Undefined 0 (2015) 1–0
IOS Press

One Size Does Not Fit All: Logic-based Clustering for On-the-fly Semantic Web Service Composition and Verification

Khai Huynh, Tho Quan, and Thang Bui
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology, VietNam
E-mail: {htkhai, qttho, thang}@cse.hcmut.edu.vn

Abstract. Recently, web service composition has been emerging widely since it is obviously hopeless to develop a specific web service which can single-handedly fulfill completely a requirement posed from clients. Preferably automatic, this task requires an efficient mechanism to semantically well-define the web services, which are perfectly fulfilled by the usage of ontology in *semantic web services*. Since ontological concepts are commonly comprehensive among computer-based systems, semantic web services can be not only composed precisely, but also verified efficiently from their functionality and QoS (Quality of Services) constraints.

However, composition and verification tasks always suffer from huge computational cost, which make *clustering* approaches naturally considered. However, typical clustering techniques neither ensure the soundness nor completeness of a composition solution. In this paper, we suggest a logic-based approach for clustering of semantic web services. The clustering results are then further applied for service composition and verification in an on-the-fly manner. In theoretical aspect, our approach achieves both soundness and completeness. As for practical result, our metric of logic-based similarity generates more reasonable clusters, resulting in significant performance improvement enjoyed in our experiments.

Keywords: Logic-based Semantic Web Service Clustering, Logic-based Semantic Web Service Similarity, Web Service Clustering, Web Service Composition, Web Service Formal Definition

1. Introduction

1.1. Web Service Composition and Verification

Nowadays, advent of *web service* is considered as a technology bringing a revolution operations of online B2B (Business to Business) and B2C (Business to Customer) applications. The basic value of web services is that it provides standard ways to access to the packaged and stand-alone systems. Software was written by various programming languages and running on various platforms can use web services to communicate with each others in order to handle and process data via the Internet.

Service-Oriented Architecture (SOA) is an approach used to create an architecture based upon the use of services [1]. However, when a web service usually provides a simple functionality, a single service can not meet the client requirement in many practical cases. For example, if a user wants to travel, he or she not only wants to book a flight to a $City$, but also be being exposed with $Sightseeing$ of this city, and in the meantime taking care of the price of nearby $Hotel$. Once locating suitable hotel, the user may want to make the $Reservation$ and perform a $Payment$. It prompts the issue of *Web Service Composition (WSC)*, the process to produce *composite web service*, which is a collection of services that will be executed in a specific order to serve a user requirement.

Each web service has functional and non-functional (or Quality of Service – QoS) properties [2]. Functional properties are the input and output of a web ser-

vice. They represent the functionality of the service. The user requirement on functional properties is called *hard constraint*. In addition to those functional properties, there are many QoS properties of web services, such as the *response time*, *execution cost*, *availability*, etc. For example, all web services described in Table 1, which provide the functions related to a travel booking process, have their QoS property on response time (*respTime*). The user requirement on QoS properties is called *soft constraint*.

There have been much studies on WSC recently. These studies are based on different approaches [3]. Some approaches only deal with the hard constraint using the theory of artificial intelligence (AI) planning as [4] [5] [6] [7]. There are typical frameworks in this approach have been proposed such as PORSCE II [4], OWLS-XPlan [5], etc.

However, as QoS is concerned, it is significant that the composed web service should satisfy the soft constraints. This is a complex verification task since we need to take into account all of QoS of each individual web service, which may yield different results when combined in different manners [2]. As such, the integrated task of web service composition and verification suffers from a huge complexity in order to find all of possible compositions and verifies each of them.

Model checking [8] has been attracted much attention, since it provides a solid mathematical background to handle the above discussed issue. There are various model-checking-based approach for web service composition such as [9], [10], [11], [12]. However, most of them (such as [9] or [12]) rely on *composition schema* which is a combination of all possible composition solutions (based on the hard constraint). A *model* is then developed from this schema to be verified. A model is basically a finite state machine based on which a model checker can verify a property by exploring all of possible states. In our context, the goal of verification is to choose the composition which satisfies all of QoS.

These approaches have two drawbacks. The first one is *computation complexity*. That is, we have to create the composition schema based on hard constraint. This is a NP-hard problem [13]. The second one is *the frequent changes in user requirements*. Whenever the requirements change, a new schema and the corresponding model have been built again before we can verify and choose the suitable composition for the user. This is very costly.

Our previous work presents in [14] has addressed the second problem. In [14], we adopt the formal definition of *Labelled Transition System* (LTS) [15] to formally represent all of concerned web services as a single LTS-based model, known as *LTS4WS*. In order to compose and verify web services, [14] uses an on-the-fly approach combining with some heuristics allowing early termination to be applied for unfeasible composition. Moreover, the *LTS4WS* is only needed to be generated once and remained unchanged (until the repository changes). When it supports various composition purpose, thus we do not need to rebuild the model for every user requirement.

1.2. Semantic Web and Semantic Web Service

The *Semantic Web* identifies a set of technologies, tools and standards that support the vision of the web associated with meaning form the basic building blocks of an infrastructure [16]. It provides a process level description for the web service that in addition to functional information, models the preconditions and post conditions of the process that can be inferred the evolution of the domain logically.

Semantic web service is the software component that provide dynamic service discovery, composition and invocation of web services and facilitate automated handling of web services for the users. Semantic web and web services are synergistic: the semantic web transforms web into a repository of computer readable data, while web services provide tools for the use of that data. Concept of Semantic Web Service (SWS) has been established [17][18] officially for a decade.

The SWS research area started in the early 2000s. The main goal of SWS approaches is the automation of service discovery and service composition in a SOA [19]. The web service descriptions are described semantically using *ontology* [20], thus specifying their interface in a machine-readable manner.

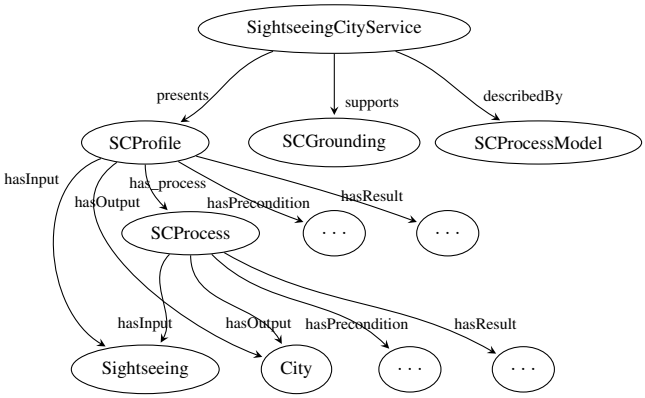
For example, the service *SightseeingCityService* in Table 1 is represented as in Fig. 1. In which, Fig. 1a is the WSDL [21] description of web service. It does not contain semantic aspect. The input (output) parameters are specified as the atomic data types, such as *string*, *int*, *float*, etc., or the structure of these data types. Fig. 1b shows the specification of web service in OWL-S [22]. A semantic web service in OWL-S is also considered as an ontology. It is an upper ontology of OWL [23] and visually represented as in Fig. 1c. The inputs and outputs are specified as ontologies. These ontologies are taken from an ontology structure which is graphically visualized rely on the idea of T-Box and A-Box [24] as in Fig. 1d. Basically, a T-Box captures

```
[...]
<wsdl:definitions>
  <wsdl:types>
    <xsd:element name="City" type="CityType"/>
    <xsd:element name="Sightseeing" type="SSType"/>
    <xsd:simpleType name="CityType">
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="SSType">
      <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
  </wsdl:types>
  <wsdl:message name="get_CITYResponse">
  [...]
  </wsdl:message>
  [...]
  <wsdl:portType name="SightseeingCitySoap">
  [...]
  </wsdl:portType>
  <wsdl:binding name="SightseeingCitySoapBinding">
  [...]
  </wsdl:binding>
  <wsdl:service name="SightseeingCityService">
  [...]
  </wsdl:service>
</wsdl:definitions>
```

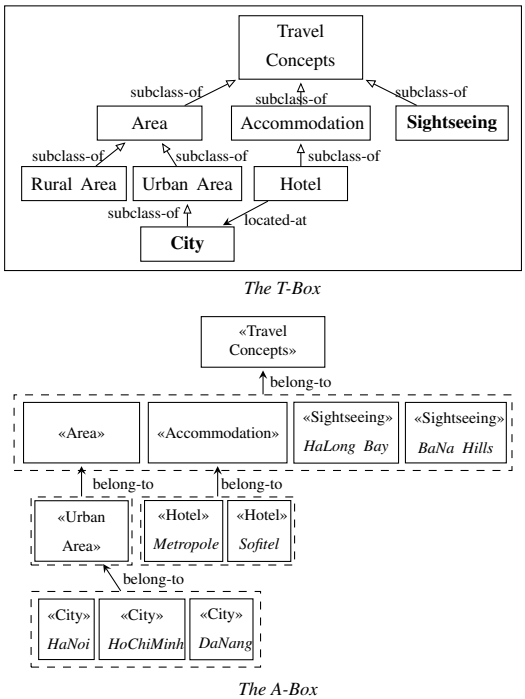
(a) The WSDL description

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#">
  <service:Service rdf:ID="SightseeingCityService">
    <service:presents rdf:resource="#SCProfile"/>
    <service:describedBy rdf:resource="#SCProcessModel"/>
    <service:supports rdf:resource="#SCGrounding"/>
  </service:Service>
  <profile:Profile rdf:ID="SCProfile">
    <profile:hasInput rdf:resource="#Sightseeing"/>
    <profile:hasOutput rdf:resource="#City"/>
    <profile:has_process rdf:resource="SCProcess" />
  </profile:Profile>
  <process:AtomicProcess rdf:ID="SCProcess">
    <process:hasInput rdf:resource="#Sightseeing"/>
    <process:hasOutput rdf:resource="#City"/>
  </process:AtomicProcess>
  <process:Input rdf:ID="Sightseeing">
    <process:parameterType rdf:datatype="&Sch;#anyURI">
      http://127.0.0.1/ontology/travel.owl#Sightseeing
    </process:parameterType>
  </process:Input>
  <process:Output rdf:ID="City">
    <process:parameterType rdf:datatype="&Sch;#anyURI">
      http://127.0.0.1/ontology/travel.owl#City
    </process:parameterType>
  </process:Output>
</rdf:RDF>
```

(b) The OWL-S description



(c) A part of semantic web service ontology (OWL-S)



(d) The domain ontology

Fig. 1. An example about description of web service *SightseeingCityService*

the relations between *concepts* and an A-Box describes *instances* of concepts.

1.3. Web Service Clustering

Even though the research on [14] has achieved some improvements, it still suffers from the state space explosion problem in verifying large-scale repository of web services. It is also a big issue in this area, which is commonly addressed by clustering. In which, the calculation of distance between web services when performing clustering will based on the features extracted from textual web service description, such as [25], [26], etc., or based on ontological semantic, such as [27], [28], [29], [30], etc., or combined both of them [31].

The typical approach by most of those works is to cluster web services into several clusters, mostly based on the information extracted from the input, output and description of the services. The most suitable clusters to the user requirement are then selected to be used in composition.

Although those clustering-based approaches reduce the number of web services considered as candidates for the composition solution, they still face the difficulty on deciding the size and number of the selected clusters. If only a few clusters are selected, we may miss some solutions. In contrast, if too many clusters are selected, the problem space may be enlarged unnecessarily. In other words, one cannot ensure the soundness nor the completeness of the solutions suggested from clustering approaches.

In this paper, we propose a novel approach on logic-based clustering for the semantic web service composition and verification. In this approach, web service is represented by a logical expression. A dataset of web services is grouped into clusters based on the calculation of the similarity between their logical expression. Each cluster is also represented by a logical expression so-called *Representative Logical Expression*. We then prove that our clustering method ensures the soundness and completeness of the composition solution.

Contributions. The contributions of this paper are summarized as follows:

- Proposed an approach which combines clustering and model checking for composition and verification of web services. This approaches extends the work in [14] in which web services can be composed and verified in on-the-fly manner w.r.t various kinds of constraint. This work is enhanced with clustering technique to enjoy significant improvement of performance.
- Presented a logic-based approach to present web services and clustering results. This approach contributes in threefold as follows: (i) logic-based representation that allows us to integrate clustering results with formal representation of model checking effectively; (ii) introduce a logic-based similarity of web services which presents more reasonable clustering result; and (iii) the composition approach has been proved to be sound and complete.

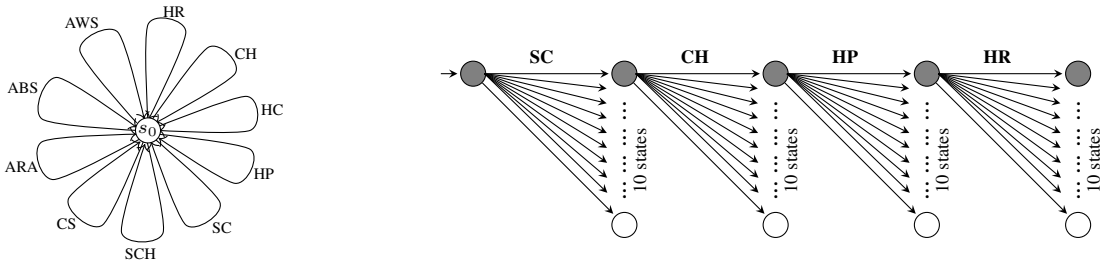
Outline. The rest of the paper is organized as follows: Section 2 presents the motivating example. In Section 3, we present detail about the preliminaries. Section 4 presents about logic-based web service clustering. This is the main section of this paper. Section 5 supplies the case study and the experiments are represented in Section 6. Section 7 is for the related works and the conclusion and future work are presented in Section 8.

2. Motivating Examples

Suppose that we have a web services repository with 10 web services as in Table 1, and the user would like to book a tour. The user then provides the place to travel to ($Sightseeing$) and the date of the travel ($Dates$), and requires the prices of some hotels ($Price$) near by the $Sightseeing$ and their reservation ($HotelReservation$). Besides these functional requirements (*hard constraint*), the user also ask for the quality of service (*soft constraint*), such as the response time ($respTime$) of web services must not exceed 30 seconds. In addition, the user also wants to know about $Price$ before the $HotelReservation$ when the price is the important information to choose the hotel. This requirement can be represented as a temporal relationship between web services. These requirements are shown in Table 2.

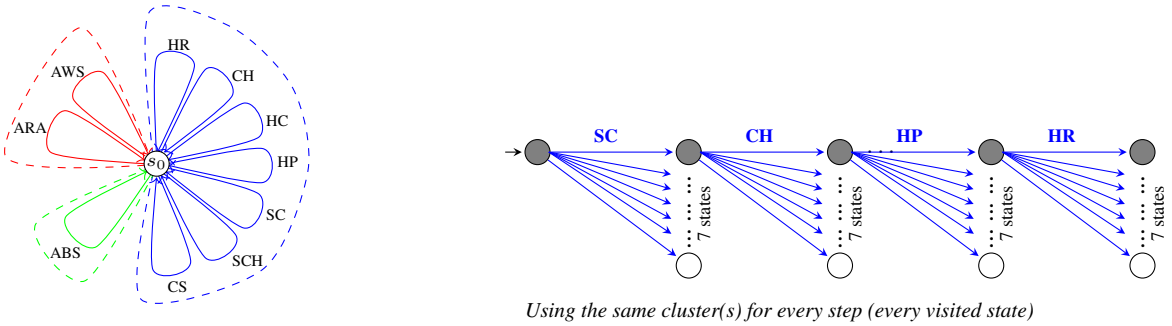
Example 1. *The advantage of web service clustering*

As mentioned in the earlier text, the work of [14] will build the LTS model for the input web service (*LTS4WS*) and explore the state space in order to compose and verify the web services. The LST4WS model of the web services in Table 1 is illustrated in Fig. 2a. At every composing step, [14] will examine all web services to generate the real states (and their heuristic



(a) The LTS4WS model [14] (b) The expanded state space

Fig. 2. The model and expanded state space of unclustered web service repository

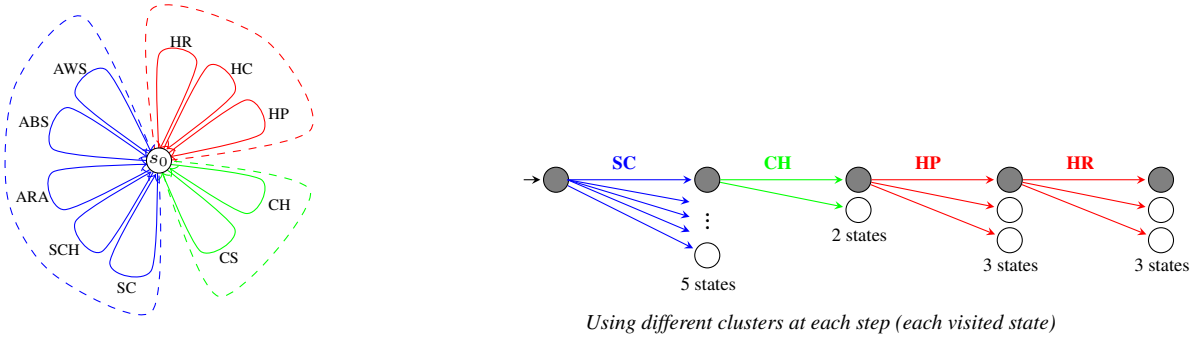


(a) The LTS4WS model and its normal clustering

Using the same cluster(s) for every step (every visited state)

(b) The expanded state space

Fig. 3. The model and expanded state space of combined feature and semantic-based clustered web service repository [31]



(a) The LTS4WS model and its logic-based clustering

Using different clusters at each step (each visited state)

(b) The expanded state space

Fig. 4. The model and expanded state space of Logic-based clustered web service repository

values) before choosing a best state (ie. the current best composition). This process is visually represented in Fig. 2b. It is easy to see that, the number of expanded (examined) states is very large, although [14] returns a very good composition (that has also been verified). In general, [14] will expanded at least $n * m$ states, where m is the length of the composition and n is the number of input web services. The approach in [14] still suffers from the memory usage and time consuming.

In this example, [14] returns the composition as $\{SC \bullet CH \bullet HP \bullet HR\}$. It also means that, many web services never involve in any step of the composition process. Therefore, some studies proposed some solutions to reduce the set of candidate web services that can be used in a web service composition. It can be done by clustering the input web services into clusters such that only some related clusters (of web services) are used in a case. The other clusters will be

Table 1
The Travel Booking web service repository

#	Service Name	Input(s)	Output(s)	respTime
1	HotelReserve-Service (HR)	Dates, Hotel	HotelRe-servation	5
2	CityHotelService (CH)	City	Hotel	3
3	HotelCityService (HC)	Hotel	City	3
4	HotelPriceInfo-Service (HP)	Hotel	Price	10
5	SightseeingCity-Service (SC)	Sightseeing	City	2
6	SightseeingCity-HotelService (SCH)	Sightseeing	City, Hotel	16
7	CitySightseeing-Service (CS)	City	Sightseeing	4
8	AdventureRural-AreaService (ARA)	Adventure	RuralArea	5
9	ActivityBeach-Service (ABS)	Activity	Beach	5
10	AreaWeather-Service (AWS)	Area	Weather	5

Table 2
Requirements for Travel Booking web service

Kinds of constraint	Value(s)
Hard constraint:	Input: $Dates, Sightseeing$ Output: $Price, HotelReservation$
Soft constraint:	$ResponseTime \leq 30$
Temporal relationship:	$\Box(\neg HotelReservation \cup Price)$

put away in that case. Thus, the number of web services that will be expanded in each composition step is smaller, and the performance of the whole process will be improved.

The work in [31], a combined feature and semantic-based web services clustering approach, groups the web services in Table 1 into 3 clusters: the first consists of 7 web services when their functional properties are related, the second consists of the 8th and 10th web services when their semantic related on the $Area$ and $RuralAreal$ and the last consists of only the 9th. The clustering is represented in Fig. 3a.

Applying this clustering approach to the composition and verification method in [14], only 7 web services in the first cluster will be consider as the input web services. Obviously, the number of expanded states at every composition step is reduced to 7, the total expanded states will be only $\sum(n_i) * k$, where $\sum(n_i)$ is the number of web services of the chosen clusters. The expanded state space in this case is shown in Fig. 3b.

Unfortunately, in some cases, the throw away web services may be the answer. And the composition resulted from the approach may not be optimized (in term of quality of service) when the set of input web services has been restricted before hand.

Instead of examining all web services in the chosen clusters at every composition step, we can examine web services in the most appropriate cluster first. The web services in the next appropriate cluster and the rest can be considered later in the backtrack stage of the composition process. So, in most of the case, when only the number of web services in one cluster is expanded then this approach has the same performance as the above. When there is no restriction on the input web services, there is no missing solution and there is a chance for optimization.

Example 2. *The advantage of logic-based web service clustering*

Considering only semantic and features may lead to the case that in the same group (cluster), some web services may be contradictory in term of logical meaning, even though they have the same semantic and features. For example, the web services *CityHotelService* (take *Hotel* and return *City*) and *HotelCityService* (take *City* and return *Hotel*) are in the same cluster when they have the same semantic. At a particular composition step, if we have the *City* and need the *Hotel*, then, obviously, we will consider only the service *CityHotelService*, not *HotelCityService*. They are logically opposite and should not be in the same cluster. The same observation could be applied to the pair of *CitySightseeingService* and *SightseeingCityService*.

Using our observation on logical aspect, the set of web services in Table 1 can be clustered into three groups consist of five web services, three web services and two web services, respectively. They are illustrated in Fig. 4a. It is easy to observe that, the logical opposite web services *HotelCityService* (HC) and *CityHotelService* (CH) are now groups into two different clusters. So then, there is no contradiction when examining web services of a selected cluster at every composition step.

The proposed clustering approach is in Section 4.4. In general, the similarity of any two web services is calculated using both logic-based similarity and

semantic-based similarity. In this example, HR, HC, HP are grouped into one cluster when there are similar both in logic and semantic. Similarly, the web services CH and CS are grouped into one cluster and web services AWS, ABS, ARA, SCH, SC are grouped into another.

Moreover, we can assign a logic expression to each cluster as an abstract web service. For example, the logic expressions of the above clusters are "$Hotel \rightarrow something$", "$City \rightarrow something$", and "$SightSeeing \wedge Area \rightarrow something$", respectively. Note that, the $SightSeeing$ and $Area$ are grouped into one when on our ontology, the $Area$ is more similar to the $SightSeeing$ than $Hotel$ or $City$.

At every composition step, a cluster can be selected if its abstract web service matches the current logical expression representing the user requirement. As mentioned in Example 1, when there are many matched clusters at the same time, the most appropriate cluster will be selected, the rest will be asked to wait.

In this special example, the composition only expanded 13 states in total (see Fig. 4b). It is much better than the previous approach (13 states vs. 28 states in Fig. 3b). Of course, when all the web services in the selected cluster are checked without success (in building a composition), the web services in a waiting cluster will be considered and expanded. In this case, the performance will be affected.

Importantly, there is no missing solution when there is no banned web service and there is a chance of optimization as mentioned earlier in this text.

Before proposing our approach on logic-based clustering, the background knowledge on ontology, semantic web service, modeling web service will be represented in the next Section.

3. Preliminaries

3.1. Ontology and Semantic Web Service

Ontology is a shared conceptualization of the world. Ontologies provide a common understanding of a particular domain and also provide a set of well-founded constructs to building meaningful higher level knowledge for specifying the semantics of terminology systems [16]. The ontology is defined as follow:

Definition 1 (Ontology)**.** *An **ontology** consists of four elements (C, A^C, R, X), where C represents a set of concepts, A^C represents a collection of attributes sets, one for each concept, and $R = (R_T, R_N)$ represents a set of relationships, which consists of two elements: R_N is a set of* nontaxonomy relationships *and R_T is a set of* taxonomy relationships*. Each concept c_i in C represents a set of objects, or* instances*, of the same kind. Each object o_{ij} of a concept c_i can be described by a set of attributes values denoted by $A^C(c_i)$. Each relationship $r_i(c_p, c_q)$ in R represents a binary association between concepts c_p and c_q, and the instances of such a relationship are pairs of (c_p, c_q) concept objects. X is a set of axioms. Each axiom in X is a constraint on the concept's and relationship's attribute values or a constraint on the relationships between concept objects. The constraints can be described using the SWRL [32] format.*

Example 3. Let TO is the Travel ontology, $TO = (C, A^C, R, X)$, where its components are endowed as follows:
C={"Travel Concepts", "Area", "Rural Area", "Urban Area", "Accommodation", "City", "Hotel", "Sightseeing"}
A^C("City")={"Name", "Weather"}
A^C("Hotel")={"Name", "Rating"}
A^C("Sightseeing")={"Name", "Type"}
R_T={subclass-of("Hotel","Accommodation"),
subclass-of("City", "Urban Area"),
subclass-of("Sightseeing","Travel Concepts"),
subclass-of("Accommodation","Travel Concepts"),
subclass-of("Area","Travel Concepts"),
subclass-of("Rural Area","Area"),
subclass-of("Urban Area","Area")}
R_N={located-at("Hotel","City"),
near-to("City", "City"),
near-to("Sightseeing","City")}
X={Implies(Antecedent(subclass-of(I-variable(x1) I-variable(x2))) Consequent(superclass-of(I-variable(x2) I-variable(x1))))
Implies(Antecedent(located-at(I-variable(x1) I-variable(x2))) Consequent(belong-to(I-variable(x2) I-variable(x1))))
Implies(Antecedent(near-to(I-variable(x1) I-variable(x2))) Consequent(near-to(I-variable(x2) I-variable(x1)))) }

Generally speaking, TO consists of a set of concepts, start with *Travel Concepts*, whose instances can be any concepts related to travel domain. In our context, we consider some of concepts: *Area*, *Rural Area*, *Urban Area*, *Accommodation*, *Sightseeing*. *City* is a subconcept of *Urban Area* and *Hotel* is a subconcept of *Accommodation*. In this example, we have two in-

stances of *Sightseeing*: *HaLong Bay* and *BaNa Hills*; three instances of *City*: *HaNoi*, *HoChiMinh*, *DaNang*; and two instances of *Hotel*: *Metropole*, *Sofitel*. To graphically visualize this ontology, we use the T-Box and A-Box structure [24] as in Fig. 1d.

OWL-S [22] is an upper ontology based on OWL (Ontology Web Language [23]). OWL-S is used in order to describe knowledge concerning semantic web services. Using OWL-S, the semantic description of web services can be understood by machines as well as humans. Therefore, it enables intelligent agents to discover, invoke and compose web services automatically [4]. A web service description in OWL-S is comprised of [22]:

- *Service Profile*: contains the information describing what the service accomplishes, limitations on service applicability and quality, and requirements that the service requester must satisfy to use the service.
- *Process Model*: describes the way an agent communicate and use the service.
- *Service Grounding*: contains the detailed description of how the client can access a service, such as communication protocols.

In particular, the information on the *Service Profile* is used in the process of discovering and composing services.

Definition 2 (Web Service)**.** *A **web service** $\mathcal{W}$ is a 5-tuple $\mathcal{W} = (\mathcal{N}, \mathcal{D}, \mathcal{P}, \mathcal{E}, \mathcal{Q})$, where:*

- *$\mathcal{N}$ is the unique name of $\mathcal{W}$,*
- *$\mathcal{D}$ is the description of web service,*
- *$\mathcal{P}$ is the pre-condition that must be satisfied before $\mathcal{W}$ is invoked,*
- *$\mathcal{E}$ is the expression describing the effect (or post-condition) after $\mathcal{W}$ is invoked,*
- *$\mathcal{Q}$ is a set of QoS properties, each QoS property comes in name/value pair, such as $\{respTime = 3\}$.*

Definition 3 (Semantic Web Service)**.** *A **semantic web service** $\mathcal{W}$ is a web service, associated with an ontology, $\mathcal{SWS} = (\mathcal{W}, \mathcal{O})$, where:*

- *$\mathcal{W}$ is the web service is defined in Definition 2,*
- *$\mathcal{O}$ is the ontology used to semantic description for the semantic web service, $\mathcal{O}$ is defined in Definition 1.*

Example 4. Consider the semantic web service *SightseeingCityService*, which takes the name of place to travel to (*Sightseeing*) and returns the name of city (*City*) near by or contains the *Sightseeing*. Its response time (*respTime*) is 5 seconds. The *SightseeingCityService* is given by the Definition 3 as follows:

SightseeingCityService = $(\mathcal{W}, \mathcal{O})$, where:

- $\mathcal{W}$ = ($SightseeingCityService$, *"Return the City which the Sightseeing belong to"*, $Sightseeing == 1$, $City = 1; respTime+ = 5$, $respTime = 5$)
- $\mathcal{O}$ is an ontology is given in Example 3.

3.2. Representation of Web Service as Model

In order to acquire the model checking into the composition and verification, the web service problem must be represented as model. In this section, the definitions of *Labelled Transition System* (*LTS*) and *LTS for web services* (*LTS4WS*), which are used in representing the ws system as model checking model, will be defined.

Definition 4 (Labelled Transition System)**.** *A **Labelled Transition System** (LTS) is a 4-tuple $\mathcal{L} = (S, s_0, L, \delta)$, where:*

- *S is a set of states,*
- *$s_0 \in S$ is the initial state,*
- *L is a set of action labels,*
- *$\delta : S \times L \times S$ is a transition relation. For convenience, we use $s \xrightarrow{a} s'$ to denote $(s, a, s') \in \delta$.*

A transition may also have a *pre-condition*, a condition that must always be true before firing the transition and an *effect* expression that expresses the effect after the transition is fired. An LTS with the transitions have *pre-condition* and *effect* expression is called *guarded LTS*.

Definition 5 (LTS for Web Services)**.** *Let $WS = \{\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_n\}$ be a set of web services. A **Labelled Transition System for Web Services (LTS4WS)** of L_w is an LTS $\mathcal{L}^{WS} = (\{s_0\}, s_0, L, \delta)$, where:*

- *$\{s_0\}$ is a set of states which has only one state s_0,*
- *s_0 is the initial state,*
- *$L = \{\mathcal{W}_1, \mathcal{W}_2, ..., \mathcal{W}_n\}$, w.r.t the notation described in Definition 2,*
- *δ is a set of transition relations of the form $s_0 \xrightarrow{\mathcal{W}_i} s_0$.*

Example 5. The LTS4WS model for 10 web services in Table 1 is defined as follows:

- Number of state(s): 1 (s_0)
- The initial state: s_0
- The set of label actions: $L = \{HR, CH, HC, HP, SC, SCH, CS, ARA, ABS, AWS\}$
- The set of transition relations: $\delta = \{s_0 \xrightarrow{HR} s_0, s_0 \xrightarrow{CH} s_0, s_0 \xrightarrow{HC} s_0, s_0 \xrightarrow{HP} s_0, s_0 \xrightarrow{SC} s_0, s_0 \xrightarrow{SCH} s_0, s_0 \xrightarrow{CS} s_0, s_0 \xrightarrow{ARA} s_0, s_0 \xrightarrow{ABS} s_0, s_0 \xrightarrow{AWS} s_0\}$

This LTS4WS model is represented visually in Fig. 2a.

Using this model, a model checker will search over the state space generated from this LTS model for states that satisfy the user requirement specified by hard, soft and temporal relationship constraints. For example, with the model as in Fig. 2a and the user requirement as in Table 2, the searching state space is visually represented as in Fig. 2b. Note that we can always prune the paths that violate the constraints at any time. As mentioned in Section 1, work of [14] has an effective searching strategy when it integrated the heuristics into searching and traversing the state space.

4. Logic-based Web Service Clustering

4.1. Logic Representation of Web Service

Definition 6 (Logic Expression of Web Service)**.** *A **web service** is formally defined by the features on input and output as follow:*

$$WS \equiv f_{in} \rightarrow f_{out} \equiv$$
$$f_{in_1} \wedge \ldots \wedge f_{in_m} \rightarrow f_{out_1} \wedge \ldots \wedge f_{out_n} \quad (1)$$

where:

- f_{in}: the logic expression represented the input functional properties and the preconditions,
- f_{out}: the logic expression represented the output functional properties and the effects.
- f_{in_i}, f_{out_j} are the terms or ontologies which belong to the domain of $2^{(C \times A^C \times R \times X)}$ – the power set of $(C \times A^C \times R \times X)$. Where, C, A^C, R, and X are defined in Definition 1.

Example 6. Let WS_1 be the *HotelReservationService* service. $WS1$ takes $Hotel$ and $Dates$, returns information about $HotelReservation$. We have:

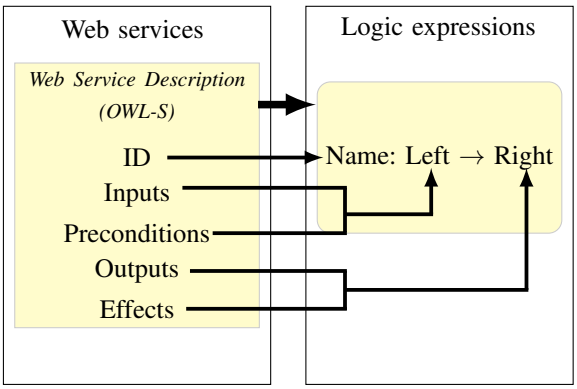


Fig. 5. Web service to logic expression mapping

- $f1_{in}$: $Hotel \wedge Dates$
- $f1_{out}$: $HotelReservation$.

or: $WS1 \equiv Hotel \wedge Dates \rightarrow HotelReservation$

4.2. OWL-S to Logic Expression Translation

The OWL-S to logic expression translation is the process which extracts the information about *name*, *input*, *output*, *precondition*, and *effect* of web services and converts them to the elements of logic expressions. Let WSP_i be a profile of web service ith and f_i be the transformed logic expression, respectively. Each element is transformed as follows:

- The name of logic expression is the ID of service profile: $name(f_i) = WSP_i.ID$
- The left-hand side of logic expression is the input and precondition of the web service: $left(f_i) = \bigcup_{k=1}^{n} WSP_i.hasInput_k \cup \bigcup_{k=1}^{m} WSP_i.hasPrecondition_k$
- The right-hand side of logic expression is the output and effect of the web service: $right(f_i) = \bigcup_{k=1}^{n} WSP_i.hasOutput_k \cup \bigcup_{k=1}^{m} WSP_i.hasEffect_k$

This translation process is described as a mapping in Fig. 5. An example of an OWL-S to logic expression transformation is presented in Fig. 6. In Fig. 6, the OWL-S description describes the *SightseeingCityService*, which has *Sightseeing* ontology as the input, *City* ontology as the output, and the relation *popular* as the precondition. This precondition describes that the *Sightseeing* must be a popular place. The OWL-S description is then transformed into the following logic expression: $Sightseeing \wedge popular(Sightseeing) \rightarrow City$.

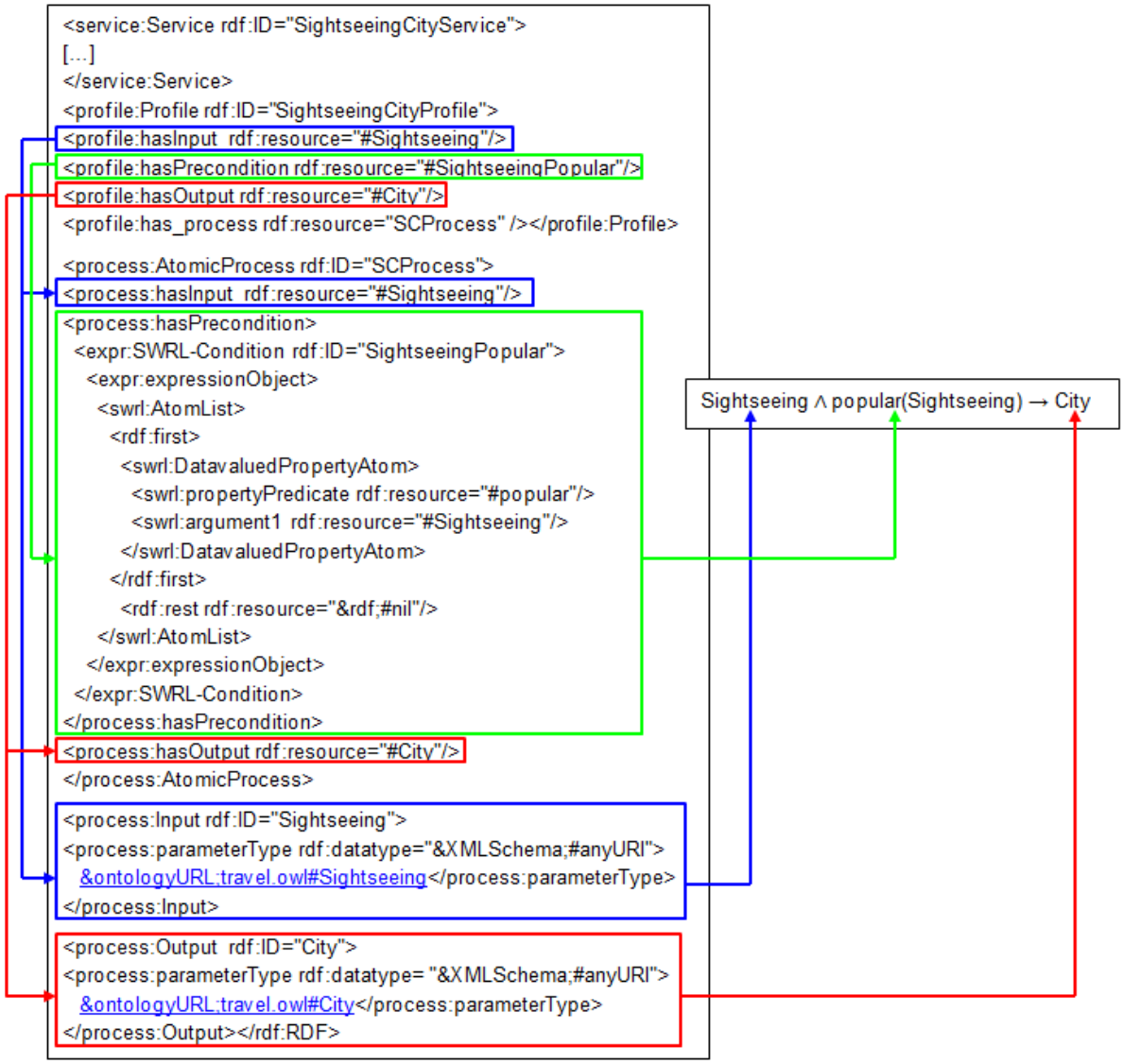


Fig. 6. OWL-S to logic expression translation example

4.3. Logic-based Semantic Web Service Similarity

Web service clustering is based on the similarity of web services. In our approach, we propose a novel metric of logic-based similarity between two semantic web services being represented as two logic formulas. Intuitively, our metric is evolved from classical similarity metrics as follows:

- *Feature-based similarity:* We extract features from the representative logic formulas, based on which their similarity is computed.
- *Ontology-based similarity:* We use ontology-based measure to evaluate the similarity the concepts involved the logic formulas. Thus, two similar ontological concepts will be deemed higher similarity degree.
- *Logic-based similarity:* Finally, logic interpretation is employed for the final similarity measure. We use *over-approximation* to get an over-approximated formula from the original formulas, based on which the similarity is computed. The idea is that if two formulas are logically similar, their over-approximated formula would also be similar to the original ones.

4.3.1. Feature-based Similarity

To calculate the feature similarity between two logic expressions, we have to determine *the common part* and *the different part* of two expressions. The common part and the different part are determined by the following definitions:

Definition 7 (Common part of logic expressions)**.** *Let f and g are two logic expressions. The **common part** (C) of two expressions is defined as follow:*

$$C = f \cap g = \{\forall f_i \in f\} \cap \{\forall g_j \in g\} \quad (2)$$

Whereas, f_i, g_j are atomic terms in f and g, respectively. Let a be the common factor between f and g.

$$a = |C|$$

Example 7. Let f_1 and f_{13} be the two logic expressions in Example 13, we have:
$f_1 = Dates \wedge Hotel \rightarrow HotelReservation$
$f_{13} = \neg Hotel \vee \neg Dates \vee City \vee HotelReservation$

$C = \{Dates, Hotel, HotelReservation\}$
$a = |C| = |\{Dates, Hotel, HotelReservation\}|$
$= 3$

Definition 8 (Different part of logic expressions)**.** *Let f and g be two logic expressions. The **different part** (D) of two expressions is defined as follow:*

$$D = (f \cup g) \backslash C = (\{\forall f_i \in f\} \cup \{\forall g_j \in g\}) \backslash C \quad (3)$$

Whereas, f_i, g_j are atomic terms in f and g respectively. Let b be the different factor between f and g.

$$b = \frac{|D|}{2}$$

Example 8. Let f_1 and f_{13} are two logic expressions in Example 13, we have:
$f_1 = Dates \wedge Hotel \rightarrow HotelReservation$
$f_{13} = \neg Hotel \vee \neg Dates \vee City \vee HotelReservation$

$$D = \{City\}$$
$$b = \frac{|D|}{2} = \frac{|\{City\}|}{2} = \frac{1}{2}$$

Definition 9 (Feature-based Similarity)**.** *Let f and g be two logic expressions represented of two web services. The **feature-based similarity** (Sim_{Fe}) of f and g is calculated based on the following formula:*

$$Sim_{Fe}(f, g) = \frac{a}{(a + b)} \quad (4)$$

Whereas, a and b are defined in Definition 7 and Definition 8.

Example 9. With the f_1 and f_{13} logic expression in Example 13 and the value of common part, different part is calculated in Example 7 and Example 8, we have the feature similarity of them:

$$Sim_{Fe}(f_1, f_{13}) = \frac{3}{(3 + \frac{1}{2})} = \frac{6}{7} \approx 0.86$$

Similarly, we have $Sim_{Fe}(f_3, f_{13}) = \frac{2}{3} \approx 0.67$.

4.3.2. Ontology-based Similarity

In feature-based similarity calculation, one needs to identify the common part and different part of two logic formulas. It is done by simply matching the terms in two formulas.

However, as the terms involved in logic formula of a semantic web service are in fact ontological concepts, we can enhance the feature-based similarity by introducing the ontology-based similarity. It is very similar to feature-based similarity, however, two terms are considered in the common part if their corresponding concepts are ontologically similar.

The semantic similarity between two sets of ontologies is calculated based on the average similarity of each pair of ontologies:

$$Sim_{On}(OnSet_1, OnSet_2) = \frac{1}{(n \times m)} \sum_{i=1}^{n} \sum_{j=1}^{m} Sim_c(On1_i, On2_j) \quad (5)$$

Whereas, $Sim_c(C_i, C_j)$ is the concept similarity of two concept C_i and C_j on the ontology hierarchical tree.

The influencing factors of concept similarity based on hierarchy are the following [28]:

1. *Depth of the hierarchy*: The domain ontology hierarchy is deeper, shows that the more exhaustive is the concept classification, and the more similar is the two concepts in the hierarchy.
2. *The steps of the shortest path*: The more steps of two concepts, the distance of them is farther, and the less similar is the two concepts.
3. *Depth of concept*: If the two concepts are deeper, it means the concepts are analyzed more comprehensive. So the semantic should be more similar. The similarity is positive with the sum depths of two concepts, and inverse proportion with the difference depths between two concepts.

4. *The concepts density*: Similar with the depth of concepts, if the concept has more brother concepts, it means the concepts are analyzed more comprehensive. So the concepts which have greater density, the similarity should be higher.

Through the analysis above, the depth and path of concepts measurements is calculated as follows:

$$Dep(C_i, C_j) = \frac{1}{2} \times \frac{(d_i + d_j)}{depth} \times \frac{1}{(max(|d_i - d_j|, 1)} \quad (6)$$

$$Path = \frac{1}{shortest\_path(C_i, C_j)} \quad (7)$$

- d_i, d_j: depth of C_i, C_j respectively.
- $shortest\_path(C_i, C_j)$: number of nodes on the shortest path from C_i to C_j
- $depth$: depth of ontology hierarchical tree.

Concepts density measurement depends on two areas: the amount of information, mean that the total number of ontologies on the ontology hierarchical tree; and the number of concepts belong to the ontologies of the subtree starting from the current ontology of the ontology hierarchical tree. The similarity on density is calculated as:

$$Den(C_i, C_j) = \frac{(BroCnt(C_i) + BroCnt(C_j))}{NodeCnt} \quad (8)$$

whereas:

- $BroCnt(C_i)$ is the number of concepts belong to the same concept as C_i (including C_i itself)
- $NodeCnt$ is the number of all concepts in the domain ontology hierarchy.

Example 10. Support that we have the ontology tree as in Fig. 7, the values of $Dep, Path, Den$ of concept H and G is calculated as follow:

- $Dep(G, H) = \frac{1}{2} \times \frac{1+2}{3} \times \frac{1}{1} = \frac{1}{2}$
- $Path(G, H) = \frac{1}{3}$
- $Den(H, G) = \frac{3+2}{17} = \frac{5}{17}$

After calculating the similarity Dep, $Path$, and Den, the similarity of two ontologies as follow:

$$Sim_c(C_i, C_j) = \varphi_1 Dep(C_i, C_j) + \varphi_2 Path(C_i, C_j) + \varphi_3 Den(C_i, C_j) \quad (9)$$

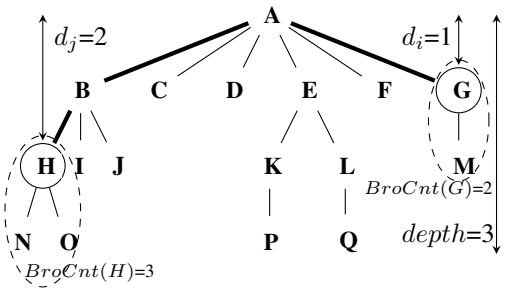


Fig. 7. Example about ontology tree

$\varphi_1, \varphi_2, \varphi_3$ are the weights used for adjusting the effects of Dep, Path and Den, respectively. $\varphi_1 + \varphi_2 + \varphi_3 = 1$.

Example 11. The concept similarity of G and H in Fig. 7 is:

$$Sim_c(H, G) = \frac{1}{3} \times \frac{1}{2} + \frac{1}{3} \times \frac{1}{3} + \frac{1}{3} \times \frac{5}{17} = \frac{115}{306}$$

4.3.3. Logic-based Similarity

Even though the ontology-based similarity, enhanced from feature-based similarity, can improve the similarity between two similar ontological concepts, this similarity measure still does not sufficiently reflect the similarity of the logic in data processing of the involved web services. We handle this by introducing the ultimate similarity measure: logic-based similarity of semantic web services.

Definition 10 (Over approximation of two logic expression)**.** *Let f_1 and f_2 are two logic expressions represented of the two web services. The **over approximation of two logic expressions** is defined as follow:*

$$f_{12} = f_1 \oplus f_2 = f_{1_{in}} \wedge f_{2_{in}} \rightarrow f_{1_{out}} \vee f_{2_{out}} \quad (10)$$

Where, $\oplus$ is an *over approximation operator*. Then, f_{12} will be simplified by a prover, such as Z3 [33].

Example 12. Support that we have three logic expressions:
$f_1 = A \rightarrow B$
$f_2 = B \rightarrow A$
$f_3 = A \rightarrow C$

The over approximation of f_1 and f_2 is calculated as follow:
$f_{12} = f_1 \oplus f_2 = A \wedge B \rightarrow A \vee B$
$= \neg A \vee \neg B \vee A \vee B = true$

The over approximation of f_1 and f_3 is f_{13} which is calculated as follow:

$f_{13} = f_1 \oplus f_3 = A \wedge A \rightarrow B \vee C$
$= \neg A \vee B \vee C$

Example 13. Consider the first three web services in Table 1. Their logic expressions are as follows:

- $f_1 = Dates \wedge Hotel \rightarrow HotelReservation$
- $f_2 = City \rightarrow Hotel$
- $f_3 = Hotel \rightarrow City$

Some over approximation of these expressions as follow:

- $f_{23} = f_2 \oplus f_3$
  $= (City \rightarrow Hotel) \oplus (Hotel \rightarrow City)$
  $= (City \wedge Hotel \rightarrow Hotel \vee City) = true$
- $f_{13} = f_1 \oplus f_3$
  $= (Dates \wedge Hotel \rightarrow HotelReservation) \oplus (Hotel \rightarrow City)$
  $= (Dates \wedge Hotel) \wedge Hotel$
  $\rightarrow City \vee HotelReservation$
  $= \neg Hotel \vee \neg Dates \vee City \vee HotelReservation$

Definition 11 (Logic-based web service similarity)**.** *Let WS_A, WS_B be two web services, f_A, f_B be the logic expressions represented for WS_A, WS_B, respectively, f_{AB} be the over approximation of f_A and f_B as in Definition 10. The **logic-based web service similarity** of WS_A and WS_B is calculated as follows:*

$$Sim_{Lo}(WS_A, WS_B) = \frac{Sem_{Fe}(f_A, f_{AB})}{2} + \frac{Sem_{Fe}(f_B, f_{AB})}{2} \quad (11)$$

where $Sem_{Fe}(f_1, f_2)$: the feature similarity between f_1 and f_2

The process of calculating the logic-based web service similarity of a pair of web services is implemented in Algorithm 1.

Example 14 (Logic-based web service similarity)**.** The application of Algorithm 1 to the first three web services in Table 1 is as follows:

The over approximation logic expression of each pair of web services:

- $f_{12} = f_1 \oplus f_2 = true$
- $f_{13} = f_1 \oplus f_3 = \neg Hotel \vee \neg Dates \vee City \vee HotelReservation$
- $f_{23} = f_2 \oplus f_3 = true$

The feature-based similarity of them:

- $Sem_{Fe}(f_1, f_{12}) = 0; \quad Sem_{Fe}(f_2, f_{12}) = 0$

Algorithm 1 Calculates the logic-based similarity of two logical expression services

Input: A pair of web services – WS_1, WS_2
Output: The value of logic-based web service similarity of WS_1, WS_2 – S
Process:

1: Build the logic expression for each web service – f_1, f_2 as in Definition 6;
2: Calculate the over approximation expression f_{12} of f_1, f_2;
3: Using Z3 prover [33] to simplify f_{12};
4: Calculate the number of common factor (a) and different factor (b) of each pair $f_1 - f_{12}$, $f_2 - f_{12}$ follow the Equation 2 and Equation 3;
5: Calculate the feature-based similarity $Sim_{Fe}(f_1, f_{12})$ and $Sim_{Fe}(f_2, f_{12})$ by Equation 4; with the common part is calculated by the semantic-based similarity by Equation 5;
6: Calculate the logic-based web service similarity – S of WS_1, WS_2 from $Sim_{Fe}(f_1, f_{12})$ and $Sim_{Fe}(f_2, f_{12})$ by Equation 11;
7: Return S;

- $Sem_{Fe}(f_2, f_{23}) = 0; \quad Sem_{Fe}(f_3, f_{23}) = 0$
- $Sem_{Fe}(f_1, f_{13}) = \frac{6}{7}; \quad Sem_{Fe}(f_3, f_{13}) = \frac{2}{3}$

And the logic-based similarity of each pair of web services:

- $Sim(WS_1, WS_2)$
  $= \frac{Sem_{Fe}(f_1, f_{12}) + Sem_{Fe}(f_2, f_{12})}{2} = 0$
- $Sim(WS_2, WS_3)$
  $= \frac{Sem_{Fe}(f_2, f_{23}) + Sem_{Fe}(f_3, f_{23})}{2} = 0$
- $Sim(WS_1, WS_3)$
  $= \frac{Sem_{Fe}(f_1, f_{13}) + Sem_{Fe}(f_3, f_{13})}{2} = \frac{\frac{6}{7} + \frac{2}{3}}{2} = \frac{16}{21}$

4.4. Logic-based Web Service Clustering

To perform clustering, we apply the K-means algorithm [34], one of the most popular clustering technique. Determining the number of clusters is an important problem of clustering algorithms in general. In the web service composition, suppose that we have N services clustered into k clusters. Each cluster has the average of N/k services. At each step of the composition, we need k comparisons to select the most appropriate cluster from k clusters. Then, we perform N/k comparisons in examining all web services in the selected cluster. Thus, the number of comparisons in each composition step will be $k + N/k$. Our objec-

tive is minimizing this value, and this value minimized when $k = \lfloor \sqrt{N} \rfloor$.

Another important problem is how to form the representative element for each cluster. Note again that our web services are represented by logic expressions. Therefore, our representative is also defined using logic expression.

Definition 12 (Representative Element)**.** *Let $F = \{f_1, f_2, \ldots, f_k\}$ is a cluster of logic expressions, the **representative element** (f_r) of F is defined as follow:*

$$f_r = \biguplus f_i = f_{1_{in}} \vee f_{2_{in}} \vee \cdots \vee f_{k_{in}} \rightarrow f_{1_{out}} \wedge f_{2_{out}} \wedge \ldots \wedge f_{k_{out}} \quad (1 \leq i \leq k) \quad (12)$$

where $\biguplus$ is the *representative operator* of every logic expression f_i.

The proof about the soundness and completeness of the representative element is given in Fig. 8.

Example 15. Support that three web services *HotelReservationService*, *CityHotelService*, and *HotelPriceInfoService* (the first three web services in Table 1) are clustered into a cluster. Then, the representative logical expression of this cluster is:

$$\begin{aligned} f_r &= f_1 \uplus f_2 \uplus f_3 \\ &= (Dates \wedge Hotel) \vee City \vee Hotel \rightarrow \\ &\quad HotelReservation \wedge Hotel \wedge Price \\ &= City \vee Hotel \rightarrow \\ &\quad HotelReservation \wedge Hotel \wedge Price \end{aligned}$$

After determining the number of clusters and the method of identifying cluster representative element, the K-means algorithm [34] is used to implement our logic-based web service clustering. The result of this algorithm is a set of logic expression clusters, each representing by a logic expression (the representative). It is also the group of the corresponding web services.

Example 16. The result of the logic-based clustering for the web services in Table 1 is in Table 3. In that table, the third column is for the logic expressions representing the corresponding clusters.

5. Case Study

For the set of web services in Table 1, the results of web service composition using mentioned approaches are shown on Table 4. It is easy to see that, the *OnTheFlyWSCV* tool [14] with logic-based clustering only fired 13 transitions, visited 10 states ($\sim$ 9 steps), and expanded 34 states (around $34/9 \sim 3.8$ states at each step in average). It is much better than the original tool itself and the normal clustering approach, which expanded 150 states and 105 states, respectively.

Table 4
The case study results

Approach	Expanded states	Visited states
Unclustered	150	16
Combined feature and semantic-based clustering [31]	105	16
Our approach: Logic-based clustering	34	10

Table 5
The experimentation datasets

Dataset	No. of web service	Description
Travel Booking (TB)	20	Including web services providing information to serve the travel booking
Book Store (BS)	40	Including services for books purchasing, payment, delivery to customer
Online Film Store (OFS)	60	Including services support to search, purchase and watch the online movies
Medical Services (MS)	80	Services support to lookup hospital, treatment, medicine, etc.
Education Services (EDS)	100	Services related to education such as scholarship, courses, degrees, etc.
Economy Services (ECS)	200	Including services provided information on goods, restaurant, food, etc.
Global	1000	1000 random services from OWLS-TC [35].

6. Experimentation

In this section, we will present the experimental results of our approach. To evaluate, we use the framework *OnTheFlyWSCV* [14] to compose web services. Our experiments work on the real datasets obtained from the project OWLS-TC [35]. OWLS-TC provides over 1000 semantic web services classified into different domains and described by OWL-S [22]. In this dataset, we select seven sub datasets with the number of web services is varied 20 to 1000 services as shown in Table 5.

We conduct the experiment scenario based on three approaches. These are *unclustered* approach; *Com-*

Table 3
The Travel Booking clustered web service repository

Clusters	**Web services**	**The representation logical expression**
Cluster 1	Activity BeachService (ABS) AdventureRuralAreaService (ARA) AreaWeatherService (AWS) SightseeingCityService (SC) SightseeingCityHotelService (SCH)	$Activity \vee Adventure \vee Area \vee Sightseeing \rightarrow Beach \wedge RuralArea \wedge Weather \wedge City$
Cluster 2	CityHotelService (CH) CitySightseeingService (CS)	$City \rightarrow Hotel \wedge Sightseeing$
Cluster 3	HotelReserveService (HR) HotelCityService (HC) HotelPriceInfoService (HP)	$Hotel \rightarrow City \wedge Price \wedge HotelReservation$

Proof. Soundness: $f_r \models f_i, \forall i \in \{1, \ldots, k\}$
We need to prove f_r ($f_{r_{in}} \rightarrow f_{r_{out}}$) represented for all f_i ($f_{i_{in}} \rightarrow f_{i_{out}}$). It means, suppose that we have the representative element (f_r), we will have all f_i in the cluster which f_r represented, or $f_r \models f_i$ ($\forall i \in 1, \ldots, k$)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f_{i_{in}}}\,assumption}{f_{1_{in}} \vee ... \vee f_{i_{in}} \vee ...}\vee i \qquad \dfrac{}{f_r \equiv f_{1_{in}} \vee ... \vee f_{i_{in}} \vee ... \rightarrow f_{1_{out}} \wedge ... \wedge f_{i_{out}} \wedge ... \wedge f_{k_{out}}}\,premise}{f_{1_{out}} \wedge \ldots \wedge f_{i_{out}} \wedge \ldots \wedge f_{k_{out}}} \rightarrow e}{f_{i_{out}}} \wedge e}{f_{i_{in}} \rightarrow f_{i_{out}} \equiv f_i} \rightarrow i$$

□

Proof. Completeness: $f_{g_{in}} \rightarrow f_{i_{in}} \models f_{g_{in}} \rightarrow f_{r_{in}}$
Suppose that there is a web service (represented by f_i) in cluster that can be used to compose the goal service (represented by f_g), $f_{g_{in}} \rightarrow f_{i_{in}}$. Then, the representative element (f_r) must satisfy the goal service, $f_{g_{in}} \rightarrow f_{r_{in}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f_{g_{in}}}\vee i \qquad \dfrac{}{f_{g_{in}} \rightarrow f_{i_{in}}}\,premise}{f_{i_{in}}} \rightarrow e}{f_{1_{in}} \vee \ldots \vee f_{i_{in}} \vee \ldots \vee f_{k_{in}} \equiv f_{r_{in}}} \vee i}{f_{g_{in}} \rightarrow f_{r_{in}}} \rightarrow i$$

□

Fig. 8. Proving the soundness and completeness of representation element

bined feature and semantic-based web service clustering approach – the approach was proposed in [31]; and *Logic-based web service clustering* approach – the approach is proposed in this paper. The experiments were executed on a PC with core i5-5200 processor (4 x 2.7 Ghz), 8.0 Gb of RAM, running on Windows 7 64-bit operating system. The result of experiments is evaluated in three aspects: The number of expanded states, the number of visited states, and the execution time. The experiment results are analyzed statistically in Table 6 and Fig. 9.

The experimental results confirm our hypothesis that the clustering helps in reducing the number of expanded states (see Fig. 9a). Of course, our logic-based approach reducing that number significantly.

Note that, the heuristic algorithm in the OnTheFly-WSC tool always chooses the best way to travel in the state space. The number of visited states in general shows the number of best states that have been chosen during the search. For the original and the second approach, when the "best" states (based only on features and semantic) are the same, the behaviors of the algorithms in both approaches are the same, then the number of visited states are the same. (Of course, they could be slightly different in practice.) (Fig. 9b)

In our approach, when the contradictory web services cannot be in the same cluster, then the number of "best" states are smaller, then the number of visited states obviously smaller. Unfortunately, it is just a slightly improvement on the performance. (Fig. 9b)

When the execution time mostly depends on the number of expanded states and visited states, the second approach is faster than the original one and our approach is the best (Fig. 9c)

7. Related Works

7.1. Web Service Clustering

In this section, we discuss the works related to web service clustering. Kumara et al. [31] proposed a term-similarity approach to calculating the semantic similarity of Web services. Firstly, it uses an ontology-learning method. If this fails to calculate the similarities, it then uses an information retrieval (IR)-based method. The ontology learning uses Web service description language (WSDL) documents to generate ontologies by examining the hidden semantic patterns that exist within the complex terms used in service features. In IR method, authors use both thesaurus-based and search-engine-based (SEB) term similarities. To address the second issue in service clustering, [31] proposes an approach that identifies the cluster center as cluster representative by combining the service-similarity value with the term frequency-inverse document frequency (TF-IDF) value of the service name, which reflects the importance of a service to its cluster.

The research in [27] proposed a non-logic-based matchmaking approach that uses correlated topic model [36] to extract topics from semantic service descriptions and search for services in the topic space where heterogeneous service descriptions are all represented as a probability distribution over topics. In this research, authors utilized the Correlated Topic Model (CTM) to extract latent factors and the correlation between these topics to propose an efficient Web Service Hierarchical Clustering. Based on the hierarchical nodes (clustered groups) and the extracted topics, a set of matched services can be returned by comparing the similarity between the query and the related cluster, rather than computing the similarity between the query and each service in the dataset. To archive this, this approach uses the Formal Concept Analysis (FCA) [37] formalism to organize the constructed hierarchical clusters into concept lattices according to their topics. Thus, service discovery may be achieved more easily using the concept lattice.

Xie et al. [28] proposed an ontology-based semantic web service clustering approach. To cluster web services, it calculates the semantic Web service similarity based on function similarity. The authors then design the matching method for inputs and outputs of web service based on the functional similarity. The web service similarity is based on an accurate concept semantic similarity of the domain ontology. A domain ontology hierarchy is defined to describe the concept semantic information. In this research, the concept semantic similarity is discussed from several aspects, such as the path between two concepts, the path weight, the density of concepts, and the antisense relationship. In the kernel, [28] uses the K-means clustering technique to cluster web services, and then recommends the similar services to Web service consumer. The cluster will improve the efficiency of Web service discovery.

7.2. Web Service Composition

Research community has a lot of work related to WSC problem. WSC only involves the functional properties (hard constraints) is the classic problem of

Table 6
Experimentation results

Datasets	**Approaches**	**Expanded states**	**Visited states**	**Execution time (s)**
TB (20)	Unclustered	1,100	56	0.250
	Combined feature and semantic-based clustering [31]	825	56	0.210
	Logic-based clustering	316	35	0.155
BS (40)	Unclustered	3,880	98	0.913
	Combined feature and semantic-based clustering [31]	3,007	98	0.726
	Logic-based clustering	781	60	0.545
OFS (60)	Unclustered	9,060	152	1.102
	Combined feature and semantic-based clustering [31]	6,946	152	0.915
	Logic-based clustering	1,425	89	0.682
MS (80)	Unclustered	16,720	210	2.597
	Combined feature and semantic-based clustering [31]	12,331	210	2.022
	Logic-based clustering	2,143	119	1.538
EDS (100)	Unclustered	27,400	275	5.570
	Combined feature and semantic-based clustering [31]	20,824	275	4.579
	Logic-based clustering	3,021	151	3.294
ECS (200)	Unclustered	102,800	515	28.198
	Combined feature and semantic-based clustering [31]	76,586	515	23.030
	Logic-based clustering	8,324	287	18.273
Global (1000)	Unclustered	out-of-memory	out-of-memory	out-of-memory
	Combined feature and semantic-based clustering [31]	out-of-memory	out-of-memory	out-of-memory
	Logic-based clustering	91,457	1,429	597.228

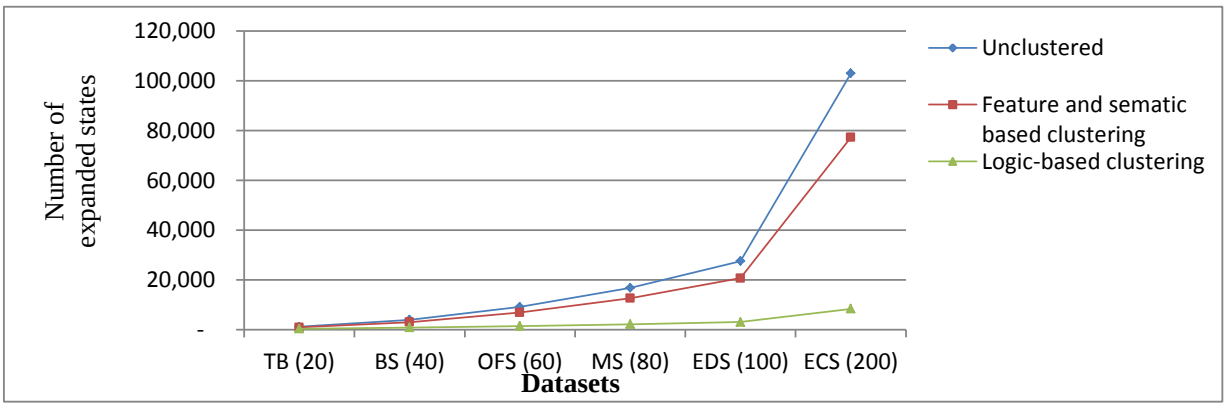
SOA, which are mostly based on the theory of planning of the artificial intelligence field (AI Planning) such as [4] and [5]. PORSCE II [4] is a framework implementing the WSC based on the requirements on input and output of the services. Similarly, OWLS-XPlan [5] also uses web services expressed by OWL-S to transform the problem from WSC domain to planning domain and uses the planner named XPlan, constructed by author. Some recent studies are based on abstract models such as Petri net or Colored Petri net [10], [11] to compose web services.

Fan et al. [10] proposed an approach to constructing the reliable service composition. It formalized the component web service as a Petri net, which provides means to observe the behaviors and the relationship of components. The transaction attributes, reliability and failure processing mechanisms are articulated. The composition mechanism systematically integrates these schemas into a transaction mapping model. Based on this, [10] proposed a reliable composition strategy and its enforcement algorithm, which can verify the behaviors of service composition at design time or after runtime to repair design errors. The operational semantics and related theories of Petri nets help prove the effectiveness of the proposed method.

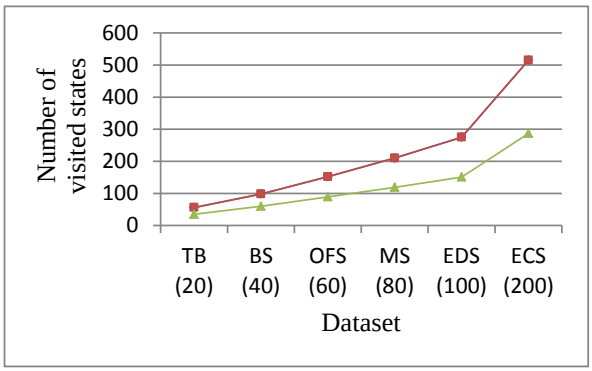
Maung et al. [11] had the approach similar to Fan et al. [10]. However, [11] used Colored Petri net (CPN) instead of Petri net. A Colored Petri net is a high level Petri net that provides a significant increase in the expressiveness and compactness of Petri net models.

Jingjing et al. [38] proposed the web service composition model based on timed automata. The proposed approach designed the formal model and its construction algorithm; provide a web service interface description language and composition automation engine. In particular, Jingjing et al. defined the Timed Automata for Web Service (TAW) and Timed Automata for WSC (TAC) models. In order to build these models from web service descriptions, [38] also provided the corresponding algorithms. After all, [38] built the composition automation engine to compose web services automatically.

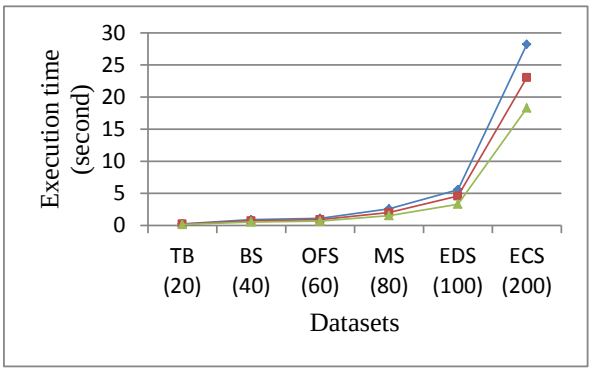
WSC methods which combine functional properties (hard constraints) and QoS properties (soft constraints) has been proposed in [2]. In [2], the authors applied genetic algorithm (GA) to solve the problem with each possible composition encoded as a gene, in



(a) Visual comparison between approaches according to the number of expanded states



(b) Visual comparison according to the number of visited states



(c) Visual comparison according to execution time

Fig. 9. Visual representation of the experimental results

order to calculate value for specific kinds of QoS properties. However, this study only provides us a mechanism to choose the best (possible) composition from a set of composition ways (full composition schema) rather than composes from the component web services. Besides, the application of genetic algorithms has increased the complexity of the problem and thus very difficult to apply in practice.

A different approach proposes of automated recovery when a WSC fall into the failure state (a web service could not be accessed or unsatisfied the user requirement) [12]. With this approach, we have to have a full composition schema described in BPEL [39] language, which is transformed into a Labelled Transition System (LTS), monitored by a monitor automata. When an error arises (a state that can not be reached, corresponding to a web service can not be accessed), the system will start calculating to choose the recovery plan by using the genetic algorithm. The difference between [12] and [2] is that the size of gene in [12] is unfixed, which depends on the number of backtracking steps from the error state.

7.3. Web Service Verification

In the field of web service verification, most of current approach verify hard or soft constraints separately. WS-Engineer [40] is a typical work for the web service verification based on functional properties. In [40], authors described a model-based approach to the analysis of service interactions for web service choreography and their coordinated compositions. The approach employs several formal analysis techniques and perspectives, and applies these to verify the web service composition. The resulted web service composition was described as a BPEL process. Then, this process will be specified using the Finite State Process (FSP) algebra notation. The verification was done on this FSP model by using the *Finite State Machine* (FSM).

Another study that intends to verify combined functional and non-functional requirements of WSC, is in-

troduced by Chen et al. [41]. This approach was implemented as the VeriWS tool [9]. VeriWS [9] takes in a full composition schema expressed in BPEL schema, then transforms that schema into a LTS model and uses a model checker to verify (the model). VeriWS will show the composition which satisfies the user constraints, both hard and soft constraints.

Recently, Khai et al. [14] proposed a fast and formalized approach for web service composition and verification, named *Heuristics-based On-the-fly Web Service Composition and Verification*. In the proposed approach, authors simultaneously compose and verify web services on both hard, soft, and temporal relationship constraints in an on-the-fly manner. [14] used the model checking theory and built the model from all of component web services (not from a composition schema as in [9]), so-called LTS4WS model – an extension of LTS model. In addition, [14] also proposed some heuristics based on web service characteristics to improve the state space searching performance of the model checker. This approach was implemented as a framework, known as *OnTheFlyWCV*.

8. Conclusion

In this paper, we proposed a logic-based web services clustering method. The clustering method improves the web service composition and verification when it allows the system chooses the small set of web services which suitable for user requirement on logical aspect. Moreover, when the clustering based only on the web services repository, it can be done before hand independently with the service to the user requirements. In order words, the cluster web services can be used for any kind of user requirements and be only changed when the repository changes.

Our clustering approach is proven the soundness and completeness rigorously, and certainly does not miss any solution as the others because we do not discard any cluster. At each composition step, we preferred to use the cluster which is evaluated as the best suitable.

Also through this paper, we have some contributions about the logical representation of Web service; the method to calculate the logic-based similarity between logical expressions represented for web services through the over approximation operator; and the method to build the representative logical expression that represent for web service cluster.

Our logic-based web service clustering approach was implemented and run experiments through tool *OnTheFlyWSCV*[14]. The results shown that the system parameters such as the number of expanded states, the number of visited states, and the processing times are better than other existing approaches.

However, in this work, we are inspired by the work of [28] to proposed our semantic-based similarity methods as part of our web service similarity method. We do believe that there may be many other methods than that we have proposed. It could be investigated in the near future. Also, in this work, we have use the K-means clustering algorithm, when it is the most popular algorithm in the field. Unfortunately, in this research area, many other algorithms have been proposed such as K-medoids [42], agglomerative hierarchical clustering methods [43], etc. The experimentation on the pros and cons of using other clustering algorithms has not been done yet. It is another future work.

References

[1] W. B. Williams, *Service-oriented architecture*, Information Security Management Handbook: Social Networking, vol. 6, 2012, pp. 317.

[2] M. AllamehAmiri, V. Derhami, and M. Ghasemzadeh, *Qos-based web service composition based on genetic algorithm*, Journal of AI and Data Mining, vol. 1, no. 2, pp. 63-73, 2013.

[3] N. H. Rostami, E. Kheirkhah, and M. Jalali, *Web services composition methods and techniques: A review*, International Journal of Computer Science, Engineering & Information Technology, vol. 3, no. 6, 2013.

[4] O. Hatzi, D. Vrakas, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas,*The PORSCE II framework: Using AI planning for automated semantic web service composition*, The Knowledge Engineering Review, vol. 28, no. 02, pp. 137-156, 2013.

[5] Klusch, Matthias, A. Gerber, and M. Schmidt, *Semantic web service composition planning with OWLS-Xplan*, In Proceedings of the AAAI Fall Symposium on Semantic Web and Agents, Arlington VA, USA, AAAI Press. 2005.

[6] Doliwa, Dariusz, et al., *PlanICS – A web service composition toolset*, In Fundamenta Informaticae, vol. 112, no. 1, pp. 47-71, 2011.

[7] Niewiadomski, Artur, W. Penczek, and J. Skaruz, *Hybrid Approach to Abstract Planning of Web Services*, In SERVICE COMPUTATION 2015: The Seventh International Conferences on Advanced Service Computing, pp. 35-40, 2015.

[8] C. Baier, and J. P. Katoen, *Principles of model checking*, Cambridge: MIT press, vol. 26202649, 2008.

[9] M. Chen, T. H. Tan, J. Sun, Y. Liu, and J. S. Dong, *VeriWS: A tool for verification of combined functional and non-functional requirements of web service composition*, in Proceedings of the 36th International Conference on Software Engineering. ACM, 2014, pp. 564-567.

[10] G. Fan, H. Yu, L. Chen, and D. Liu, *Petri net based techniques for constructing reliable service composition*, Journal of Systems and Software, vol. 86, no. 4, pp. 1089-1106, 2013.

[11] Y. W. M. Maung, and A. A. Hein, *Colored petri-nets (CPN) based model for web services composition*, IJCCER, vol. 2, pp. 169-172, 2014.

[12] T. H. Tan, M. Chen, É. André, J. Sun, Y. Liu et al., *Automated runtime recovery for QoS-based service composition*, in Proceedings of the 23rd international conference on World wide web. International World Wide Web Conferences Steering Committee, 2014, pp. 563-574.

[13] E. Pejman et al., *Web service composition methods: A survey*, Proceedings of the International MultiConference of Engineers and Computer Scientists, vol. 1, 2012.

[14] H. T. Khai, Q. T. Tho, and B. H. Thang, *Fast and formalized: Heuristics-based on-the-fly web service composition and verification*, in The Second NAFOSTED Conference on Information and Computer Science (NICS), 2015, pp. 174-179.

[15] J. Tretmans, *Model based testing with labelled transition systems*, In Formal methods and testing, pp. 1-38, Springer Berlin Heidelberg, 2008.

[16] J. Cardoso, *Semantic Web Services: Theory, Tools, and Applications*, New York, United States of America, 2007.

[17] Y. Charif, and N. Sabouret, *An overview of semantic web services composition approaches*, Electronic Notes in Theoretical Computer Science, vol. 146, no.1, pp. 33-41, 2006.

[18] E. Paikari, E. Livani, M. Moshirpour, B. H. Far, and G. Ruhe, *Multi-Agent system for semantic web service composition*, In Knowledge Science, Engineering and Management, pp. 305-317, Springer Berlin Heidelberg, 2011.

[19] S. McIlraith, T. Son, and H. Zeng, *Semantic web services*. Intelligent Systems, vol. 16, no. 2, pp: 46-53, April 2001.

[20] P. Giaretta, and N. Guarino, *Ontologies and knowledge bases towards a terminological clarification*, Towards very large knowledge bases: knowledge building & knowledge sharing, vol. 25, 1995.

[21] R. Chinnici, J. J. Moreau, A. Ryman, and S. Weerawarana, *Web services description language (WSDL) version 2.0 part 1: Core language*, W3C recommendation 26: 19, 2007.

[22] D. Martin et al., *OWL-S: Semantic markup for web services*, W3C Member Submission, 2004.

[23] D. L. McGuinness, and F. V. Harmelen, *OWL web ontology language overview*, W3C recommendation 10.10, 2004.

[24] T. R. Gruber, *A translation approach to portable ontology specifications*, Knowledge acquisition, vol. 5, no. 2, pp: 199-220, 1993.

[25] Z. Zhang, *Research on Web Service Clustering Based on Feature Model*, Information Technology Journal vol. 13, no.9, pp. 1668-1672, 2014.

[26] L. Chen, Q. Yu, P. S. Yu, and J. Wu, *WS-HFS: A Heterogeneous Feature Selection Framework for Web Services Mining*, In Web Services (ICWS), 2015 IEEE International Conference on. IEEE, pp. 193-200, June 2015.

[27] A. Mustapha, M. Quafafou, and Z. Jarir, *Leveraging Formal Concept Analysis with Topic Correlation for Service Clustering and Discovery*, In Web Services (ICWS), 2014 IEEE International Conference on. IEEE, 2014.

[28] L.L. Xie, F.Z. Chen, and J.S. Kou, *Ontology-based semantic web services clustering*, Industrial Engineering and Engineering Management (IE&EM), IEEE 18th International Conference on. IEEE, 2011.

[29] Y. Y. Du, Y. J. Zhang, and X. L. Zhang, *A Semantic Approach of Service Clustering and Web Service Discovery*, Information Technology Journal, vol. 12, no. 5, pp. 967-974, 2013.

[30] B. T. Kumara, I. Paik, K. R. Koswatte, and W. Chen, *Ontology learning with complex data type for Web service clustering*, In Computational Intelligence and Data Mining (CIDM), IEEE Symposium on. IEEE, pp. 129-136, December 2014.

[31] B. T. Kumara, I. Paik, W. Chen, and K. H. Ryu, *Web Service Clustering using a Hybrid Term-Similarity Measure with Ontology Learning*, International Journal of Web Services Research (IJWSR), vol. 11, no.2, pp. 24-45, 2014.

[32] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean, *SWRL: A semantic web rule language combining OWL and RuleML*. W3C Member submission 21: 79, 2004.

[33] L. De Moura, and N. Bjørner, *Z3: An efficient SMT solver*, In Tools and Algorithms for the Construction and Analysis of Systems, Springer Berlin Heidelberg, pp. 337-340), 2008.

[34] A. K. Jain, *Data clustering: 50 years beyond K-means*, Pattern recognition letters, vol. 31, no.8, pp. 651-666, 2010.

[35] M. Klusch et al., *OWLS-TC: OWL-S service retrieval test collection, version 2.1*, available at: http://projects.semwebcentral.org/projects/owls-tc/.

[36] D. Blei, and J. D. Lafferty, *A Correlated Topic model of Science*, In AAS 2007, pp. 17-35, 2007.

[37] Z. Azmeh, M. Huchard, C. Tibermacine, and C. Urtado, *Using Concept Lattices to Support Web Service Compositions with Backup Services*, in Proc. of ICIW'10, 2010.

[38] H. Jingjing, Z. Wei, Z. Xing, and Z. Dongfeng, *Web Service Composition Automation based on Timed Automata* Appl. Math, vol. 8, no.4, 2017-2024, 2014.

[39] D. Jordan, J. Evdemon, Alves et al., *Web services business process execution language version 2.0*, OASIS standard, vol. 11, p. 10, 2007.

[40] H. Foster, S. Uchitel, J. Magee, and J. Kramer, *WS-Engineer: A model-based approach to engineering web service compositions and choreography*, in Test and Analysis of Web Services. Springer, 2007, pp. 87-119.

[41] M. Chen, T. H. Tan, J. Sun, Y. Liu, J. Pang, and X. Li, *Verification of functional and non-functional requirements of web service composition*, In Formal Methods and Software Engineering, pp. 313-328, Springer Berlin Heidelberg, 2013.

[42] H. S. Park, and C. H. Jun, *A simple and fast algorithm for K-medoids clustering*, Expert Systems with Applications, vol. 36, no. 2, pp: 3336-3341, 2009.

[43] W. H. Day, and H. Edelsbrunner, *Efficient algorithms for agglomerative hierarchical clustering methods*, Journal of classification, vol. 1, no. 1, pp: 7-24, 1984.