbol $|$ represents distributed disjunction.

$$ARule \equiv ((\exists hasB_i \cdot B_i) \cup (\exists hasB_j \cdot B_j))$$

for $1 \leq i \leq n$ and for $1 \leq j \leq n$

$$ARule\equiv \neg ((\exists hasB_i \cdot B_i) \cap (\exists hasB_j \cdot B_j))$$

for $1 \leq i \leq n$ and for $1 \leq j \leq n$

Fig. 1 shows that features $CardService$ and $BankTrnsService$ are alternative features of $PaymentService$. We model this relation as follows.

```xml
PaymentService ⊑ T
hasPaymentService ⊑ ObjectProperty
PaymentServiceRule ⊑ T
\equiv ∃ hasPaymentService · PaymentService
```

```xml
CardService ⊑ T
hasCardService ⊑ ObjectProperty
CardServiceRule ⊑ T
\equiv ∃ hasCardService · CardService
```

```xml
BankTrnsService ⊑ T
hasBankTrnsService ⊑ ObjectProperty
```

214
BankTrnsServiceRule $\subseteq \top$
$\top \subseteq \text{hasBankTrnsService} \cdot \text{BankTrnsService}$

BankTrnsServiceRule
$\equiv \exists \text{has BankTrnsService} \cdot \text{BankTrnsService}$

PaymentServiceRule
$\subseteq \exists \text{has BankTrnsService} \cdot \text{BankTrnsService}$

PaymentServiceRule
$\subseteq ((\exists \text{has Cardservice} \cdot \text{Cardservice})$
$\lor ((\exists \text{has BankTrnsService})$
$\land (\exists \text{has BankTrnsService}))$

The last two restrictions ensure that one and only one feature from the set of alternative features can be included.

D. Or
At least one from a set of available features is included, if the parent feature is included. For a set of Or features $B_1, B_2, ..., B_n$ of a parent feature $A$, we need to use a disjunction of someValuesFrom restrictions to model this relation.

$\text{ARule} \sqcup (\exists \text{hasBi} \cdot \text{Bi}), \text{ for } 1 \leq i \leq n$

It may be noticed that the definition of Or is very similar to that of Alternative, with the omission of the negation of distributed disjunction to allow for multiple or features to be included. In Fig. 1, the feature Reservation Charge has a number of Or features. We use the following constructs to model it.

TicketService $\subseteq \top$
$\text{hasTicketService} \subseteq \text{ObjectProperty}$

TicketServiceRule $\subseteq \top$

TicketServiceRule
$\subseteq \exists \text{hasTicketService} \cdot \text{TicketService}$

AirService $\subseteq \top$
$\text{hasAirService} \subseteq \text{ObjectProperty}$

AirServiceRule $\subseteq \top$
$\top \subseteq \text{hasAirService} \cdot \text{AirService}$

AirServiceRule $\equiv \exists \text{hasAirService} \cdot \text{AirService}$

TicketServiceRule $\subseteq \exists \text{hasAirService} \cdot \text{AirService}$

TicketServiceRule
$\subseteq ((\exists \text{hasAirService} \cdot \text{AirService})$
$\lor (\exists \text{hasTax} \cdot \text{Tax})$
$\lor (\exists \text{hasBasicCharge} \cdot \text{BasicCharge})$
$\lor (\exists \text{hasDiscount} \cdot \text{Discount})$

E. Optional Or
One or more optional feature may be included when parent feature is included. There is no Optional Or feature in our

\footnote{For brevity, definitions of Rule classes are omitted from here.}

brief case study but a detailed feature model consists of such features. If $A$ is parent feature of child features $B$ and $C$. The OWL-DL representation is defined as follow:

$B \subseteq \top$
$\text{hasB} \subseteq \text{ObjectProperty}$

$\text{BRule} \subseteq \top$
$\text{BRule} \equiv \exists \text{hasB} \cdot B$

$C \subseteq \top$
$\text{hasC} \subseteq \text{ObjectProperty}$

$\text{CRule} \subseteq \top$
$\text{CRule} \equiv \exists \text{hasC} \cdot C$

$\text{ARule} \equiv ((\exists \text{hasB} \cdot B) \lor (\exists \text{hasC} \cdot C))$

F. Optional Alternative
One feature from a set of alternative features may or may not be included if parent is included. Consider $A$ is parent feature of child features $B$ and $C$. We can write the OWL-DL representation as follow:

$A \subseteq \top$
$\text{hasA} \subseteq \text{ObjectProperty}$

$\text{ARule} \subseteq \top$
$\top \equiv \forall \text{hasA} \cdot A$

$\text{ARule} \equiv \exists \text{hasA} \cdot A$

$B \subseteq \top$
$\text{hasB} \subseteq \text{ObjectProperty}$

$\text{BRule} \subseteq \top$
$\text{BRule} \equiv \exists \text{hasB} \cdot B$

$C \subseteq \top$
$\text{hasC} \subseteq \text{ObjectProperty}$

$\text{CRule} \subseteq \top$
$\text{CRule} \equiv \exists \text{hasC} \cdot C$

$\text{ARule} \equiv ((\exists \text{hasB} \cdot B) \lor (\exists \text{hasC} \cdot C))$

G. Requires
A feature may depend on some other features, hence its presence in a feature configuration requires the appearance of the others. For a given feature $A$ and a set of features $B_1, B_2, ..., B_n$ that $A$ requires, besides the NS condition that binds $\text{ARule}$ to $A$, we make sure that each of the $B_i$ features appears in a configuration if $A$ is present.

$\text{ARule} \equiv \exists \text{hasBi} \cdot B_i \text{ for } 1 \leq i \leq n$

H. Excludes
The presence of a feature may be inhibited by that of some other features. We say the appearance of a feature in a configuration excludes the appearance of some other features. For a given feature $A$ and a set of features $B_1, B_2, ..., B_n$ that $A$ excludes, we make sure, using the negation of someValuesFrom restriction on $\text{hasBi}$ property, that $\text{ARule}$ does not have any $B_i$ feature.

$\text{ARule} \equiv \neg ((\exists \text{hasB} \cdot B) \lor (\exists \text{hasC} \cdot C))$

We encoded the OWL-DL models of the Hall Booking System features into Proté gé tool. Proté gé gives an immediate feedback to the encoded logic by showing the corresponding feature graph ensuring the preliminary
correctness of the syntax. Fig. 7 shows the feature graph created in Proté gé.

![Fig. 7. Proté gé screenshot.](image)

VI. FEATURE VERIFICATION

Consistency checking is a mechanism for checking whether the rules are semantically consistent. Consistency checking has to ensure that inconsistent requirements do not become part of the requirements specification of a given product. Our aim is to ensure that all the products derived checking has to ensure that inconsistent requirements do not become part of the requirements specification of a given product. Our aim is to ensure that all the products derived from a feature model are consistent with requirements specifications. We can achieve this through the description and verification of semantic relationships between feature model and use scenarios. We input our ontology into Proté gé and use RACER to check its consistency. For the initially encoded ontology, RACER checks for consistency and shows that encoded definitions are consistent (Fig. 8).

![Fig. 8. Consistency checking in RACER.](image)

In feature modeling, an instance of a concept is a configuration derived from the feature model. In order to detect inconsistency in a configuration OWL classes are used, and features and concept instances are then simulated. When an instance is checked, the reasoned tool not only can check inconsistency but also shows which class/classes are inconsistent. We use an existential restriction for each feature instance is checked, the reasoned tool not only can check inconsistency but also shows which class/classes are inconsistent. We use an existential restriction for each feature and use RACER to check its consistency. For the initially encoded ontology, RACER checks for consistency and shows that encoded definitions are consistent (Fig. 8).

\[ X \subseteq ARule \]

\[ X \equiv \prod(\exists hasB_j \cdot B_j, \text{ for } 1 \leq j \leq i) \]

\[ \prod(\neg \exists hasB_k \cdot B_k, \text{ for } i < k \leq n) \]

VII. CONCLUSION

Successful analysis and management of requirement plays a crucial role in the success of any software product. In this paper our emphasis is on the analysis of product line based web service composition. We explained that product line ideas can be successfully applied to web service to apply the reuse mechanism of software product line. We performed a detailed analysis of a travel agent web service in the light of software product line. Besides analyzing various feature requirement as well as service composition, we have also carried out a formal analysis of the feature diagram by applying semantic web technique.

Our future plan is to extend the analysis model of the web service composition. In the goal model we only describe the success state of the goals. We also need to analyze the conditions for which a goal may fail. Analyzing the failed conditions we can improve the service composition. Our plan is to perform failure analysis and use any supporting tool to automate the analysis operations.

In OWL verification, we only performed consistency checking. We are also conducting experiment for instantiation checking for the web services. Our plan also includes representing the web service by using OWL-S which is a dialect of OWL to represent specifically web service compositions.

REFERENCES


Nakul Chandra Das completed his B.Sc in computer science and engineering from East West University, Dhaka, Bangladesh in 2013. He is a member of Software Engineering and Formal Method Research Group, East West University. He is interested in web service composition, semantic web service, software engineering, and software product line.
Shamim Ripon is an associate professor in the Department of Computer Science and Engineering, East West University, Dhaka, Bangladesh where he leads Software Engineering and Formal Method Research Group. Previously, he was a Research Associate in the Department of Computing Science, University of York, UK and Research Fellow in the Department of Computing Science, University of Glasgow, UK. He also served as a lecturer in Khulna, University, Bangladesh. He is a member of IAENG, Senior member of IACSIT.

Dr. Ripon holds a B.Sc. in computer science and engineering from Khulna University, MSc in Computer Science from National University of Singapore and PhD in Computer Science from University of Southampton, UK. His research interests focus on the requirement engineering, software product line, semantic web, natural language processing. His current research examines the formal representation and verification of knowledge based requirement specification.

Orin Hossain completed her B.Sc in computer science and engineering from East West University in 2013. She is a member of Software Engineering and Formal Method Research Group, East West University. She is currently working towards formal verification of web services composition. She is also interested in semantic web based requirement analysis of product line requirements.

Mohammad Salah Uddin is a lecturer in the Department of Computer science and Engineering, Central Women’s University, Dhaka, Bangladesh. He published several journals and international conference paper from his research work. He is a member of IACSIT and Software Engineering and Formal Method Research Group of East West University. He received his B.Sc in computer science and engineering from East West University, Dhaka, Bangladesh in 2012. He is interested in web service composition, semantic web service, knowledge representation, Mobile apps, Software Engineering, Software Product Line, Modeling and Verification.