

Calculating the Semantic Conformance of Processes

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. Verifying process properties has been an important research topic in last several years. The idea is to support humans in modeling processes by checking whether their processes are correct according to certain criteria (e.g. always terminate, adhere to a predefined specification). But these approaches were mostly limited to verifying syntactic properties of the process ignoring semantics and functionality of the contained activities.

In this paper we introduce a new property called semantic conformance that ensures that a process has the intended functionality. A process is semantically conformant to a process specification if it fulfills the intended functionality in all situations described in the process specification and if every activity of the process is actually invocable whenever it can be invoked.

1 Motivation

Modeling business processes is an inevitably complex task. It includes identifying the right activities, their ordering, and the data flow between them. Depending on the concrete setting, it can also include modeling the organizational structure of the company or of quality of service properties and constraints. Assisting human modelers is hence crucial. In earlier work[1,2] we identified three such supporting features: filter inappropriate services, suggest partial plan, and verify semantic correctness. In these works we described the first two features in great details but left the exact implementation to verify semantic correctness open. In this paper we will close this gap.

A great deal of research was and still is targeted at verifying process properties like soundness, conformance, and compatibility [3–8]. But they only verify syntactical properties of processes (e.g. that they are deadlock-free). They cannot verify whether the process actually implements the intended functionality. Additionally, they can only show whether a process is syntactically correct. But a syntactically correct process can still contain semantic errors (e.g. the precondition for an activity is not fulfilled). This paper therefore introduces a new process property called semantic conformance. A process is semantically conformant to a process specification if it provides the intended functionality and the preconditions of all contained activities are satisfied whenever they are enabled.

The idea is that the specification exists prior to modeling the process. It is either created by the process modeler himself, given by the requirements of future users of the process or defined by industry standards. The process modeler then models the process trying to adhere to the specification. When he is finished he can check using semantic conformance whether he modeled it correctly.

The calculation of semantic soundness of processes is based on earlier work on algorithms to calculate the preconditions and effects of service compositions [9]. Processes are modeled as so-called semantic workflow nets, a special form of Petri nets extended by semantic annotations. While the approach presented here is independent of services and service oriented architectures, it can easily be mapped to such a setting: service compositions are the processes consisting of special activities called service invocations. In [9] we actually motivated the foundations using SOA: the ability to automatically calculate precondition and effect of a service composition enables us to easily publish such a service composition as a service.

The next section refreshes some fundamentals on Petri nets. The main contribution of this paper, the definition of semantic conformance and the approach to calculate it, is presented in section 3. The paper closes with a look on related work and the summary.

2 Semantic Workflow nets

In this section, we introduce Petri nets and workflow nets but also the advanced concepts from [9]. Using petri nets to model (business) processes or workflows is a common approach [3] and serves as the foundation for many process verification techniques. 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 flow relation

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

The transitions of a Petri net are active components. They represent the activities or service invocations. Places contain tokens to represent the current state of the net. 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.*

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 $v \in P \cap T$ is on a path from the input place to the output place.

In the following we will only use workflow nets. Another important concept are logical expressions. In this paper we will use a fairly simple notion of logical expressions: a logical expression is a collection of facts and negated facts. We will use them to express the semantics of the activities of a process. As the activities of a process are represented by the transitions of the workflow net, we add a labeling function that attaches the semantics to each transition:

Definition 5. 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 an activity instance.

An activity 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 activity is a logical expression and must be satisfied in order to invoke the activity.
- eff : The effect of the activity is a logical expression. It describes the changes to the current state resulting from the invocation of the activity.

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

Activity instances, the atomic elements of a process, are instances of activity specifications containing the actual semantics. As the last step in this section, we will describe briefly the algorithms to calculate preconditions and effects of processes modeled as semantic workflow nets. The idea is to assign a logical state to each marking of the workflow net. But before the algorithms can be described, we need to have a look at what actually happens if an activity is invoked:

Definition 6. An activity instance $i = (s, z)$ with $s = (\mathcal{I}, \mathcal{O}, pre, eff)$ is **invokable** in logical state a if $a \models pre$. **Invoking activity** s with variable assignment z in logical state a leads to a state transition. This is defined by the state transition function $\gamma(a, i) = a \cup eff \setminus (\{x \mid \neg x \in eff\} \cup \{\neg x \mid ifx \in eff\})$. γ is a partial function only defined if $a \models pre$.

How invocation is performed formally, is out of scope of this paper. But approaches exist to base it on transaction logics [10]. Other approaches to express this are based on the notion of abstract states [11]. Calculating the effect of a process can then be achieved by simulating the invocation of all contained activities. To do this, we will use the reachability graph of the semantic workflow net. The reachability graph is the state transition graph of the markings of a semantic workflow net:

Definition 7. *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 $rg = (V, E, l_V, l_E)$. 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 the final marking of a workflow net, we can then traverse the reachability graph backwards until we reach the initial marking adding up all the effects of contained activities. While doing so, we must take splits and joins into account. This can be achieved with a recursive algorithm that calculates the logical state for a given marking based on the logical states of its preceding markings. If the marking is the final one, this logical state is the process' effect:

Definition 8. *Given a semantic workflow net $n = (P, T, f, l_s)$ and its reachability graph $rg = (V, E, l_V, l_E)$, the logical state s for a marking m is given as $s_m = \bigvee \gamma(s_{m'}, i')$ 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 process is the logical state assigned to its final marking.

This algorithm is able to deal with processes containing sequences, and-split and -join, and xor-splits and -joins. Cyclic nets are not supported as the algorithm would not terminate. Additionally it still needs to be investigated what a loop actually means in terms of preconditions and effects.

Calculating the precondition works quite similar: instead of starting with the final marking, one starts with the initial marking and instead of adding up the effects one add up the preconditions. While these changes seem obvious, there is another, more subtle one. If the precondition of a activity is satisfied by preceding activities, this precondition should not be added to the precondition of the whole process. It is an internal precondition that is imposed and satisfied by the process itself. Hence, when calculating the precondition for a marking, already satisfied preconditions are not added. This is achieved by calculating the already known information (using the previous effect calculation algorithm) and subtract it from the precondition to be added:

Definition 9. *Given a semantic workflow net $n = (P, T, f, l_s)$ and its reachability graph $rg = (V, E, l_V, l_E)$, 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 process, is the precondition pre_{m_1} of its initial marking.

3 Semantic Conformance

In the previous section, we introduced the means to calculate the precondition and effect of a process. In this section, we apply them to verify the semantic conformance of a process to the previously defined process specification. The process specification describes the intended functionality under certain assumptions. Formally the process specification is just an activity specification describing the intended functionality of the whole process. Recapturing what a activity specification is, it consists of input parameters, output parameters, precondition and effect. The input parameters and the precondition define the situations in which the process should work and the output parameters and the effect define what changes the process may yield in these situations.

A process is semantically conformant to a process specification if the precondition of the process is always fulfilled in the precondition of the specification, if the effect of the process fulfills the effect of the specification, and if each activity in the process is actually invocable when it can be invoked. Formally:

Definition 10. *A process $n = (P, T, f, l_s)$ with a reachability graph $rg = (V, E, l_V, l_E)$ is semantically conform to a process specification $R = (I, O, pre, eff)$ if:*

1. $pre_R \models pre_n$ with pre_n the precondition of the process,
2. $eff_n \models eff_R$ with eff_n the effect of the process, and
3. $(pre_R \cup s_m) \models pre_i$ for all activity instances i and according markings m with $\exists(m, m') \in E$ and $l_E(l_s((m, m'))) = i$.

Checking the first two properties is rather straightforward using the algorithms from the previous section to calculate pre_n and eff_n . The third property is more complicated. For all activities in the process we need to determine the markings in which they are invocable ($l_E(l_s((m, m'))) = i$), calculate the logical state of the the marking (s_m) and check whether this logical state together with the precondition of the process specification entails the precondition of the activity ($(pre_R \cup s_m) \models pre_i$).

One might question whether the last property is actually necessary. We are already checking whether the process has any preconditions that are not satisfied in the process specification. Depending on the expressiveness of the used logical formalism, checking the third property can be unnecessary. This is for example the case if the logical formalism does not support negation. Then every time property one is violated, property three is violated, too. The same is true for the other direction. The only use of the third property without negation is that it gives detailed hints where the problem lies.

With more expressive logical formalisms, the third property becomes crucial. Then the first property is only a necessary (but not sufficient) criterion for the third property. If the precondition of the process is not satisfied by the process specification we know that the precondition of at least one activity instance is not satisfied. But it can be the case that the precondition of one activity instance is not satisfied while still the precondition of the overall process is satisfied. How this can happen, will be illustrated in the next section.

3.1 Example

Let us now look at an example illustrating how semantic conformance can be checked. We will use an example process to demonstrate subsequently violations of each property. The example we will use is a very simple order shipment process. After the order is received, the receipt and the actual goods are shipped separately and finally the order is closed. Shipping of the goods can be performed by one of two different shippers. Let us first specify the specification for this process. The precondition of the specification is $shipper_1 \vee shipper_2$ meaning that the customer must have specified which shipper to use. The effect of the specification is $receipt_sent \wedge shipped$ meaning that the receipt should be send and the goods should be on their way to the customer.

Transition	Precondition	Effect
order		<i>ordered</i>
send_receipt	<i>ordered</i>	<i>receipt_sent</i>
shipping1	$ordered \wedge shipper_1$	<i>shipped</i>
shipping2	$ordered \wedge shipper_2$	<i>shipped</i>
close	$receipt_sent \wedge shipped$	<i>order_closed</i>

Table 1. Preconditions and effects of the activities.

Now let us look at a first example. In Table 1 the preconditions and effects of each activity are specified. Figure 1 illustrates our first try at modeling a correct process. But checking whether it fulfills the process specification's effect will actually show us that it is not sending the receipt.

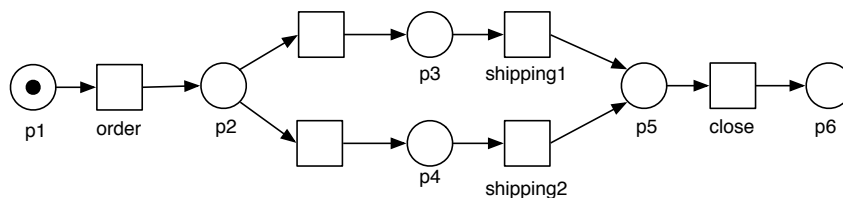


Fig. 1. First version of the modeled process.

We start with the final marking in which the only token is in the place after *order* and traverse the reachability graph backwards until we reach the initial marking (the one depicted). The reachability graph in this case resembles the process: it contains one marking for each place in the process. And two markings are connected if the according places are connected via a transition. Hence we know that a marking s_{m_i} is the marking in which the token is in place p_i . When calculating the effect we start with $\gamma(s_{m_5}, close)$ and continue until we reach s_{m_1} : $\gamma(s_{m_5}) = ordered_closed \wedge (\gamma(s_{m_3}, shipping_1) \vee$

$\gamma(s_{m_4}, shipping_2) = (order_closed \wedge shipped \wedge \gamma(s_{m_1}, order)) \vee (order_closed \wedge shipped \wedge \gamma(s_{m_1}, order)) = order_closed \wedge shipped \wedge ordered$. If we check this against the effect of the process specification we see that we do not achieve *receipt_sent*. Hence the process is not complete. We will actually achieve the same result when checking for the precondition. Because the order may only be closed if the goods have been shipped and the receipt has been send.

In Figure 2 an updated version of the process is displayed. It now contains the necessary sending of the receipt as well as a new activity: package gift. The idea is that we want to give each customer for a limited time a gift (to apologize for the delayed receipt sending). The new error we introduced is apparent: we only package the gift after we already shipped the goods. This will not work. To check this formally we need to know the precondition of packaging a gift. It is $\neg shipped$.

Calculating the precondition and checking it against the precondition of the specification will not help us here. If negation as failure is used, the precondition of the process is $shipper_1 \vee shipper_2$ ¹ Therefore we need to check individually for each activity whether its precondition is satisfied. As we already know that the problem is packaging the gift, we will only check the precondition for this activity. Packaging the gift has exactly one preceding marking, namely the one where there are tokens in p_6 and p_7 . We need to calculate the logical state for this marking and check whether it entails the precondition of packaging the gift. The logical state is $ordered \wedge receipt_sent \wedge shipped$. And it is $ordered \wedge receipt_sent \wedge shipped \not\models \neg shipped$. Hence, this activity is not invocable. to correct this process we need to move packaging the gift before shipping the goods.

With the, rather artificial, example in this section we have seen how using semantic conformance we can identify three different errors: unachieved effects, unsatisfied process preconditions, and not invocable activities.

3.2 Complexity

Analyzing the complexity includes analyzing the complexity of checking each property. Complexity largely depends on the expressiveness of the used logical formalism and the complexity of the reasoning tasks. Each property includes the same reasoning tasks:

- Checking for entailment: This is used at the end of precondition and effect calculation to check properties (1) and (2) and done once for each activity instance to check property (3).
- Updating the knowledge base: This is performed at each marking to update the calculated precondition and effect.

Assuming constant efforts for both reasoning tasks² checking all three properties has similar complexity. We also assume that intermediate results are

¹ With classical negation it would be $(shipper_1 \vee shipper_2) \wedge \neg shipped$ but the precondition of the specification would also include $shipper_1 \vee shipper_2$.

² In reality, this is seldom the case. But as we will soon see, the complexity of the reasoning tasks matters hardly.

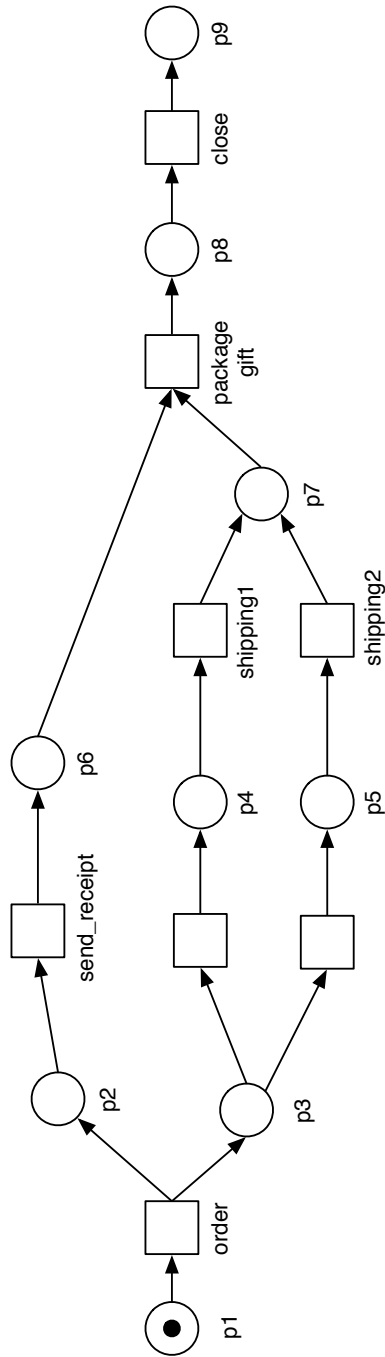


Fig. 2. Second version of the modeled process.

stored. Hence in case of a split and join, preceding markings are only visited once. Then calculating the effect of a process requires traversing the reachability graph exactly once. Calculating the precondition requires traversing the reachability graph twice (once for adding up the preconditions and once for calculating the effects to remove them from the precondition). Hence both precondition and effect calculation are linear in size of the reachability graph. This means that, in the worst case, they are exponential to the size of the process³. Checking the precondition of each contained activity instance to check the third property is actually mostly the same as calculating the precondition of the whole process. The only difference is that we need to perform additional reasoning tasks for each activity. But this does not change the complexity. Hence, complexity is still exponential to the size of the process.

4 Related Work

A lot of work exist regarding the verification of certain process properties. Based on the notion of soundness of Petri nets [3]. A Petri net is sound if it will terminate eventually leaving only a token in the final place and the net does not contain any dead tasks. Several approaches to relax this criterion were invented later on including relaxed soundness [4], weak soundness [6], and lazy soundness [7]. All have in common that they only verify syntactic properties of the process.

Additionally, all these soundness properties are properties of the process itself and not according to a specification or description. Our approach has much more in common with notions of conformance and consistency from choreography research [5, 6, 8]. In these approaches it is tested whether the concrete implementation matches the behavior specified in the interface. The interface can be the partner description of a choreography. Again these approaches only work on the syntactical level. They check whether the right action is performed at the right time. But to be the right action it has to have the same syntactical structure (e.g. same name) as in the specification. This approach is actually quite promising if you want to check whether to protocol (the sent and received messages) is adhered to. But for our use case it does not help much, because the process specification needs to model the whole process to capture all functionality. Our approach allows for structurally vastly different implementations yet achieving the same goal. But it cannot check whether the communication protocol of the process is correct.

Matteo Baldoni et al. [12] use a quite different approach to reach a similar goal. They model the interaction protocol of services using logical programming to perform reasoning to select and compose them. This approach could most probably be used as well to check semantic conformance. While expressing the interaction protocol using logical programming makes Petri nets unnecessary leading therefore to a more coherent implementation, we still favor the Petri net

³ Size of a process $n = (P, T, f, l_s)$ is $|f|$

approach. It allows us to take up the tremendous amount of work on expressing processes as Petri nets and verifying certain properties. Hence we can use the related work presented earlier to check soundness and the adherence to a predefined communication protocol.

5 Summary

In this paper, we presented the novel property of semantic conformance. It enable a process designer to check whether a process is not only syntactically correct but also achieves the the intended functionality. A process is conformant to a request template if the precondition of the process if entailed in the precondition of the specification, the effect of the process entails the effect of the specification, and each activity is invocable whenever its invocation is possible.

In the future we want to extend the presented approach in two directions. On the one hand it should allow for checking more complex processes and on the other hand more errors should be detected. To achieve the first goal we want to support cyclic processes in the future. Our current idea to realize this is by defining a fix point semantic for loops. Errors that are currently not detected are for example errors in parallelism. If for example two activities depending on each other or who are in conflict invoked in parallel, the result is undetermined. Finally, we are also working on embedding our approach to check semantic conformance into a larger framework checking not only functional but non-functional properties on a semantic level as well.

References

1. 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) 54–65
2. Schaffner, J., Meyer, H., Weske, M.: A formal model for mixed initiative service composition. In: Proceedings of The IEEE International Conference on Services Computing (SCC 2007). (2007)
3. van der Aalst, W.: The application of petri nets to workflow management. The Journal of Circuits, Systems and Computers **8(1)** (1998) 21–66
4. Dehnert, J., Rittgen, P.: Relaxed soundness of business processes. In: Proceedings of the 13th International Conference on Advanced Information Systems Engineering. Volume 2008 of Lecture Notes In Computer Science., London, UK, Springer (2001) 157–170
5. Basten, T., van der Aalst, W.: Inheritance of behavior. JLAP **47** (2001) 47–145
6. Martens, A.: On Compatibility of Web Services. Petri Net Newsletter **65** (2003) 12–20
7. Puhmann, F., Weske, M.: Investigations on soundness regarding lazy activities. In Dustdar, S., Fiadeiro, J.L., Sheth, A., eds.: Business Process Management (BPM 2006). Volume 4102 of Lecture Notes In Computer Science., Heidelberg, Springer (2006) 81–96

8. Decker, G., Weske, M.: Behavioral consistency for b2b process integration. In: Proceedings of the 19th International Conference on Advanced Information Systems Engineering (CAISE). (2007)
9. Meyer, H.: On the semantics of service compositions. In: Proceedings of The First International Conference on Web Reasoning and Rule Systems (RR 2007). (2007)
10. Kifer, M., Lara, R., Polleres, A., Zhao, C., Keller, U., Lausen, H., Fensel, D.: A logical framework for web service discovery. In: ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, CEUR Workshop Proceedings. Volume 119., Hiroshima, Japan (2004)
11. Keller, U., Lausen, H., Stollberg, M.: On the semantics of functional descriptions of web services. In: Proceedings of the 3rd European Semantic Web Conference (ESWC), Budva, Montenegro. (2006)
12. Baldoni, M., Baroglio, C., Martellia, A., Pattia, V.: Reasoning about interaction protocols for customizing web service selection and composition. *Journal of Logic and Algebraic Programming* **70** (2007) 53–73