ITC 1/49
Information Technology and Control
Vol. 49 / No. 1 / 2020
pp. 5-27
DOI 10.5755/j01.itc.49.1.23403

A Formal Technique for Composing Cloud Services

Received 2019/05/19 | Accepted after revision 2019/12/17

http://dx.doi.org/10.5755/j01.itc.49.1.23403

HOW TO CITE: Barati, M. (2020). A Formal Technique for Composing Cloud Services. *Information Technology and Control*, 49(1), 5-27. https://doi.org/ 10.5755/j01.itc.49.1.23403

A Formal Technique for Composing Cloud Services

Masoud Barati
Cardiff University, Department of Computer Science and Informatics, Cardiff, CF24 3AA, UK;
e-mail: Baratim@cardiff.ac.uk

Corresponding author: Baratim@cardiff.ac.uk

Recent cloud search engines lack a formal method in their service composition mechanisms to automatically build composite services realizing user requirements. This paper prescribes behavior composition framework as a formal tools for the search engines. The framework automatically synthesizes a controller that delegates the service operations requested by a cloud user to the proper available cloud services whose operations satisfy the request. Since most cloud search engines support semantic and ontology to discover similar service operations, the paper extends the framework to be more adaptable with such search engines through the use of resource reasoning. Several experiments are provided to demonstrate how the extended framework outperforms the original one in terms of realizing users' requirements.

KEYWORDS: Behavior composition, cloud computing, resource reasoning, controller synthesis, formal method.

1. Introduction

The tremendous increase in the number of cloud customers demanding their numerous requirements stimulates cloud providers to publish services with a great diversity of functional and non-functional properties. In many cases, a single service may not necessarily realize the requirements. Hence, the combination of several services as a composite service is created to fulfil them.

The architecture of cloud search engines recently benefit from the notions of semantic and ontology in order to offer a set of composite services being similar to customers' requirements. For instance, a distributed architecture was designed for cloud service discovery in [26]. The architecture benefits from a semantic Web technology using the Infrastructure and Network Description Language (INDL) and Standard Protocol and RDF Query Language (SPARQL) to exchange information among providers. In [13], a semantic based search engine architecture–called Cloudle–was proposed. It utilized a cloud ontology to measure the degree of similarity between customers' requirements and cloud services. The search engine leveraged an agent-based paradigm to build a test bed for service management. Furthermore, in light of de-

veloping cloud search engine architectures, a model using semantic Web and quality of service (QoS) was presented in [20]. It matches and composes cloud services for realizing customers' requirements. Although these approaches discover matchable or similar composites services for cloud customers, they integrate cloud services in an ad-hoc way and suffer from the lack of a formal method to automatically construct composite services.

Automatic service composition has lately become an interesting research area in both industrial and academic centres. Its purpose is the automatically determination of a strategy in order to compose available services realizing a desired service seen as user requirements [1, 24]. Current automated service composition approaches not only consider the functional specification of services, but also take into account the conversation specification of them through which the behavior of a service is described [8, 15]. Behavioral description of services consists of describing the order of invocation of service operations. To support such descriptions, services are represented by finite state machines, which in turn, facilitate their verification and lay a basis for their composition in an automatic way.

A promising automatic composition approach that acts as a formal tool for web or cloud service integration is behavior composition [9]. This approach, first, constructs a framework in which both customer requirements, called target service, and available services are abstracted as finite state machines. Then, it provides a sound and complete technique for synthesizing a controller that delegates the operations of the target service to the proper available services being able to realize the target. The behavior composition framework generates composite services automatically. Its synthesis technique guarantees that all possible composite services realizing the target service are obtained. One of the shortcomings of this framework, however, is that its synthesized controller does not delegate the target service operations to the available services that their operations are similar to those in target. Hence, the framework only builds composite services whose operations have a fully match with the target service operations. Note that, two operations can be similar if they have different names but the same functionalities.

This paper introduces a semantic-based framework for behavior composition in which similar operations to user requests can appear in composite services. The framework benefits from ontology and resource reasoning to semantically define service operations. Three kinds of reasoning methods, namely similarity, compatibility, and numerical reasoning are taken into account for calculating the degrees of match among operations. For the aim of providing a match making mechanism in the semantic-based framework, such degrees are used as references in the controller synthesis method proposed by the framework. The paper also gives come clues about the implementation of framework via model checking tools at hand: Symbolic Model Verifier (SMV) and Temporal Logic Verifier (TLV).

The rest of this paper is structured as follows. Section 2 reviews the related work. Section 3 briefly presents a background about the original behavior composition framework and the notion of resource reasoning. Section 4 proposes a semantic-based framework for behavior composition. The section also implements the framework by the model checking tools. Section 5 provides some experimental results to show the effects of resource reasoning on the realization of user requirements in the semantic-based behavior composition framework. Finally, Section 6 concludes the paper and gives some indications about future work.

2. Literature Review

In recent years, many contributions have been proposed for the problems of service composition and service discovery in the cloud and web. The strategies applied for such kind of problems can be classified into informal, semi-formal, and formal.

A brief review of some research using an informal strategy can be found in [42], which introduces the description of a Cloud-based Middleware for Service Composition, called *CM4SC*. In this approach, the middleware appears as a new layer between the application layer and the platform layer in the conventional cloud architecture to permit automatic composition planning and accelerate dynamic service composition. A flexible open source middleware has been proposed to support adaptive enactment of complex service composition in the cloud [16]. It facilitates the deployment of a large number of composite services and provides the capability of runtime

support for monitoring how they have been built. A semantic-based search engine, called *Cloudle*, that takes advantage of a cloud ontology to determine the measure of similarity between the user requirements and the available services, was devised for service composition and discovery [13]. Consequently, an agent-based architecture showing how the *Cloudle* search engine is organized was put forward in [31]. Generally speaking, regardless of the advantages that such contributions may bring about, the correctness of service composition cannot be verified.

Usually the solutions exploiting a semi-formal strategy are subcategorized into syntactic-based and semantic-based methods. The methods in the first category are also decoupled with respect to the methods that integrate services with the aid of an orchestrator or a choreographer. For service orchestration, *Business Process Execution Language* (BPEL) is used to define an interoperable integration model [37, 38]. For instance, a management-based adaptive and configurable service composition method was proposed in [37] with the aid of a development in BPEL, called as VxBPEL, to support variability in service compositions. For service choreography, *Web Service Choreography Description Language* (WS-CDL),[1] being an XML-based specification language is used for composing peer-to-peer, interoperable collaborations among participants. The methods in the second category include *Ontology Language for Web Services* (OWL-S) to define an ontology for the semantic markup of web services to enable service invocation and composition through supplying the proper semantic descriptions [12, 19]. Furthermore, *Web Service Modelling Ontology* (WSMO)[2] explicitly defines a conceptual model and also provides an ontology graph for the description of different aspects that are related to the semantic web services to solve the problem of service integration. A framework for automatic discovery, selection and composition of RESTful Web services which utilized linked open data was presented in [21]. The framework applied RDF (Resource Description Framework) to represent the state of linked data services through which SPARQL (Standard Protocol and RDF Query Language) queries has been used to compose RESTful services. In general, there is no standard or a universal interfacing language that facilitates the composition of services written in different languages.

There are several solutions based on a formal strategy with theoretical models. Four of them, respectively based on transition systems with formal verification, proof system, process calculus, and AI planning are briefly presented. Such approaches ensure soundness, completeness or correctness of service composition. The first one aims to be applied in the domain of grid computing. A resource discovery approach to address multi-attribute queries is introduced [36]. The technique to get replies to queries are very fast and decreases the number of nodes examined during the resource discovery process. The proposed model for resource discovery has been decoupled into data gathering, discovery, and control to simplify the formal verification of properties expressed by CTL (computational tree logic) and LTL (linear temporal logic) formulas. The second one suggests to exploit the *X-UNITY* language for representing services in the cloud and a proof theory for proving temporal properties over service specifications needed for cloud applications [10]. Moreover, a compositional proof-system was extended in [39] with a number of inference rules and proven system properties. The approach proposed a semantic formalization using SWSpec language in which the complexity of composing workflows was reduced. The third one includes a formal definition of composition in terms of a process calculus and also provides an implementation through the extension of the *Jolie* language [7]. This approach was applied in the context of service-oriented architectures (SOAs). In the same spirit, a new model and verification mechanism that relies on a process calculus for orchestration of web services, which combined with UML, provides a solution for the problem of formal verification of cloud manufacturing services composition [43]. Following that, a new extended process calculus for cloud manufacturing service composition was proposed [44]. Through this contribution, six elements of quality of service have been evaluated. Furthermore, a formal schema of service composition and a BPEL code generation method have been provided. The fourth one proposes to exploit planning to optimize both composition and the underlying collection of information in order to obtain high-quality composite services in-line [34]. In this solution, irrelevant actions, based on the

1 http://www.w3.org/TR/ws-cdl-10

2 http://www.wsmo.org

user preferences, are eliminated to reduce the search space. An automatic service composition by using a HTN (Hierarchical Task Network) planner JSHOP2 (Java Simple Hierarchical Ordered Planner) was proposed in [35] which considered both functional and non-functional properties of services.

Although the correctness of service composition is guaranteed in the contributions based on a formal strategy, they still suffer from the lack of exploiting controller synthesis techniques for service composition. To this end, behavior composition can provide a sound and complete technique. More precisely, the problem of behavior composition has been widely studied in the areas of web services [5], verification [18], and even multi-agent systems [29]. Among recent studies which are able to be applied in service-oriented computing, an automatic behavior composition synthesis framework is quite significant [9], as the authors have extensively investigated a particular type of behavior composition. In this framework, a specific controller is generated to coordinate the parallel execution of available services, so as to simulate a given target service with respect to constraints imposed on operations by an environment. A model without environment constraints, called *Roman,* was also introduced latter by the same research team [8]. Furthermore, an effective procedure, computing realizable target fragment, has been developed in the case that a behavior composition problem is unsolvable [28, 40].

3. Preliminaries

This section devotes a brief review to the original behavior composition framework and resource reasoning.

3.1. Behavior Composition

The main elements of behavior composition framework are available services $\mathcal{B}_1,\ldots,\mathcal{B}_n$, a target service $\mathcal{B}_t$, an environment $\mathcal{E}$, and controller generator *CG* [9]. [3] The asynchronous product of available services makes a system *S*. Formally, the target service is defined as a tuple $\langle B_t, b_{t0}, A_t, F_t, \eta_t\rangle$, where B_t is a finite set of states, b_{t0} is the initial state, A_t is the set of operations, F_t is a set of final states, and $\eta_t \subseteq B_t \times A_t \times B_t$ is a transition relation. Each available service in the system is a tuple $\langle B_i, b_{i0}, A_i, F_i, \eta_i\rangle$, where A_i is the set of operations defined by $\mathcal{B}_i$. The operations set $A_s = \bigcup_{i=1}^{n} A_i$ is defined to indicate all available operations in the system such that $A_t \subseteq A_s$. Given the target and system, an algorithm called as largest Nondeterministic-simulation (ND-simulation for short) is introduced to trigger the process of controller synthesis. Through the algorithm, at each step, a state from the system does not simulate the one in the target service is eliminated. This algorithm leads to an ND-simulation relation *R* in which the relation of $b_t \preceq s_s$ denotes the state b_t of $\mathcal{B}_t$ is simulated by the state s_s in *S*. The output of the algorithm is a particular controller, called as controller generator. Given the execution of a current operation, this controller delegates the operation to the appropriate available services being able to handle it. Technically, the controller generator *CG* of *S* for $\mathcal{B}_t$ is a tuple $\langle \Sigma, A_t, I_n, \xi, \omega\rangle$, where $\Sigma=\{\langle b_t, s_s\rangle \in B_t \times S_s \mid b_t \preccurlyeq s_s\}$ is the set of *CG* states, $I_n \in \{1\ldots n\}$ is the indices of available services, $\xi \subseteq \Sigma \times A_t \times I_n \times \Sigma$ is the transition relation, and $\omega : \Sigma \times A_t \to 2^{I_n}$ is the output function.

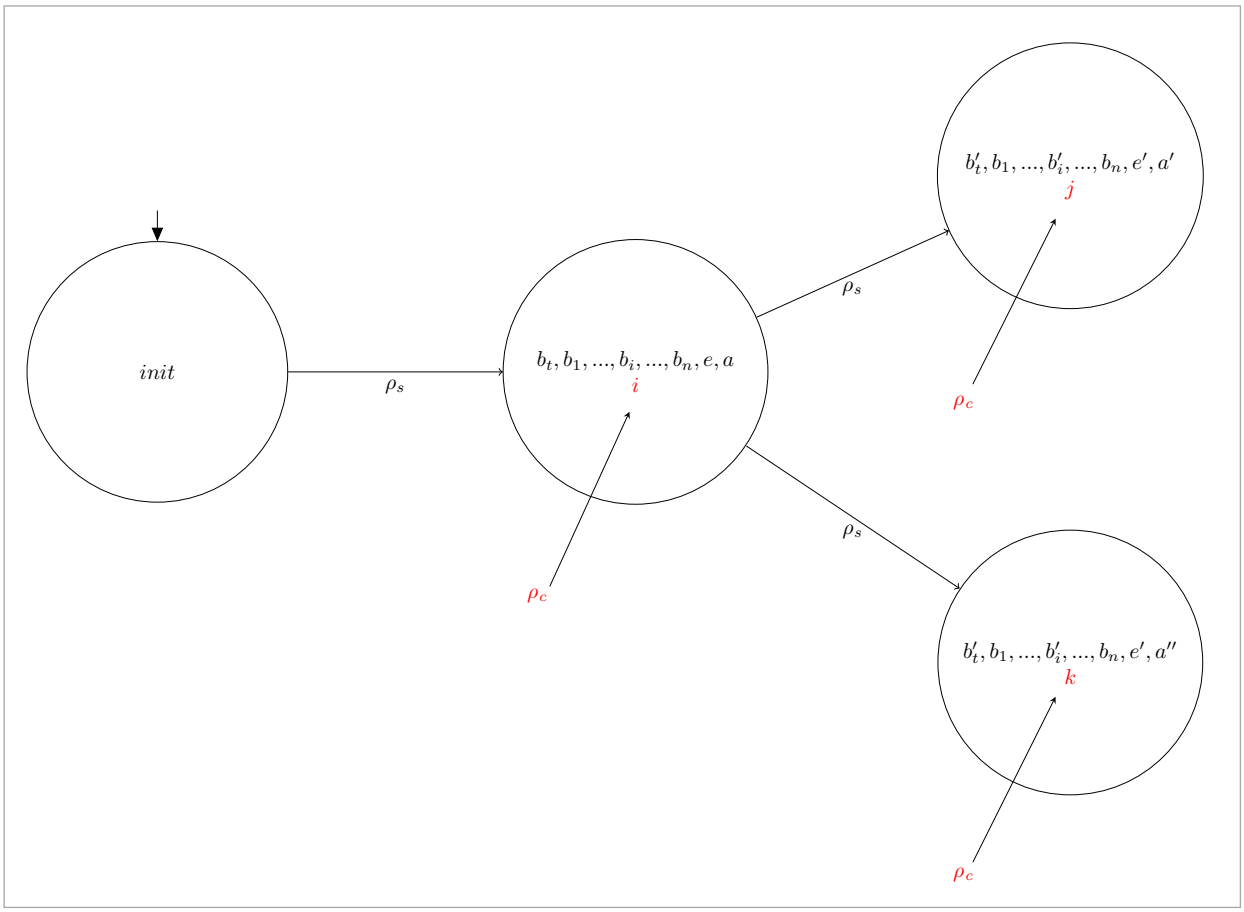
When a controller generator has been emerged, the notions of trace and history can be defined. A *CG* trace τ_{CG} is a sequence $\sigma^0 \xrightarrow{a_t^1, k^1} \sigma^1 \xrightarrow{a_t^2, k^2} \cdots$, where $\sigma \in \Sigma$, and a *CG* history is a finite prefix of a *CG* trace.

Let $\mathcal{H}_{CG}$ be the set of all histories on *CG*. A selection function $C : \mathcal{H}_{CG} \times A_t \to I_n$ is defined from the output function ω in order to select one service among those that are able to execute the current operation with respect to the last state of a given history. If *CG* contains the initial state $\sigma_0 = \langle b_{t0}, s_{s0}\rangle$, and $b_{t0} \preccurlyeq s_{s0}$, then a set of controllers–called generated controllers, which are compositions of $\mathcal{B}_t$–can be extracted from *CG*.

For the purpose of controller synthesis with the aid of an available model checking tool, the approach based on the largest ND-simulation is replaced by the calculation of a winning strategy of a corresponding two players in a safety game [9]. As depicted in Fig. 1, in such a game structure, one plays the role of the system and the other plays the role of the controller. The former keeps the information about the current state of the target service, available services, and environment, and at each step, releases an operation that must be ex-

3 Since the use of environment is not very widespread in practice, the definition of this element is not discussed in our paper.

Figure 1
Two players of a safety game



ecuted. The latter returns an index indicating which available service in the system is able to perform the requested operation. These data appear in two lines inside each state of the transition system in Fig. 1. The state *init* indicates an initial state. In this state, all available services and the target service are in their initial states, the current operation is empty and there is no controller reply, namely the index is zero.

The transition relations ρ_s and ρ_c represent system moves and controller replies, respectively. More precisely, $\rho_s \subseteq X \times Y \times X$, where $X = B_t \times B_1 \times \ldots \times B_n \times E \times (A \cup \varnothing)$ and $Y = I_n \cup \{0\}$, and $\rho_c \subseteq X \times Y \times X \times Y$, where

$\langle\langle b_t, b_1, \ldots, b_n, e, a\rangle, i, \langle b'_t, b'_1, \ldots, b'_n, e', a'\rangle, j\rangle \in \rho_c$

if and only if $j \neq 0$. Notice that, $b_k = b'_k$ for all $k \in I_n \setminus \{i\}$. The reader is referred to [9] for the detailed procedure that shows how to derive a safety-game structure from a behavior composition problem.

3.2. Reasoning Based on Semantics

To provide a more flexible framework for behavior composition, a compatibility relation $\ll \subseteq A \times A$ over the set of operations can be introduced. The relation substitutes for the present equality between operations in the definition of the largest ND-simulation relation proposed in [9] and the underlying algorithm that computes it. An operation a' can now be carried out by an available service, if it is compatible with the delegated operation a, that is, $a \ll a'$. No more details were given about this issue by the authors.

The use of resource reasoning metrics constitutes a first appealing solution [2, 31]. It evaluates the degree

of match between any two service operations. Generally, such metrics fit with a domain ontology graph, which has a well-formed structure to determine the multipaths connecting two concepts. Hence, the main effort must be concentrated on building ontologies.

3.2.1. Ontology and Resource Reasoning

Ontology is a representational artifact whose purpose is the exhibition of entities, defined classes, and relations between them [3]. An ontology can offer meta-information to describe semantics of data and allows for building knowledge bases. Furthermore, it is a formal structure that supports the communication between a user and a computer agent [2].

The kinds of ontology are classified into domain ontology, reference ontology, top-level ontology, and application ontology [3]. The intended class of ontology for better representation and classification of resources included in a specific system is the domain one. It provides a taxonomy with a hierarchical structure for such resources, considered as concepts, together with a set of axioms identifying several rules to show how the concepts and relations can be comprehended [3]. A typical example is a cloud ontology which supplies a taxonomy for its computing resources [31].

Different kinds of relations can be defined between the concepts in a domain ontology. For instance, "*is-a*", "*part-of*", "*is-subtype-of*", "*is-member-of*", "*participates-in*", "*has-output*", and "*precedes*" are some examples of such relations. A domain ontology is formally defined as follows.

Definition 1 *[11]. An ontology in a specific domain O is a tuple of* $\langle C, \leq c, R, \leq r, A\rangle$*, where C is a set of concepts, R is a set of relations,* $\leq c$ *is a partial order on C that is called the concept hierarchy,* $\leq r$ *is a partial order on R that is called the relation hierarchy, and A is a set of axioms including rules in the logical forms to describe the relationships among the concepts.*[4]

Based on a domain ontology, a graph, called an ontology graph, is drawn to demonstrate a taxonomy [2]. In this graph, each concept is represented as a node and each edge indicates a relationship between two concepts. More precisely, each edge illustrates a relation such as "*is-a*" or "*part-of*". For example, Fig. 2 depicts a simple ontology graph for an online agency providing travel services. In this graph, the *Travel agency services* is considered as the root node having subnodes including *Accommodation reservation, Transportation reservation,* and *Meal reservation.*

In cloud computing, where service operations are defined semantically, the notion of resource reasoning is put forward, which includes similarity, compatibility, and numerical reasoning [31]. In similarity reasoning, to measure the degree of similarity between two different concepts, several semantic similarity functions have been introduced. Among those proposed in [2, 25, 30], there is one that defines a function being compatible with a hierarchical structure of well-formed concepts that can be found in a domain ontology graph [2]. In comparison to Jaccard [23], cosine [41], and Slimani [33] similarity measures' functions, Knappe in [2] defined a function that better considers path length, depth, and local density of an ontology graph [32]. This function takes into account specialization or generalization of one concept with respect to another.

Definition 2 *[2]. The semantic similarity function* $sim: C \times C \rightarrow [0,1]$ *is defined as:*

$$sim(x,y)=\rho\frac{|\alpha(x)\cap\alpha(y)|}{|\alpha(x)|}+(1-\rho)\frac{|\alpha(x)\cap\alpha(y)|}{|\alpha(y)|}, \quad (1)$$

where the constant $\rho \in [0,1]$ determines the degree of influence of generalization[5] that depends on a hierarchical ontology; the parameter ρ permits to tailor the similarity function, and hence can conform to the generalization property. The term $\alpha(x)$ is considered as the set of upward nodes reachable from x (including the node labeled by x), and the expression $\alpha(x)\cap\alpha(y)$ is the reachable common nodes between x and y.

For instance, in Fig. 2, the concept of *Meal reservation* has two reachable upward nodes from itself, whereas this is four for *Cliff hotel reservation*. Hence,

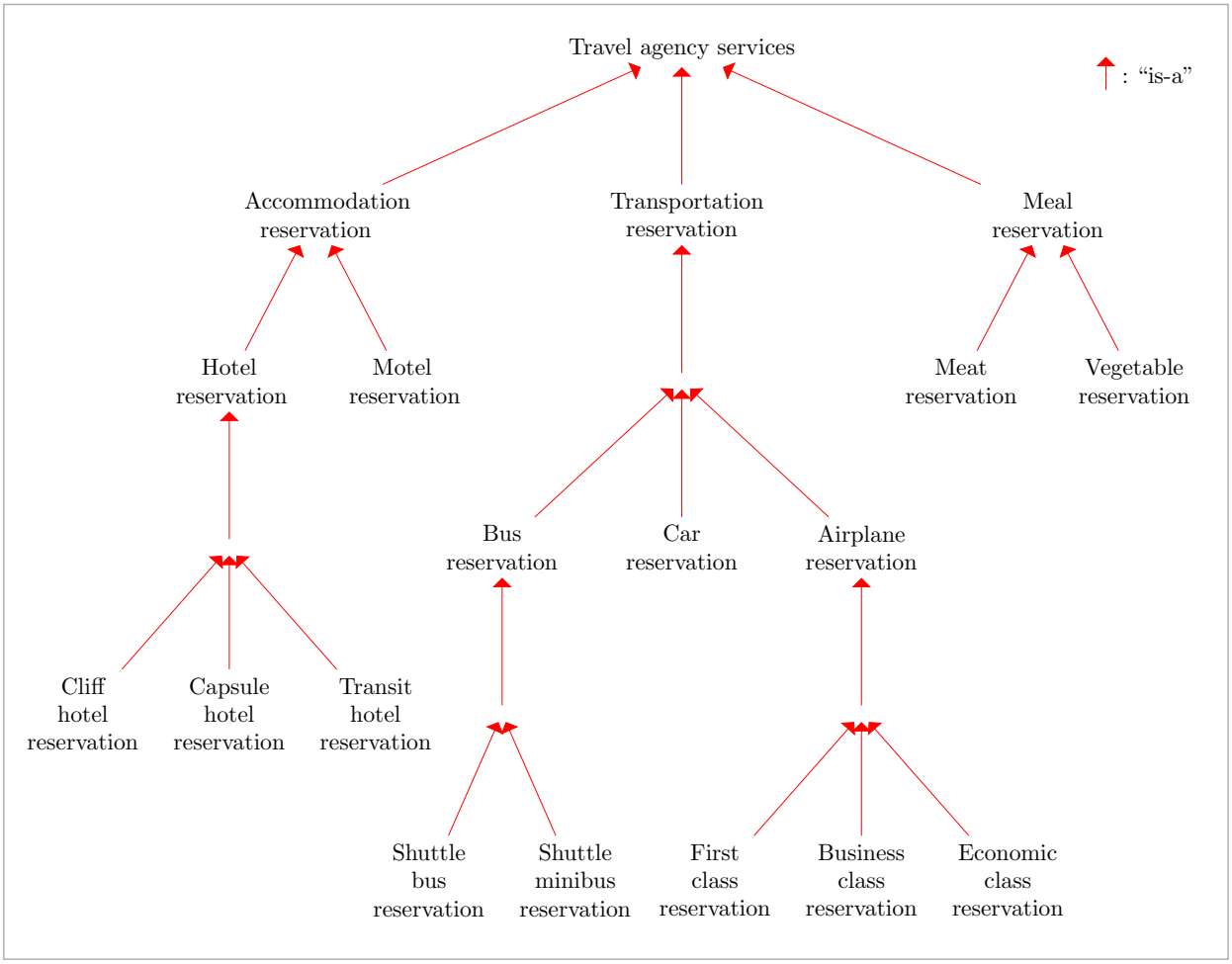
$$|\alpha(Mealreservation)|=2 \text{ and}$$
$$|\alpha(Cliffhotelreservation)|=4.$$

In addition, the number of common nodes for *Cliff hotel reservation* and *Transit hotel reservation* is more than *Cliff hotel reservation* and *Bus reservation,* which are calculated as follows:

4 The notations $\leq c$ and $\leq r$ could be replaced by $\leq c$ and $\leq r$, respectively. Although the latter are better, the former were adopted to avoid confusion with [11].

5 Generalization is the opposite of specialization.

Figure 2
A part of simple ontology graph of a travel agency service



$|\alpha(Cliffhot.reserv.) \cap \alpha(Transithot.reserv.)| = 3$

and $|\alpha(Cliffhot.reserv.) \cap \alpha(Busreserv.)| = 1.$

As indicated in Eq. 1, the semantic similarity function maps two concepts into the unit interval, and its output shows the degree of similarity between x and y. So, the value 0 means no similarity and 1 means full similarity.

Given the ontology graph and semantic similarity function, a square matrix (similarity matrix) of order n of similarities among concepts is constructed:

$$SIM = \begin{bmatrix} sim(a_1,a_1) & \dots & sim(a_1,a_n) \\ \vdots & \ddots & \vdots \\ sim(a_n,a_1) & \dots & sim(a_n,a_n) \end{bmatrix},$$

where n is the number of concepts. It is important to note that each element in this matrix indicates a real number in [0,1] giving the degree of similarity between related concepts. Moreover, a threshold in the interval (0,1] is defined to accept the minimum measure of similarity between two concepts.

The similarity reasoning was introduced to measure the degree of similarity for functional requirements with the aid of a semantic similarity function (see Def. 2). To calculate the degree of match for technical requirements in computing systems, both the compatibility and numerical reasoning were proposed.

The compatibility reasoning is appropriate for comparing two sibling nodes in a domain ontology graph, for example, the compatibility between two different

versions of a software program in the cloud ontology.

Definition 3 *[13]. The compatibility reasoning function* $compat : C \times C \to (0,2)$ *is defined as:*

$$compat(x,y) = sim(x,y) + \frac{0.8^{|c_x - c_y|}}{10}, \quad (2)$$

where The terms c_x and c_y indicate the chronological orderings of different versions of a software program. The expression $0.8^{|c_x - c_y|}/10$ is a fine-grain measurement, because x and y have a small degree of difference.

In [31], the compatibility reasoning function was generalized as follows.

$$compat(x,y) = sim(x,y) + \frac{\mu^{|c_x - c_y|}}{\theta}, \quad (3)$$

where $0 < \mu < 1$ and $1 < \theta < \infty$ can be assigned by an arbitrary value. The terms c_x and c_y indicate the chronological orderings of different versions of a software program. The most important component in $\frac{\mu^{|c_x - c_y|}}{\theta}$ is the term $|c_x - c_y|$.

More precisely, the term $sim(x,y)$ in Eq. 3 is computed based on Eq. 1 and the main significant value comes from the expression $|c_x - c_y|$. When this value is large, it means that x and y are less compatible; otherwise, they are more compatible. Appendix B provides an example to show how compatibility reasoning is calculated.

The numerical reasoning is about the similarity between two numeric values of a concept such as CPU speed or RAM size.

Definition 4 *[31]. Let a and b be numeric values and c a concept. The numerical reasoning function* $Sim : \mathbb{R} \times \mathbb{R} \times C \to [0,1]$ *is defined as:*

$$Sim(a,b,c) = 1 - \left| \frac{a-b}{Max_c - Min_c} \right|, \quad (4)$$

where Max_c and Min_c are the minimum and maximum values being available for c.

As an example, consider the concept RAM with three different instances whose sizes are 1GB, 4GB, and 8GB. Given Eq. 4, $Max_{RAM} = 8$ and $Min_{RAM} = 1$. In case of calculating numerical reasoning between RAM 1GB and RAM 4GB (i.e., $a = 1$ and $b = 4$), it is:

$$Sim(a,b,RAM) = 1 - \left| \frac{1-4}{8-1} \right| \approx 0.58.$$

Placing concepts inside an ontology graph is manually performed and the degree of similarity between two concepts can be retrieved from a similarity matrix. For locating concepts in an automatic way, a hierarchy matching method proposed in [22] can be suggested. The details of such a method is, however, out of the scope of this paper.

4. Semantic-Based Behavior Composition

Given the notion of ontology and reasoning, each operation handled by an available service is considered as a concept [4]. To have a matchmaking between the operation requested by a target and those available in the system, two sets of operations are defined in the framework. One set, denoted by A_t, contains the requested operations, and the other set, denoted by A_s, includes all operations handled by available services in the system. Given such sets, the target service $\mathcal{B}_t$, available services $\mathcal{B}_i$ $(1 \le i \le n)$, and the system $\mathcal{S}$ are as defined in Sect. 3.1, while $A_t \not\subseteq A_s$.

Example 1. *Consider requirements expressed as a target service and depicted in Fig. 3. The requested operations belong to the set:*

$A_t = \{StorageSpace210GB,$

$Windows7, SQL\text{-}Server2008\}$.

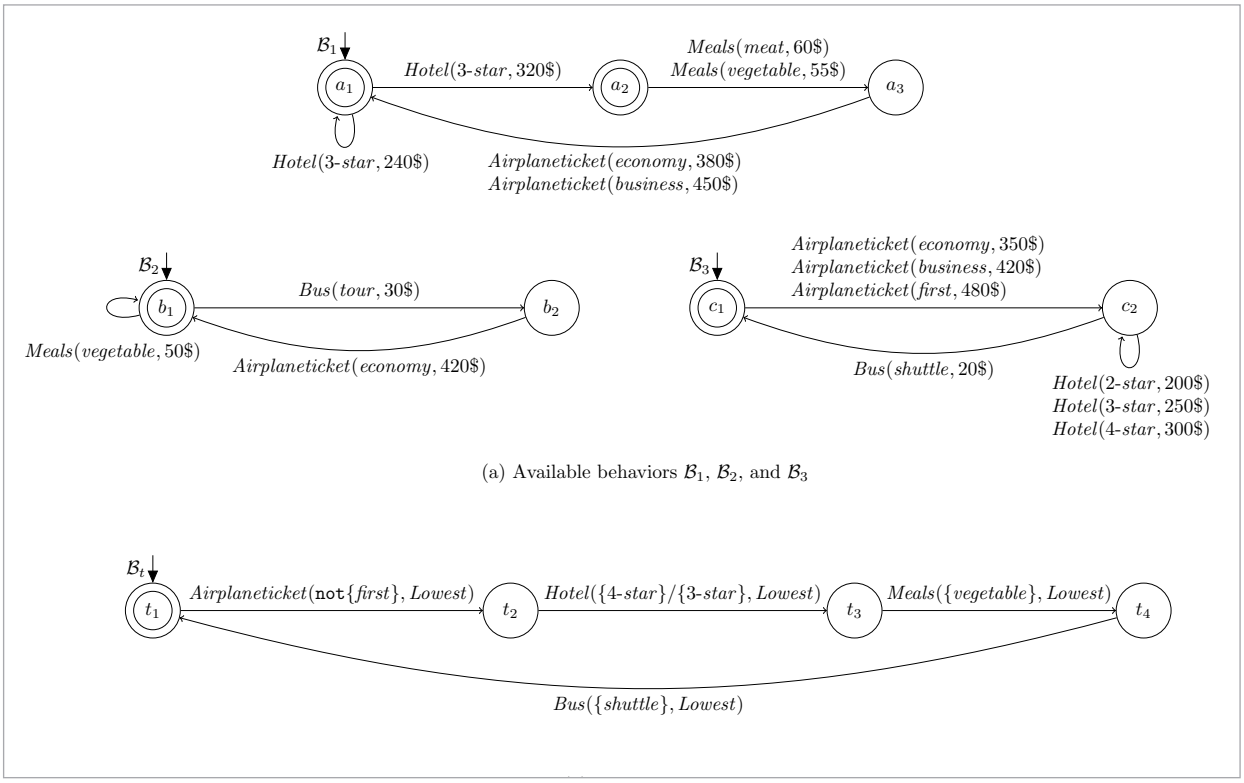
Many of them are not available in the cloud. Moreover, there are three available services able to meet the target. For instance, the service $\mathcal{B}_1$ is able to offer the operation set:

$A_1 = \{Windows8, SQL\text{-}Server2005\}$.

Though such resources do not have the same name as those of the requested resources, they can be similar or have the same functionality. In this end, a part of cloud ontology graph is illustrated in Fig. 4 to indicate the relations among service operations.

Taking into account the three types of resource reasoning, the definition of largest ND-simulation relation must be revisited to support them. Let v_s and v_t be the numeric values for a_s and a_t, respectively, and c is a concept carried by both a_t and a_s. Moreover, assuming that τ_1, τ_2, and τ_3 be thresholds. An ND-simu-

Figure 3
A target service and available services handling cloud service operations



(a) Available behaviors $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$

lation relation of $\mathcal{B}_t$ by $\mathcal{S}$ is a relation $R \subseteq B_t \times S$ such that $\langle t, s\rangle \in R$ implies:

Extension of largest ND-simulation relation proposed in [9]

1. if $t \in F_t$, then $s \in F$;
2. for all transitions $\langle t, a_t, t'\rangle \in \delta_t$:

- there exists a transition $\langle s, a_s, k, s'\rangle \in \delta$;
- for all transitions $\langle s, a_s, k, s'\rangle \in \delta$, $\langle t', s'\rangle \in R$ and:
  - $sim(a_t, a_s) \geq \tau_1$ if reasoning is similarity,
  - $compat(a_t, a_s) \geq \tau_2$ if reasoning is compatibility,
  - $Sim(v_s, v_t, c) \geq \tau_3$ if reasoning is numerical.

For each type of reasoning, a condition is provided to ensure that at each step of operation execution, the degree of match between a requested operation and an available operation is equal to or greater than a predefined threshold.

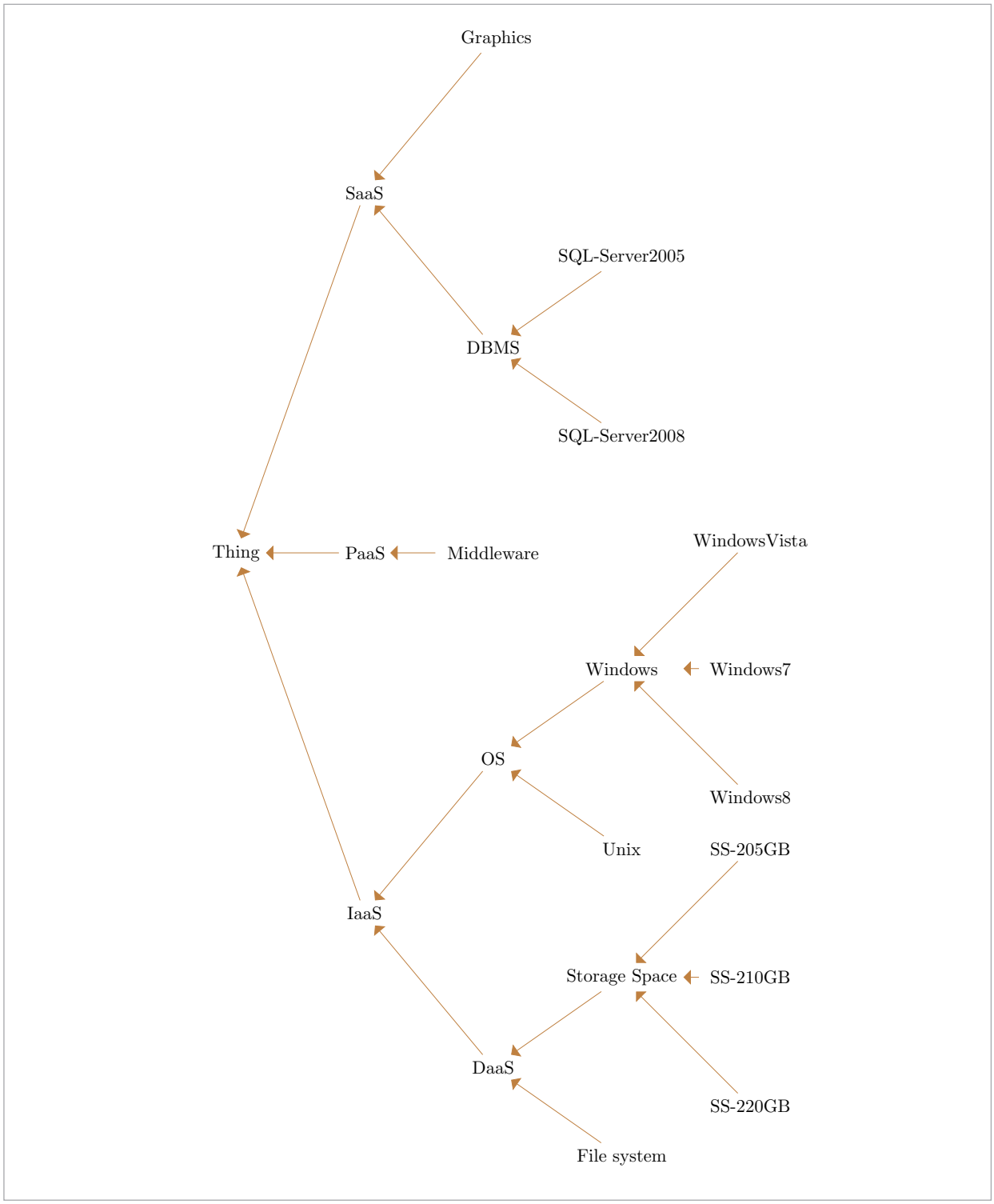
Based on the extension of the largest ND-simulation relation, the notion of controller generator also requires a revision to consider the reasoning conditions. Formally, the controller generator CG of $\mathcal{B}_t$ on $\mathcal{S}$ is $\langle \Sigma, A_t, A_s, I_n, \xi, \omega\rangle$, where:

1 $\Sigma = \{\langle t,s\rangle \in B_t \times S \mid t \preccurlyeq s\}$ is the set of CG states made by all pairs of $\mathcal{B}_t$ and $\mathcal{S}$ states that belong to the largest ND-simulation relation;

2 ξ is the transition relation, where $\sigma \xrightarrow{a_t, a_s, k} \sigma'$ in ξ, if and only if:

- there is a transition $t \xrightarrow{a_t} t'$ in $\mathcal{B}_t$;
- there is a transition $s \xrightarrow{a_s, k} s'$ in $\mathcal{S}$;
- in the case of similarity reasoning, the condition $sim(a_t, a_s) \geq \tau_1$ holds;
- in the case of compatibility reasoning, the condition $compat(a_t, a_s) \geq \tau_2$ holds;
- in the case of numerical reasoning, the condition $Sim(v_s, v_t, c) \geq \tau_3$ holds;
- for all $\langle t'', s''\rangle \in B_t \times S$, such that $s \xrightarrow{a_s, k} s''$ in $\mathcal{S}$ and $t \xrightarrow{a_t} t''$ in $\mathcal{B}_t$, it is the case that $\langle t'', s''\rangle \in \Sigma$;

Figure 4
A part of an ontology for the cloud service operations [14]



3 $\omega : \Sigma \times A_t \times A_s \to 2^{I_n}$ is the output function with $\omega(\sigma, a_t, a_s) = \{k \mid \exists \sigma' \in \Sigma \text{ such that } \langle \sigma, \langle a_t, a_s, k \rangle, \sigma' \rangle \in \xi\}$.

Example 2. *The largest ND-simulation relation for Example 1 is computed based on compatibility and numerical reasoning, which are done from the ontology graph in Fig. 4. Notably, the graph is a part of a cloud ontology graph represented in [14].*

The compatibility reasoning is used for both *Windows* and , whereas a numerical reasoning is used for *StorageSpace*. Given the simulation relation:

$$R = \{\langle t_1, \langle a_1, b_1, c_1 \rangle\rangle, \langle t_2, \langle a_1, b_2, c_1 \rangle\rangle, \langle t_2, \langle a_1, b_1, c_1 \rangle\rangle, \langle t_3, \langle a_1, b_1, c_1 \rangle\rangle, \langle t_3, \langle a_2, b_1, c_1 \rangle\rangle\},$$

the controller generator, illustrated in Fig. 5, is synthesized. All transitions of the controller generator are labeled by a pair of operations, namely a requested operation of the target and a similar operation to the request handled by an available service. For instance, due to the compatibility between *WindowsVista* and *Windows7*, the former can be offered to the target service.

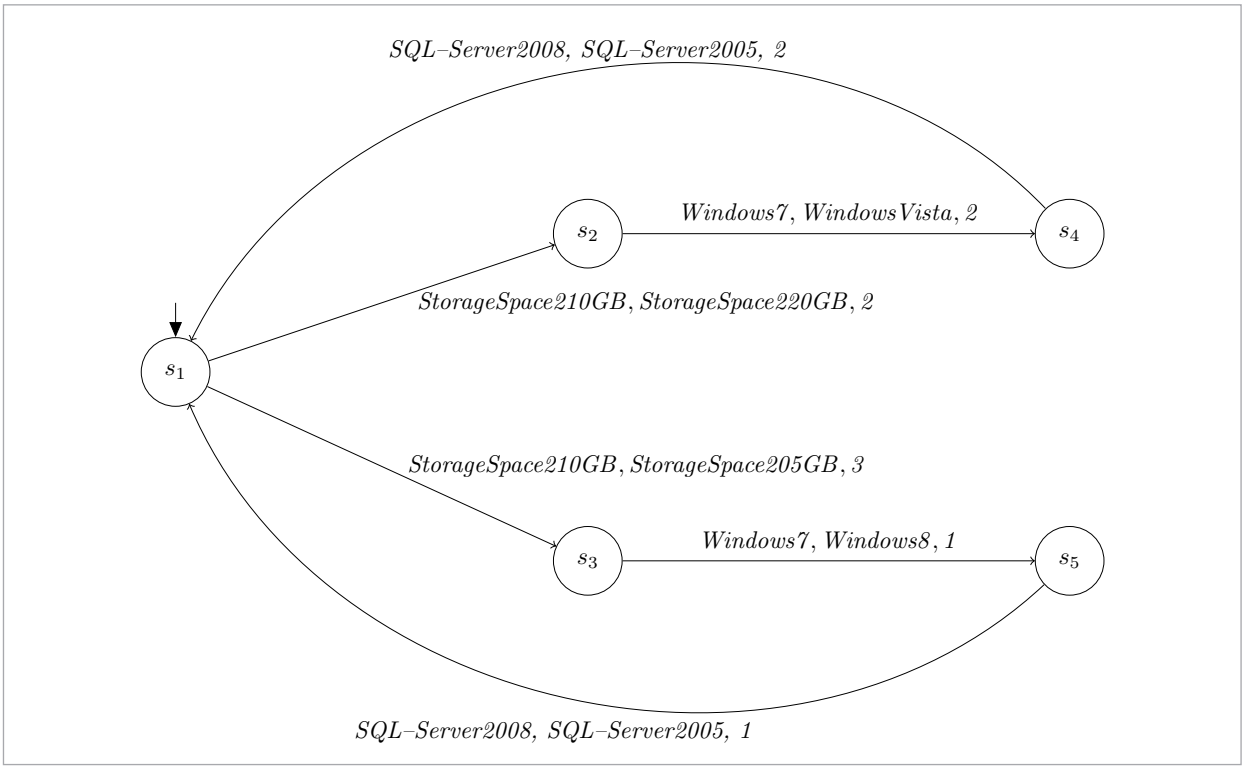
Given the possible generated controllers that can be extracted from the controller generator, the transition relations of two of them are:

$$P_1 : \{\langle s_1, StSpace210GB, StSpace220GB, 2, s_2 \rangle, \langle s_2, Windows7, WindowsVista, 2, s_4 \rangle, \langle s_4, SQL--Server2008, SQL--Server2005, 2, s_1 \rangle\}.$$

$$P_2 : \{\langle s_1, StSpace210GB, StSpace205GB, 3, s_3 \rangle, \langle s_3, Windows7, Windows8, 1, s_5 \rangle, \langle s_5, SQL--Server2008, SQL--Server2005, 1, s_1 \rangle\}.$$

It is supposed that the predefined threshold for compatibility and numerical reasoning are $\tau_2 = 0.7$ and $\tau_3 = 0.3$, respectively. Moreover, let ρ in Eq. 1 be set to 0.5, and μ and θ in Eq. 3 be set to 0.8 and 10, respectively. Given the following amounts, separately computed for each transition in the generated controllers, the average degree of match for the generated controllers P_1 and P_2 are 0.68 and 0.79, respectively. For instance, the average degree of match for P_1 is calculated as (0.33+0.88+0.83)/3.

Figure 5
The controller generator



$Sim(205GB, 210GB, StorageSpace) = 0.66,$
$Sim(220GB, 210GB, StorageSpace) = 0.33,$
$compat(Windows7, Windows8) =$
$compat(WindowsVista, Windows8) = 0.88,$
$sim(SQL--Server2008, SQL--Server2005) = 0.83,$

where $|c_{Windows7} - c_{Windows8}| = 1,$
$|c_{Windows7} - c_{WinVista}| = 1, |\alpha(Windows8)| = 5,$ and
$|\alpha(SQL--Server2008) \cap \alpha(SQL--Server2005)| = 3.$

4.1. Implementation of Semantic-Based Behavior Composition in SMV/TLV

Once behavior composition has been translated into the safety-game structure, it can be implemented with a model checking tool like TLV [27]. TLV (Temporal Logic Verifier) is a tool for the purpose of verification of LTL specifications. It uses Boolean Decision Diagrams (BDDs) for indicating state valuations and transitions. The inputs of TLV are an LTL specification written in SMV and a synthesis procedure. The latter is based on the safety-game structure and its output represents a controller generator. The procedure of controller synthesis based on a winning strategy has been implemented in TLV [9]. SMV (Symbolic Model Verifier) is a symbolic model checking tool supporting the verification of temporal logic (LTL and CTL) properties of finite-state machines. In SMV, each element of the safety-game players, namely available services, target service, and controller, is implemented as a module [9]. Figure 6 includes the main modules of services in Example 1. The module main contains two submodules: the controller Ctr and

Figure 6
Modules of system and controller in SMV

```
MODULE main
 VAR
   sys: system Sys(ctr.index);
   ctr: system Ctr;
 DEFINE
   good := (ctr.initial & sys.initial) | !(sys.failure);

MODULE Ctr
 VAR
   index : 0..3;
 INIT
   index = 0
 TRANS
   case
    index=0 : next(index)!=0;
    index!=0 : next(index)!=0;
   esac
 DEFINE
   initial := (index=0);

MODULE Sys(index)
 VAR
  operation : {start_op, StorageSpace205GB, StorageSpace210GB, StorageSpace220GB,
               Windows7, Windows8, WindowsVista, SQL-Server2005, SQL-Server2008};
  threshold : 0..10;
  target : Target(operation,threshold);
  B1 : Service1(index,operation,threshold);
  B2 : Service2(index,operation,threshold);
  B3 : Service3(index,operation,threshold);
 DEFINE
  initial := (B1.initial & B2.initial & B3.initial & target.initial &
              operation=start_op & threshold = 0);
  failure := (B1.failure |B2.failure |B3.failure) |
             (target.final & !(B1.final & B2.final & B3.final));
```

the system Sys. The former returns the index of an available service executing the requested operation of the target service. The latter chooses the next operation that must be executed. Figure 7 gives the SMV modules of target service $\mathcal{B}_t$ and the available service $\mathcal{B}_1$. The transitions part (TRANS) of target module indicates how desired operations are released. Such operations are inputs in the available service module.

More precisely, the submodules indicating the two players of the game structure are detailed. For ctr, being an instance of controller or orchestrator (Ctr), the transition relation defined via the constraints in the INIT and TRANS parts encodes an unconstrained orchestrator, assigning at each step, one operation to each available service, by assigning values to the state variables state, operation, and threshold. The goal of synthesis is to restrict such a relation in order to obtain a winning strategy. More specifically, the constraints enforced on the controller player's state are as follows. Given the INIT part, in its initial state (where variable initial holds true) the controller must instruct every service to initialize itself by performing the dummy operation start (all services initialize simultaneously). As for non-initial states, the TRANS part defines the following constraints: (i) no initialization operation can be assigned to any service; (ii)

Figure 7
Modules of target service and an available service

```
MODULE Target(operation,threshold)
 VAR
   state : {start_st,t1,t2,t3};
 INIT
   state = start_st & operation = start_op & threshold = 0
 TRANS
  case operation
    state = start_st & operation = start_op & threshold = 0:
            next(state) = t1 & next(operation) in {StorageSpace210GB}
           & next(threshold) = 3;
    state = t1 & operation in {StorageSpace210GB} & threshold = 3:
                           next(state) = t2 & next(operation) in
                           {Windows7} & next(threshold) = 8;
    state = t2 & operation in {Windows7} & threshold = 8 :
                            next(state) = t3 &  next(operation) in
                            {SQL-Server2008} & next(threshold) = 7;
    state = t3 & in {SQL-Server2008} & threshold = 7 :
                            next(state) = t1 & next(operation) in
                           {StorageSpace210GB} & next(threshold) = 3;
   esac
  DEFINE
   initial := state = start_st & operation = start_op & threshold=0;
   final := state in {t1};

MODULE Service1(index,operation,threshold)
 VAR
   state : {start_st,a1,a2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & operation = start_op & threshold = 0 & index = 0: next(state) in {a1};
   (index != 1) : next(state) = state;
   (state=a1 & operation in {Windows8,Windows7} & threshold<=8) : next(state) in {a2};
   (state=a2 & operation in {SQL-Server2005,SQL-Server2008} & threshold<=8) : next(state) in {a1};
  esac
 DEFINE
  initial := state = start_st & operation = start_op & threshold = 0 & index = 0;
  failure := index = 1 & !((state = a1 & operation in {Windows8,Windows7} & threshold<=8) |
             (state = a2 & operation in {SQL-Server2005,SQL-Server2008} & threshold<=8));
  final := state in {a1};
```

the current operation request must match at least one of the service operation (by regarding the similarity between operations); (iii) a service can be instructed to execute an operation only if that operation is the one currently requested; and (iv) at most one service can be instructed to operate at a time.

Given the module Sys, being an instance of system, it contains all the aspects of the system player. More precisely, Sys is the synchronous product of available services (submodule Service1, Service2, Service3) and the target service (submodule Target). Some abbreviations are used to define, in the DEFINE part, namely final and failure states. In fact, the system fails (failure) when any of the available service is instructed to perform an operation that it cannot run based on its current state. Prohibiting such situations, by properly constraining sys transition relation, is the synthesis procedure goal. Furthermore, the entire system does not respect the final-state condition (final) when the target is in a state where it terminates its execution but the available services do not.

For providing a matchmaking between operations of a target service and those of available services, a variable, named threshold, is declared for each type of resource reasoning in the system module of SMV. Through this variable, target service is able to assign its acceptable threshold for the degree of match between its requested operation and the one in system $\mathcal{S}$. The type of this variable must be integer, since real numbers are not supported in SMV. For instance, a real number 0.7 is considered as an integer number 7. Given the ranges of resource reasoning functions, the domain of threshold for both similarity and numerical reasonings is integer numbers in an interval [0,10] and for compatibility reasoning is integer number in an interval (0,20). Such domains are appropriate if only one digit of decimal precision is taken into account for thresholds. Note that, the values returned from reasoning functions defined from Def. 2 to 4 cannot be computed through SMV/TLV.

The variable threshold is considered as an argument in both target module and available service modules. In the former, it is regarded as an output argument in which at each step of an operation request, its threshold, denoting the acceptable degree of match between the operation and the ones in available services, is released. In contrast, this variable is an input argument in the latter through which the precomputed degree of match between a requested operation and similar ones in the current state of an available service is compared with the released threshold. In the transition part of the available service module, the condition for the threshold is associated with an operation handled by the service and the one requested by the target. Hence, in each transition of the module, a set is provided to encompass both operations.

Example 3. *Given the transitions in the target service module depicted in Fig. 7 , it is supposed that:*

- the threshold τ_1 for the degree of match between *SQL–Server2005* and the similar operation in the system is 7,
- the threshold τ_2 for the degree of match between *Windows7* and the similar operation in the system is 8,
- the threshold τ_3 for the degree of match between *StorageSpace210GB* and the similar operation in the system is 3.

For simplification, although three different variables should be declared for τ_1, τ_2 and τ_3 in the implementation of this example, only one variable threshold is declared. Given the transition part of the target service module, this variable is assigned with three values of 3, 8, and 7 to consider the acceptable thresholds for numerical, compatibility, and similarity reasonings, respectively. These values are inputs in the module of available services. For instance, the requested resource *SQL–Server2008* can be matched with the operation *SQL–Server2005* in the module Service1, since the current input of threshold in this module has the value of 7, and the degree of similarity between the operations is 8 ().

Appendix A provides the details about the rest of SMV modules of Example 1 along with a part of TLV output indicating controller generator.

5. Experimental Results

Through the implementation of resource reasoning in SMV, three different experiments have been provided. Given a fixed number of target services, the first one is related to the effects of similarity reasoning on the number of realized target services. The second one is calculating the average time that it takes for the realization of target services. Finally, the third one evaluates the relationship between similarity reasoning and

the synthesized generated controllers for a target service under different rates of available services' failures.

5.1. Effect of Similarity Reasoning on Realized Target Services

The assumption is that there is one available service, carrying 16 different operations. Furthermore, it is supposed that number of target services varies from 2 to 12. Each target service requests four different operations so that from those, one operation is randomly chosen to be possibly matched with a similar one in the available service. The predefined threshold in the target service can be 2, 4, 6, or 8. Moreover, in the available service, the degree of match between an operation and the one requested by target service is randomly selected between 1 and 10.

Given the graphs illustrated in Fig. 8, the horizontal axes represent the number of target services, and the vertical axes indicate the average number of realized target services, where for each datum (point in a curve), it is calculated after 10 times execution of TLV program. To investigate the average number of realized target services in the larger scale of operations, the graphs depicted in Fig. 8(b) and Fig. 8(c) have been provided. In the former, the available service carries 32 different operations, and in the latter, the number of operations is extended to 48. Note that, in both graphs, the scenario for the requested operations of target services is similar to the one assumed for Fig. 8(a).

It can be comprehended from the graphs that when the rate of threshold increases, the average number of realized target services decreases sharply. Furthermore, the growth in the number of operations leads to an increase in the average number of realized target services. Besides, a compare between the curves and the threshold rates indicates that curves, having lower rates of threshold, are closer to each other, and such curves are mapped onto each other when the number of operations increases. For instance in Fig. 8(c), when the amount of thresholds were 2, 4, and 6, the curves were mapped onto each other and all released target services were realized. The reason of such fully realizable target services is that the probability of finding an operation whose degree of match with a requested operation of target service is high when target requests lower rates of thresholds and we have larger scale of operations. Notably, the fluctuations in the curves of the graphs are simply due to the random selection of degree of match between operations.

Figure 8
The relationships between realized target services and similarity reasoning under different scale of operations

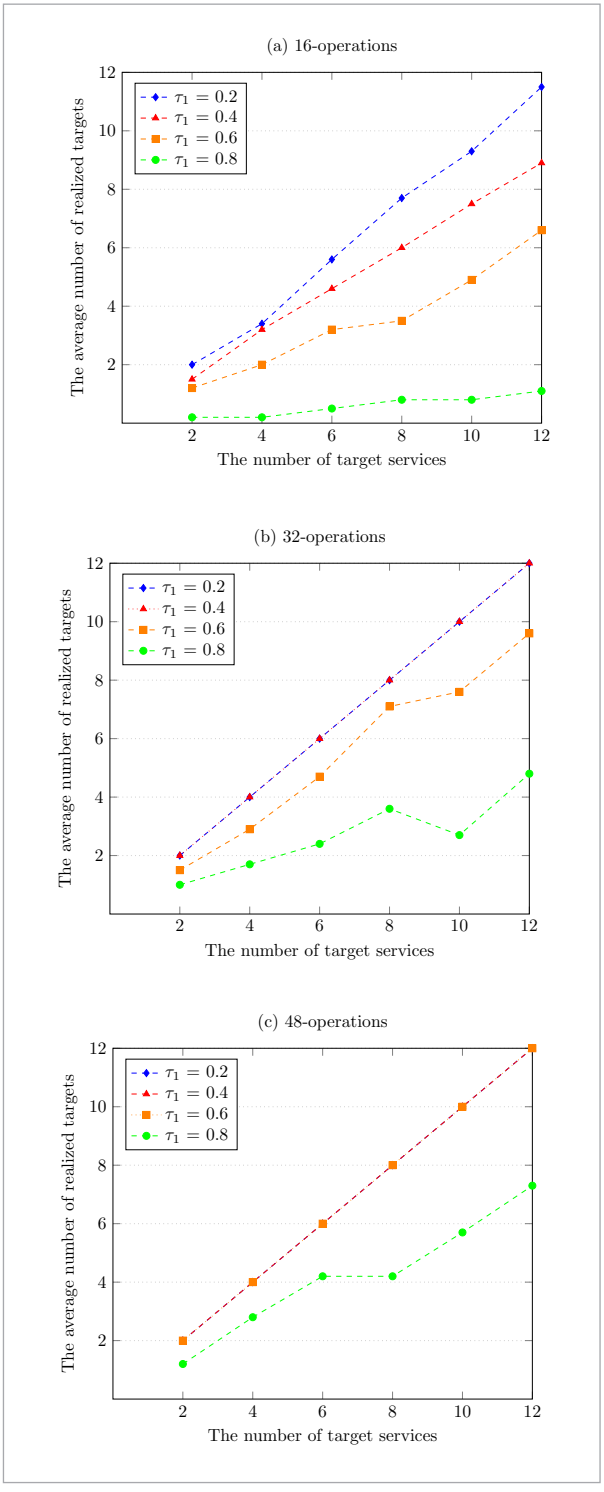


Table 1
A comparison between semantic-based framework and original behavior composition framework (ABCS)

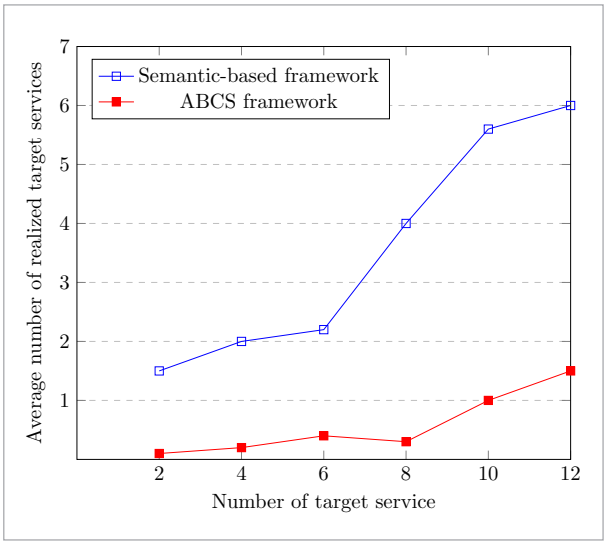
Number of target services	Average realization time in semantic-based framework	Average realization time in original framework (ABCS)
2	0.05	0.04
4	0.08	0.06
6	0.16	0.13
8	0.52	0.41
10	4.63	3.52
12	67.22	65.74

To have a compare between our semantic-based framework and the original automatic behavior composition synthesis (ABCS) framework[6] proposed in [9], the graph depicted in Fig. 9 is represented. In the experiment, the assumption is that we have one available service handling 32 operations and the rate of threshold released by target service is 7 in the semantic-based framework. Moreover, the number of released target services ranges from 2 to 12 and the degree of match between an operation of each target service and the one in available service is randomly selected between 1 and 10. As seen from Fig. 9, the chance of realization of target services in our approach is much more than the original framework proposed in [9], since our approach considers the operations which are similar to those requested by target service during building composite services.

5.2. Realization Time of Target Services

This experiment evaluates the average of time taken for the realization of targets services in both semantic-based and original behavior composition frameworks. It is assumed that there is one available service carrying 32 different operations and the number of target services varies from 2 to 12, each of which requests four different operations. Furthermore, in the available service, the degree of match between an operation and the one demanded by target service is randomly selected between 1 and 10. Target service requests only one similar operation along with a threshold randomly selected between one and ten.[7] Table 1 represents the average time (in seconds) which is taken for the realization of target services. The calculation of the average realization time was obtained after ten times execution of TLV program. As seen from the table, by increasing the number of target services, the realization time rises drastically. Moreover, the realization time in the original behavior composition is totally shorter than our semantic-based framework. This is due to the fact that the number of realized targets in the semantic-supported framework is much more than those realized by the original framework (see Fig. 9).

Figure 9
A comparison between realization time in semantic-based framework and original behavior composition framework



5.2.1. Effect of Similarity Reasoning on Controller Synthesis Under Different Rates of Service Failure

The assumption is that a target service requests 5 different operations. Some requested operations are randomly chosen to be possibly matched with the similar ones in the system. The number of requested similar operations varies from 1 to 5, and the predefined threshold can be 2, 4, 6, 8, or 10. There are 10 available services in the system handling totally 25 different operations, and for each operation, there exists at least two instances. More precisely, in this ex-

6 It is called also as original behavior composition framework.

7 When the target demands a threshold 10, it means a fully match should be found. Such demands are simply allowed in the original behavior composition framework.

periment, available services have been classified into three groups:

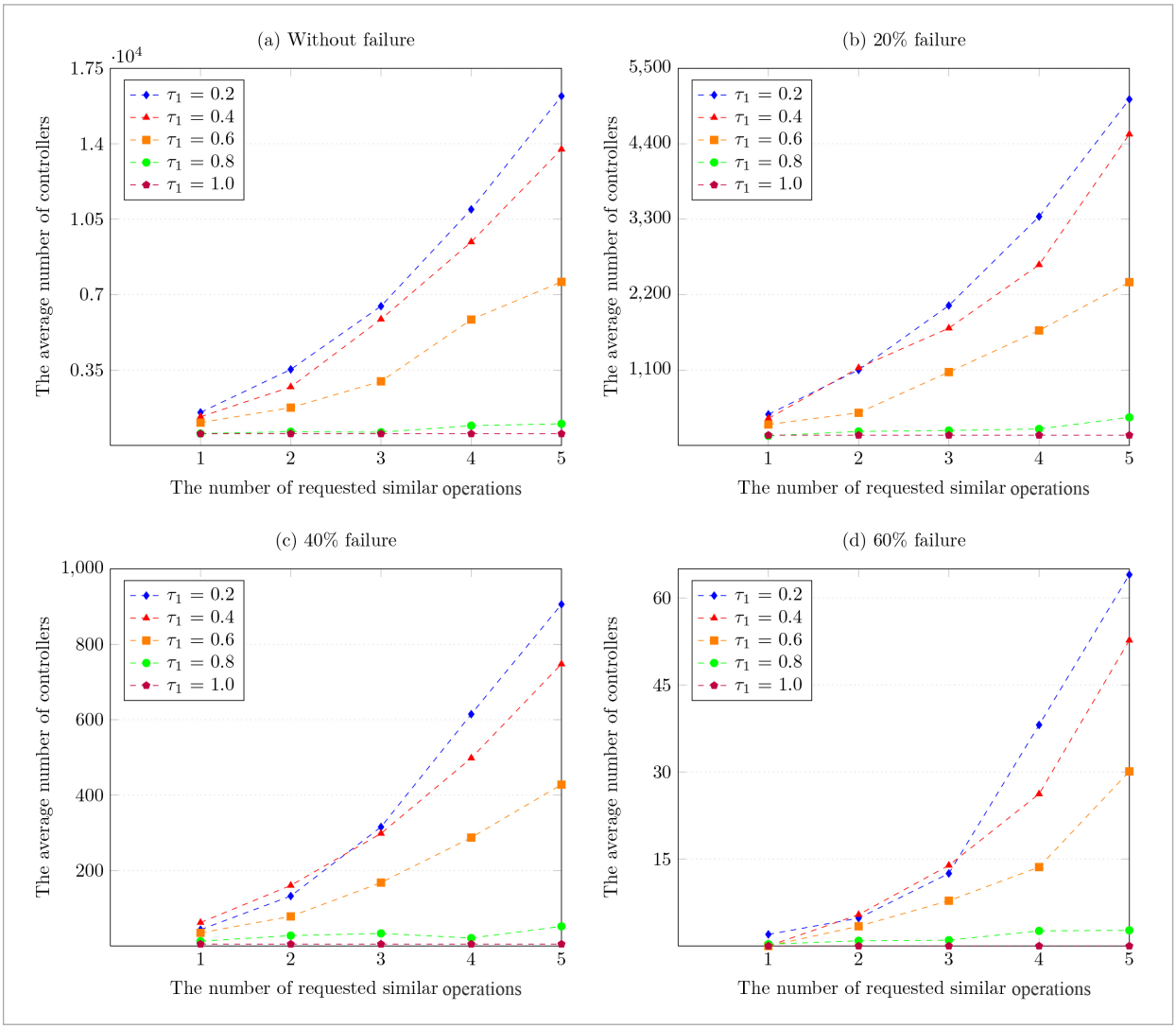
- the group with five services each of which has one state and handles 5 different operations;
- the group with three services each of which has two states and handles 10 different operations;
- the group with two services each of which has three states and handles 15 different operations.

In each available service, the degree of match between an operation and the similar one requested by target service is randomly chosen between 1 and 10.

Given the graphs illustrated in Fig. 10, the horizontal axes represent the number of similar operations requested by target service, and the vertical axes indicate the average number of synthesized controllers (generated controllers). For each datum (point in a curve), the average is calculated after 10 times execution of TLV program. To evaluate the average number of controllers when some available services randomly encounter with failures, the depicted graphs in Fig. 10(b)-(d) are provided. The results after the failures of 2, 4, and 6 available services are illustrated by the graphs, respectively.

Figure 10

The relationships between controller synthesis and similarity reasoning under different rates of failure



As seen from the figures, there is a significant decrease in the number of synthesized controllers when the rate of failures increases. Although the average number of controllers is nearly zero for the threshold one, the chance of synthesis rises when lower thresholds are requested. As another result, the increase in the number of requested similar operations leads to the increase in the number of controllers.

6. Conclusion

This paper introduced a semantic-based framework for the problem of behavior composition whereby service operations that have different names but the same functionality are taken into account during building composite services. Our semantic-based framework extended the original behavior composition framework [9] by developing synthesis procedure, leveraging three types of resource reasoning, namely similarity, compatibility, and numerical reasoning. Such an extension not only provided a more flexible framework for the behavior composition framework to match similar service operations, but also enabled semantic-supported cloud ecosystems to compose their services in an automatic way. Moreover, the paper gave some clues about the implementation of our framework via model checking tools in hand: SMV and TLV.

The kind of control exercised in our framework is a control by delegation. Other formal methods have been proposed for service composition when supervision (i.e., disabling controllable events) is adopted as kind of control (e.g., [6]), but the theoretical framework behind them do not consider resource reasoning behind their synthesis mechanism. Some formal methods for service composition, very different from the aforementioned ones, use a probabilistic (e.g., [17]), preferences-based planning (e.g., [34]) or process calculus(e.g., [7]) approach. These sorts of methods, however, did not synthesize automatic controllers to orchestrate a community of services.

The experiments showed that the possibility of realizing user requirements in our semantic-based version is much more than the original framework due to the flexible match between similar operations in the former. However, the realization time of specifications in our framework was fairly longer than the original one. Given possible failures of some available services, our framework indicated that has a higher rate of fault tolerance compared to the original one in terms of building composite services.

Although our framework prescribed a promising formal tools for composing cloud services in an automatic way, it still requires further work or investigations to be readily applied in a real cloud environment. In our present approach, the values returned from the resource reasoning functions, showing the degree of match between service operations, were not automatically computed by SMV/TLV. In fact, these values were manually calculated and inserted as integer numbers in the SMV module skeletons extended for the implementation of our framework. Hence, this problem may raise a question about how to establish a link between SMV/TLV and the available ontological engineering tools for building ontology graphs and knowledge-based solutions. As another potential research for future work, the integration of real-time constraints in our framework can be examined. By supporting such constraints, manufacturing cloud and Internet of Things (IoT) industries, often determining deadlines for the use of their services, can benefit from our proposed formal method to integrate objects or services. Furthermore, investigating the scalability of our semantic-based framework in terms of orchestrators synthesis will be another possible research direction.

Appendix

Appendix A. Implementation in SMV/TLV

Figure 11 represents the SMV modules of two available services $\mathcal{B}_2$ and $\mathcal{B}_3$ in Example 1. The controller generator that is obtained for the example is illustrated in Fig. 12. The TLV output indicates an automaton with 7 states and 10 transitions that was successfully synthesized.

Appendix B. An example of compability reasoning

Given the graph in Fig. 13, the values 1 to 9 show the chronological orderings of Windows. *Windows*95 is

assigned 1 to demonstrate that it is the oldest version of *Windows* and *WindowsServer*2008 is assigned 9 to represent the latest version. The degree of compatibility between *Windows*98 and *WindowsVista*, and the one between *Windows*7 and *WindowsVista* are calculated as follows.

$$\alpha(Windows98) = \alpha(Windows7) = \alpha(WindowsVista) = 2,$$

$$\alpha(Windows98) \cap \alpha(WindowsVista) = \alpha(Windows7) \cap \alpha(WindowsVista) = 1,$$

and

$$sim(Windows98, WindowsVista) = sim(Windows7, WindowsVista) = (0.5 + 0.5)/2 = 0.5.$$

The label values of $Windows98, WindowsVista,$ and $Windows7$ are $c_{w98} = 2, c_{wv} = 7,$ and $c_{w7} = 8$. For the aim of experimentation, μ and θ are set to 0.8 and 10, respectively. Given the compatibility reasoning function, $compat(Windows98, WindowsVista) = 0.533$ and $compat(Windows7, WindowsVista) = 0.58$. Hence, compared to $Windows98$, $Windows7$ is more compatible with $WindowsVista$.

Figure 11
The SMV modules of available services

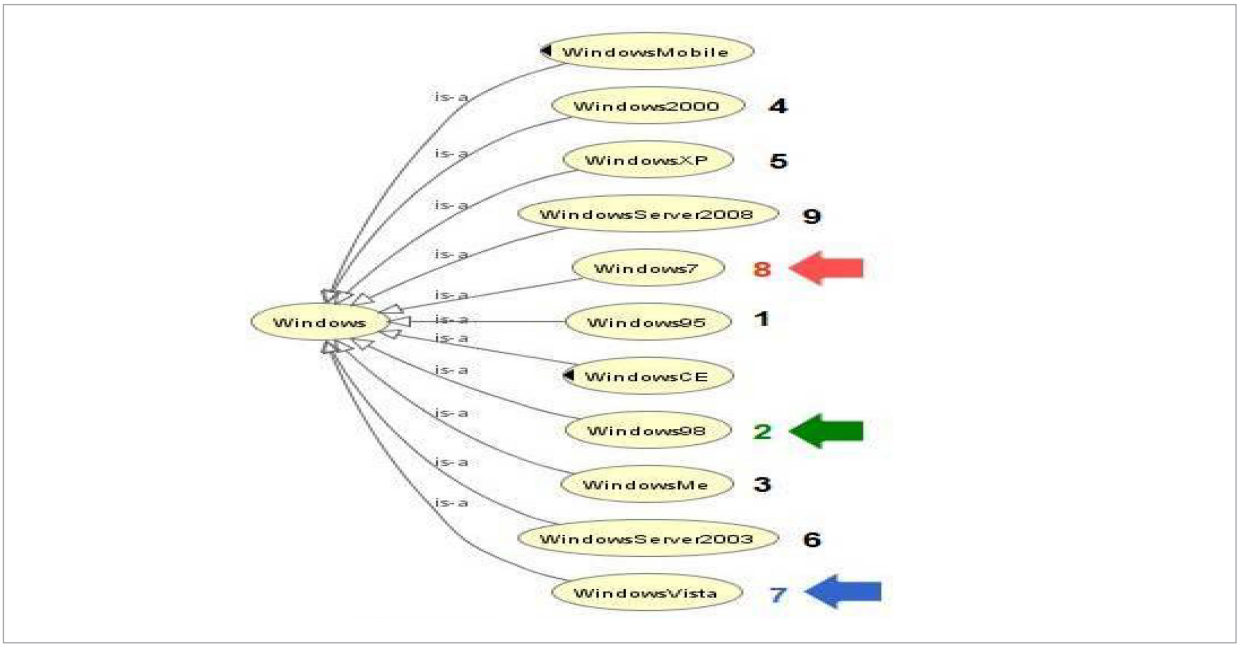
```
MODULE Service2(index,operation,threshold)
 VAR
   state : {start_st,b1,b2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & operation = start_op & threshold = 0 & index = 0 : next(state) in {b1};
   (index != 2) : next(state) = state;
   (state=b1 & operation in {StorageSpace210GB,StorageSpace220GB} & threshold<=3) : next(state) in {b2};
   (state=b2 & operation in {WindowsVista,Windows7} & threshold<=8) : next(state) in {b1};
   (state=b1 & operation in {SQL-Server2005,SQL-Server2008} & threshold<=8) : next(state) in {b1};
  esac
 DEFINE
  initial := state = start_st & operation = start_op & threshold = 0 & index = 0;
  failure := index = 2 & !((state = b1 & operation in {StorageSpace210GB,StorageSpace220GB} & threshold<=3) |
             (state = b1 & operation in {SQL-Server2005,SQL-Server2008} & threshold<=8) |
             (state = b2 & operation in {WindowsVista,Windows7} & threshold<=8));
  final := state in {b1};

MODULE Service3(index,operation,threshold)
 VAR
   state : {start_st,c1,c2};
 INIT
   state = start_st
 TRANS
  case
   state = start_st & operation = start_op & threshold = 0 & index = 0 : next(state) in {c1};
   (index != 3) : next(state) = state;
   (state=c1 & operation in {SQL-Server2008} & threshold==10) : next(state) in {c2};
   (state=c2 & operation in {SQL-Server2008} & threshold==10) : next(state) in {c1};
   (state=b1 & operation in {StorageSpace205GB,StorageSpace210GB} & threshold<=3) : next(state) in {b1};
  esac
 DEFINE
  initial := state = start_st & operation = start_op & threshold = 0 & index = 0;
  failure := index = 3 & !((state = c1 & operation in {SQL-Server2008} & threshold==10) |
             (state = c1 & operation in {StorageSpace205GB,StorageSpace210GB} & threshold<=3) |
             (state = c2 & operation in {SQL-Server2008} & threshold==10));
  final := state in {c1};
```

Figure 12
TLV output (controller generator)

```
All winning states satisfy invariant
Automaton States
State 1
 sys.operation = start_op            sys.threshold = 0             sys.target.state = start_st
 sys.B1.state = start_st            sys.B2.state = start_st        sys.B3.state = start_st
 ctr.index = 0
State 2
 sys.operation = Storage210GB        sys.threshold = 3             sys.target.state = t1
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 2
State 3
 sys.operation = Storage210GB        sys.threshold = 3             sys.target.state = t1
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 3
State 4
 sys.operation = Windows7            sys.threshold = 8             sys.target.state = t2
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 1
State 5
 sys.operation = SQL_Server2008      sys.threshold = 7             sys.target.state = t3
 sys.B1.state = a2                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 1
State 6
 sys.operation = Windows7            sys.threshold = 8             sys.target.state = t2
 sys.B1.state = a1                  sys.B2.state = b2              sys.B3.state = c1
 ctr.index = 2
State 7
 sys.operation = SQL_Server2008      sys.threshold = 7             sys.target.state = t3
 sys.B1.state = a1                  sys.B2.state = b1              sys.B3.state = c1
 ctr.index = 2
Automaton Transitions
 From 1 to  2  3
 From 2 to  6
 From 3 to  4
 From 4 to  5
 From 5 to  2  3
 From 6 to  7
 From 7 to  2  3
Automaton has 7 states, and 10 transitions
BDD nodes allocated: 7955
max amount of BDD nodes allocated: 7955
Bytes allocated: 589888
```

Figure 13
A simple ontology graph [31]



References

1. Amato, F., Moscato, F. Automatic Cloud Services Composition for Big Data Management. Proceedings of IEEE 30th International Conference on Advanced Information Networking and Applications Workshops, Crans-Montana, Switzerland, 2016, 46-51. https://doi.org/10.1109/WAINA.2016.169
2. Andreasen, T., Bulskov, H., Knappe, R. From Ontology Over Similarity to Query Evaluation. In Bernardi R., Moortgat, M. (Eds.), 2nd CoLogNET-ElsNET Symposium - Questions and Answers: Theoretical and Applied Perspectives, Amsterdam, The Netherlands, 2003, 39-50.
3. Arp, R., Smith, B., Spear, A. D. Building Ontologies with Basic Formal Ontology. MIT Press, Cambridge, Massachusetts, 2015. https://doi.org/10.7551/mitpress/9780262527811.001.0001
4. Barati, M. Composition of Dynamic Components Based on Behavioral Descriptions. Ph.D. Thesis, Département d'informatique, Université de Sherbrooke, 2018.
5. Berardi, D., Cheikh, F., De Giacomo, G., Patrizi, F. Automatic Service Composition via Simulation. International Journal of Foundations of Computer Science, 2008, 19, 429-451. https://doi.org/10.1142/S0129054108005759
6. Bertoli, P., Pistore, M., Traverso, P. Automated Composition of Web Services via Planning in Asynchronous Domain. Artificial Intelligence, 2010, 174, 316-361. https://doi.org/10.1016/j.artint.2009.12.002
7. Dalla Preda, M., Gabbrielli, M., Guidi, C., Mauro, J., Montesi, F. Interface-Based Service Composition with Aggregation. In De Paoli, F., Pimentel, E., Zavattaro, G. (Eds.), First European Conference on Service-Oriented and Cloud Computing, Lecture Notes in Computer Science, Bertinoro, Italy, 7592, 2012, 48-63. https://doi.org/10.1007/978-3-642-33427-6_4
8. De Giacomo, G., Mecella, M., Patrizi, F. Automated Service Composition Based on Behaviors: The Roman Model. In: Bouguettaya, A., Sheng, Q. Z., Daniel, F. (Eds.), Web Services Foundations, Springer, New York, 2014, 189-214. https://doi.org/10.1007/978-1-4614-7518-7_8
9. De Giacomo, G., Patrizi, F., Sardina, S. Automatic Behavior Composition Synthesis. Artificial Intelligence, 2013, 196, 106-142. https://doi.org/10.1016/j.artint.2012.12.001

10. Hale, M. L., Gamble, M. T., Gamble, R. F. A Design and Verification Framework for Service Composition in the Cloud. Proceedings of 2013 IEEE 9th World Congress on Services, Santa Clara, CA, 2013, 317-324. https://doi.org/10.1109/SERVICES.2013.46

11. Han, L., Dave, B. Semantic-Supported and Agent-Based Decentralized Grid Resource Discovery. Future Generation Computer Systems, 2008, 24, 806-812. https://doi.org/10.1016/j.future.2008.04.005

12. Huangab, C., Wangab, X., Wang, D. Type Theory Based Semantic Verification for Service Composition in Cloud Computing Environment. Information Sciences, 2018, 496, 101-118. https://doi.org/10.1016/j.ins.2018.08.042

13. Kang, J., Sim, K. M. Cloudle: An Agent-Based Cloud Search Engine That Consults a Cloud Ontology. Proceedings of Annual International Conference on Cloud Computing and virtualization, Singapore, 2010, 312-318. https://doi.org/10.5176/978-981-08-5837-7_224

14. Kang, J., Sim, K. M. Cloudle: An Ontology-Enhanced Cloud Service Search Engine. Proceedings of 1st International Workshop Cloud Information System Engineering, Collocated with 11th International Conference Web Information System Engineering, Hong Kong, 2010, 416-427. https://doi.org/10.1007/978-3-642-24396-7_33

15. Kil, H., Nam, W., Lee, D. Behavioural Description Based Web Service Composition Using Abstraction and Refinement. International Journal of Web and Grid Services, 2013, 9, 54-81. https://doi.org/10.1504/IJWGS.2013.052849

16. Leite, L., Moreira, C. E., Cordeiro, D., Gerosa, M. A., Kon, F. Deploying Large-Scale Service Compositions on the Cloud with the CHOReOS En actment Engine. Proceedings of 2014 IEEE 13th International Symposium on Network Computing and Applications, Cambridge, MA, 2014, 121-128. https://doi.org/10.1109/NCA.2014.25

17. Li, L., Jin, Z., Li, G., Zheng, L., Wei, Q. Modeling and Analyzing the Reliability and Cost of Service Composition in the IoT: a Probabilistic Approach. Proceedings of 19th International Conference on Web Services, Honolulu HI, 2012, 584-591. https://doi.org/10.1109/ICWS.2012.25

18. Lustig, Y., Vardi, M. Y. Synthesis from Component Libraries. International Journal on Software Tools for Technology Transfer, 2012, 15, 603-618. https://doi.org/10.1007/s10009-012-0236-z

19. Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D. L., Sirin, E., Srinivasan, N. Bringing Semantics to Web Services with OWL-S. World Wide Web Journal, 2007, 10, 243-277. https://doi.org/10.1007/s11280-007-0033-x

20. Modi, K. J., Garg, S. A QoS-Based Approach for Cloud-Service Matchmaking, Selection and Composition Using the Semantic Web. Journal of Systems and Information Technology, 2019, 21, 63-89. https://doi.org/10.1108/JSIT-01-2017-0006

21. Modi, K. J., Garg, S., Chaudhary, S. An Integrated Framework for RESTful Web Services Using Linked Open Data. International Journal of Grid and High Performance Computing, 2019, 11, 24-49. https://doi.org/10.4018/IJGHPC.2019040102

22. Nayak, G., Dutta, S., Ajwani, D., Nicholson, P., Sala, A. Automated Assessment of Knowledge Hierarchy Evolution: Comparing Directed Acyclic Graphs. Information Retrieval Journal, 2019, 22, 256-284. https://doi.org/10.1007/s10791-018-9345-y

23. Niwattanakul, S., Singthongchai, J., Naenudorn, E., Wanapu, S. Using of Jaccard Coefficient for Keywords Similarity. Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, 2013.

24. Paik, I., Chen, W., Huhns, M. N. A Scalable Architecture for Automatic Service Composition. IEEE Transactions on Services Computing, 2014, 7, 82-95. https://doi.org/10.1109/TSC.2012.33

25. Pirró, G. A Semantic Similarity Metric Combining Features and Intrinsic Information Content. Data & Knowledge Engineering, 2009, 68, 1289-1308. https://doi.org/10.1016/j.datak.2009.06.008

26. Pittaras, C., Ghijsen, M., Wibisono, A., Grosso, P., Der Ham, J. V., Laat, C. Semantic Distributed Resource Discovery for Multiple Resource Providers. Proceedings of 8th International Conference on Semantics Knowledge and Grids, Beijing, China, 2012, 225-228. https://doi.org/10.1109/SKG.2012.46

27. Pnueli, A., Shahar, E. A Platform for Combining Deductive with Algorithmic Verification. Proceedings of International Conference Computer Aided Verification (CAV), New Brunswick, NJ, 1996, 184-195. https://doi.org/10.1007/3-540-61474-5_68

28. Ramirez, M., Yadav, N., Sardina, S. Behavior Composition as Fully Observable Non-Deterministic Planning. Proceedings of 23rd International Conference on Automated Planning and Scheduling, Rome, Italy, 2013, 180-188.

29. Randelli, G., Marchetti, L., Marino, F. A., Iocchi, L. Multi-Agent Behavior Composition Through Adaptable Software Architectures and Tangible Interfac-

es. In Ruiz-del Solor, J., Chown, E., Plöger, P. G. (Eds.), RoboCup 2010: Robot Soccer World Cup XIV, Lecture Notes in Computer Science, Singapore, 6556, 2011, 278-290. https://doi.org/10.1007/978-3-642-20217-9_24

30. Rodriguez, M. A., Egenhofer, M. J. Determining Semantic Similarity Among Entity Classes from Different Ontologies. IEEE Transactions on Knowledge and Data Engineering, 2003, 15, 442-456. https://doi.org/10.1109/TKDE.2003.1185844
31. Sim, K. M. Agent-Based Cloud Computing. IEEE Transactions on Services Computing, 2012, 5, 564-577. https://doi.org/10.1109/TSC.2011.52
32. Slimani, T., Description and Evaluation of Semantic Similarity Measures Approaches. International Journal of Computer Applications, 2013, 80, 25-33. https://doi.org/10.5120/13897-1851
33. Slimani, T., Ben Yaghlane, B., Mellouli, K. A New Similarity Measure Based on Edge Counting. World Academy of Science, Engineering and Technology, 2006, 34-38.
34. Sohrabi, S., McIlraith, S. A. Preference-Based Web Service Composition: A Middle Ground Between Execution and Search. In Patel-Schneider, P. F., Pan, Y., Hitzler, P., Mika P., Zhang, L., Pan, J. Z., Horrocks, I., Glimm, B. (Eds.), Proceedings of 9th International Semantic Web Conference, Lecture Notes in Computer Science, Shanghai, China, 6496, 2010, 713-729. https://doi.org/10.1007/978-3-642-17746-0_45
35. Song, Y., Sun, Q., Zhou, A., Wang, S., Li, J. QoS-Aware Service Composition Using HTN Planner. Proceedings of 8th International Symposium on Cloud and Service Computing, Paris, France, 2018, 107-110. https://doi.org/10.1109/SC2.2018.00022
36. Souri, A., Navimipour, N. J. Behavioral Modeling and Formal Verification of a Resource Discovery Approach in Grid Computing. Expert Systems with Applications, 2014, 41, 3831-3849. https://doi.org/10.1016/j.eswa.2013.11.042
37. Sun, C., Wang, Z., Wang, K., Xue, T., Aiello, M. Adaptive BPEL Service Compositions via Variability Management: a Methodology and Supporting Platform. International Journal of Web Services Research, 2019, 16, 37-69. https://doi.org/10.4018/IJWSR.2019010103
38. Ter Beek, M. H., Bucchiarone, A., Gnesi, S. Formal Methods for Service Composition. Annals of Mathematics, Computing & Teleinformatics, 2007, 1, 1-10.
39. Viriyasitavat, W., Xu, L. D., Bi, Z. The Extension of Semantic Formalization of Service Workflow Specification Language. IEEE Transactions on Industrial Informatics, 2019, 15, 741-754. ttps://doi.org/10.1109/TII.2018.2807400
40. Yadav, N. K. Behavior Composition Optimisation. PhD thesis, School of Computer Science and Information Technology, RMIT University, Melbourne, Australia, 2014.
41. Ye, J. Cosine Similarity Measures for Intuitionistic Fuzzy Sets and Their Applications. Mathematical and Computer Modelling, 2011, 53, 91-97. https://doi.org/10.1016/j.mcm.2010.07.022
42. Ylianttila, M., Riekki, J., Zhou, J., Athukorala, K., Gilman, E. Cloud Architecture for Dynamic Service Composition. International Journal of Grid and High Performance Computing, 2012, 4, 17-31. https://doi.org/10.4018/jghpc.2012040102
43. Yongxiang, L., Xifan, Y., Jie, Z., Bin, L. Cloud Manufacturing Service Composition Modeling and Formal Verification Based on Calculus for Orchestration of Web Service. Proceedings of 25th Chinese Control and Decision Conference, Guiyang, China, 2013, 2806-2810. https://doi.org/10.1109/CCDC.2013.6561422
44. Yongxiang, L., Xifan, Y., Xiangmin, X., Hong, J. Formal Verification of Cloud Manufacturing Service Composition and BPEL Codes Generation Based on Extended Process Calculus. Information Technology Journal, 2014, 13, 1779-1785. https://doi.org/10.3923/itj.2014.1779.1785