

On the Semantics of Service Compositions

Harald Meyer

Hasso-Plattner-Institute for IT-Systems-Engineering
Prof.-Dr.-Helmert-Strasse 2-3, 14482 Potsdam, Germany
`harald.meyer@hpi.uni-potsdam.de`

Abstract. Supporting service discovery by semantic service specifications is currently an important research area. While the approaches for the annotation of individual services are well researched, determining the semantics of compositions of services remains an open research issue.

In this paper, we present an approach to generate the semantics of service compositions from the semantics of the contained services. To do this we assume a formal Workflow net model of the service composition. With an example use case we show how this works in practice.

1 Introduction

Discovery, composition, and management of services are the most important tasks in a service-oriented architecture (SOA) [1, 2]. To perform them successfully in a large service landscape, semantic specifications of service functionality are important [3–5]. While several approaches for the description of individual service functionality exist, how to determine the functionality of service compositions remains largely unclear. It is the aggregation of the functionality of the individual services inside the composition. But determining this in the presence of complex control flow constructs is challenging. In this paper, we provide a formal model for the calculation of service composition functionality based on Petri nets that allows us to calculate the functionality for processes containing and- or xor- splits and joins.

There are several approaches for the description of individual service functionality through semantic service specification. The two most prominent approaches are OWL-S [6] and WSMO [7]. In OWL-S a service has a service profile that contains the functionality description consisting of input, output, precondition, and effect. These four elements describe the functionality of the service. The input describes the input parameters of the service; the output describes what the service returns. With precondition and effect logical expressions can be formulated to describe the states in which the service is invocable and that are reached by invoking the service. WSMO has a very similar model: A service has capability consisting of precondition, assumption, postcondition, and effect. While precondition and postcondition correspond to input and output in OWL-S, assumption and effect correspond to precondition and effect in OWL-S. So while they differ in syntax, terminology, and in the logical foundations used to express service functionality, they both agree on extending service descriptions

by logical expressions to describe the situation in which a service is invocable and the effects on the situation by invoking the service [8]. Having semantic service specifications, facilitates the discovery of services. As the functionality of services is known, matchmaking can be used to find services matching given preconditions and effects. Composition and management both heavily rely on discovery so semantic service specifications support them as well.

We argue that knowing what a service composition does is equally important as knowing what individual services do. A service composition is not the task of creating the process but the resulting process that can be enacted. Knowing the functionality of a service composition is important if discovery is not only performed on individual services but also on service compositions. While no individual service fulfills the request, an existing composition might fulfill it. Finding it, saves us from re-modeling it. Also service compositions can be exposed as services again. This service requires a semantic service specification. By giving means to calculate the functionality of the new service from the functionality of the services it is composed of, the publication of the new service is simplified.

One might be interested in what a service composition does to verify that it works as expected. After modeling a service composition, one can calculate the functionality and check, manually or automated, whether it provides the desired functionality. This verification task can also be part of a semi-automated service composition approach. In [9] and [10] we introduced three mixed initiative features for semi-automated composition: filter inappropriate services, suggest partial plans, and check validity. While the two first features can be implemented using existing service matchmaking and automated service composition approaches, the last one can be realized by the work presented in this paper. A last use case for the verification of service composition is the verification of automated service composition results. While proofs for the correctness of automated service composition approaches exist, bugs in the implementation or incorrect service specifications can lead to erroneous service compositions. Checking whether the created service composition serves the intended purpose, can detect some of problems.

In the next section, we will give a short overview of Petri nets and workflow nets. Then in Section 3 we will present our formal model on how to express the semantics of services inside a Petri net. This model will serve as the foundations for the algorithms presented in Section 4 for the calculation of service composition preconditions and effects. Afterwards, in Section 5, we will show how this algorithm works with a service composition example and demonstrate our tool that implements these algorithms. The paper closes with an overview of related work in Section 6 and the conclusion in Section 7.

2 Preliminaries: Petri nets & Workflow nets

This section describes the notions of Petri nets and workflow nets. A Petri net is a directed bipartite graph. It contains two sets of vertices: places and transitions.

The directed edges connect either a place with a transition or a transition with a place.

Definition 1. A Petri net is a triple $n = (P, T, f)$ with:

- P : set of places
- T : set of transitions
- $f \subseteq (P \times T) \cup (T \times P)$

The sets of places and transitions are disjoint : $P \cap T = \emptyset$

The input and output places of a transition t are denoted with $\cdot t$ and $t \cdot$. Each place can host multiple token. The assignment of tokens to the places of a Petri net is called the marking:

Definition 2. A marking is function $m : P \rightarrow N$ that assigns to each place the number of tokens in this place.

The marking represents the state of the Petri net. State changes of the Petri net result in different markings. State changes can occur when an *enabled* transition *fires*:

Definition 3. A transition t is enabled if all its input places $\cdot t$ contain at least one token.

If a transition t fires one token is removed from each input place and one token is added to each output place.

Petri nets are a very generic concept to describe processes. Workflow nets restrict the notation of Petri nets to a subset sufficient for modeling the control flow of workflows:

Definition 4. A Petri net $n = (P, T, f)$ is a workflow net iff:

- The net has one input place i (no incoming transitions)
- The net has one output place o (not outgoing transitions)
- Every vertex $v \in P \cup T$ is on a path from the input place to the output place.

A workflow net is sound if it terminates with one token in place o and no tokens in any other place and it does not contain any dead tasks. In the following we will limit ourselves to sound workflow nets to model service compositions.

3 Formal Model

To support the calculation of service composition functionality, workflow nets need to be extended to incorporate service semantics. Before this can be done, we first need to define a few basic concepts. A service is a discrete business functionality. It is described by a service specification:

Definition 5. A *service specification* $s = (\mathcal{I}, \mathcal{O}, pre, eff)$ is a tuple with

- \mathcal{I} : List of input parameters consisting of variables $\in V$
- \mathcal{O} : List of output parameters consisting of variables $\in V$
- pre : The precondition of the service is a logical expression and must be satisfied in order to invoke the service.
- eff : The effect of the service is a logical expression. It describes the changes to the current state resulting from the invocation of the service.

A **service invocation** $i = (s, z)$ is a pair consisting of a service specification s and a variable assignment $z : \mathcal{V} \rightarrow \mathcal{T}_{ground}$ that assigns every variable a ground term. Variables $v \in V$ are all the elements from \mathcal{I} and \mathcal{O} plus the variables in pre and eff .

INV is the set of all possible service invocations for a given set of service specifications.

Service invocations represent the atomic elements of service compositions. They are ordered in such a way to reach the goal of the service composition. In a workflow net they are the transitions. Hence, to represent them in a workflow net, the definition for workflow nets needs to be extended:

Definition 6. A semantic workflow net is a 4-tuple $sn = (P, T, f, l_s)$ is a Petri net $n = (P, T, f)$ with a function $l_s : T \rightarrow INV$ that maps each transition to a service invocation.

Finally, we need to refine the concept of states. In a workflow net the state is the marking assigning tokens to the place of the net. But given logically specified service semantics, a logical state exists as well. The logical state is a logical expression assigned to each marking of a workflow net. How these states are calculated will be presented in the next section.

4 Calculation of Service Composition Precondition and Effect

How to calculate the precondition and effect of given a semantic workflow net is described in this section. First the effect calculation will be introduced. Afterwards it is shown how this algorithm can be modified and extended to calculate preconditions. To do this, we need to define what happens when a service is invoked:

Definition 7. A service invocation $i = (s, z)$ with $s = (\mathcal{I}, \mathcal{O}, pre, eff)$ is **invokable** in state a if $a \models pre$. **Invoking service** s variable assignment z in state a leads to a state transition. This can be defined by the state transition function

$\gamma(a, i) = a \cup eff \setminus (\{x | \neg x \in eff\} \cup \{\neg x | i f x \in eff\})$. γ is a partial function only defined if $a \models pre$.

If an invokable service is invoked, all its effects are added to the state. Then all the facts of the state which are negated in the effects and all negated facts of the state which are not negated in the effects are removed from the new state.

4.1 Calculation of Effects

The calculation of the effects of a service composition works by recursively defining the current state for markings of the workflow net. Before we can show how this is done, we need to introduce the *reachability graph*:

Definition 8. *Given a semantic workflow net $n = (P, T, f, l_s)$ and an initial marking m_1 the reachability graph is a directed, labeled graph $mg = (V, E, l_V, l_E)$. The vertices V represent the possible markings. The labeling function $l_V : V \rightarrow M$ assigns to each vertex the according marking. The edges E represent the transitions of the Petri net. The labeling function $l_E : E \rightarrow T$ assigns a transition to every edge.*

Given this notion of a reachability graph the logical state for a given marking can be defined based on the logical states of its preceding markings:

Definition 9. *The logical state s for a marking m is given as $s = \bigvee \gamma(s_{m'}, i')$ with $i' = l_s(l_E(e))$ and $e = (m', m) \in E$ where $s_{m'}$ is the logical state of m' . The initial marking has the empty logical state s_0 .*

The effect of a service composition is the logical state assigned to its final marking.

This means that the state for a marking is given by the states of its preceding markings and the effects of the transitions leading from them to the current marking. If a marking has more than one incoming marking, several actual states are possible. To express this in one state, all possible states are combined disjunctively. With this possibility to calculate the logical state for a marking, the effect of a workflow net is the state assigned to its final marking. To show how this works we will now look at three prominent examples: sequence, and-split and -join, xor-split and join. For each, its effect will be calculated.

First, Figure 1 shows a workflow net containing a sequence and all possible markings. The initial marking is m_1T with $s_{m_1} = \{\}$; m_3 is the final marking and we want to calculate s_3 . To calculate it we need to solve $\gamma(s_{m_2}, t_2)$ which can be reduced to $\gamma(\gamma(s_{m_1}, t_1), t_2) = \gamma(\gamma(\{\}, t_1), t_2) = a \wedge b$

Similarly compositions with and-splits and -joins work (Fig. 2). This process has the following markings:

- $m_1 = p_1$ (initial marking)
- $m_2 = p_2 + p_3$
- $m_3 = p_3 + p_4$
- $m_4 = p_2 + p_5$
- $m_5 = p_4 + p_5$
- $m_6 = p_6$ (final marking)

This results in the reachability graph displayed in Figure 3. To calculate the service composition effect, the logical state assigned to the final marking M_6 needs to be calculated. This state is given by $s_{m_6} = \gamma(s_{m_5}, t_4) = \gamma(\gamma(s_{m_3}, t_3) \vee \gamma(s_{m_4}, t_2), t_4) = \gamma(\gamma(\gamma(s_{m_2}, t_2), t_3) \vee \gamma(\gamma(s_{m_2}, t_3), t_2), t_4) = \gamma(\gamma(\gamma(\gamma(s_{m_1}, t_1), t_2), t_3) \vee$

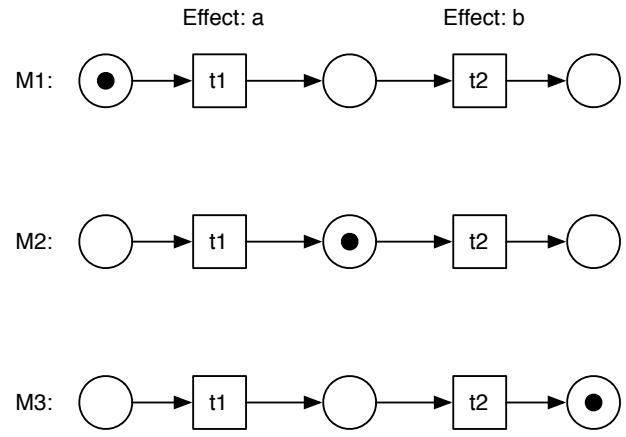


Fig. 1. Workflow net with a sequence

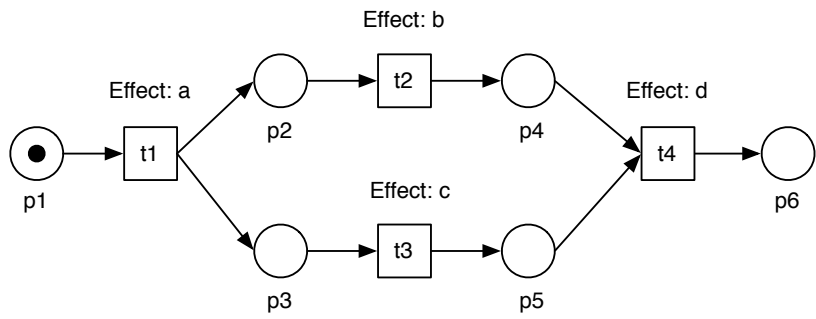


Fig. 2. Workflow Net with And-split and-join

$\gamma(\gamma(\gamma(s_{m_1}, t_1), t_3), t_2), t_4) = ((a \wedge b \wedge c) \vee (a \wedge c \wedge b)) \wedge d = a \wedge b \wedge c \wedge d$. As you can see, even though the algorithm introduced a disjunction, it can be reduced to the expected outcome.

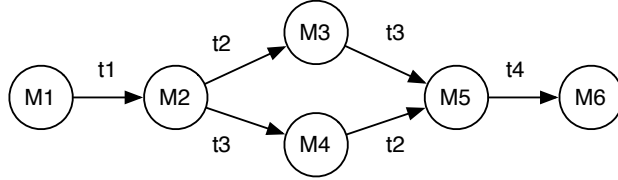


Fig. 3. Reachability Graph for And-split and -join

Finally, let us look at a composition with xor-splits and -joins. This composition is depicted in Figure 4. It has the following markings:

- $m_1 = p_1$ (initial marking)
- $m_2 = p_2$
- $m_3 = p_3$
- $m_4 = p_4$ (final marking)

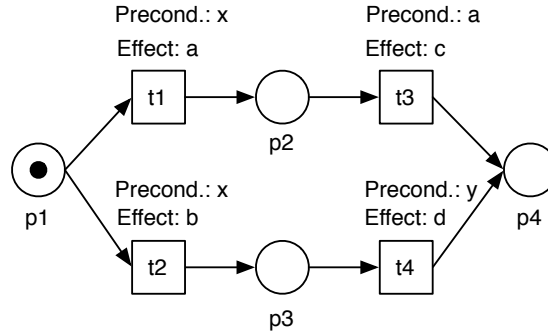


Fig. 4. Workflow Net with Xor-split and -join

The initial marking has two succeeding markings: m_2 and m_3 . And both markings have exactly one succeeding marking m_4 , the final marking. Calculating the state s_{m_4} works as follows: $s_{m_4} = \gamma(s_{m_2}, t_3) \vee \gamma(s_{m_3}, t_4) = \gamma(\gamma(s_{m_1}, t_1), t_3) \vee \gamma(\gamma(s_{m_1}, t_2), t_4) = (a \wedge c) \vee (b \wedge d)$. So this results in an effect for the composition one might expect. Either t_1 and t_3 or t_2 and t_4 are invoked. Actually, the service effect also allows for a third possibility: all four transitions are invoked. This does not map exactly to the semantics of the Workflow net, which forbids this possibility. While using exclusive or would solve this problem, it opens up

another one. The previous Workflow net contained an and-split and an and-join. In it the transitions t_2 and t_3 can fire in arbitrary order leading to distinct markings. In logics, the order of propositions is irrelevant, rendering $(a \wedge b \wedge c)$ and $(a \wedge c \wedge b)$ equivalent. Using inclusive or, this means that they can be merged. But with an exclusive or this results in a contradiction. Therefore, we choose the inclusive or leading to composition effects that are less restrictive than implied by the Workflow net. One possible solution to overcome this problem, is to replace the inclusive or by an exclusive or only after the whole calculation is finished and possible contradictions are resolved. This is allowed if only those axioms of the boolean algebra (A, \wedge, \vee) are used that are also defined in the corresponding boolean ring (A, \oplus, \wedge) plus idempotence.

In all examples listed above, the service effects were quite simple consisting of a singular fact. But service effects can be more complicated. They can contain conjunctions and disjunctions. In the next section, we will look at such a more complicated example. They may also contain first-order literals containing variables. Of course, these variables must be grounded in the service invocations assigned to the transitions.

The main restriction the algorithm currently has is that it does not allow for cycles in the workflow net. Because if it encounters a workflow net with cycles, the algorithm is not guaranteed to terminate. But for acyclic workflow nets termination is guaranteed.

4.2 Precondition Calculation

If it is possible to calculate the effects of a service composition, calculating the precondition should work similarly. In principle, this works like the effect calculation in reverse. Instead of calculating the state achieved when invoking the service composition, we want to determine the requirements on a state necessary to invoke the service. This works by starting from the initial marking and adding up all the preconditions until reaching the final marking. But there is one important distinction. Preconditions of services inside the compositions can be fulfilled by the effects of preceding services. These preconditions do not need to be specified as part of the service composition precondition.

Definition 10. *The precondition pre_m for a marking m is given by $pre_m = \bigvee pre_{m'} \cup (pre_{i'} - s_m)$ for all pairs $pre_{m'}$ and i' with $i' = l_s(l_E(e))$ where $e = (m, m') \in E$ and $pre_{m'}$ is the precondition of m' . The final marking has the empty precondition pre_{final} .*

The precondition of a service composition, is the precondition pre_{m_1} of its initial marking.

So instead of traversing the reachability graph backward until reaching the initial marking, the graph is traversed forward until the final marking is reached. Before adding the precondition of a service to the precondition of a marking, all the facts already known in the current logical state are removed. This ensures that preconditions satisfied by other services are not added to the precondition

of the service composition. The current logical state can be calculated given the method for service composition effects described earlier.

As this algorithm is quite similar to the effect calculation algorithm, only one example will be demonstrated: the process with and-splits and -joins from Figure 4. In contrast to the other figures, it also lists the preconditions for the services. To calculate the precondition of composition, we need to calculate the precondition for m_1 . It is given by $m_1 = (pre_{m_2} \cup (pre_{t_1} - s_{m_1})) \vee (pre_{m_3} \cup (pre_{t_2} - s_{m_1})) = ((pre_{m_4} \cup (pre_{t_3} - s_{m_2})) \cup (pre_{t_1} - s_{m_1})) \vee ((pre_{m_4} \cup (pre_{t_4} - s_{m_3})) \cup (pre_{t_1} - s_{m_1})) = (\{\} \cup (p_{t_3} - s_{m_2})) \cup (pre_{t_1} - s_{m_1}) \vee (\{\} \cup (pre_{t_4} - s_{m_3})) \cup (pre_{t_1} - s_{m_1}) = (\{\} \cup (\{a\} - s_{m_2})) \cup (\{x\} - s_{m_1}) \vee (\{\} \cup (\{y\} - s_{m_3})) \cup (\{x\} - s_{m_1}) = (\{\} \cup (\{a\} - \{a\})) \cup (\{x\} - \{\}) \vee (\{\} \cup (\{y\} - \{b\})) \cup (\{x\} - \{\}) = (\{\} \cup \{\} \cup \{x\}) \vee (\{\} \cup \{y\} \cup \{x\}) = (x) \vee (x \wedge y)$. Intuitively, we connect the two possible paths disjunctively and remove the precondition of t_3 because it is already fulfilled by the effect of t_1 .

To sum up, in this section we have seen how service composition preconditions and effects can be calculated. Both algorithms, are based on traversing the reachability graph containing all possible state transitions. The algorithm to calculate service composition preconditions requires the calculation of logical states for each marking except for the final marking. Having calculated the precondition, the effect calculation is reduced to determining the logical state for this final marking without traversing the reachability graph.

5 Example & Tooling

In this section a more complicated example is introduced and it is shown how it can be analyzed with our tool. The tool is a command line tool implemented in Java that reads Petri nets specified in the LoLA format [11]. We extended this format to incorporate preconditions and effects of transitions. An extract from our example looks like this:

```
PLACE s1, s2, s3, s4, s5, s6, s7,s8;
```

```
MARKING s1: 1;
```

```
TRANSITION order
```

```
CONSUME s1: 1;
```

```
PRODUCE s2: 1, s3: 1;
```

```
PRECONDITION ;
```

```
EFFECT ordered;
```

```
...
```

```
TRANSITION close
```

```
CONSUME s6: 1, s7: 1;
```

```
PRODUCE s8: 1;
```

```

PRECONDITION rsent and shipped;
EFFECT order_closed;

```

Such a file defines places, transitions and the initial marking for a Petri net. Each TRANSITION contains two additional sub-elements, PRECONDITION and EFFECT, describing the semantics of this transition. The example used in this section is an order process (Fig. 5). After ordering, shipment and sending the receipt are done in parallel. Depending on the requested shipper, one out of two allowed shippers is selected. Preceding the two shipment services are empty transitions. They do not perform any functionality but just decide which route should be taken. This example contains an and-split, an and-join, an xor-split, and an xor-join. To not overload the diagram, it does not contain the semantics for the transitions. They are instead depicted in Table 5.

Transition	Precondition	Effect
order		<i>ordered</i>
send_receipt	<i>ordered</i>	<i>rsent</i>
shipping1	<i>ordered</i> \wedge <i>shipper1</i>	<i>shipped</i>
shipping2	<i>ordered</i> \wedge <i>shipper2</i>	<i>shipped</i>
close	<i>rsent</i> \wedge <i>shipped</i>	<i>order_closed</i>

Table 1. Preconditions and Effects of the Services

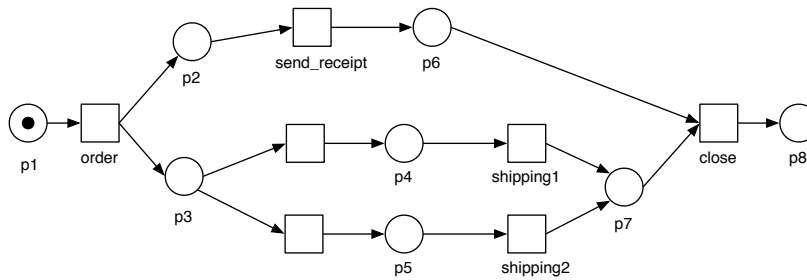


Fig. 5. Petri net of Order Service Composition

Using the tool, it is possible to calculate the semantics of the composition. As the effect it calculates $order_closed \wedge ordered \wedge rsent \wedge shipped$. The interesting fact about this effect is that it does not contain a xor even though the composition contained an xor-split. But as both possible paths have the same semantics (the order is shipped) this is explainable. The calculated precondition is $(ordered \wedge rsent \wedge shipped \wedge shipper1) \oplus (ordered \wedge rsent \wedge shipped \wedge shipper2)$. Obviously, the preconditions fulfilled by preceding services are not removed from

the composition's precondition. This is a weakness of the current implementation. It uses a simplistic, propositional reasoner developed specifically for this tool. One of the next steps, is to replace it by a full fledged reasoner like Pellet.

6 Related Work

The algorithms presented allow the calculation of service composition preconditions and effects. This allows for the automated generation of service specification and the verification of service compositions.

In spite of these possible applications, only a few similar approaches exist. Koschmider and Ried [12] present a work that sounds quite similar to the one we present in this paper: *Semantic Annotation of Petri nets*. They show how Petri nets can be expressed in OWL to allow reasoning about their properties. So they are using Semantic Web technologies to prove properties about the syntactic structure of the process. Our approach instead extends the syntactic structure by semantic service annotations and uses known syntactic properties of the control flow to derive the semantics of the whole composition. In [13] they apply their formalism to allow for the auto completion of business processes.

Much more similar is the work by Narayanan and McIlraith [14]. They present an execution semantics for DAML-S (now OWL-S) based on Petri nets. The main difference to our approach is that they express atomic DAML-S processes (or services) as Petri nets. Each service consists of several places and transitions mapping the preconditions and effects to Petri net constructs. In our approach each service corresponds to exactly one transition in the Petri net. Another difference is that we define formally how the Petri net is build up from the atomic services whereas Narayanan and McIlraith only state this informally.

Many approaches for the automated composition of services exist [15–19]. They can be seen as related work as their algorithms create a service composition with a requested functionality. But they all fail to make this explicit. This gap is filled by this paper.

In the Tools4BPEL project the relationship between Petri nets and service compositions in general is investigated. This includes the mapping of BPEL processes to Petri nets [20]. This work can be used to create the Workflow nets necessary for the presented algorithm.

7 Conclusion

In this paper we presented an approach to calculate the preconditions and effects of service compositions based on semantics of individual services and a Petri net representation of the service composition. Using this approach it is now possible to automatically calculate the preconditions and effects for service compositions. This allows us to verify the correctness of compositions and their publication as services. We also introduced a tool that implements the described algorithms.

In the future we want to extend the precondition and effect calculation to cyclic Workflow nets. To reach this goal, both algorithms need to be adjusted to

detect previously visited markings in the reachability graph. This functionality can then be used to ensure termination of the algorithm. Additionally, it needs to be discussed what a cycle or loop means semantically. If the only modification to the algorithms is to prevent visiting the same marking multiple times, the semantics of a composition with loops is equivalent to the semantics of an identical composition without the backward transition. This is not very useful.

Reasoning about loops only makes sense if service preconditions and effects are described using first-order logic. Otherwise, multiple runs through the same loop do not change anything. Therefore, a new reasoner is necessary as the very simple reasoner currently used is not sufficient for this. So a next step is to integrate a full-fledged reasoner.

References

1. Burbeck, S.: The tao of e-business services. IBM developerWorks (2000)
2. Papazoglou, M.P., Georgakopoulos, D.: Service-oriented computing: Introduction. *Communications of the ACM* **46** (2003) 24–28
3. McIlraith, S.A., Son, T.C., Zeng, H.: Semantic web services. *IEEE Intelligent Systems* **16** (2001) 46–53
4. Paolucci, M., Kawamura, T., Payne, T.R., Sycara, K.P.: Semantic matching of web services capabilities. In: *Proceedings of the First International Semantic Web Conference on The Semantic Web*, London, UK, Springer-Verlag (2002) 333–347
5. Li, L., Horrocks, I.: A software framework for matchmaking based on semantic web technology. In: *Proceedings of the 12th International Conference on World Wide Web (WWW03)*, New York, NY, USA, ACM Press (2003) 331–339
6. <http://www.daml.org/services/owl-s/1.0/>: OWL-S 1.0 Release. (2003)
7. <http://wsmo.org>: Web Service Modeling Ontology. (2005)
8. Keller, U., Lausen, H., Stollberg, M.: On the semantics of functional descriptions of web services. In: *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*. (2006) 605–619
9. Schaffner, J., Meyer, H.: Mixed initiative use cases for semi-automated service composition: A survey. In: *Proceedings of the International Workshop on Service Oriented Software Engineering*. (2006)
10. Schaffner, J., Meyer, H., Tosun, C.: A semi-automated orchestration tool for service-based business processes. In: *Proceedings of the 2nd International Workshop on Engineering Service-Oriented Applications: Design and Composition*. (2006)
11. Schmidt, K.: LoLA: A Low Level Analyser. In Nielsen, M., Simpson, D., eds.: *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*. Volume 1825 of *Lecture Notes in Computer Science.*, Springer-Verlag (2000) 465–474
12. Koschmider, A., Ried, D.: Semantische annotation von petri-netzen. In: *Proceedings des 12. Workshops Algorithmen und Werkzeuge fr Petrinetze (AWPN'05)*. (2005) 66 – 71
13. Betz, S., Klink, S., Koschmider, A., Oberweis, A.: Automatic user support for business process modeling. In: *Proceedings of the Workshop on Semantics for Business Process Management*. (2006) 1–12

14. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: Proceedings of the 11th International Conference on World Wide Web, New York, NY, USA, ACM Press (2002) 77–88
15. Zeng, L., Benatallah, B., Lei, H., Ngu, A., Flaxer, D., Chang, H.: Flexible Composition of Enterprise Web Services. *Electronic Markets – Web Services* **13** (2003) 141–152
16. Pistore, M., Barbon, F., Bertoli, P., Shaparau, D., Traverso, P.: Planning and monitoring web service composition. In: Workshop on Planning and Scheduling for Web and Grid Services (held in conjunction with The 14th International Conference on Automated Planning and Scheduling. (2004) 70 – 71
17. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using shop2. *Journal of Web Semantics* **1** (2004) 377–396
18. Berardi, D., Calvanese, D., Giacomo, G.D., Mecella, M.: Composition of services with nondeterministic observable behaviour. In: Proceedings of the Third International Conference on Service-Oriented Computing. Volume 3826 of Lecture Notes In Computer Science., Heidelberg (2005) 520–526
19. Meyer, H., Weske, M.: Automated service composition using heuristic search. In Dustdar, S., Fiadeiro, J.L., Sheth, A., eds.: Proceedings of the Fourth International Conference on Business Process Management (BPM 2006). Volume 4102 of Lecture Notes In Computer Science., Heidelberg, Springer (2006) 81–96
20. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri Nets. In van der Aalst, W., Benatallah, B., Casati, F., Curbera, F., eds.: Proceedings of the Third International Conference on Business Process Management (BPM 2005). Volume 3649 of Lecture Notes in Computer Science., Springer-Verlag (2005) 220–235