Incremental Service Composition based on Partial Matching of Visual Contracts

Muhammad Naeem[1], Reiko Heckel[1]*, Fernando Orejas[2]**, and Frank Hermann[3]

[1] Department of Computer Science, University of Leicester, United Kingdom
mn105|reiko@mcs.le.ac.uk
[2] Departament de L.S.I., Universitat Politècnica de Catalunya, Barcelona, Spain
orejas@lsi.upc.edu
[3] Fakultät IV, Technische Universität Berlin, Berlin, Germany
frank.hermann@tu-berlin.de

Abstract. Services provide access to software components that can be discovered dynamically via the Internet. The increasing number of services a requesters may be able to use demand support for finding and selecting services. In particular, it is unrealistic to expect that a single service will satisfy complex requirements, so services will have to be combined to match clients' requests.
In this paper, we propose a visual, incremental approach for the composition of services, in which we describe the requirements of a requester as a *goal* which is matched against multiple *provider offers*. After every match with an *offer* we decompose the *goal* into satisfied and remainder parts. We iterate the decomposition until the *goal* is satisfied or we run out of *offers*, leading to a resolution-like matching strategy. Finally, the *individual offers* can be composed into a single *combined offer* and shown to the requester for feedback.
Our approach is based on visual specifications of pre- and postconditions by graph transformation systems with loose semantics, where a symbolic approach based on constraints is used to represent attributes and their computation in graphs.

1 Introduction

Service-oriented Architecture (SOA)[1] supports dynamic discovery and binding based on matching requesters' requirements with providers' offers. Both requirements and offers can be expressed as specifications of the (expected or given) semantics of a service's operations in terms of their pre- and postconditions. At a *technical level* this is supported by semantic web technologies (e.g. OWL-S [2], WSML [3]), at *modeling level* visual contracts have been suggested to describe service semantics [4].

* Partially supported by the project Sensoria IST-2005-016004
** Partially supported by the MEC project (ref. TIN2007-66523), the MEC grant PR2008-0185, the EPSRC grant EP/H001417/1 and a Royal Society travel grant

However, expecting to find a single service for each requirement is unrealistic. Often services need to be combined to satisfy the demands of clients. For example, let us consider a scenario, where a requester is looking to book a trip for attending a conference. The requester may be interested in flight and hotel reservation. Rather than using a single service, the requester may have to use two separate service providers.

In this paper we propose an incremental approach for service composition, where we assume that the requirements are expressed by a *single goal* stating pre- and postconditions. A variety of *offers* could contribute to the *goal*, each described by pre- and postconditions as well. We propose a notion of partial matching of *offers* with *goal*. After every partial match we compute the remaining requirements by decomposition of the *original goal* into the *satisfied subgoal* and its *remainder*. We iterate this process until the goal is achieved or we do not have any more *offers*.

As a result of this procedure we produce a *combined offer* which can be visualized and reviewed by the client. Our approach thus supports *incremental matching procedure* which is based on partial matching of visual contracts.

The rest of the paper is organized as follows. Section 2 discusses related approaches. Section 3 introduces the basic framework for specification and matching of services. Section 4 presents our approach towards incremental composition based on the construction of the remainder rule and generation of composed operation from individual ones, and Section 5 discusses the main results and concludes the paper.

2 Related Work

As motivated above, we work on the assumption that it is unrealistic to expect a single offer to be sufficient to satisfy a goal, i.e., several offers will have to be combined. This raises the first three of the following questions to serve as criteria for approaches to dynamic service composition. The fourth question derives from the desired integration into mainstream modelling techniques such as the UML, which use diagrammatic languages to specify software. For service specification to be integrated into standard software engineering processes, they have to use compatible visual notations.

- **Partial Match:** Does the approach support partial matching of an offer with a goal, or is full satisfaction of all requirements necessary for each match?
- **Flexibility:** Does the approach allow to match offers in flexible order or does it follow a given control flow?
- **Completeness:** Is the approach decidable, i.e., does it provide a complete and terminating procedure to find out if there are combinations of offers satisfying a goal?
- **Visualisation:** Does the approach provide a visual language for service specification and feedback on the result of the matching?

Table 1. How existing approaches realize the proposed requirements

Approach of:	Language	Partial Match	Flexibility	Completeness	Feedback
[7]	OWL, DAML-S	✓	×	×	✓
[8]	DAML-S, SHOP-2	✓	✓	✓	×
[9]	FOL	✓	✓	×	×
[10]	WSML	✓	✓	✓	×
Our Approach	Graph Theory, GTS	✓	✓	✓	✓

While there are many approaches to composition of services, we limit our discussion to semantics-based approaches using pre- and postconditions, disregarding process or workflow-based orchestration, see [5, 6] for a more complete picture. We summarise the results of our analysis in Table 1.

In [7] the authors propose a semi-automatic approach for filtering and composition of services using OWL and DAML-S. An inference engine performs an exact match with available services and shows the resulting list to the user who selects the ones to be composed. The approach is highly dependent on user input and so avoids the need for a decidable composition procedure required for automation. In contrast, we would require feedback on the end result of the automated composition only.

In a number of works, AI planning models are used to construct process models from goals and operations described by pre- and postconditions. For example, [8] is based on DAML-S. Their approach is decidable for a finite number of services / operations. Partial matching is possible based on the semantic description. The main difference with our approach is the use of logic-based (rather than visual) descriptions, which makes it difficult to provide feedback on the result of the composition to domain and business experts.

The work in [9] is representative of approaches based on first-order specification of goals and services. It allows partial matching and flexibility in ordering in addition to *goal templates* which abstract from the actual input parameters for invoking services and can thus be matched at design time. Our goals are at the level of goal templates in [9] in that they are generic with regard to the actual parameters.

Approaches such as [10] use semantic service web markup languages such as WSML-MX that are both specialised for the task of service description and matching and limited in expressiveness to guarantee computability.

3 Graphical Service Specification and Matching

Following [4, 11], in this section we review the basic notions of service specification and matching used in the rest of the paper.

Visual Service Specification. According to [12], a web service describes a collection of operations that are network accessible, each specified by a pre- and a postcondition. As usual, a precondition denotes the set of states where that operation is applicable and the postcondition describes how any state satisfying the given precondition is changed by the operation. In our case the states can be

seen as typed attributed graphs. This means graphs that may include values (the attributes) in their nodes or edges, all typed by a fixed type graph. For instance, Fig. 1 describes the type graph of the running example of a travel agency that we will use along the paper and Fig. 3 is an example of an attributed (instance) graph typed over that type graph after booking a flight.

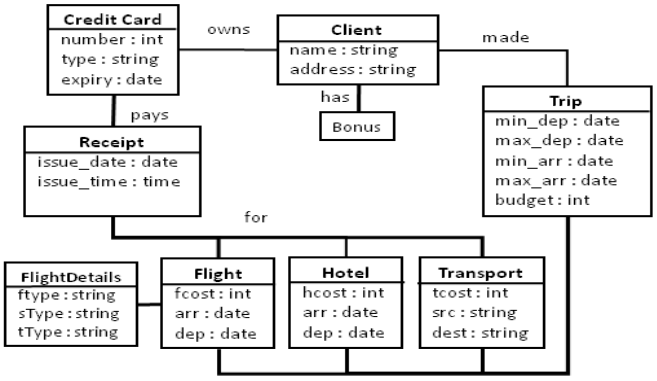

Fig. 1. Type Graph

In [4, 11] the specification of an operation Op, denoted $Op : PRE \Rightarrow POST$ is given by typed attributed graphs, PRE and $POST$ related by an injective partial graph morphism represented as a pair of injective morphisms, $pre : COM \to PRE$ and $post : COM \to POST$. Here COM provides the intersection of PRE and $POST$, and pre, $post$ are the corresponding injections into PRE and $POST$. Usually, attributed graphs PRE, $POST$ and COM include variables, values or complex expressions as attributes. However, in this paper attributes will be restricted to variables and basic values only, ruling out complex expressions. These will be captured, together with other constraints, by a formula α which constrains the possible values of the variables occurring as attributes in a graph G and we call (G, α) a constrained graph. In our case the graphs of an operation have a common condition α relating the variables occurring in PRE and $POST$. Hence, an operation specification is denoted by a pair $\langle PRE \Rightarrow POST, \alpha \rangle$ (or $\langle PRE \overset{pre}{\leftarrow} COM \overset{post}{\rightarrow} POST, \alpha \rangle$ if we are interested in the intersection COM). For instance, Fig. 2 describes an operation $BookFlight$ for booking a flight with a travel agency service.

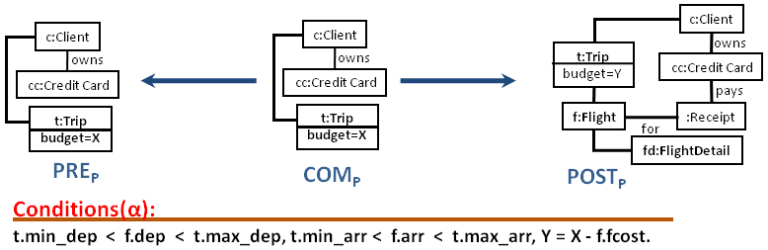

Fig. 2. *BookFlight* Operation for Travel Agency (Provider)

In the example, graphs PRE_P and $POST_P$ are the pre- and postconditions for operation $BookFlight$ whereas graph COM_P represents the intersection of PRE_P and $POST_P$. Condition α constrains the operation to be flexible enough to choose the departure date $f.dep$ within the range of $t.dep\_min$ and $t.dep\_max$ as well as the arrival date $f.arr$ within $t.arr\_min$ and $t.arr\_max$. This way of dealing with attributed graphs, introduced in [**?**], has proved essential in order to support the declarative (as opposed to computational) description of attribute operations appropriate to the use of graphs as pre- and postconditions (rather than rewrite rules only). In particular, we allow attributed graphs representing states to include variables and conditions on attributes, rather than just values, thus providing a symbolic representation where not all the attributes are fully evaluated. We refrain from the use of application conditions [13] in the specification of operations to keep the presentation simple, but believe that they would not add an essential difficulty.

The semantics of the operation $\langle PRE \overset{pre}{\leftarrow} COM \overset{post}{\rightarrow} POST, \alpha \rangle$ is described using graph transformation. More precisely, given an attributed graph $\langle G, \alpha' \rangle$ representing the current state and given a matching morphism $m : PRE \rightarrow G$ such that α' implies $m(\alpha)$, with $m(\alpha)$ the formula obtained by replacing each variable X in α by its image $m(X)$, the result will be the attributed graph $\langle H, \alpha' \rangle$ with H defined by the following double pushout:

$$\begin{array}{ccccc} PRE & \prec^{pre}\!\!- & COM & -^{post}\!\!\succ & POST \\ \downarrow m & po & \downarrow & po & \downarrow \\ G & \longleftarrow & D & \longrightarrow & H \end{array}$$

Intuitively, graph D is obtained by deleting from G all the elements (nodes, edges, or attributes) which are matched by an element that is present in PRE but not in COM. Then, the graph H is obtained adding to D all the elements which are present in $POST$, but not in COM. For instance, if we apply the operation $BookFlight$ to graph G in the left of Fig. 3, the result will be the graph H in the right of Fig. 3. Notice that, for readability, these graphs include some values, such as 200, instead of a variable X and the equality $X = 200$ included in the associated formula.

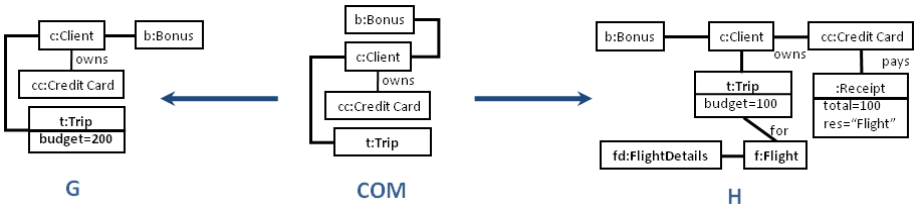

Fig. 3. Transformation due to $BookFlight$ of Travel Agency

A requester looking for a service must specify the operations they want to use. Requester specifications have the same form as provider specifications. They are seen as inquiries that the requester is making, with the aim of entering into

a contract with the provider. In particular, PRE would denote the data and resources that the requester would accept to provide and $POST$ would describe the expected result of the operation. In this use case, the semantics of these specifications is a different one because the requester may not (need to) know all the details of the provider state and cannot thus describe completely all the changes caused by the operation. Such a semantics has been studied for graph transformations in terms of double pullbacks [14]. More precisely, given an attributed graph $\langle G, \alpha' \rangle$ and a matching morphisms $m : PRE \to G$ such that α' implies $m(\alpha)$, a graph H is the result of a double pullback transition if we can build a double pullback diagram of the same shape like the double pushout above, but replacing the *po* by *pb* squares.

Intuitively, in a double pullback transition, the rule $PRE \Rightarrow POST$ describes a lower bound to the effects the operation should cause when applied to G. This means, if an element a is present in PRE but not in COM, then $m(a)$ must be deleted from G, and if a is present in $POST$, but not in COM, then it must be added to G. And if a is present in COM, then it must remain unchanged. However, G may suffer other changes not specified by the operation. For instance, if the specification of the operation *BookFlight* in Fig. 2 would be part of a requester specification, then applying that operation to the graph G in Fig. 3 could yield the graph H in Fig. 3 as a result of a double pullback transition.

Matching Visual Contracts. In order to match the requirements for an operation $Op_R = \langle PRE_R \Rightarrow POST_R, \alpha_R \rangle$ of a requestor against a description $Op_P = \langle PRE_P \Rightarrow POST_P, \alpha_P \rangle$ supplied by a provider, we have to guarantee that all effects required by Op_R are implemented by Op_P. More precisely, assuming that $\langle G, \alpha \rangle$ represents the given state, the following conditions must be satisfied:

a) Whenever a transition for Op_R can take place, a corresponding transformation associated to Op_P must be possible.
b) If Op_R prescribes that some element must be deleted from the current state, that element must also be deleted by Op_P.
c) If Op_R prescribes that some element must be added to the current state, that element must also be added by Op_P.
d) If Op_R prescribes that some element remain in the current state, that element should be part of COM_P.

Technically, this means to ask for the existence of three injective morphisms $h_{PRE} : PRE_P \to PRE_R$, $h_{COM} : COM_R \to COM_P$, and $h_{POST} : POST_R \to POST_P$, such that α_R implies $h_{PRE}(\alpha_P)$, h_{PRE} and pre_R are jointly surjective[4], diagram (1) commutes, and diagram (2) is a pullback.

$$\begin{array}{ccccc} PRE_R & \xleftarrow{pre_R} & COM_R & \xrightarrow{post_R} & POST_R \\ \uparrow{\scriptstyle h_{PRE}} & (1) & \downarrow{\scriptstyle h_{COM}} & (2) & \downarrow{\scriptstyle h_{POST}} \\ PRE_P & \xleftarrow{pre_P} & COM_P & \xrightarrow{post_P} & POST_P \end{array}$$

[4] This means that every element in PRE_R is the image of an element in COM_R or of an element in PRE_P

In particular, given $\langle G, \alpha \rangle$, the existence of $h_{PRE} : PRE_P \to PRE_R$ such that α_R implies $h_{PRE}(\alpha_P)$ ensures that if there is a match $m : PRE_R \to G$ such that α implies $m(\alpha_R)$ then we also have a corresponding match $h_{PRE} \circ m : PRE_P \to G$ such that α implies $h_{PRE}(m(\alpha_P))$. In addition if an element is in PRE_R but not in COM_R then that element should also be in PRE_P, because h_{PRE} and pre_R are jointly surjective, but not in COM_P since diagram (1) commutes. This means that according to both rules, Op_R and Op_P, that element must also be deleted. If an element is in $POST_R$ but not in COM_R then that element should also be in $POST_P$ but not in COM_P because diagram (2) is a pullback. Finally, if an element is in COM_R its image through h_{COM} would also be in COM_P.

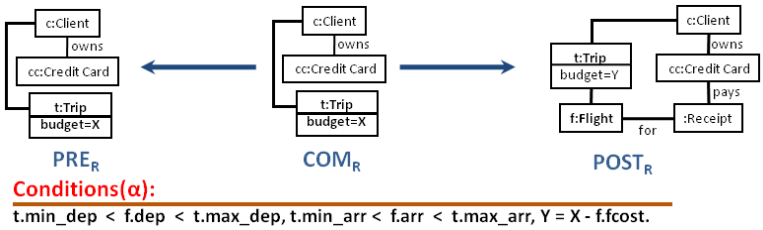

Fig. 4. *BookFlight* Operation of Requester

For instance, in our example the specification in Fig. 2 matches the specification in Fig. 4. All the elements in PRE_P, COM_R, and $POST_R$ have an image in PRE_R, COM_P, and $POST_P$ respectively. So there exist three injective morphisms [15, 13] between PRE_P and PRE_R, COM_P and COM_R, and $POST_R$ and $POST_P$.

We have discussed how to match a request with a service offered by a provider in an *ideal* situation, where the granularity of requirements and offered services coincide and the matching is complete. However, such a lucky outcome is unlikely in practice.

On one hand, as seen above, a service precondition must describe the data and resources that may be needed to run that service (and, in addition, through the associated condition α_R, it may also describe the conditions under which a given service is considered to be acceptable, e.g. its cost). This means to assume that the requester knows, a priori, all the data and resources that may be required to satisfy his needs. This may be unrealistic in many cases. For instance, when booking a trip, the requester may describe in its precondition some basic data, like their name, date and destination of the travel, credit card or bank account number, etc. In addition, the requester may specify an overall budget for the travel. However the provider may also need to know the age of the traveller, to see if some discount applies, or whether the requester has a discount bonus that would be consumed when using the service.

On the other hand, the postcondition describes the effect of using a given service. In this sense, the requester will describe everything they expect to get when binding to a certain service. However, there may be two problems here. On one hand, there may not be a single provider that offers a service covering all the requester needs. For instance, the requester for a travel may want, not only to book a flight and a hotel, but also to get tickets for a play and to have a dinner in a well-known restaurant. Then, there may be no travel agency that can take care of all these activities. On the other hand, the specification level of the requester and the provider may have different granularity. In particular, a requester may describe as a single operation booking a flight and a hotel room, while a given provider, in his specifications, may consider these two bookings as independent operations. Then, matching this request would mean for that provider finding an appropriate combination of the two operations that satisfies the customer needs.

So, we believe there is a need for matching a request with multiple offers. In the next section we discuss such an incremental procedure for the composition of services, where we will discuss the partial match of single *requester operation* with multiple *provider offers*.

4 Incremental Service Composition

Given a *goal* of a requester as well as a set of provider *offers*, both expressed by pre- and postconditions, first we select an offer providing a partial match of the goal. Then, we compute the remainder of the goal with respect to this offer, containing all the requirements not yet satisfied, and post the result as a new goal. We iterate these steps until all requirements are satisfied or we run out of offers to match. Finally, we compose all offers used into one global offer summarising the overall effect of the combined services. Next we describe this approach in detail.

Partial Matching. Given a request $\langle PRE_R \Rightarrow POST_R, \alpha_R \rangle$ a *partial match* with a provided description $\langle PRE_P \Rightarrow POST_P, \alpha_P \rangle$ is given by a partial embedding of PRE_P into PRE_R and of a partial embedding of $POST_R$ into $POST_P$. Following the previous discussion, the idea is that, on the one hand, not everything included in the provider's precondition needs to be present in the requester's precondition, since the latter may have to be completed later. On the other hand, not everything in the requester's postcondition needs to be present in the provider's postcondition, since not every effect demanded by the requester may be covered by a single provided operation.

Definition 1. ***(partial match, common suboperation)*** *Given requester and provider operations* $Op_R = \langle PRE_R \overset{pre_R}{\leftarrow} COM_R \overset{post_R}{\rightarrow} POST_R, \alpha_R \rangle$ *and* $Op_P = \langle PRE_P \overset{pre_P}{\leftarrow} COM_P \overset{post_P}{\rightarrow} POST_P, \alpha_P \rangle$ *a* partial match *m consists of embed-*

dings:

$$\begin{array}{ccccc} PRE_R & \xleftarrow{pre_R} & COM_R & \xrightarrow{post_R} & POST_R \\ \uparrow{\scriptstyle m_{PRE_R}} & (1) & \uparrow{\scriptstyle m_{COM_R}} & (2) & \uparrow{\scriptstyle m_{POST_R}} \\ PRE_C & \xleftarrow{pre_C} & COM_C & \xrightarrow{post_C} & POST_C \\ \downarrow{\scriptstyle m_{PRE_P}} & (3) & \downarrow{\scriptstyle m_{COM_P}} & (4) & \downarrow{\scriptstyle m_{POST_P}} \\ PRE_P & \xleftarrow{pre_P} & COM_P & \xrightarrow{post_P} & POST_P \end{array}$$

such that diagram (3) commutes, and diagrams (1), (2), and (4) are pullbacks.

The operation $Op_C = \langle PRE_C \overset{pre_C}{\leftarrow} COM_C \overset{post_C}{\rightarrow} POST_C, \alpha_C \rangle$*, where* α_C *is a condition such that* α_R *and* α_P *imply* $m_{PRE_R}(\alpha_C)$ *and* $m_{PRE_P}(\alpha_C)$*, respectively, is called a* common suboperation *of the provider and the requester operations, since it can be considered to be embedded in both operations. For example, the* common suboperation *is shown by unshaded background in Fig. 5 in both embeddings, where the circled arrows denote the partial embeddings between the providers and requesters pre and postconditions.*

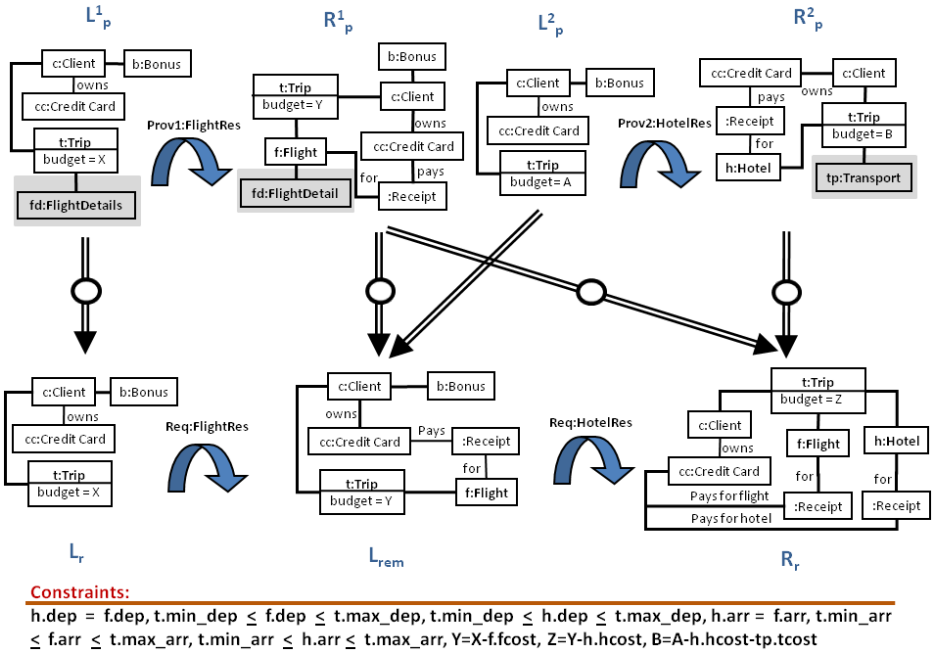

Fig. 5. Requester goal jointly matched by two consecutive offers

The condition α_C is obtained from α_P when some of its free variables are not present. In particular, α_C may be $\exists X \alpha_P$ or some stronger condition, where X is the set of variables included in the provided operation which are not present in the common suboperation. The fact that we do not ask diagram (3) to be a

pullback, while we ask diagrams (1), (2), and (4) to be so, is a consequence of the fact that we want to express the condition that Op_P implements partially the effects of Op_R on their common elements. This means, on one hand, that if a common element is deleted by Op_P then that element must also be deleted by Op_R, but not necessarily the other way round. Conversely, this means that every common element preserved by Op_R must also be preserved by Op_C (and hence by Op_P), which means that (1) is a pullback. However, the fact that not every common element preserved by Op_P must also be preserved by Op_R means that (3) is not necessarily a pullback. On the other hand, (2) and (4) are pullbacks, because we consider that if a common element in $POST_C$ is produced by Op_P, then it should also be produced by Op_R, and vice versa, That is, it makes no sense to think that an element that is considered to be information used by the requester's rule is produced by the provider's rule, or the other way round.

The common suboperation of Op_P and Op_R, while being embedded into both operations, does not implement their common behaviour. When Op_C is applied to a given state $\langle G, \alpha \rangle$, it adds all common elements added by both Op_P and Op_R, but it only deletes common elements that are deleted by Op_R, but not necessarily by Op_P. For instance in Figure 5, the requestor goal requires the deletion of $b : Bonus$ but $Prov1 :: FlightRes$ does not. The common suboperation of $Prov1 :: FlightRes$ and $Req :: FlightRes$ is constituted by all elements of $Prov1 :: FlightRes$ not shaded in grey. Hence, $b : Bonus$ is in the precondition of the common *common suboperation* but not in the postcondition, i.e., it is deleted. However, we are interested in a common operation that describes the shared effects of Op_P and Op_R, i.e., that deletes all elements deleted by both Op_P and Op_R and adds all elements added by both operations.

Definition 2. ***(shared behaviour suboperation)*** *Given requester and provider operations Op_R and Op_P, and given their common suboperation Op_C with respect to a* partial match *m, we define the shared behaviour operation of Op_R and Op_P with respect to m as $Op_{SB} = \langle PRE_C \overset{pre_{SB}}{\leftarrow} COM_{SB} \overset{post_{SB}}{\rightarrow} POST_{SB}, \alpha_C \rangle$, where COM_{SB} and $POST_{SB}$ are given by the following pullback and pushout diagrams:*

$$\begin{array}{ccc} PRE_C & \xleftarrow{pre_{SB}} & COM_{SB} \\ {\scriptstyle m_{PRE_P}}\downarrow & pb & \downarrow{\scriptstyle m_{COM_{SB}}} \\ PRE_P & \xleftarrow{pre_P} & COM_P \end{array} \qquad \begin{array}{ccc} COM_C & \xrightarrow{post_C} & POST_C \\ {\scriptstyle m_{COM_C}}\downarrow & po & \downarrow{\scriptstyle m_{POST_C}} \\ COM_{SB} & \xrightarrow{post_{SB}} & POST_{SB} \end{array}$$

with m_{COM_C} defined by the universal property of the pullback defining COM_{SB}.

Intuitively, COM_{SB} includes the elements which are shared by PRE_P and PRE_R and are not deleted by Op_P, and $POST_{SB}$ includes all the elements of $POST_C$ plus the elements that are not deleted by Op_{SB}. Fig. 6 depicts the *shared behaviour suboperation* based on the requestor goal and $Prov1 :: FlightRes$ and their common sub operation (shown in Fig. 5).

We may consider several special kinds of partial matches which are of interest.

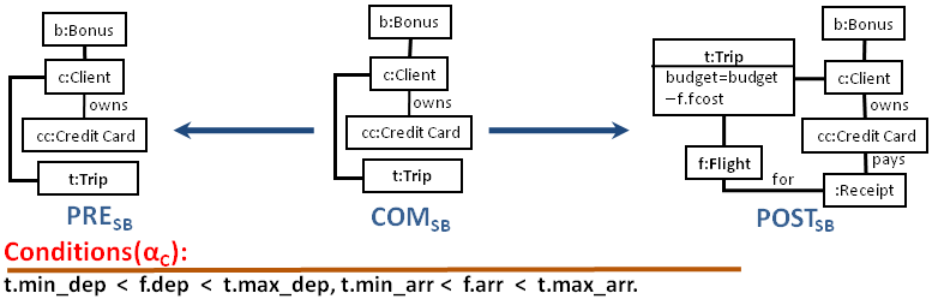

Fig. 6. Shared behaviour operation of goal and Prov1::FlightRes of Fig. 5

Definition 3. ***(classes of partial matches)*** *Given a partial match m as above:*

- *m provides* positive progress *if $post_C$ is not an isomorphism (or, equivalently, if $post_{SB}$ is not an isomorphism).*
- *m provides* negative progress *if pre_{SB} is not an isomorphism.*
- *m provides* progress *if m provides positive progress or negative progress.*
- *m is* demanding *if m_{PRE_P} is not an isomorphism.*
- *m is* weakly complete *if m_{POST_R} is an isomorphism.*
- *m is* complete *if diagram (3) is a pullback, m_{PRE_R} and pre_R are jointly surjective, and m_{POST_R} is an isomorphism.*

Two completely unrelated rules may be bound by a trivial partial match. For instance, a partial match where the common rule is empty, or if the precondition and the postcondition of the common rule coincide. In this sense, the first three cases describe partial matches where the provider's rule satisfies partially some of the goals of the requester. In particular, if m provides positive progress this means that the provider's rule produces some of the elements that the requester asks to be produced. Similarly, if m provides negative progress then the provider's rule consumes some of the elements that the requester asks to be consumed. Finally, m provides progress if it provides any progress at all.

If m is weakly complete then this means that the provider's rule produces all the elements that are asked by the requester but it may not consume all the elements that are specified to be consumed. If m is complete and not demanding this means that the provider's rule fully satisfies the requester's needs, i.e. m is a match. A partial match is demanding if the provider's rule demands the requester to strengthen its precondition. Conversely, this means that if m is not demanding then the provider's precondition is embedded in the requester's precondition, which means that the former can be considered stronger than the latter. This kind of situation may be part of a negotiation between the provider and the requester: the contract defined by the requester has specified some resources to satisfy his needs, but the provider is answering that, to satisfy this needs more resources are needed. If m is weakly complete then the provider's rule produces everything that the requester's rule asks to be produced. So this means that the requester's postcondition is embedded in the provider's postcondition. However, notice that this not necessarily means that the provider's rule consumes

everything that the requester's rule asks to be consumed. This only happens if, in addition, diagram (3) is a pullback and m_{PRE_R} and pre_R are jointly surjective, i.e. m is complete. In particular, the condition that diagram (3) is a pullback ensures that a common element cannot be preserved by Op_P and be deleted by Op_R, and the condition that m_{PRE_R} and pre_R ensures that there are no elements in PRE_R which are deleted by Op_R and which are not common elements (and, hence, cannot be deleted by Op_P).

Remainder of Requester Operation. If the match is not complete, then we may want to know what remains to be done to satisfy the rest of the requester's needs. In particular, the provider may want to use other operations to satisfy their requirements. This can be done by computing what we call the *remainder* of the requester's rule with respect to the shared behaviour rule.

Definition 4. ***(Remainder of an operation)*** *Given operation specifications $Op_R = \langle PRE_R \Rightarrow POST_R, \alpha_R \rangle$ and $Op_P = \langle PRE_P \Rightarrow POST_P, \alpha_P \rangle$, we define the remainder of Op_R with respect to Op_P and a partial match m as the operation $\langle PRE_{Rem} \Rightarrow POST_R, \alpha_R \rangle$, where PRE_{Rem} is the result of applying the operation Op_{SB} to PRE_R with match m_{PRE_R}.*

The idea is quite simple. We know that PRE_R denotes the class of states where Op_R is expected to be applicable, but also that Op_{SB} specifies the shared behaviour of Op_P and Op_R, i.e., all the deletions and additions which are shared by both operations. Then, PRE_{Rem} would describe the states after these deletions and additions, and $\langle PRE_{Rem} \Rightarrow POST_R, \alpha_R \rangle$ would specify the effects that are yet to be implemented by another provider operation.

For example in Fig. 5, the left-hand side PRE_{Rem} of the remainder rule is obtained by applying the *shared behaviour suboperation* (shown in Fig. 6) to the left-hand side PRE_R of the goal, while the remainder's postcondition is $POST_R$. Op_{Rem} may be matched with $Prov2 :: HotelRes$ in the same way, leaving empty remainder.

It is not difficult to prove that the remainder is the trivial operation, i.e. $PRE_{Rem} = POST_R$, if and only if the match m is complete. Moreover, we can also prove that if a provider operation Op_P^1 can be partially matched via m_1 to a request Op_R leaving Op_{Rem} as a remainder, and if another provider operation Op_P^2 can be partially matched via m_2 to Op_{Rem} leaving as a new remainder Op'_{Rem}, then we can compose Op_P^1 and Op_P^2 to form a new operation $Op_P^3 = \langle PRE_P^3 \Rightarrow POST_P^3, \alpha_P^3 \rangle$ that can be partially matched to Op_R via m_3, which is built from m_1 and m_2, directly leaving as a remainder Op'_{Rem}. This means that the global effect of this new operation is the same one as the sequential application of Op_P^1 and Op_P^2. In particular, this means that if m_2 is complete then m_3 is also complete.

The operation Op_P^3 is built analogously to the so-called concurrent rule for the consecutive application of two graph transformation rules [15]: Intuitively, PRE_P^3 is the union of PRE_P^1 and (PRE_P^2 minus $POST_P^1$) and $POST_P^3$ is the union of $POST_P^2$ and ($POST_P^1$ minus PRE_P^2), where in each case elements from

different graphs are identified if they are mapping to the same elements of the requester specification. As for conditions, α_P^3 is the conjunction of α_P^1, α_P^2, and all equations $x_1 = x_2$, for all variables x_1 from α_P^1 and x_2 from α_P^2 which are bound to the same variable x from α_R via m_1 and m_2, respectively. For our example, Fig. 7 shows the resulting composed operation.

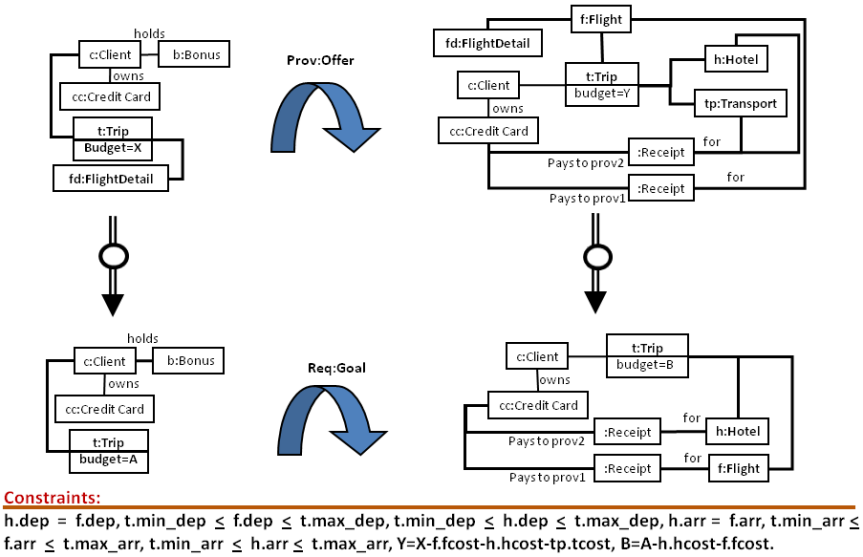

Fig. 7. Composed Operation of both Provider Offers

Finding Complete Solutions. We describe a resolution-like procedure for building a complete match to a requester specification. This procedure is terminating, correct and complete, i.e., given a requester operation Op_R, if at all possible this procedure will combine suitable provider operations into a single composed one which forms a complete match for the requestor's goal. Moreover, this is done in a finite number of steps. The procedure is presented as an inference rule whose application is non-deterministic, although in practice we could use heuristics to guide the search and to produce first the results which are considered better according to some criteria.

Let us describe this procedure in detail. We describe the computation states of our procedure as 3-tuples $\langle Op_P, m, Op_{Rem}\rangle$, where Op_P is a provider operation (perhaps built from more basic operations), m is a partial match from Op_P to Op_R, and Op_{Rem} is the remainder associated to m. Intuitively, Op_P represents the partial solution that we have built up to that point, m is the partial match that tells us in which way Op_P partially satisfies the request, and Op_{Rem} is the the part of the request that we still have to satisfy. In this context, we consider that the initial state is the 3-tuple $\langle Triv, triv, Op_R\rangle$, where $Triv$ is the trivial (empty) operation, $triv$ is the trivial empty match and, obviously, the remaining part to satisfy is the whole request. Then, the procedure is based on the following inference rule:

$$\frac{\langle Op_P^1, m_1, Op_{Rem}^1 \rangle}{\langle Op_P^2, m_2, Op_{Rem}^2 \rangle}$$

If there is a provider operation Op_P^3, and a partial match m_3 from Op_P^3 to Op_{Rem}^1, such that m_3 provides progress, Op_{Rem}^2 is the remainder associated to m_3, and Op_P^2 and m_2 are, respectively, the composition of Op_{Rem}^1 and Op_{Rem}^3 and the associated partial match from Op_P^2 to Op_R.

An execution of this procedure is a sequence:

$$\langle Triv, triv, Op_R \rangle \Longrightarrow \langle Op_P^1, m_1, Op_{Rem}^1 \rangle \Longrightarrow \ldots \Longrightarrow \langle Op_P^n, m_n, Op_{Rem}^n \rangle$$

where, for each i, $\langle Op_P^{i+1}, m_{i+1}, Op_{Rem}^{i+1} \rangle$ can be inferred from $\langle Op_P^i, m_i, Op_{Rem}^i \rangle$. Then, an execution is successful if the final match m_n is complete.

It is not difficult to show that the above procedure is correct, complete and terminating. In particular, it is sound in the sense, for every i, m_i is a partial match from Op_P^i to Op_R, and Op_{Rem}^i is the corresponding remainder. It is complete in the sense that if there is a way of satisfying completely the request by applying a sequence of provider operations then there exists an execution that will return a composed operation, together with a complete match. Finally the procedure is terminating, i.e. there are no executions of infinite length and, moreover, there is a finite number of executions, provided that the graphs involved are finite and that there is a finite number of provider operations. This is due to the fact that the number of additions and deletions requested in a goal is finite. Since we are assuming that all the matchings involved provide progress, the length of each execution is bounded by the number of additions and deletions specified in the request. Moreover, with a finite number of provider operations and finite graphs only, there is a finite number of partial matches between requester operation and provider operations.

5 Conclusion

In this paper, we have proposed an approach to the incremental composition of services using visual specifications of pre- and postconditions. The procedure is based on the repeated partial matching of provider offers with a requestor goal, which is reduced in the process until all requirements are satisfied or there are no more offers to consider. As a result, the procedure constructs a combined offer, which can be presented to the requestor to confirm if it is acceptable.

The formalization of these notions and constructions is provided in the appendix for information. In summary, the main theoretical results are as follows.

1. A definition of partial matching allowing the comparison of individual offers of services with the global goal of the requestor.
2. An incremental matching procedure based on the construction of a remainder of a goal with respect to a chosen partial match. Assuming a finite number of offers, the incremental matching procedure terminates. Thus, partial matching is decidable.

3. Each combined offer constructed as result of the matching has the same overall effect as executing the sequence of offers from which the combined offer is derived. That means, for each sequence of applications of individual offer rules there exists an application of the combined offer rule with the same effect, and vice versa.

In general, there will be several combined offers computed for a given request. These could be presented to the client to let them choose the most suitable one. Alternatively, the selection could be automated based on a specification of preferences (non-functional properties) by the client. Once an offer is computed it can be stored in the repository of services, such that new requests can be served more quickly, matching them against existing combined offers. Future work will address the use of non-functional requirements for the selection of offers, as well as a proof-of-concept implementation of the approach.

References

1. Papazoglou, M.P.: Service-oriented computing: Concepts, characteristics and directions. In: Fourth International Conference on Web Information Systems Engineering (WISE'03), Roma, Italy December 2003, IEEE Computer Society (2003)
2. Kim, I.W., Lee, K.H.: Describing semantic web services: From UML to OWL-S. In: IEEE International Conference on Web Services ICWS 2007), Salt Lake City, UT, USA, July 2007, IEEE Computer Society (2007) 529–536
3. de Bruijn, J., Lausen, H., Polleres, A., Fensel, D.: The web service modeling language WSML: An Overview. In Sure, Y., Domingue, J., eds.: Proc. of the 3rd European Semantic Web Conference (ESWC 2006). Volume 4011 of LNCS., Springer (2006) 590–604
4. Hausmann, J.H., Heckel, R., Lohmann, M.: Model-based development of web service descriptions: Enabling a precise matching concept. International Journal of Web Services Research **2**(2) (2005) 67–84
5. Peltz, C.: Web services orchestration and choreography. Computer (2003) 46–52
6. van der Aalst, W.: Don't go with the flow: Web services composition standards exposed. (2003)
7. Sirin, E., Hendler, J., Parsia, B.: Semi-automatic composition of web services using semantic descriptions. In: Workshop on Web Services: Modeling, Architecture and Infrastructure. (2002) 17–24
8. Wu, D., Parsia, B., Sirin, E., Hendler, J.A., Nau, D.S.: Automating DAML-S web services composition using SHOP2. In Fensel, D., Sycara, K.P., Mylopoulos, J., eds.: International Semantic Web Conference. Volume 2870 of Lecture Notes in Computer Science., Springer (2003) 195–210
9. Stollberg, M., Keller, U., Lausen, H., Heymans, S.: Two-phase web service discovery based on rich functional descriptions. In Franconi, E., Kifer, M., May, W., eds.: The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings. Volume 4519 of Lecture Notes in Computer Science., Springer (2007) 99–113
10. Klusch, M., Kaufer, F.: Wsmo-mx: A hybrid semantic web service matchmaker. Web Intelli. and Agent Sys. **7**(1) (2009) 23–42

11. Heckel, R., Cherchago, A.: Structural and behavioral compatibility of graphical service specifications. Journal of Logic and Algebraic Programming **70**(1.1) (2007) 15–33
12. Kreger, H.: Web services conceptual architecture (2001) IBM Software Group, http://www.ibm.com.
13. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation (Monographs in Theoretical Computer Science. An EATCS Series). Springer (2006)
14. Heckel, R., Llabrés, M., Ehrig, H., Orejas, F.: Concurrency and loose semantics of open graph transformation systems. Mathematical Structures in Computer Science **12** (2002) 349–376
15. Rozenberg, G., ed.: Handbook of graph grammars and computing by graph transformation: vol. 3: Concurrency, parallelism, and distribution. World Scientific Publishing Co., Inc., River Edge, NJ, USA (1999)