# Extending the Service Composition Formalism with Relational Parameters

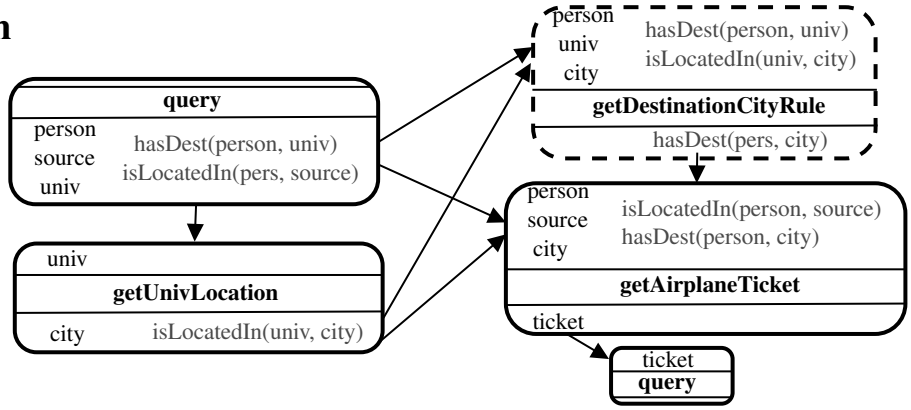
Paul Diac, Liana Ţucăr, Radu Mereuţă
Alexandru Ioan Cuza Unviersity of Iaşi, România
{paul.diac, liana.tucar, radu.mereuta}@info.uaic.ro

Web Service Composition deals with the (re)use of Web Services to provide complex functionality, inexistent in any single service. Over the state-of-the-art, we introduce a new type of modeling, based on ontologies and relations between objects, which allows us to extend the expressiveness of problems that can be solved automatically.

## 1 Introduction

Web Service Composition is a complex research area, involving other domains such as: web standards, service-oriented architectures, semantics, knowledge representation, algorithms, optimizations, and more [4]. We propose an extended model that allows the specification of relationships between parameters, as a generalization of previous models such as [1]. Moreover, it allows working with different instances of the same type of concept within the automatic composition; a feature that is fundamental in manual composition, and also defines *inference rules*. The formalism defined in Section 3 is a complete specification of the model presented in our previous work [3]. We motivate the proposed model by an intuitive example and verify its effectiveness by implementing and testing a composition algorithm.

## 2 Motivation




We present a simple query, where a user wants to travel to a university located in a different city. Each rectangle represents a web service with input at the top and output at the bottom. We also represent the query twice as a services with no input and respectively, no output. The dashed rectangle is an inference rule, handled by the algorithm as a virtual web service.

Because we cannot directly get the answer to the query, we must use the information provided by different services and rules found in the ontology to compose an answer. We see that in order to buy a ticket we must know the source city and the destination city, the latter found indirectly. We must first call a different web service which finds the city where the university is located and by using the inference rule we can finally match the precondition of the web service which can return the plane ticket. Arrows show parameters and relations matching. The algorithm in 4 finds the correct order of calls; in this case: *(1) getUniversityLocation*, applies *(2) getDestinationCityRule* and then *(3) getAriplaneTicket*.

Submitted to:
FROM 2019

# 3 Formal Model

We define the model in three steps: the original composition formalism matches parameters by name; the semantic level defining concepts over a taxonomy; and finally the new relational level, enhancing the taxonomy to a full ontology. On these three levels, expressiveness increases allowing for more and more natural composition examples to be resolvable by composition algorithms if appropriately modeled. The first two are well-known and were used in the Composition Challenges in [2] and [1]. The last level is our contribution, introducing two important concepts: parameter relations and, as a consequence of the first, type instances as separate matching objects.

## 3.1 Name-based Parameter Matching

The initial and simplest model for *Web Service Composition* uses parameter **names** to match services. Each **name** represents a **concept**. Expecting that parameters are chosen from a set of predefined concepts, the output of a previously called service can be used as input to another service. The user specifies a composition request by a list of known concepts and a list of required concepts. A request has the structure of a service but expresses the need for such a service. A satisfying composition is a (partially ordered) list of services generating the requested concepts starting from known concepts.

**Definition 1. Concept**s are elements from the predefined set of all concepts $\mathbb{C}$.

**Definition 2. Web Service**s are triplets $\langle name, I, O\rangle$ consisting of: the service name and two disjoint sets of concepts, also referred to as parameters: input and output. The **User Request** has the same structure and specifies a required functionality, possibly solvable by a list of services. $I \cap O = \emptyset$ and $I, O \subseteq \mathbb{C}$.

**Definition 3.** The **Repository** is the set of all services, also written as $\mathbb{S}$.

**Definition 4. Parameter Matching**. If $C$ is a set of (known) concepts and $ws = \langle ws.name, ws.I, ws.O\rangle$ is a web service, then $C$ matches $ws$ iff $ws.I \subseteq C$.
The result of matching, or the union of $C$ and $ws.O$ is $C \oplus ws = C \cup ws.O$.

**Definition 5. Chained Matching**. If $C$ is a set of concepts and $\langle ws_1, ws_2, \ldots, ws_n\rangle$ a list of services then: $C \oplus ws_1 \oplus ws_2 \oplus \cdots \oplus ws_n$ is a chain of matching services, generating $C \cup \bigcup_{i=1}^{n} ws_i.O$; valid iff:

$$ws_i.I \subseteq \left(C \cup \left(\bigcup_{j=1}^{i-1} ws_j.O\right)\right), \forall i = \overline{1..n}$$

**Definition 6. Web Service Composition Problem**. Given a repository of services $\mathbb{S}$ and a user request $r = \langle r.name, r.I, r.O\rangle$, all with parameters defined over the set of concepts $\mathbb{C}$; find a chain of matching services $\langle ws_1, ws_2, \ldots, ws_n\rangle$ such that $r.I \oplus ws_1 \oplus ws_2 \oplus \cdots \oplus ws_n \subseteq r.O$.

## 3.2 Taxonomy-based Parameter Matching

Subsequent models extend the definitions of **concept**s and **parameter matching** (1 and 4), and the rest of the definitions adapt to these changes.

**Definition 7. Concept**s (in model 3.2) are elements of the set of concepts $\mathbb{C}$, over which the binary relation $subtypeOf$ is defined. $subtypeOf \subseteq \mathbb{C}^2$ and $subtypeOf$ is *transitive*.

**Definition 8. Parameter Matching** (in 3.2). If $C \in \mathbb{C}$ is a set of concepts with $subtypeOf$ relation, and $ws = \langle ws.name, ws.I, ws.O\rangle$ a service, then $C$ matches $ws$ iff:
$\forall\ c \in ws.I,\ \exists\ spec \in C,\ such\ that\ (spec, c) \in subtypeOf$.
The result of matching is $C \oplus ws = C \bigcup \left\{gen \in \mathbb{C} \;\middle|\; \exists\ c \in ws.O,\ such\ that\ (c, gen) \in subtypeOf\right\}$.

### 3.3 Ontological Level: Relational and Contextual Model

The main contribution of the paper is the introduction of two elements: *relations* and *objects*. Relations are a generalization of the $subtypeOf$ relation of the previous level. Multiple relations are allowed between concepts, defined in the semantic *ontology*. Service providers do not define new relations; they can only use existing relations defined in the *ontology* to describe their parameters. Relations can be *transitive* and/or *symmetric*.

Concepts can now be described with more semantic context, and it is useful to allow updates on it. Therefore, we also introduce *objects*, that are similar to instances of concepts. Instances are not concrete values of concept types, but distinct elements that are passed trough service workflow, distinguished by their provenance and described by a set of semantic relations.

*Inference rules* are also introduced as a generalization of relation properties. *Inference rules* generate new relations on objects if some preconditions are met. Similarly, web service calls that exclusively generate objects can also generate relations. Service input can define preconditions that include relations on objects matching input parameters.

**Definition 9.** An **Object** is an element of the set of objects $\mathbb{O} = \{o = \langle id, type \rangle\}$. $id$ is a unique identifier generated at object creation. The $type$ is a concept: $type \in \mathbb{C}$.

**Definition 10. Relation**. A relation $r$ is a triple consisting of: the name as a unique identifier, relation properties, and the set of pairs of objects that are in that relation. The latter is dynamic, i.e., can be extended through the composition process.

$$\mathbb{R} = \left\{ \langle name, properties, objects \rangle \;\middle|\; properties \subseteq \{transitivity,\ symmetry\} \text{ and } objects \subseteq \mathbb{O}^2 \right\}$$

**Definition 11.** The **knowledge** $\mathbb{K}$ is a dynamic structure consisting of objects and relations between the objects. Knowledge describes what is known at a stage of the composition workflow, i.e., at a time when a set of services have been added to the composition. $\mathbb{K} = \langle \mathbb{O}, \mathbb{R} \rangle$.

**Definition 12. Web Services** (in model 3.3) are tuples $\langle name, I, O, relations \rangle$ with $I, O$ defined as in def. 2 and $relations$ specifying preconditions and postconditions (effects) over objects matched to service inputs or generated at output. *relations* within service definitions are pairs consisting of: the *name* used to refer to an existing relation (the relation from $\mathbb{R}$ with the same name), and a binary relation over all service parameters. Relations between inputs are preconditions, and relations between output are effects, i.e., they are generated after the call. Relations between input and output parameters are effects.

$$ws.relations = \left\{ \langle name, parameters \rangle \;\middle|\; names \text{ from } \mathbb{R} \text{ and } parameters \subseteq (ws.I \cup ws.O)^2 \right\}$$

**Definition 13. Inference Rules** (in 3.3) are tuples $rule = \langle name, parameters, preconditions, effects \rangle$ where $parameters$ is a set of parameter names with local visibility (within rule), and preconditions and effects are relations defined over $parameters$. More precisely:

$$rule.preconditions, rule.effects \subseteq \left\langle \bigcup_{rel \in \mathbb{R}} rel.name, rule.parameters^2 \right\rangle$$

The set of all inference rules is written as $\mathbb{I}$. Preconditions must hold before applying the rule for the objects matching rule parameters, and relations in the effects are generated accordingly. Rules are structurally similar to services, but they apply automatically and, conceptually, with no cost. For example, *transitivity* and *symmetry* are particular rules, the following expresses that $equals$ is symetric:

$$equals_{symmetric} = \left\langle \{X, Y\}, \{\langle equals, \{(X, Y)\} \rangle\}, \{\langle equals, \{(Y, X)\} \rangle\} \right\rangle$$

**Definition 14. Ontology** $\mathbb{G}$ consists of: concepts organized hierarchically, relations and inference rules. $\mathbb{G} = \langle\ \mathbb{C}, subtypeOf, \mathbb{R}, \mathbb{I}\ \rangle$. At ontological level, relations are static and defined only by names and properties. At **knowledge** level, relations are dynamic in what objects they materialize to. We refer to both using $\mathbb{R}$.

**Definition 15. Parameter Matching** (in 3.3). In ontology $\mathbb{G} = \langle\ \mathbb{C}, subtypeOf, \mathbb{R}, \mathbb{I}\ \rangle$, a web service *ws* matches (is "callable" in) a knowledge state $\mathbb{K} = \langle \mathbb{O}, \mathbb{R} \rangle$, iff:

$$\exists\ function\ \ f : ws.I \to \mathbb{O}\ \ such\ that: \\ \forall\ i \in ws.I, \big(f(i), i\big) \in subtypeOf\ \ and \tag{1}$$

$$\forall\ \big(\ i, j \in ws.I\ \ and\ \ r_{ws} \in ws.relations,\ \ with\ \ (i,j) \in r_{ws}.parameters\big) \\ \exists\ \big(r_{obj} \in \mathbb{R}\ \ with:\ \ r_{obj}.name = r_{ws}.name\ \ and\ \ \big(f(i), f(j)\big) \in r_{obj}.objects\big) \tag{2}$$

We skip other similar definitions in model 3.3 that are intuitively similar, such as $\mathbb{K} \oplus ws$, **chained matching**, **user request** and the **composition problem**.

# 4 Composition Algorithm

**Overview** The algorithm takes as input a query from the user, the repository and ontology and returns a composition of services that answers the query. We start by populating a set (called "knowledge") with objects and relations based on the information provided by the user. We then repeat the process of adding new objects and relations until no more service calls can be made or until the query can be answered.

*init*: data structures and create virtual services for inference rules;
**while** $\neg canAnswerQuery(query)$ *And* $compositionUpdated = True$ **do**
  $compositionUpdated \leftarrow False$ ;
  **foreach** $service \in repository$ **do**
    $possibleCalls \leftarrow searchForPossibleCalls(service)$ ;
    **foreach** $servCall \in possibleCalls$ **do**
      **if** $providesUsefulInformation(servCall)$ **then**
        $makeCall(servCall)$; $compositionUpdated \leftarrow True$ ;
**if** $canAnswerQuery(query)$ **then return** $composition$ ;
**else return** *Not Solved* ;

**Construction Phase** At each step, we iterate over all the services and search for all possible calls. We then add to the composition the service calls that provide new information: i.e. a service call is excluded if all the new objects added are semantically similar to others already present in the knowledge. To obtain the similarity between objects, we represent the knowledge as a labeled directed graph where vertices are objects (labeled with the type of the object), and edges are relations, and we consider two objects similar if their associated connected components are isomorphic.

When a service call is made, new objects and relations corresponding with the service output and postconditions are created and added in the knowledge.

**Search for service calls** Finding all possible service calls of a given service *serv* means finding all combinations of objects that can be used as input parameters for the service: i.e. finding for each input parameter a corresponding object in the *knowledge* that has a type that is equal or more general with the parameter type. Besides this condition regarding the types, all relations from preconditions need to hold between corresponding objects used to call the service.

This problem reduces to finding all subgraph isomorphisms in the following problem instance:

- $q = (V, E, L)$, where $V = serv.inputParams, E = serv.preConditions$, $L(u) = u.type, \forall u \in V$ and $L(e) = e.name, \forall e \in E$;
- $g = (V', E', L')$, where $V' = knwoledge.objects, E' = knowledge.relationsBo$, $L(u') = \{type \mid (u'.type, type) \in ontology.subType\}, \forall u' \in V'$ and $L(e') = e'.name, \forall e' \in E'$;

The associated decision problem is known to be NP-Complete and an optimized backtracking procedure was implemented to solve it. In real-world use cases, we expect that instances that are computationally hard are rare. This is because service and rule preconditions are checked at each step of the backtracking, pruning many execution paths. Moreover, the *inference rules* that are more generic - i.e. without typed parameters - are defined in the ontology that cannot be updated by service developers or other users, so expensive rules can be safely avoided.

**canCallService** To check if service calls provide useful information and if the query is solved, virtual services with corresponding parameters and conditions are constructed and checked if can be called. This problem is similar to the one described above, except only one solution is needed, not all possible service calls.

An optimization implemented in the backtracking procedure is to **split the query graph into connected components** and to search each of them independently in the data graph. If the query graph is formed by multiple connected components this optimization helps by reducing a cartesian product of tested solutions of each component to a union of them. This has been proven to have a significant benefit on the runtime on tests (depending on how the tests are generated).

# 5 Conclusion

Current Web Service Composition models include limited semantics in expressing how service parameters are matched. Particularly, there is no way to express any relationships between parameters, and parameter typing models do not allow distinguishing the separation between instances of the same concept. In this paper, we propose a formalism that solves both of these limitations. We also implemented an efficient automatic composition algorithm that produced valid compositions on generated tests, using all elements in the proposed model.

# References

[1] Ajay Bansal, M Brian Blake, Srividya Kona, Steffen Bleul, Thomas Weise & Michael C Jaeger (2008): *WSC-08: continuing the web services challenge*. In: *IEEE Conference on E-Commerce Technology and the Fifth IEEE Conference on Enterprise Computing, E-Commerce and E-Services*, IEEE.

[2] M Brian Blake, Kwok Ching Tsui & Andreas Wombacher (2005): *The EEE-05 challenge: A new web service discovery and composition competition*. In: *IEEE International Conference on e-Technology, e-Commerce and e-Service*, IEEE.

[3] Paul Diac, Liana Ţucăr & Andrei Netedu (2019): *Relational Model for Parameter Description in Automatic Semantic Web Service Composition*. In: *International Conference on Knowledge-Based and Intelligent Information & Engineering Systems*, Elsevier.

[4] Jinghai Rao & Xiaomeng Su (2004): *A survey of automated web service composition methods*. In: *International Workshop on Semantic Web Services and Web Process Composition*, Springer, pp. 43–54.