Application of Graph Transformation for Automating Web Service Discovery*

Reiko Heckel and Alexey Cherchago

University of Paderborn, Paderborn, Germany
reiko|cherchago@upb.de

Abstract. The paper represents current achievements of an ongoing research that aims to develop a formal approach supporting an automatic selection of a Web service sought by a requestor. The approach is based on the matching the requestor's requirements for a "useful" service against the service description offered by the provider. We focus on the checking behavioral compatibility between operation contracts specifying pre-conditions and effects of required and provided operations.
Graph transformation rules with positive application conditions are proposed as a visual formal notation for contracts. The desired dependence between requestor and provider contracts is determined by the semantic compatibility relation and syntactic matching procedure that is sound w.r.t. this relation.

1 Introduction

The Web Services technology adopts the WWW for application to application communication that, in its turn, enables applications to interact with each other either to use services delivering by other business entities or to provide complex services based on their own capabilities. The ability of applications to discover automatically useful services and select those of them that can be safely integrated with the existing components leads Web services to their full potential.

Much work has been already done: The interface of the provided service can be specified in the Web Service Description language (WSDL). This specification as well as additional descriptive information about the delivered service is placed by a provider into a UDDI-registry being an information repository. A service requestor can query the registry to get descriptions of services delivering facilities for specific business domains.

While WSDL and UDDI (partially) support the dynamic discovery of services, an automatic selection of candidates which can be adequately integrated into the requestor application is not captured by the Web services standards. The service descriptions obtained from the UDDI-registry is manually analyzed by a developer of the requestor application. Our objective is to enable techniques allowing automation of the selection process. The present achievements regarding this global goal are discussed in the paper.

* Research funded in part by European Community's Human Potential Programme under contract HPRN-CT-2002-00275, [SegraVis].

Dagstuhl Seminar Proceedings 04101
http://drops.dagstuhl.de/opus/volltexte/2005/12

The selection process in our research is based on the matching provided and required service specifications describing the sets of (provided and required) operations. Each required operation must be associated with the structurally and behaviorally compatible provided operation. The *structural compatibility* is ensured by the matching operation signatures. Since this topic has been thoroughly investigated in the area of Component-Based Software Engineering (CBSE) [10], the existing techniques can be transfered to the Web Services domain. Our work basically contributes to the problem of the *behavioral compatibility*. We employ a concept of *contracts* [5] to represent behavioral information about operations. Graph transformation rules with positive application conditions are proposed as a visual formal notation for contract specification.

An operation contract contains potentially incomplete information about transformation of a system state, therefore a loose semantic of contracts must be adequately reflected in an approach to graph transformation. In the classical double-pushout (DPO) approach [4] it is assumed that nothing is changed in the transformation beyond what is explicitly specified in the rule. It does not fit our purposes. The double-pullback (DPB) approach [6] defines *graph transitions* and generalizes DPO by allowing additional changes, not encoded in the rule. Such alternations require refinements of the structure of the rules via application conditions. Besides the elements whose deletion, creation, and preservation are specified, we would like to specify those elements which must be present in the rule to be applicable, but are not necessarily affected by the rule application. These elements compose positive application conditions. In the DPO approach such positive application conditions could be included in the left-hand sides of the rules, because due to the implicit frame condition of the approach all that is not explicitly affected is preserved.

To check whether the provided operation contract matches the required one we will establish a *semantic compatibility relation* between the contract rules. Since this relation will be based on an infinite set of transitions and, therefore, can not be computed directly, a *syntactic matching relation* will be defined. We will demonstrate that the syntactic matching relation provides sufficient conditions with respect to the semantic one.

The paper is organized as follows: Section 2 describes the basics of the DPO and DPB approaches to graph transformation. In Section 3, we consider ingredients of a service specification along with the informal and formal presentations of the dependencies between the rules, i.e., the semantic compatibility and syntactic matching relations. In the last section, we conclude with a short summary and list the open problems in our approach.

2 Preliminaries

In this section, we review some of the main concepts of the *double-pushout* (DPO) [4] approach (see [3] for a survey) and the *double-pullback* (DPB) approach [6] to graph transformation. Also, we provide several examples being reused in Sec-

tion 3 to demonstrate the matching of service specifications. We start with the brief discussion of the DPO approach.

2.1 The Double-Pushout Approach to Graph Transformation

Given a graph TG, called *type graph*, a *TG-typed (instance) graph* consists of a graph G together with a typing homomorphism $g : G \to TG$ (cf. Fig. 1 on the left) associating with each vertex and edge x of G its type $g(x) = t$ in TG. In this case, we also write $x : t \in G$. A TG-typed graph morphism between two TG-typed instance graphs $\langle G, g\rangle$ and $\langle H, h\rangle$ is a graph morphism $f : G \to H$ which preserves types, that is, $h \circ f = g$.

The DPO approach to graph transformation has originally been developed for vertex- and edge-labelled graphs [4]. Here, we present the typed version [2].

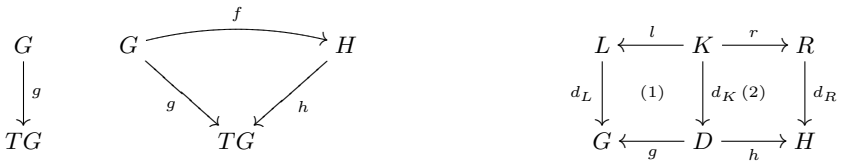

Fig. 1. Typed graph and graph morphism (left) and double-pushout diagram (right).

Example 1. UML class diagram in Fig. 2 on the left represents the domain data model of a Web Service for booking a hotel room that plays a part of a sample scenario throughout out presentation. A customer (class Customer) intends to book a room (class Room) in a hotel (class Hotel). A booking information (class BookingInfo) and possibly a business license code (class LicenseInfo) of a customer (e.g. travel bureau) are required to be provided to make a reservation. A result of the booking process is represented by an acknowledgment in the form of a reservation tag (class ReservTag) and/or a reservation document (class Reservation) containing all details of the reservation.

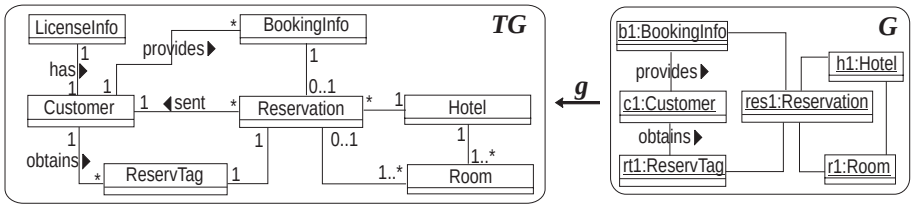

Fig. 2. Type graph representing the data model of sample scenario (left) and instance graph (right).

In the context of graph transformation, a class diagram is considered as a directed graph, whose vertices contain type declarations. Their relation with object diagrams (cf. Fig. 2 on the right) representing run-time states is expressed by the notion of a *type graph* (TG) and corresponding *instance graphs* [2].

According to the DPO approach, graph transformation rules, also called graph productions, are specified by pairs of injective graph morphisms ($L \xleftarrow{l} K \xrightarrow{r} R$), called rule spans, between TG-typed instance graphs L, K and R. The *left-hand side* L contains the items that must be present for an application of the rule, the *right-hand side* R those that are present afterwards, and the *interface graph* K specifies the "gluing items", i.e., the objects which are read during application, but are not consumed.

Definition 1. (rule, graph transformation system) *A* rule span typed over TG, *in short TG-typed rule span,* $s = (L \xleftarrow{l} K \xrightarrow{r} R)$ *is a span of injective TG-typed graph morphisms.*

A graph transformation system $GTS = \langle TG, P, \pi \rangle$ *consists of a* type graph TG, *a set of rule names* P, *and a mapping* π *associating with each rule name* p *a TG-typed rule span* $\pi(p)$. *If* $p \in P$ *is a rule name and* $\pi(p) = s$, *we say that* $p : s$ is a rule of GTS.

The upper part of Fig. 3 demonstrates the span representation of the rule bookHotel().

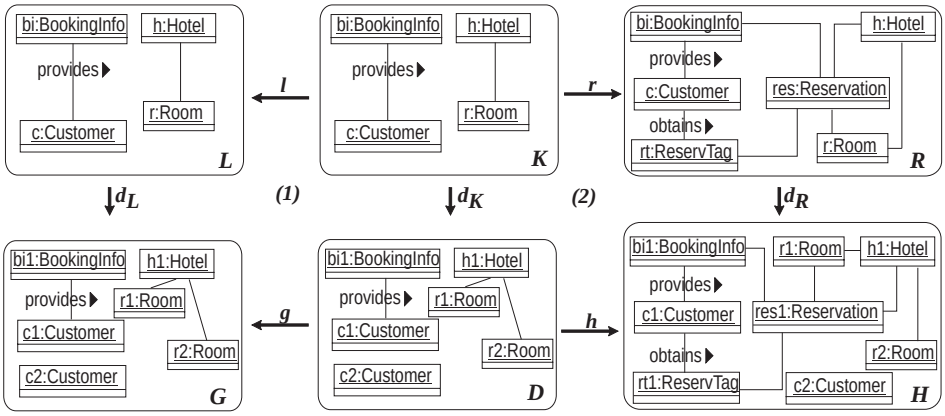

Fig. 3. DPO graph transformation step using rule bookHotel().

The transformation of graphs is defined by a pair of pushout diagrams, a so-called double-pushout construction.

Definition 2. (DPO graph transformation) *A* double-pushout (DPO) diagram d *is a diagram as in Fig. 1 on the right, where (1) and (2) are pushouts.*

Given a type graph TG and a rule $p : s$ *with* $s = (L \xleftarrow{l} K \xrightarrow{r} R)$ *the corresponding* (DPO) transformation step *from* G *to* H *is denoted by* $G \overset{p/d}{\Longrightarrow} H$*, or simply* $G \overset{p}{\Longrightarrow} H$ *if the diagram* d *is understood.*

An example of a transformation step via the rule bookHotel() is given in Fig. 3.

Operationally speaking, the application of the rule proceeds as follows. Given the occurrence d_L of the left-hand-side L in G, the application consists of two steps: The elements of G matched by $L \setminus l(K)$ are removed, that does not change graph G in Fig. 3. Then, the elements matched by $R \setminus r(K)$ are added to D which leads to the derived graph H additionally containing the vertecies res1, rt1 and the corresponding edges.

Gluing the graphs L and D over their common part K yields again the given graph G, i.e., D is a so-called *pushout complement* and the left-hand square (1) is a pushout square. Only in this case the application is permitted. Similarly, the derived graph H is the gluing of D and R over K, which forms the right-hand side pushout square (2).

This formalization implies that only vertices that are preserved can be merged or connected to edges in the context. It is reflected in the *identification* and the *dangling conditions* of the DPO approach which characterize, given a rule $p : s = (L \xleftarrow{l} K \xrightarrow{r} R)$ and an occurrence $d_L : L \to G$ of the left-hand side, the existence of the pushout complement (1), and hence of a transformation step $G \overset{p/d}{\Longrightarrow} H$. The *identification condition* states that objects from the left-hand side may only be identified by the match if they also belong to the interface (and are thus preserved). The *dangling condition* ensures that the structure D obtained by removing from G all objects that are to be deleted is indeed a graph, that is, no edges are left "dangling" without source or target node.

2.2 The Double-Pullback Approach to Graph Transformation

The DPO approach guarantees that the changes to the given graph H are exactly those specified by the rule. However, operation contracts represent specifications of operations that are, in general, incomplete, that is, additional effects should be allowed in the transformation. Therefore, a more liberal notion of rule application is required which ensures that *at least* the elements of G matched by $L \setminus l(K)$ are removed, and *at least* the elements matched by $R \setminus r(K)$ are added. This kind of the rule interpretation is supported by the double-pullback (DPB) approach to graph transformation [6].

Graph transitions have been proposed to provide a looser interpretation of graph transformation rules. The DPB approach introduces graph transitions and generalizes DPO by allowing additional changes, not encoded in the rule. Graph transitions are defined by replacing the double-pushout diagram of a transformation step with a double-*pullback*.

Definition 3. (graph transition) *Let* $p : s$ *be a rule span with* $s = (L \xleftarrow{l} K \xrightarrow{r} R)$*. Then, a* graph transition *from* G *to* H *via* p*, denoted by* $G \overset{p/d}{\rightsquigarrow} H$*,*

is a diagram like the right part of Fig. 4 where both (1) and (2) are pullback squares. A graph transition (or briefly transition) is called injective *if both g and h are injective graph morphisms. It is called* faithful *if it is injective, and the morphisms d_L and d_R satisfy the following condition: for all $x, y \in L$, $y \notin l(K)$ implies $d_L(x) \neq d_L(y)$, and analogously for d_R* [1].

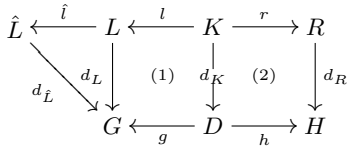

Fig. 4. DPB graph transition and positive application condition.

Notice that any pushout square of two given morphisms such that one of them is injective is also a pullback square. Thus, every DPO transformation is also a DPB transition. Each faithful transition can be regarded as a transformation step plus a change-of-context [6]. This is modeled by extra deletion and creation of elements before and after the actual step.

A rule under the DPB interpretation may also contain, so-called "don't care" elements. They must be present before the rule application, but the specification of their behavior is beyond the scope of the rule. The "don't care" elements compose an extra restriction on the using of the rule called a positive application condition. A *graph transformation rule with positive application condition* $\hat{p}$ also contains the graph $\hat{L}$ specifying extensions of L by such elements. $\hat{L}$ represents a pattern for positive application condition.

Definition 4. (rule with positive application condition) *A* graph transformation rule with positive application condition $\hat{p}$ *is a pair* $(p, \hat{L})$*, where* $p : s$ *is a graph transformation rule with* $s = (L \xleftarrow{l} K \xrightarrow{r} R)$ *and* $\hat{L}$ *is a TG-typed graph, such that* L *is a subgraph of* $\hat{L}$ *and* $\hat{l} : L \to \hat{L}$ *is the corresponding inclusion (cf. Fig. 4 on the left).*

Example 2. Fig. 5 shows two graph transformation rules with positive application conditions representing a booking procedure from the viewpoint of the requestor (upper rule) and provider (lower rule). The interface graphs of these rules are omitted in the figure. $\hat{L}_p$ and $\hat{L}_r$ specify the patterns needed for the rule application and contain information about a customer (verticies c and cus), booking details (verticies bi), etc. While the graphs $\hat{L}_p$ and L_p in the provider rule are identical, the graph $\hat{L}_r$ in the requestor rule additionally holds the vertex

[1] The last condition means that d_L and d_R satisfy the identification condition of the DPO approach [3] with respect to l and r.

l:LicenseInfo denoting a business license code of a customer. This vertex, being typical example of the "don't care" element, is required to be present, but does not participate in the following transformations. The reservation of a room is shown by newly created verticies res, rs, rt and the corresponding edges between them in the right-hand sides of the rules. The extra association sent in the lower rule reflects the fact that the reservation document is sent to the customer by default that is not assumed in the upper rule.

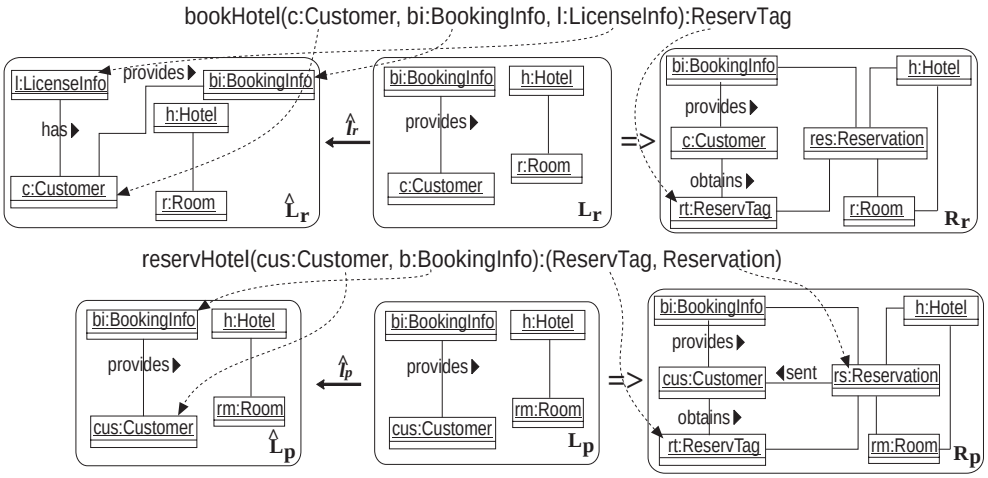

Fig. 5. Graph transformation rules for provided operation bookHotel() and required operation reservHotel().

Definition 5. (graph transition via rule with positive application condition) *Let $\hat{p} = (p, \hat{L})$ be a graph transformation rule with positive application condition, where $s = (L \xleftarrow{l} K \xrightarrow{r} R)$. A* graph transition *from G to H via the rule $\hat{p}$, denoted by $G \overset{\hat{p}/d}{\rightsquigarrow} H$, is a graph transition via a rule p, such that there exists $d_{\hat{L}}$ satisfying $d_L = d_{\hat{L}} \circ \hat{l}$ (cf. Fig. 4 on the left).*

Faithful transitions capture our intuition about a loose interpretation of contracts which can be specified by graph transformation rules with positive application conditions.

The purpose of the next section is to present a formal approach to service specification matching based on graph transformation.

3 Service Specification Matching

This section discusses constituents of a service specification and addresses service specification matching basically at the level of behavioral descriptions. The

semantic compatibility and syntactic matching relations determine a foundation for the matching procedure. The formal definition of these relations comes out of the ideas obtained in the sample scenario of a hotel booking Web Service which will be also considered.

3.1 Service Specification

First of all, let us introduce the basic ingredients of the service specifications. We start with the data model of the application expressed by the UML class diagram in Fig. 2. This data model has been already discussed in Section 2.1. To avoid additional complications, we assume that service requestor and provider are working with the same data model, agreed upon in advance.

According to [8], a Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging. The next part of the service specification is represented by an interface. An example of the provided and required interfaces is given in Fig. 6.

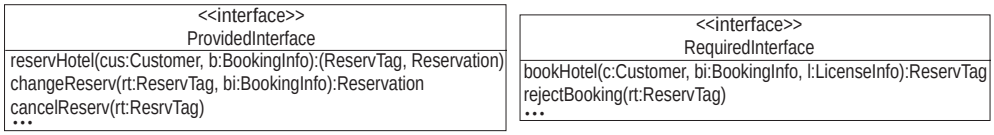

Fig. 6. Provided and required interfaces.

An interface contains structural information about operations. The behavior of these operations can be specified by contracts. The concept of *contracts* is widely used to describe behavior of Web services and their constituents (see [5]). A contract consists of a pre-condition specifying the system state before some behavior is executed and a post-condition describing the system state after the execution of the behavior. There are different approaches employing formal techniques (e.g., description logic [9], algebraic specification languages [10], etc.) to contract specification. The main obstacle of these approaches is their lack of usability in the software industry, where knowledge and skills in the application of logic formalisms are scarce. A notation that is close to the standard software modeling languages (e.g., UML) and has, at the same time, a formal background allowing to provide automation is given by *typed graph transformation* [2]. Graph transformation rules with positive application conditions can be taken as a visual formal notation for the contract specification.

The positive application condition $\hat{L}$ represents the pre-condition. The left-hand side L and the the right-hand side R describe the post-condition and effect. Fig. 5 demonstrates the graph transformation rules for the required operation bookHotel() and the provided operation reservHotel(). Among other things, $\hat{L}_r$ and $\hat{L}_p$ contain elements stating for the input parameters of these operations, i.e., information about a customer (verticies c and cus), booking details (verticies

bi), and a business license code (vertex l:LicenseInfo). Notice that the graph transformation rules with application conditions enable to clearly distinguish a pre-condition of an operation from its post-condition and effect. Besides, one can describe the input parameters of the operations which can be submitted by the requestor, but are not necessarily affected by the further computations (e.g., LicenseInfo).

To summarize, a service specification consists of a data model, structural (operation signatures) and behavioral (operation contracts represented by graph transformation rules) specifications of operations constituting a service. In the next section we discuss service specification matching and consider an example of matching required and provided operation contracts.

3.2 Specification Matching

In general, specification matching has to deal with all three aspects of a specification, i.e., data, signatures, and contracts. For simplicity, we ignore the matching of data models and discuss the matching of signatures only briefly (see [10] for a general discussion).

As an example, we consider the relation between the required operation bookHotel() and the provided operation reservHotel() whose signatures and contracts are depicted in Fig. 5.

The signatures of the operations have several distinctions. The required operation has the extra input parameter l:LicenseInfo, while the provided operation contains an extra output parameter of the type Reservation. This does not violate compatibility, because the input of the requestor, which is not required, may simply be ignored by the provider. Similarly, the output of the provided, which is not expected by the requestor, may be skipped.

To determine the relation between signatures and contracts, we require that input and output parameters of each operation are represented by vertices with corresponding types in the rules. These dependencies are indicated by the dashed arrows in Fig. 5.

Now we consider behavioral compatibility which amounts to check compatibility of pre-conditions ($\hat{L}_r$ and $\hat{L}_p$) and effects ($L_r => R_r$ and $L_p => R_p$). In order to perform an operation successfully, the provider requires certain input data from the requestor as well as a certain properties to hold in the current state. In the provider rule of Fig. 5 the input data is given by information about a customer and booking details. The requestor has to be prepared to deliver this data and to guarantee these properties. Hence, the pre-condition of the requestor must entail the pre-condition of the provider, which is expressed by an occurrence (formally a graph homomorphism) from $\hat{L}_p$ to $\hat{L}_r$.

A requestor wants to have some benefit form the invocation of a service operation. If an operation does less then expected by a requestor, it is not considered to be useful. In other words, the effect of the provided operation must not be less than the effect specified by the requestor. That means, the requestor rule must be embedded in the provider as it is the case with the rules in Fig. 5. For

example, the operation reservHotel() additionally creates the edge sent that denotes default delivering a reservation document to the customer. This vertex is not presented in the requestor contract, because it is sufficient for him to obtain only a reservation tag. Nevertheless, the effect of the provided operation fits the client requirements.

Next, we will present a (partial) formalization of the intuitive ideas obtained from the example.

3.3 Towards Formalization

The concept of transition allows us to formalize semantically the desired notion of compatibility: Provider and requestor rules are semantically compatible if (1) every transition via the provider rule can be regarded as a transition via the requestor rule, and (2) applicability of the requestor rule implies applicability of the provider rule.

Definition 6. (semantic compatibility) *Let $\hat{p}_1 = (p_1, \hat{L}_1)$ and $\hat{p}_2 = (p_2, \hat{L}_2)$ be graph transformation rules with positive application conditions, where $s_1 = (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ and $s_2 = (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$. We say that $\hat{p}_1$, is semantically compatible with $\hat{p}_2$, in symbols $\hat{p}_2 \models \hat{p}_1$, iff*

1. *for all spans $t = (G \xleftarrow{g} D \xrightarrow{h} H)$ and transitions $G \overset{p_2/d_2}{\rightsquigarrow} H$, there exists a transition $G \overset{p_1/d_1}{\rightsquigarrow} H$ using the same bottom span t , where $d_1 = (d_{L_1}, d_{K_1}, d_{R_1})$ and $d_2 = (d_{L_2}, d_{K_2}, d_{R_2})$ (cf. Fig. 7 on the right), and*
2. *if there exists $d_{\hat{L}_1} : \hat{L}_1 \to G$ such that $d_{L_1} := d_{\hat{L}_1} \circ \hat{l}_1$ satisfies the identification condition of p_1, then there exists $d_{\hat{L}_2} : \hat{L}_2 \to G$ such that $d_{L_2} := d_{\hat{L}_2} \circ \hat{l}_2$ satisfies the identification condition of p_2.*

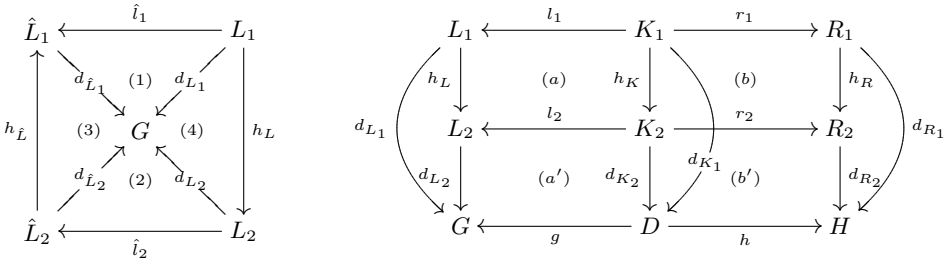

Fig. 7. Matching graph transformation rules.

This definition reflects the desired relation between contracts, but can hardly be applied for an algorithm determining contract compatibility. Therefore, we introduce a relation of *syntactic matching* which encompasses ideas presented in Section 3.2 and has more constructive character.

Definition 7. (syntactic matching) *Let* $\hat{p}_1 = (p_1, \hat{L}_1)$ *and* $\hat{p}_2 = (p_2, \hat{L}_2)$ *be graph transformation rules with positive application conditions, where* $s_1 = (L_1 \xleftarrow{l_1} K_1 \xrightarrow{r_1} R_1)$ *and* $s_2 = (L_2 \xleftarrow{l_2} K_2 \xrightarrow{r_2} R_2)$*. We say that* $\hat{p}_1$ *syntactically matches with* $\hat{p}_2$*, in symbols* $\hat{p}_2 \vdash \hat{p}_1$*, iff*

1. *there exist graph homomorphisms* $h_L : L_1 \to L_2$*,* $h_K : K_1 \to K_2$*, and* $h_R : R_1 \to R_2$ *such that the diagrams (a) and (b) in Fig. 7 on the right represent a faithful transition, and*
2. *there exists an injective graph homomorphism* $h_{\hat{L}} : \hat{L}_2 \to \hat{L}_1$ *such that* $h_{\hat{L}} \circ \hat{l}_2$ *satisfies the identification condition of* p_2 *and the outer square in Fig. 7 on the left commutes.*

An example of syntactic matching is given in Section 3.2 for the graph transformation rules specifying the contracts of the required operation bookHotel() and the provided operation reservHotel().

Finally, we demonstrate the soundness of our approach.

Theorem 1. (soundness of matching) *Assume two graph transformation rules with positive application conditions* $\hat{p}_1$ *and* $\hat{p}_2$*. Then* $\hat{p}_2 \vdash \hat{p}_1$ *implies* $\hat{p}_2 \models \hat{p}_1$*.*

Proof. (Sketch) We show that Def. 7 entails Def. 6.1/2, respectively.

1. It is necessary to demonstrate that for each faithful transition via the second rule there is a faithful transition via the first rule. By assumption, there exist graph homomorphisms between the first and the second rules (h_L, h_K, h_R), forming a faithful transition (cf. Fig. 7 on the right). Now, both transitions can be vertically composed using the composition of the underlying pullback squares, and faithfulness of the composed transition follows from the fact that the identification condition of d_{L_1} follows from that of h_L and d_{L_2}, and analogously for the right-hand side.
2. Given $d_{\hat{L}_1} : \hat{L}_1 \to G$, we obtain $d_{\hat{L}_2} : \hat{L}_2 \to G$ by $d_{\hat{L}_1} \circ h_{\hat{L}}$ resulting in the commutativity of diagram (3). Morphism $d_{L_2} = d_{\hat{L}_2} \circ \hat{l}_2$ satisfies the identification condition of p_2 because of this commutativity and the fact that $h_{\hat{L}} \circ \hat{l}_2$ satisfies the identification condition of p_2. Moreover, we can show that the remaining diagrams in Fig. 7 on the left commute, thus relating the compatibility conditions for pre-condition and effect.

Completeness of syntactic matching requires a more refined relation at the semantic level, establishing a connection between statements (1) and (2), that we have not yet fully worked out. The final section summarizes the main results and discusses more open problems.

4 Conclusion and Future Works

The paper has outlined basic ideas standing behind a UML-based approach to service specification matching. This approach employs graph transformation

rules with positive application conditions as a visual formal notation for modeling operation contracts. A loose interpretation of the rules based on DPB graph transitions has been used to obtain an operational understanding of contracts and a corresponding semantic compatibility relation. Since this relation is defined over the infinite set of transitions and, therefore, can not be directly computed, we have established a syntactic relation providing sufficient conditions for the semantic one.

In the future, we intend to refine the semantic relation adding constraints on the compatibility between pre-conditions and effects, that will guarantee the completeness of our approach. Also, it is desirable to extended the existing formalization to typed graphs with attributes [7] and sub-typing [1].

The practical application of the theoretical concepts presented in our work is stipulated by finding an adequate XML-representation of contracts, and tool support for computing the syntactic matching relation.

References

1. R. Bardohl, H. Ehrig, J. de Lara, O. Runge, G. Taentzer, and I. Weinhold. Node type inheritance concepts for typed graph transformation. Fachberichte Informatik 2003-19, Technical University Berlin, 2003.
2. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.
3. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation, Part I: Basic concepts and double pushout approach. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*, pages 163–245. World Scientific, 1997. Preprint available as Tech. Rep. 96/17, Univ. of Pisa, `http://www.di.unipi.it/TR/TRengl.html`.
4. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
5. D. Fensel and C. Bussler. The web service modeling framework. 2003.
6. R. Heckel, H. Ehrig, U. Wolter, and A. Corradini. Double-pullback transitions and coalgebraic loose semantics for graph transformation systems. *Applied Categorical Structures*, 9(1), January 2001. See also TR 97-07 at `http://www.cs.tu-berlin.de/cs/ifb/TechnBerichteListe.html`.
7. R. Heckel, J. Küster, and G. Taentzer. Confluence of typed attributed graph transformation systems. In A. Corradini and H.-J. Kreowski, editors, *Proc. 1st Int. Conference on Graph Transformation (ICGT 02), Barcelona, Spain*, Lecture Notes in Comp. Science. Springer-Verlag, October 2002.
8. H. Kreger. Web services conceptual architecture (WSCA 1.0), May 2001. `http://www-3.ibm.com/software/solutions/webservices/pdf/WSCA.pdf`.
9. C. Pahl. An ontology for software component matching. In M. Pezze, editor, *Fundamental approaches to software engineering: 6th international conference, FASE 2003*, volume 2621 of *LNCS*, pages 6–21. Springer, 2003.
10. A.M. Zaremski. *Signature and specification matching*. PhD thesis, Carnegie Mellon University, Pittsburg, Pa., January 1996.