

Information Gathering for Semantic Service Discovery and Composition in Business Process Modeling

Norman May and Ingo Weber

SAP Research, Karlsruhe, Germany
{norman.may | ingo.weber}@sap.com

Abstract. When creating an execution-level process model today, two crucial problems are how to find the right services (service discovery and composition), and how to make sure they are in the right order (semantic process validation). While isolated solutions for both problems exist, a unified approach has not yet been available. Our approach resolves this shortcoming by gathering all existing information in the process, thus making the basis of semantic service discovery and task composition both broader and more targeted. Thereby we achieve the following benefits: (i) less modeling overhead for semantic annotations to the process, (ii) more information regarding the applicability of services, and (iii) early avoidance of inconsistencies in the interrelation between all process parts. Consequently, new or changed business processes can be realized in IT more efficiently and with fewer errors, thus making enterprises more agile in response to new requirements and opportunities.¹

Key words: Semantic Business Process Management, Web Service Discovery & Composition

1 Introduction

When enterprises need to adapt to changes in their environment they often have to adjust their internal business processes. In such cases, a business expert with little technical knowledge designs a changed or new business process model, which should then be implemented. Today, the refinement of this model into an executable process is carried out manually, requiring time and communication between the domain expert and the respective IT expert with technical expertise. Hence, enterprises are confronted with limited adaptability of IT-supported business processes, leading to low agility and high cost for changes [3, 18].

In this paper we present an improvement addressing this problem by extending and combining previous work on service discovery, service composition, and process validation. In our approach we use all available process information to

¹ This work has been funded through the German Federal Ministry of Economy and Technology (01MQ07012) and the European Union (IST FP6-026850, <http://www.ip-super.org>).

reduce the modeling effort, to improve the efficiency of the refinement into an executable process, and to provide richer feedback on inconsistencies in models.

Our solution relies on semantic technologies which promise a simpler, cheaper and more reliable transformation from the business level into the technical level [3]. On the business level, the domain or business expert models a process in a graphic modeling notation, e.g. BPMN or UML activity diagrams, or in a process description language, e.g. a graphical representation of BPEL, and all tasks in the process are annotated with concepts of a common ontology.

Assuming that all tasks of the business process can be implemented by a set of Semantic Web services², an automatic translation of the business process into an orchestration of Semantic Web Services is possible. Efficient algorithms for such a translation require the Web services to be annotated with the same ontology as the business process [1, 4, 16]; i.e. every Web Service is annotated with its preconditions and its effect, i.e. the postcondition it establishes.

Finding the implementation for a process task is done in the following way: First, service discovery attempts to find a matching Web service. If this is not successful (or for other reasons not satisfactory, e.g., too costly), service composition is performed (e.g., [4]). Later on, the resulting combination of discovered or composed services and the process model needs to be validated to assure that the preconditions of all services hold in any possible case and that parallel services are not in conflict with one another. This is needed since in previous work discovery and composition is performed for every task in isolation and hence potential conflicts between multiple tasks are not taken into account directly.

Currently, modeling of a complex process is laborious due to the need to annotate all tasks with preconditions and postconditions manually. Also, by regarding the manually annotated precondition of a task, service composition may miss attractive solutions that could be applied by regarding the global process context (but not from the local task context). In addition, the supposedly "intelligent" tool may suggest faulty solutions, thus jeopardizing user acceptance.

Our solution is based on logical state summaries that may be encountered during any valid execution of the process. These states can, e.g., model real world behavior by using terms from the common ontology. The executable process is in an invalid state if the execution states of two parallel execution paths contradict each other. Such a contradiction arises in two cases: (1) One activity invalidates the precondition of another activity that is executed in parallel. (2) One activity relies on facts, some of which may be not true. We extend state-of-the-art approaches by integrating service composition and process validation to distribute information about the process model across all activities. More precisely, the contributions of this paper are the following:

1. Our method reduces the effort for modeling semantic business processes: The burden for annotating tasks is reduced because all context informa-

² In this paper we focus on the description based on Semantic Web services for the sake of readability. However, any kind of known execution-level process building blocks which are described in an analogous same way may be used here, e.g., automated workflow tasks, or standardized but entirely manual activities.

tion is taken into account. This is done, e.g. by expanding preconditions of the activities in the business process based on the postconditions of their predecessors.

2. During service composition our approach takes into account constraint-sets which restrict the states considered as intermediate states during service composition. The pruned states would otherwise lead to inconsistencies with states computed in parallel tasks.
3. We perform process validation in parallel with service composition. As a consequence, we are able to detect semantically invalid process parts much earlier than it is possible currently. Moreover, composition does not generate conflicts, thereby improving the effectiveness of the overall approach.

The remainder of the paper is organized as follows. First we describe the underlying formalism and existing work on composition and process validation in Section 2. On this basis we detail how our solution improves these techniques in Section 3. Section 4 discusses related work before Section 5 concludes.

2 Foundations

In order to discuss the concepts in this paper thoroughly, we now introduce the underlying formalism verbally. Subsequently, we discuss shortly today's discovery, composition, and semantic process validation techniques. A running example is used for illustration purposes, and the shortcomings of today's techniques are discussed at the end of the section.

2.1 Semantics for Business Process Models

In the following, the basic execution semantics of the control flow aspect of a business process model are defined using common token-passing mechanisms, as in Petri Nets. The definitions used here are basically equivalent to the ones in [20], which extends [19] with semantic annotations and their meaning. For a more formal presentation please refer to [20].

A process model is seen as a graph with nodes of various types – a single start and end node, task nodes, XOR split/join nodes, and parallel split/join nodes – and directed edges (expressing sequentiality in execution). The number of incoming (outgoing) edges are restricted as follows: start node 0 (1), end node 1 (0), task node 1 (1), split node 1 (>1), and join node >1 (1). The location of all tokens, referred to as a *marking*, manifests the state of a process execution. An execution of the process starts with a token on the outgoing edge of the start node and no other tokens in the process, and ends with one token on the incoming edge of the end node and no tokens elsewhere (cf. *soundness*, e.g., [23]). Task nodes are executed when a token on the incoming link is consumed and a token on the outgoing link is produced. The execution of a XOR (Parallel) split node consumes the token on its incoming edge and produces a token on one (all)

of its outgoing edges, whereas a XOR (Parallel) join node consumes a token on one (all) of its incoming edges and produces a token on its outgoing edge.

As for the semantic annotations, we assume there is a background ontology O out of which two parts are used: the vocabulary as a set of predicates P , and a logical theory T as a collection of formulae based on literals over the predicates. Further, there is a set of process variables, over which logical statements can be made, again as literals over the predicates. The logical theory is basically interpreted as a rule base, stating e.g., that all monkeys are animals, or that no man may be married to two women. These rules can then be applied to the concrete process variables, e.g., a particular monkey or a particular man. Further, the task nodes can be annotated using *preconditions* (pre) and *postconditions* (post, also referred to as *effects*), which are also formulae over literals using the process variables. A task can only be orderly executed if its precondition is met, then changing the state of the world according to its postcondition. The postcondition states the explicit effects, and together with the current state and the theory we may derive implicit effects, e.g.: if one carries away a table (with the explicit effect being that the table has been moved), and the theory tells us that everything which is standing on some other thing moves with this other thing, and the local state includes a bottle standing on the table, then the implicit effect is that the bottle is moved too. For any set of literals L we refer to \bar{L} as the union of L and this implications of L from the theory (e.g. $\bar{\text{post}}_i$ is the union of the explicit and implicit effects of T_i).

The question to which degree an explicit effect triggers implicit effects, i.e., further changes the state of the world, yields the well-understood frame and ramification problems. We deal with it by following Winslett’s possible models approach [22], resulting in a notion of *minimal changes*, which can be described as a kind of local stability: if there is no indication that something changed, then we assume it did not change. Further, we assume that all changes (unless explicitly inconsistent with the theory) trigger all applicable implications from the theory directly. For more details please refer to [20].

Example 1. *We motivate our solution based on the example process in Fig. 1, represented in BPMN [11]. In this process, every Task T_i is annotated with precondition pre_i and postcondition post_i . As outlined above, these annotations allow for an automatic³ transformation into an executable orchestration of Semantic Web Services and for validating the consistency of the process model.*

Let us assume that the ontology contains the literals haveCar, poor, rich, paysBills, billsPaid, haveProgram, and that the theory says that being rich and being poor are mutually exclusive as well as that iff you are rich you usually pay your bills⁴:

$$- \forall x : \neg\text{rich}(x) \vee \neg\text{poor}(x),$$

³ In the ideal case, full automation is possible. In the general case the transformation will be semi-automatic.

⁴ We refer to process variables as concrete entities, e.g., me, whereas the variables in the services are not bound yet, e.g., x.

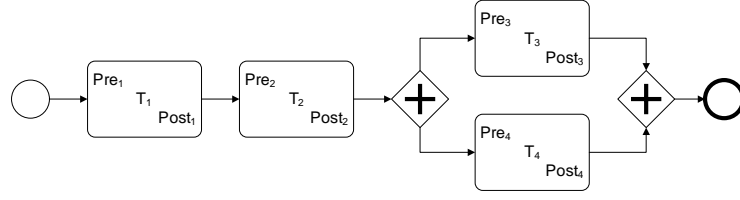


Fig. 1. Example BPMN process

- $\forall x : \text{rich}(x) \Rightarrow \text{paysBills}(x)$, and
- $\forall x : \text{poor}(x) \Rightarrow \neg \text{paysBills}(x)$.

Say, task T_2 is the task of selling your car, and thus annotated with the precondition $\text{pre}_2 := [\text{haveCar}(\text{me}) \wedge \text{poor}(\text{me})]$ and the postcondition $\text{post}_2 := [\neg \text{haveCar}(\text{me}) \wedge \text{rich}(\text{me})]$, where me is a process variable. Then, the theory allows us to derive the following implicit effects: $\neg \text{poor}(\text{me}) \wedge \text{paysBills}(\text{me})$, and $\overline{\text{post}_2} = [\neg \text{haveCar}(\text{me}) \wedge \text{rich}(\text{me}) \wedge \neg \text{poor}(\text{me}) \wedge \text{paysBills}(\text{me})]$.

Based on the execution semantics, we are now able to define the discovery of Semantic Web Services, Service Composition, and Semantic Process Validation more formally.

2.2 Service Discovery

By service discovery we mean the search for a service which matches a requirement best. Semantic Web Service discovery relies on the following input: (1) an ontology O with T and P as above, (2) a set Web services S with their preconditions and postconditions using the predicates in P , (3) a search request Q for a set of Web services which is described through its preconditions and postconditions just as the Web services. Given this input, semantic service discovery returns a set of services R that match the search request Q given ontology O . Three levels of matching are possible [12]: *exact* and *subsumption* matches may suffice to implement a task by a single Web Service; *partial* matches do not completely satisfy the request, however, a combination of them might do.

Example 2. Consider task T_4 in Fig. 1 which is supposed to implement the “PayBill” action. Task T_4 is annotated with precondition $\text{pre}_4 := [\text{paysBill}(\text{me})]$ and postcondition $\text{post}_4 := [\text{billPaid}(\text{me}) \wedge \text{poor}(\text{me})]$. Assume that our service repository contains the Web service CreditCardPayment which is annotated with $\text{pre}_{\text{PayBill}} := \text{paysBill}(x)$ and $\text{post}_{\text{PayBill}} := \text{billPaid}(x) \wedge \text{poor}(x)$.

Service discovery with pre_4 as search request is invoked to find all Web services whose preconditions hold. In this example, the Web service exactly matches the tasks precondition. Next, we check if this service establishes the postcondition of task T_4 , i.e. we have to test if the the Web Service’s postcondition subsumes the tasks postcondition. In this example, this condition holds, and thus the service CreditCardPayment can be selected as the implementation of task T_4 .

2.3 Service Composition

Service composition, in contrast, tries to find a composition of services which jointly satisfy the requirements. In many cases service composition will be performed when service discovery cannot find a single Web service – or it may be that the composition is a better match to the request than any service individually. Several algorithms for service composition exist and can be integrated into our framework, e.g. [4, 8].⁵ All of them have in common that they find a sequence⁶ of Web services $WS_{i_1}, WS_{i_2}, \dots, WS_{i_n}$ for a task T_i where the following conditions hold: (1) the precondition of each Web service is fulfilled at the point where this service is to be executed, i.e., if s is the state in which WS_{i_j} is executed, then $s \models \text{pre}_{WS_{i_j}}$ – in particular, in the context here the precondition of T_i enables the execution of the first service, $\text{pre}_i \models \text{pre}_{WS_{i_1}}$; (2) the postcondition of task T_i is satisfied by the state s after the last Web service, i.e. $s \models \text{post}_i$. Notice that for the special case of $n = 1$ the composition result is a subset of the result of service discovery, R .

Example 3. *In the example process in Fig. 1, task T_3 may be annotated with $\text{pre}_3 := [\text{rich}(\text{me}) \wedge \neg \text{haveProgram}(\text{me})]$ and $\text{post}_3 := [\text{haveProgram}]$. Say that there are, amongst others, two services available: `buyComputer` and `writeProgram` with*

$\text{pre}_{\text{buyComputer}} := [\text{rich}(x)],$
 $\text{post}_{\text{buyComputer}} := [\neg \text{rich}(x) \wedge \text{haveComputer}(x)],$
 $\text{pre}_{\text{writeProgram}} := [\text{haveComputer}(x)],$ and
 $\text{post}_{\text{writeProgram}} := [\text{haveProgram}(x)].$

Note that the resulting composition of Web services contains the literal $\neg \text{rich}(x)$ as part of its state. In fact, this is a non-obvious inconsistency which semantic process validation, as described below, would detect. When service composition is performed in isolation for every task, inconsistencies like this one can be the result.

2.4 Process Validation

As mentioned above, a process validation step is needed to detect inconsistencies that may result from performing service composition in isolation for every task. Service composition locally does not lead to any inconsistencies at execution time. Hence, process validation only needs to consider tasks that potentially are executed in parallel because the effects of a task T_1 executed in parallel to another task T_2 may violate T_2 's precondition. The basic steps for detecting these inconsistencies are the following [20]:

⁵ Note that there is another notion of service composition, namely the composition of complex behavioral interfaces of a small number of known services – see, e.g., [6, 13].

⁶ In general there may be more complex solutions than pure sequences – for the sake of brevity we omit further details.

- Computing the parallelity relation. For every task, determine the set of tasks that are potentially executed in parallel. Therefore, a matrix is computed that states which pairs of tasks T_i, T_j may be executed in parallel ($T_i \parallel T_j$).
- Detection of conflicting parallel tasks. Two parallel tasks $T_i \parallel T_j$ are in *precondition conflict* if there is a literal l such that $l, \neg l \in \text{pre}_i \cup \overline{\text{post}}_j$, or in *effect conflict* if there is a literal l' such that $l', \neg l' \in \overline{\text{post}}_i \cup \text{post}_j$.
- Determining if the preconditions will always hold. By propagation over the process model it is possible to compute the intersection of all logical states which may exist while an edge e is activated (i.e., a token resides on the edge). This intersection, which we refer to as $I^*(e)$ captures the logical literals which are *always* true when e is activated. Say, T_i is the task node whose incoming edge is e . Then we can check if the precondition of T_i will always hold when T_i may be executed, formally: $I^*(e) \models \text{pre}_i$. If not, we know that the process can be executed such that the precondition of T_i is violated. We refer to this property as *executability*.⁷

Example 4. Resuming with the example process in Fig. 1 after discovery and service composition was performed as described above, process validation produces the following results: First, the parallelity checker derives that Web service PayBill is executed in parallel with Web services buyComputer and writeProgram, and thus these two Web services can potentially be in conflict with Task PayBill. Next, we compute the states before and after executing these Web services. Among these states, state construction will derive the state after executing Web service buyComputer to be $\text{post}_{\text{buyComputer}}' := [\neg \text{rich}(\text{me}) \wedge \text{haveComputer}(\text{me}) \wedge \text{poor}(\text{me}) \wedge \neg \text{paysBill}(\text{me})]$. Clearly, this conflicts with the precondition of Web service PayBill. Notice that this inconsistency is not detected during service composition because it is performed for every task in isolation.

2.5 Shortcomings of State-of-the-Art Solutions

The sequence of steps described in this section implies a number of limitations and problems that we address with our solution. First, service discovery and composition can only exploit the annotations explicitly attached to the tasks in the process. Consider the example of a task T_i with $\text{pre}_i := \text{poor}(\text{me})$ and $\text{post}_i := \text{haveProgram}(\text{me})$. Then neither service discovery nor composition would find any applicable services. If, however, the previous step in the process was buyComputer, and the computer has not been sold already, then there is an additional (hidden) precondition $\text{haveComputer}(\text{me})$. Taking this precondition into account makes the problem solvable through discovery and composition. Second, as shown in Section 2.3 service composition is performed for a single task in the process in isolation because using process level composition would

⁷ Note that this is a design time property of a process, which indicates if or if not instances of this process may not execute due to preconditions that are not fulfilled.

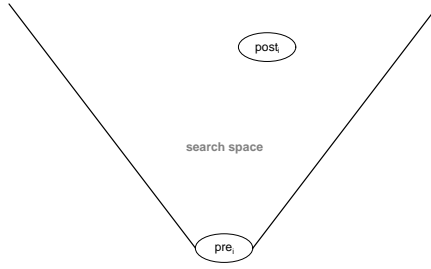


Fig. 2. Original search space

be close to automatic programming, and thus computationally prohibitive. The resulting orchestration of services is locally consistent and correct. However, inconsistencies may be caused by a lack of information about dependencies to other tasks. Therefore, a separate validation step is required to detect inconsistencies between the orchestration computed for different tasks in isolation. Third, inconsistencies induced during service composition per task or due to inconsistent annotations are detected very late in the whole procedure, wasting computing resources and slowing down the modeling process.

3 Solution Approach

In this section we present a detailed solution to the shortcomings identified in the previous section. The rough outline of the approach is the following: (1) before starting discovery or composition, all implicit information is derived from the process context; (2) based on this information we can potentially extend the preconditions of tasks that are to be implemented; (3) also due to the context information, the annotations of tasks are enriched with constraint-sets which are used to avoid the construction of compositions that are inconsistent with the rest of the process model. Given the extended preconditions and postconditions resulting from (1), service discovery and composition are able to detect more candidate services which increases the chance that any solution is found.

In the remainder of this section we focus on the effects of these extensions on the search space in service composition. This trivially relates to discovery by considering single-step compositions only. In order to better explain the implications, we visualize the composition search space. Fig. 2 shows an exemplary, unchanged search space, where the search starts from pre_i and tries to reach $post_i$.⁸ The funnel above pre_i then indicates the boundaries of the search space. This visualization of the search space serves as an intuitive and informal way to explain the idea of our approach. For a more formal treatment of search space implications, we refer to, e.g. [4, 20]. Possible solutions can be depicted as paths from pre_i to $post_i$, as depicted in Figure 3.

⁸ While the illustrations are related to forward search, the techniques can be applied analogously to backward search. For these basic planning techniques see [15].

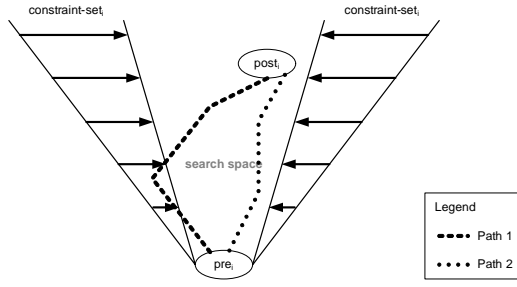


Fig. 3. Alternative solution paths, one of which violates the constraint-set

A solution path is only valid if it does not cross the boundaries of the search space. Note that Path 1 in Fig. 3 leads through a region which is out of boundaries once the constraint-set is taken into account. In other words: Path 2 is valid with and without considering the constraint-set, whereas Path 1 is only a valid solution when the constraint-set is neglected.

3.1 Modification of the Search Space

In Section 2, we explained how current solutions compute a possible solution given the precondition and a postcondition associated with a task (cf. Fig. 2). Below, we show how the expansion of the precondition and the inclusion of constraint-sets affects the shape of the search space.

First, we propose to expand the precondition of every task. For this purpose we merge its precondition with the logical facts we know must always hold when this task may be executed (given through I^* of the incoming edge, see Section 2.4). Basically, those are the literals which were established at one point in the process before the current task, i.e., that were true from the start on, or that were made true in the postcondition of another task, and which cannot be made false before the task at hand is executed. The computation of this set of literals can be done in polynomial time for our restricted class of processes (in the absence of loops), e.g., by an efficient propagation technique such as the I -propagation method described in [20].⁹

As shown in Fig. 4, our algorithm expands the precondition pre_i . In general, the extended precondition allows us to discover additional applicable Web services, because discovered Web Services can rely on more known preconditions. This increases the choices to orchestrate Web Services, and thus it becomes more likely to find any valid orchestration of Web Services. In Fig. 2, imagine that the goal, i.e. the postcondition $post_i$, was not completely inside the search space, but that in Fig. 4 it was. This would mean that a previously unsatisfiable goal becomes satisfiable by using the precondition expansion.

⁹ This algorithm is implemented and runs in well under a second for common examples. Given its polynomial time worst-case behavior, scalability is unlikely to be a problem.

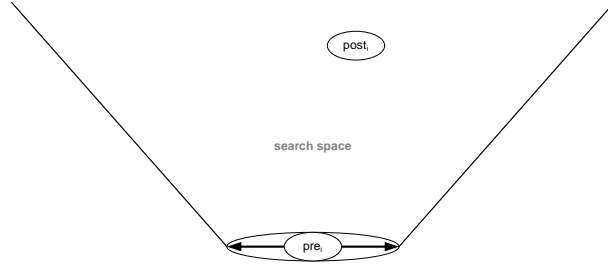


Fig. 4. Search space with expanded precondition

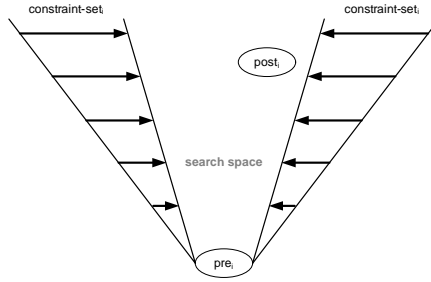


Fig. 5. Constrained search space

The constraint-set, our second extension to current solutions, may have an opposite effect. Unlike preconditions and postconditions, which are conjunctive formulae, a constraint-set is a set of literals interpreted as a disjunctive formula which expresses constraints on the states that may be reached during the execution of a task: if one of the literals from the constraint-set appears, the constraint is violated. The constraint-set is used to preemptively avoid parallelism-related conflicts. Thus, it is computed as the negated union of all preconditions and postconditions of the tasks that may be executed in parallel to the chosen task node n_i : $\text{constraint-set}_i := \bigcup_{n_j \parallel n_i} \{\neg l \mid l \in \text{pre}_j \cup \text{post}_j\}$.

Besides constraints explicitly modeled by the user (e.g. only services provided by preferred service providers may be considered), we can apply the M -propagation algorithm presented in [20] to compute the set of parallel nodes for each task node. Given that, the computation of a constraint-set _{i} is straightforward. As above, this can be done in polynomial time. Note that it is a deliberate choice to use a restricted logical formalism. While richer formalisms offer more expressivity, the here proposed extensions for modeler support are to be used during process modeling. Since long waiting times are undesirable, we propose a restricted formalism here.

As shown in Fig. 5, constraint-sets restrict the search space considered during service composition. As an important positive implication service composition avoids generating solutions that will conflict with tasks executed in parallel. This can be achieved by filtering out services whose preconditions and postconditions

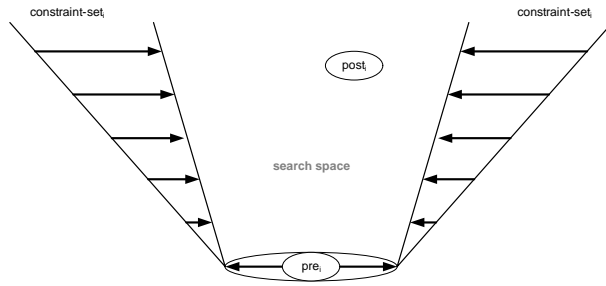


Fig. 6. Constrained search space with expanded precondition

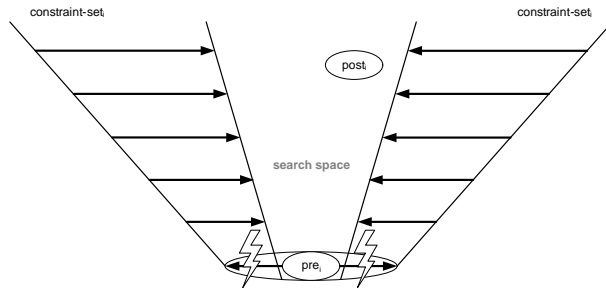


Fig. 7. Constrained search space with expanded precondition, and conflicts between the precondition and the constraint-sets

would cause a conflict with the constraint-set in a filtering step right before the actual composition. In effect, invalid orchestrations of Web Services will not even be generated and thus need not be detected (and fixed) later – e.g., as in Fig. 3.

By including both extensions (Fig. 6) we gain the best of both: while considering further relevant services we restrict the search space to valid compositions.

Fig. 7 depicts another interesting case: here, the expanded precondition is in conflict with the constraint-set. Apparently, the respective goal is not satisfiable and should not be considered. Note that in this case the conflict is between the expanded precondition and the narrowed boundaries of the constraint-set. However, the conflict can also be present between the original boundaries and the original precondition, or any combination in between.

Example 5. *Reconsidering the example from Section 2, we now indicate how the information gathering approach described in this section improves business process modeling. First, based on expanded preconditions more relevant services can be discovered, specifically when a service relies on postconditions established by predecessors of a task which cannot be derived from the precondition of a task alone. This improves upon the example mentioned in Section 2.5 for current solutions. Second, service composition does not compute an invalid orchestration here when it takes into account the constraint-sets. Thus, in Example 3 service composition in our method would not even generate a solution in which the implementation of task T_3 is in conflict with task T_4 .*

3.2 Configuration Options

An interesting aspect of our approach to semantic business process modeling is its adaptability to specific user needs along three dimensions: (1) With our method the user can, to a degree, trade completeness of the solution for efficiency. (2) The quality and granularity of error messages can be configured to the needs of the modeler. (3) It is possible to examine more suitable compositions beyond the initial solution found by service composition.

Completeness vs. Efficiency. First, the user trade-off on completeness of composition vs. efficiency: one can configure the modeling tool such that preconditions of tasks in a process are not expanded, leading to fewer Web services being considered during composition and composition is performed faster (it is exponential in the number of candidate services [4]). However, service composition may not find a valid composition of tasks that implements the modeled process even if this is possible with the available Web services (compare Fig. 2 and Fig. 4).

Error Reporting. Second, consider the problem that service composition fails because a task in the process model cannot be composed without violating its constraint-set. Then it may be useful to actually use the composition regardless of the violation, and change other parts of the process model to resolve the inconsistency. This becomes possible, e.g., by expanding the precondition but ignoring the constraint-set of a task. An explicit semantic process validation step, as it is used by state-of-the-art approaches, then highlights the violation. Thereby, error messages presented to the user can convey more information that help him to adjust the process.

Quality of the Composed Process. Finally, assume that a valid composition is already found by not expanding preconditions. One might still be interested in a more suitable service composition, if any exists that is to implement if preconditions are expanded.

These three dimensions for configuring the process underline the power of being able to adapt the process to the specific needs of the process modeler, and hence are another contribution of this paper.

3.3 Applying Search Space Modifications During Modeling

The modification of the search space outlined in this section is embedded into the overall translation procedure that we describe below. Our integrating solution extends the one presented in [21] while sharing its basic architecture.

Figure 8 summarizes the steps we follow for modeling a process. The user triggers service composition for an arbitrary task in the modeled business process. This can be done explicitly by a function invocation or implicitly, e.g. after the modeler has finished to define the task. As a result, a request is sent to derive all relevant information, which includes for each task, one after the other finding all states that can be executed in parallel, expanding the precondition,

deriving the constraint-set after which service discovery and service composition is performed. This is repeated for each of the tasks in an order that reflects the control flow over the tasks. Depending on whether or not the constraint-set is taken into account, the process needs to undergo semantic validation as a final step again. When no composition of the services was found, or when the modeler is not satisfied with the solution, the conditions attached to the tasks of the process model can be relaxed as discussed in Section 3.2.

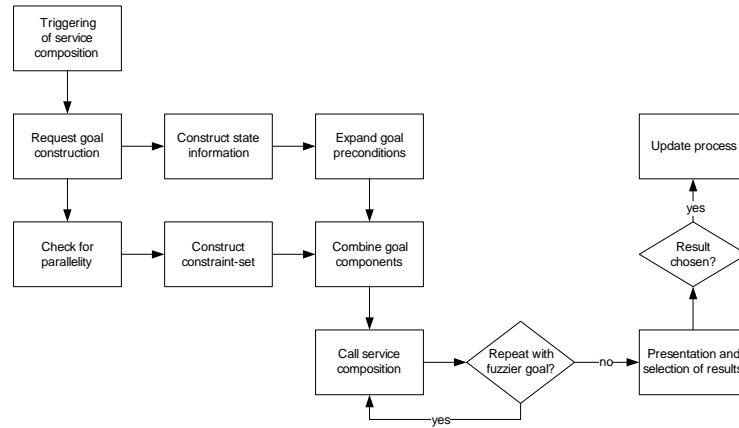


Fig. 8. Extended composition procedure

4 Related Work

In this paper we present a solution that integrates service composition with process validation to improve the efficiency of business process modeling. While to the best of our knowledge no integrated approach has been proposed yet, isolated solutions for service discovery, service composition, or process validation may be reused as building blocks in our integrated solution. The general approach and architecture that underlies our solution extends [21], which delivers implementation and configuration support in Semantic Business Process Management.

Our approach integrates well with the existing discovery mechanisms for Semantic Web Services, e.g. [7, 12] which are based on subsumption reasoning. In order to make our solution more flexible in the presence of erroneous business process models, we may extend service discovery to approximate reasoning. The feedback provided by the discovery mechanism outlined in [17] may provide guidance in case of empty results during service discovery, which can result from inconsistent use of the background ontology or distributed ontologies.

The methods for service composition [4, 8, 10] and validation [20] referred in this paper are based on the possible models approach of AI planning [22]. In principle, other algorithms or formalisms for discovery [1], composition [2, 10, 14, 16], or validation [5, 9] may be used or adapted to this context.

However, note that [9] uses a fundamentally different way to check the semantic consistency of process models: their work yields a checking method for

pairwise constraints over task co-occurrences (e.g., task *A* must always be preceded by task *B*) in block-based processes. In contrast, our underlying semantic process validation checks if declaratively annotated preconditions and postconditions are consistent with a flow-based process model and a background ontology. This is achieved by our *I*-propagation, which propagates summaries of the logical states that may be encountered. And exactly this logical state information then is used in the work presented in this paper for the precondition expansion – which disallows relying on [9] for the purposes here.

With respect to [5], it is worthwhile noting that their approach goes into a similar direction of propagating effects. Besides some formal flaws and concerns about the computational properties of their work, further differences lie in the focus on the interoperation of processes in [5] and the absence of a thorough consideration of background ontologies.

5 Conclusion

In this paper we presented a method to support the modeler of a semantic execution-level business process. For this purpose, the available semantic information in the process is collected and made available to service discovery and composition, leading to semantically consistent compositions. This is achieved by integrating previous work on service composition and process validation. In more detail, we use the process context to (i) extend the precondition which is certain to hold at a given point in the process, and (ii) derive a constraint-set, i.e., constraints on the intermediate states that may be reached inside a process activity (e.g., by service composition).

In terms of runtime, these extensions have contrary effects: on the one hand, the search space considered during service composition is reduced through the constraint-sets, as they are used to prune away solutions which would lead to inconsistent states in the resulting orchestration anyhow; on the other hand, the extended preconditions may lead to a larger search space, if the composition is performed in the manner of a forward search (i.e., starting at the precondition, searching towards the postcondition); if, however, composition is done in a backward search manner (i.e., in the opposite direction), this downside can be avoided.

While in practice the transformation of business processes into an executable process is a largely manual task, our solution is a clear step towards better automation. Thus, based on the presented solution the realization of changes to business processes may experience a significant speed-up. In future work, we plan to verify our claims experimentally by integrating and extending our previous work [4, 20] as described here.

References

1. R. Akkiraju, B. Srivastava, I. Anca-Andreea, R. Goodwin, and T. Syeda-Mahmood. Semaplan: Combining planning with semantic matching to achieve web service

- composition. In *4th International Conference on Web Services (ICWS-06)*, 2006.
2. I. Constantinescu, B. Faltings, and W. Binder. Typed Based Service Composition. In *Proc. WWW'04*, 2004.
 3. Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In *ICEBE*, pages 535–540, 2005.
 4. Jörg Hoffmann, James Scicluna, Tomasz Kaczmarek, and Ingo Weber. Polynomial-Time Reasoning for Semantic Web Service Composition. In *Intl. Workshop Web Service Composition and Adaptation (WSCA) at ICWS*, 2007.
 5. George Koliadis and Aditya Ghose. Verifying semantic business process models in inter-operation. In *Intl. Conf. Services Computing (SCC 2007)*, 2007.
 6. J. Lemcke and A. Friesen. Composing web-service-like abstract state machines (ASMs). In *Intl. Conf. Services Computing - Workshops (SCW 2007)*, 2007.
 7. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *12th World Wide Web Conference (WWW'03)*, 2003.
 8. Carsten Lutz and Ulrike Sattler. A proposal for describing services with DLs. In *International Workshop on Description Logics 2002 (DL'02)*, 2002.
 9. L.T. Ly, S. Rinderle, and P. Dadam. Semantic correctness in adaptive process management systems. In *Intl. Conf. Business Process Management (BPM)*, 2006.
 10. H. Meyer and M. Weske. Automated service composition using heuristic search. In *Intl. Conf. Business Process Management (BPM)*, pages 81–96, 2006.
 11. OMG. Business Process Modeling Notation – BPMN 1.0. <http://www.bpmn.org/>, 2006. Final Adopted Specification, February 6, 2006.
 12. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Intl Semantic Web Conf. (ISWC)*, 2002.
 13. M. Pistore, A. Marconi, P. Bertoli, and P. Traverso. Automated composition of web services by planning at the knowledge level. In *IJCAI*, 2005.
 14. Jinghai Rao and Xiaomeng Su. A Survey of Automated Web Service Composition Methods. In *SWSWPC*, pages 43–54, 2004.
 15. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1995.
 16. Evren Sirin and Bijan Parsia. Planning for semantic web services. In *Workshop "Semantic Web Services" at ISWC-04*, 2004.
 17. Heiner Stuckenschmidt. Partial matchmaking using approximate subsumption. In *AAAI*, pages 1459–1464, 2007.
 18. W. van der Aalst, A. ter Hofstede, and M. Weske. Business process management: A survey. In *Business Process Management (BPM)*, 2003.
 19. J. Vanhatalo, H. Völzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models through SESE Decomposition. In *Intl. Conf. Service-Oriented Computing (ICSOC)*, 2007.
 20. I. Weber and J. Hoffmann. Semantic business process validation. Technical report, University of Innsbruck, 2008. Available at <http://www.imweber.de/texte/tr-sbpv.pdf>.
 21. I. Weber, I. Markovic, and C. Drumm. A Conceptual Framework for Composition in Business Process Management. In *Business Inf. Systems (BIS)*, 2007.
 22. M. Winslett. Reasoning about actions using a possible models approach. In *Proc. AAAI'88*, 1988.
 23. M. Wynn, H. Verbeek, W. van der Aalst, A. ter Hofstede, and D. Edmond. Business process verification - finally a reality! *Business Process Management Journal*, 2007.