From Public Views to Private Views – Correctness-by-Design for Services

Wil M.P. van der Aalst[1], Niels Lohmann[2,⋆], Peter Massuthe[3], Christian Stahl[3,⋆⋆], and Karsten Wolf[2]

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
W.M.P.v.d.Aalst@tue.nl

[2] Universität Rostock, Institut für Informatik
18051 Rostock, Germany
{niels.lohmann, karsten.wolf}@uni-rostock.de

[3] Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
{massuthe, stahl}@informatik.hu-berlin.de

Abstract. Service orientation is a means for integrating across diverse systems. Each resource, whether an application, system, or trading partner, can be accessed as a service. The resulting architecture, often referred to as SOA, has been an important enabler for interorganizational processes. Apart from technological issues that need to be addressed, it is important that all parties involved in such processes agree on the "rules of engagement". Therefore, we propose to use a *contract* that specifies the composition of the *public views* of all participating parties. Each party may then implement its part of the contract such that the implementation (i.e., the *private view*) accords with the contract. In this paper, we define a suitable notion of *accordance* inspired by the asynchronous nature of services. Moreover, we present several *transformation rules* for incrementally building a private view such that accordance with the contract is guaranteed by construction. These rules include adding internal tasks as well as the reordering of messages and are therefore much more powerful than existing correctness-preserving transformation rules.

1 Introduction

Interorganizational cooperation is of increasing importance for enterprises to meet the new challenges of ever faster changing business conditions. Web services and service-oriented architectures (SOA) are rapidly emerging approaches to reduce the complexity of integrating systems within and between organizations. Since SOA enables dynamic binding of services at runtime, it is possible to

⋆ Funded by the German Federal Ministry of Education and Research (project Tools4BPEL, project number 01ISE08).

⋆⋆ Funded by the DFG project "Substitutability of Services" (RE 834/16-1).

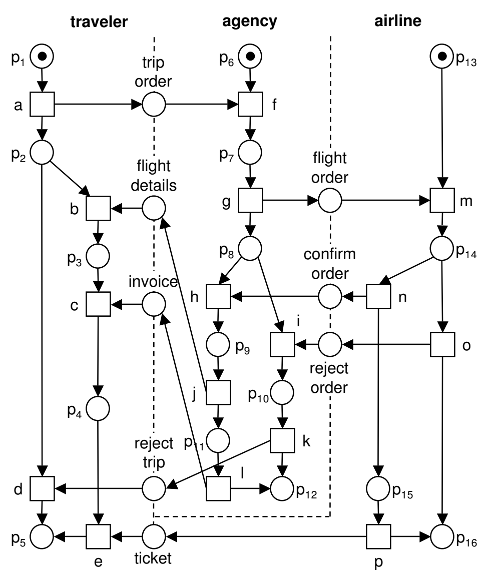
design a cooperation between parties that do not know each other. However, in practise the parties involved in a cooperation know each other. Therefore, instead of dynamically binding services at runtime, these parties agree on a common *contract*, which is the focus of this paper. This contract has the form of an agreed upon process model, similar to [1, 2], and attempts to balance the following two conflicting requirements. On the one hand, there is a strong need for coordination to optimize the flow of work in and between the different organizations. On the other hand, the organizations involved are essentially autonomous and have the freedom to create or modify workflows at any point in time. Therefore, we propose to use a process-oriented contract that defines "rules of engagement" without describing the internal processes executed within each partner.

To illustrate the idea of having a process-oriented contract, we use an example taken from [3], depicted in Fig. 1(a). The example shows a contract expressed in terms of a Petri net [4]. The Petri net is partitioned over three *parties*: traveler, agency, and airline. Each party has a part of the contract which can be seen as a *service*. Different services are connected through *interface places* for asynchronous message passing. Interface places model message buffers and are positioned on dashed lines as shown in Fig. 1(a). As a formalism, we use *open workflow nets* (oWFNs) [5] which extend the well-known concept of *workflow nets* (WFNs) [6] with interface places. However, the presented concepts are not limited to oWFNs and can be translated into other languages using message passing as a communication paradigm.

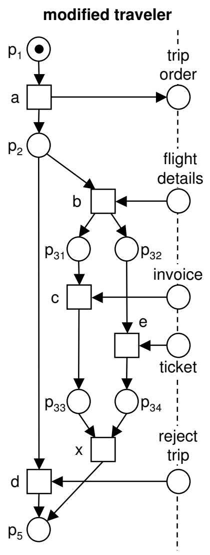
In our example, the traveler sends a trip order to the agency (transition a). As a result, the agency sends a flight order to the airline (transition g). The airline receives this order and either confirms it (message confirm order) or rejects it (message reject order). In the latter case, the rejection is forwarded to the traveler (transition k). If the flight is confirmed, the agency sends the flight details and an invoice to the traveler, and the airline sends a ticket to the traveler.

Figure 1(a) shows four oWFNs: N, N_{tr}, N_{ag}, and N_{air}. N_{tr}, N_{ag}, and N_{air} specify the *public view* of each of the parties involved. Each of the public views has interface places. For example, N_{tr} has one output place and four input places. Using these places, the public views can be merged together into the *contract*. The whole Petri net shown in Fig. 1(a) can be seen as a single oWFN N by simply ignoring the dashed lines. This oWFN has an empty interface.

After the parties agreed on such a contract, each of them needs to implement its part. A party needs to refine its part of the contract, so the resulting implementation may deviate significantly from the public view. We refer to this as the *private view* of the party. The private view may again be expressed as an oWFN. Figure 1(b) shows an example of a private view of the traveler; that is, oWFN N'_{tr} is the implementation of N_{tr}. In N_{tr}, the traveler first receives the invoice and then the ticket. In N'_{tr}, these two messages are received concurrently. Note that this example is a bit atypical since the implementation tends to have much more internal tasks. Here just transition x and places p_{31} to p_{34} are added while in more realistic scenarios there may dozens of newly added internal tasks. Clearly, N'_{tr} allows for behavior not possible in N_{tr} (even after



(a) oWFN N modeling the contract between traveler (N_{tr}), agency (N_{ag}), and airline (N_{air}).



(b) oWFN of the modified traveler (N'_{tr}).

Fig. 1. The running example.

abstracting from the new transition x). The trace a, b, e, c, x where the ticket is received before the invoice, for instance, is not possible in N_{tr} but in N'_{tr}. In this example it is harmless that N'_{tr} serves as implementation of N_{tr}. However, similar changes could lead to deadlocks and other problems. On this account, a comprehensive set of transformation rules for deriving a private view, which is correct by construction, from a public view would be very helpful for service designers.

In earlier work, we proposed to use *projection inheritance* for WFNs [7, 8] for relating the actual realization of a contract to the contract itself [1, 2]. It was proven that if private and public view are related by projection inheritance, then a party can execute its private view and no other party is effected by this change. Moreover, we defined *inheritance-preserving transformation rules* that guarantee correctness by construction; that is, the public view is extended into a correct private view by iteratively applying the rules. Unfortunately, N'_{tr} and N_{tr} are *not* related by projection inheritance. This illustrates that the inheritance notion defined is too restrictive, since it excludes private views that obviously do not jeopardize the overall correctness of the interorganizational workflow.

To address the problem, we present a more liberal notion of equivalence: *accordance*. The basic idea is that an oWFN N_2 accords with an oWFN N_1 if there is no interorganizational workflow where the replacement of oWFN N_1 by oWFN N_2 causes a problem. Accordance is weaker than projection inheritance. For example, N'_{tr} accords with N_{tr}.

The core contribution of this paper is a comprehensive set of *accordance preserving transformation rules*. They can be used to incrementally transform the public view of a party into a private view while guaranteeing that the overall process will terminate properly. These new rules are highly relevant because there are many situations where projection inheritance is too strong and accordance is a more suitable notion. In case both public and private view are given, a technique presented in [9] can be used for automatically checking accordance by using operating guidelines [5].

The paper is structured as follows. Section 2 defines oWFNs, contracts, and our accordance criterion. In Sect. 3, we present transformation rules to derive a private view which is correct by construction. A case study in Sect. 4 demonstrates the applicability and the value of these transformation rules.

2 Formalizing Contracts

The notions of a contract, public/private views, the accordance criterion, and transformation rules will be formalized using *open workflow nets* (oWFNs) [5]. Therefore, this section starts by introducing oWFNs. For more details on the formalization of these concepts we refer to a technical report [9].

We use the usual definition of a (place/transition) Petri net $N = (P, T, F)$ with a set of places P, a set of transitions T, and a flow relation $F \subseteq (P \times T) \cup (T \times P)$ representing the arcs (see [4], for instance). We also use the standard notation to denote the preset and postset of places and transitions: ${}^{\bullet}x = \{y \mid (y, x) \in F\}$ and $x^{\bullet} = \{y \mid (x, y) \in F\}$.

Definition 1 (Open workflow net). *An* open workflow net $N = (P, T, F, I, O, m_0, \Omega)$ *consists of a Petri net* (P, T, F) *together with*

- *an* interface *defined as a set* $I \subseteq P$ *of* input places *such that* ${}^{\bullet}p = \emptyset$ *for any* $p \in I$ *and a set* $O \subseteq P$ *of* output places *such that* $p^{\bullet} = \emptyset$ *for any* $p \in O$ *and* $I \cap O = \emptyset$,
- *a distinguished* initial marking m_0*, and a set* Ω *of* final markings *such that no transition of* N *is enabled in any* $m \in \Omega$*. We further require that* $m \in \Omega \cup \{m_0\}$ *implies* $m(p) = 0$ *for all* $p \in I \cup O$*; that is, in the initial and the final markings, the interface places are not marked.*

We use indices to distinguish the constituents of different oWFNs (e.g., I_j refers to the set of input places of oWFN N_j). In order to assign a reasonable meaning to *final* markings, we restrict our approach to such oWFNs where a marking in Ω does not enable any transition.

As an example, the whole process shown in Fig. 1(a) represents an oWFN with $I = O = \emptyset$, $m_0 = [\mathsf{p}_1, \mathsf{p}_6, \mathsf{p}_{13}]$, and we define $\Omega = \{[\mathsf{p}_5, \mathsf{p}_{12}, \mathsf{p}_{16}]\}$. The part of the traveler, N_{tr}, in Fig. 1(a) is an oWFN with interface: $I = \{\mathsf{flight\ details}, \mathsf{invoice}, \mathsf{reject\ trip}, \mathsf{ticket}\}$ and $O = \{\mathsf{trip\ order}\}$.

The behavior of an oWFN is defined using standard Petri net semantics [4]; that is, a transition is enabled if each place of its preset holds at least a token. An enabled transition t can fire in a marking m by consuming tokens from the preset places and producing tokens for the postset places, yielding a marking m' (denoted $m \xrightarrow{t} m'$).

For composing oWFNs, we assume that all constituents (except the interfaces) are pairwise disjoint. This requirement can be easily achieved by renaming. In contrast, the interfaces often intentionally overlap. For a reasonable concept of composition of oWFNs it is, however, convenient to require that all communication is bilateral; that is, every interface place $p \in I \cup O$ has only one party that sends into p and one party that receives from p. For a third party C, a communication taking place inside the composition of parties A and B is internal matter. These considerations lead to the following definition of composition.

Definition 2 (Composition of oWFNs). *Let $N_1, \ldots, N_k$ be oWFNs with pairwise disjoint constituents, except for the interfaces. $N_1, \ldots, N_k$ are* composable *if, for all $i \in \{1, \ldots, k\}$,*

- *$p \in I_i$ implies that there is no $j \neq i$ such that $p \in I_j$ and there is at most one j such that $p \in O_j$, and*
- *$p \in O_i$ implies that there is no $j \neq i$ such that $p \in O_j$ and there is at most one j such that $p \in I_j$.*

For markings $m_1 \in N_1, \ldots, m_k \in N_k$ which do not mark interface places, their composition $m = m_1 \oplus \cdots \oplus m_k$ is defined by $m(p) = m_i(p)$ if $p \in P_i$.

If $N_1, \ldots, N_k$ are composable, the composition *$N = N_1 \oplus \cdots \oplus N_k$ is the oWFN with the following constituents: $P = P_1 \cup \cdots \cup P_k$. $T = T_1 \cup \cdots \cup T_k$. $F = F_1 \cup \cdots \cup F_k$. $I = (I_1 \cup \cdots \cup I_k) \setminus (O_1 \cup \cdots \cup O_k)$. $O = (O_1 \cup \cdots \cup O_k) \setminus (I_1 \cup \cdots \cup I_k)$. $m_0 = m_{0_1} \oplus \cdots \oplus m_{0_k}$, $\Omega = \{m_1 \oplus \cdots \oplus m_k \mid m_1 \in \Omega_1, \ldots, m_k \in \Omega_k\}$.*

Clearly, the three oWFNs N_{tr}, N_{ag}, and N_{air} in Fig. 1(a) are composable.

Any subset of a set of composable oWFNs is composable as well. Furthermore, we have $N_1 \oplus N_2 \oplus N_3 = (N_1 \oplus N_2) \oplus N_3 = N_1 \oplus (N_2 \oplus N_3)$, and $N_1 \oplus N_2 = N_2 \oplus N_1$; that is, the composition of composable oWFNs is associative and commutative. Thus, composition of a set of oWFNs can be broken into single steps without affecting the final result.

For the oWFN depicted in Fig. 1(a) it is easy to check that the final marking is always reachable. This means that it is always possible to terminate properly. This property is formalized in the following definition.

Definition 3 (Weak termination). *An oWFN* weakly terminates *if, from every marking reachable from the initial marking, a final marking can be reached.*

For composable oWFNs whose composition is weakly terminating, we introduce the term strategy.

Definition 4 (Strategy). *An oWFN N is a* strategy *for an oWFN N' if $N \oplus N'$ is weakly terminating. $Strat(N)$ denotes the set of all strategies for N.*

Note that $\mathit{Strat}(N)$ may correspond to a large (in fact infinite) set of oWFNs; that is, it is the set of all potential partners of N. N_{tr} in Fig. 1(a) and $\mathsf{N}'_{\mathsf{tr}}$ in Fig.1(b) are two examples of strategies for the oWFN $\mathsf{N}_{\mathsf{ag}} \oplus \mathsf{N}_{\mathsf{air}}$.

Basically, we see a contract as an oWFN with empty interface where every activity is assigned to one of the involved parties. We impose only one restriction: If a place is accessed by more than one party, it should act as a directed bilateral communication place. In the following, $|X|$ denotes the cardinality of a set X.

Definition 5 (Contract). *Let $\mathcal{A}$ be a set representing the parties involved in a contract. Then, a* contract $[N, r]$ *consists of an oWFN $N = (P, T, F, I, O, m_0, \Omega)$ with an empty interface ($I = O = \emptyset$) (the overall process) and a mapping $r \in T \to \mathcal{A}$ (the partitioning) such that, for all places $p \in P$, $|\{r(t) \mid t \in {}^\bullet p\}| \leq 1$ and $|\{r(t) \mid t \in p^\bullet\}| \leq 1$. For technical purposes, we further require that N has only one final marking, $\Omega = \{m_f\}$.*

The oWFN shown in Fig. 1(a) is an example of a contract involving $\mathcal{A} = \{\mathsf{traveler}, \mathsf{agency}, \mathsf{airline}\}$. The dashed lines in the figure show the partitioning of transitions over the parties involved in the contract; $r(\mathsf{a}) = \mathsf{traveler}$, $r(\mathsf{f}) = \mathsf{agency}$ and $r(\mathsf{m}) = \mathsf{airline}$, for instance.

A contract can be cut into parts, each representing the agreed share of a single party. In accordance with terminology of service-oriented computing [10], we consider the contribution of a party as a *service*. Correspondingly, the agreed version (specification) of the service is called *public view* while an actual local implementation is called *private view* of the service.

Definition 6 (Public view). *Let $[N, r]$ be a contract with $N = (P, T, F, I, O, m_0, \Omega)$, $\Omega = \{m_f\}$, and $r \in T \to \mathcal{A}$, and let $A \in \mathcal{A}$ be a party. The* public view *of A's share in the contract is the oWFN N_A where $P_A = \{p \in P \mid \exists t \in {}^\bullet p \cup p^\bullet : r(t) = A\}$, $T_A = \{t \in T \mid r(t) = A\}$, $F_A = F \cap ((P_A \times T_A) \cup (T_A \times P_A))$, $I_A = \{p \in P_A \mid \exists t \in {}^\bullet p : r(t) \neq A\}$, $O_A = \{p \in P_A \mid \exists t \in p^\bullet : r(t) \neq A\}$, $m_{0_A} = m_{0|P_A}$ (i.e., the restriction of m_0 to the places in P_A), and $\Omega_A = \{m_{f|P_A}\}$.*

For a set $\mathcal{A} = \{A_1, \ldots, A_k\}$ of parties and a contract $[N, r]$, it is easy to see that $N_{A_1} \oplus \cdots \oplus N_{A_k} = N$. In this respect, the restriction that Ω contains only one element is indeed crucial, as otherwise $N_{A_1} \oplus \cdots \oplus N_{A_k}$ could have a final marking that results from recombining final markings of different parties but which is not a final marking of N.

Our *accordance* criterion is used to compare a public view and a private view of a party's share of a contract. The goal of the accordance notion is to preserve weak termination (see Def. 3) of the overall process N. Formally, weak termination of N and accordance of each private view N'_{A_i} with the corresponding public view N_{A_i} should imply weak termination of $N'_{A_1} \oplus \cdots \oplus N'_{A_k}$ which obviously models the overall process as actually implemented.

If $[N, r]$ is a contract with $\mathcal{A} = \{A_1, \ldots, A_k\}$ and N is weakly terminating, then $N_{A_1} \oplus \ldots \oplus N_{A_{i-1}} \oplus N_{A_{i+1}} \oplus \ldots \oplus N_{A_k}$ is a strategy for N_{A_i}. For example, $\mathsf{N}_{\mathsf{tr}} \oplus \mathsf{N}_{\mathsf{ag}} \oplus \mathsf{N}_{\mathsf{air}}$ shown in Fig. 1(a) is weakly terminating. Therefore, $\mathsf{N}_{\mathsf{tr}} \oplus \mathsf{N}_{\mathsf{air}}$ is a strategy for N_{ag}, and vice versa. These properties of the strategy concept justify the following definition of accordance.

Definition 7 (Accordance). *An oWFN N' (private view)* accords *with an oWFN N (public view) if it has the same interface ($I' = I$ and $O' = O$) and has at least the strategies that N has; that is, $\mathit{Strat}(N') \supseteq \mathit{Strat}(N)$.*

For example, the private view N'_{tr} accords with its public view N_{tr}. The following theorem shows that N_{tr} can be substituted by N'_{tr} without jeopardizing weak termination.

Theorem 1 (Implementation of a contract). *Let $[N, r]$ be a contract between parties $\{A_1, \ldots, A_k\}$ where N is weakly terminating. If, for all $i \in \{1, \ldots, k\}$, N'_{A_i} (the private view of A_i) accords with N_{A_i} (the public view of A_i), then $N' = N'_{A_1} \oplus \cdots \oplus N'_{A_k}$ (the actual implementation) is weakly terminating.*

The proof of this theorem can be found in [9]. The result is highly relevant for service composition since it gives each party a criterion (accordance of N'_{A_i} with N_{A_i}) that can be locally verified for asserting a global property (weak termination of the overall process as actually implemented). For example, any combination of arbitrary private views N''_{tr}, N''_{ag}, and N''_{air} according with the corresponding public view (i.e., N''_{tr} accords with N_{tr}, N''_{ag} accords with N_{ag}, and N''_{air} accords with N_{air}) yields a weakly terminating realization of the contract shown in Fig. 1(a).

According to Thm. 1, every party of a contract can implement its public view and finally it has to check accordance between the private and the public view. In the following, we present a different approach: The public view is incrementally transformed into a private view. To this end, fragments of the public view are incrementally replaced by other fragments until the private view is designed. In this approach, a fragment N' of a party is called a *pattern* and will be replaced by another fragment N''. We will prove that if N'' accords with N', then replacing N' by N'' preserves weak termination of the overall contract.

First of all, we formally define an oWFN pattern N' of an oWFN N. Therefore, the set of interface places of N' is divided into two sets: one set contains all places that are interface places of N for communicating with other parties (i.e., subsets of I and O) and the other set, $R \cup S$, contains all places that serve as an interface to the rest of N. R is the set of input places from the other parts of N, and S is the set of output places.

Definition 8 (oWFN pattern). *Let $N = (P, T, F, I, O, m_0, \Omega)$ be an oWFN. An oWFN $N' = (P', T', F', I', O', m'_0, \Omega')$ with $P' \subseteq P$, $T' \subseteq T$ is an* oWFN pattern *of N iff*

- $F' = F \cap ((P' \times T') \cup (T' \times P'))$,
- $m'_0 = [\,]$,
- $I' = I_{|P'} \cup R$ *with* $R \subseteq P' \setminus I$,
- $O' = O_{|P'} \cup S$ *with* $S \subseteq P' \setminus O$,
- $\Omega' = \{[\,]\}$,
- *for all* $p \in P' \setminus R$, *there is no* $t \in T \setminus T'$, $(t, p) \in F$,

– *for all $p \in P' \setminus S$, there is no $t \in T \setminus T'$, $(p,t) \in F$, and*
– *for all $t \in T'$, there is no $p \in P \setminus P'$, $(p,t) \in F$ or $(t,p) \in F$.*

The next theorem states that if the public view of a party participating in a contract has an oWFN pattern N' and there is another oWFN pattern N'' with N'' accords with N', then we can replace N' by N'' and the modified contract is still weakly terminating. Such transformations can be applied incrementally and thus we can derive a private view from a public view just by transforming the public view and the resulting private view is correct by construction.

Theorem 2 (Justification of transformation rules). *Let $[N, r]$ be a contract between parties $\{A_1, \dots, A_k\}$ where $N = N_{A_1} \oplus \cdots \oplus N_{A_k}$ is weakly terminating. Let N'_p be an oWFN pattern of N_{A_i}, $1 \le i \le k$, such that there exists N_{rest} with $N_{A_i} = N'_p \oplus N_{rest}$. Let further N''_p be an arbitrary oWFN. Then, if N''_p accords with N'_p, the modified contract $N' = N_{A_1} \oplus \cdots \oplus N_{A_{i-1}} \oplus (N''_p \oplus N_{rest}) \oplus N_{A_{i+1}} \oplus \cdots \oplus N_{A_k}$ is weakly terminating.*

We omit the proof of this theorem as it is just an application of Thm. 1.

3 Derive a Private View From a Public View

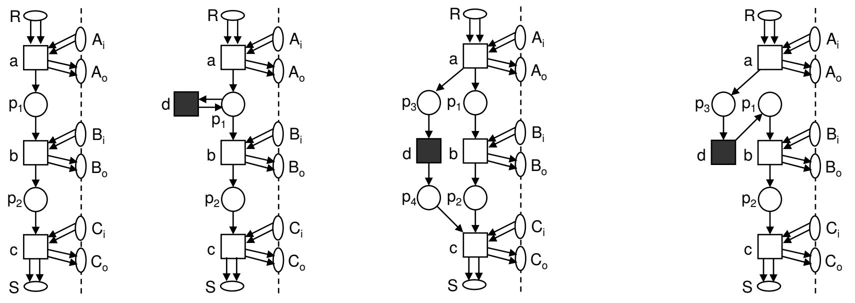
In this section, we show how a party can implement its private view by using *accordance-preserving transformation rules*. This idea is inspired by earlier work on projection inheritance [1, 2, 7, 8]. Accordance is a weaker notion than projection inheritance which was illustrated already using Fig. 1 where N'_{tr} accords with N_{tr} but N'_{tr} and N_{tr} are not related by projection inheritance. However, we will show that projection inheritance implies accordance, and thus, all inheritance-preserving transformation rules presented in [8] also preserve accordance. We will show these rules by reformulating them to fit into the setting of this paper. Afterwards, we will formulate dedicated transformation rules that allow reordering of the sending and receiving of messages still guaranteeing accordance.

3.1 Inheritance-Preserving Transformation Rules

Projection inheritance compares process models by establishing a subclass-superclass relationship. The subclass process is indeed a subclass if it inherits particular dynamic properties of its superclass. Projection inheritance is based on branching bisimulation [11] (to compare the processes) and abstraction (to hide tasks) and was formalized in [8] in terms of workflow nets. The assumption is that the subclass adds tasks to the superclass such that after hiding the additional tasks both are equivalent.

Based on the notion of projection inheritance, three *inheritance-preserving transformation rules* have been defined in [8]. These rules correspond to design patterns for extending a superclass to incorporate new behavior: (1) adding a loops (2) insert a task in-between existing tasks, and (3) put a new task in parallel with existing tasks.

It is easy to reformulate projection inheritance in terms of the setting of this paper. Instead of redefining these rules formally, we exemplify the rules in Fig. 2. Figure 2(a) represents an oWFN pattern M_0 of an oWFN M. M_0 contains transitions a, b, and c. By Def. 8, there are no other connections of a, b, c, p_1, and p_2 than those shown in Fig. 2(a). $\mathsf{A_i} = ({}^\bullet\mathsf{a}) \cap I_M$ is the set of input places of a, $\mathsf{A_o} = (\mathsf{a}^\bullet) \cap O_M$ is the set of output places of a, etc. $\mathsf{A_i}$, $\mathsf{A_o}$, $\mathsf{B_i}$, $\mathsf{B_o}$, $\mathsf{C_i}$, $\mathsf{C_o}$ do not need to be disjoint. $\mathsf{R} = ({}^\bullet\mathsf{a}) \setminus I_M$ and $\mathsf{S} = (\mathsf{c}^\bullet) \setminus O_M$ are (by Def. 8) the places connecting M_0 to the rest of M. Similar remarks hold for the other three oWFN patterns M_1, M_2, and M_3. For example, M_1 is obtained by adding transition d to M_0.



(a) M_0. (b) M_1: Adding a loop to M_0. (c) M_2: Putting task d in parallel to b. (d) M_3: Inserting task d in-between a and b.

Fig. 2. Accordance-preserving transformation rules based on projection inheritance.

If one ignores the interface places and hides transition d (i.e., all executions of d are mapped onto silent steps), then M_0, M_1, M_2, and M_3 are branching bisimular. Thus, M_0 is a superclass of M_1, M_2, and M_3. It is easy to see that projection inheritance implies accordance.

Theorem 3 (Projection inheritance implies accordance). *Let N' be an oWFN pattern and N'' be an arbitrary oWFN. If N' and N'' are related by projection inheritance, then N'' accords with N' and vice versa.*

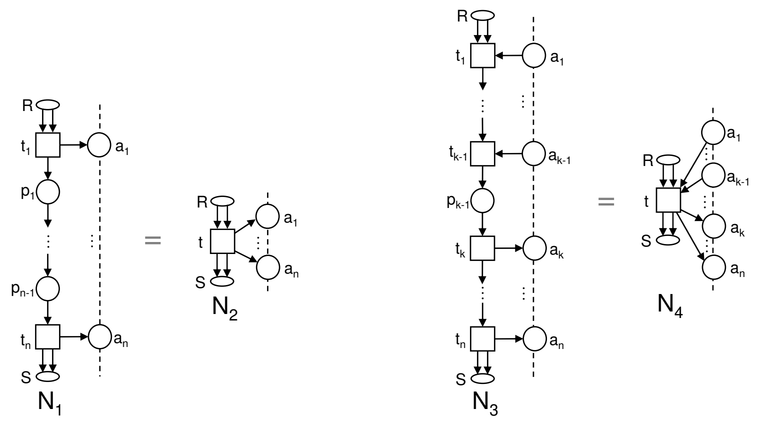
Theorem 3 justifies that all inheritance-preserving transformation rules can be used to incrementally build a private view that accords with the public view of a service. As an example, since M_1 accords with M_0, M_2 accords with M_0, M_3 accords with M_0, and vice versa, M_0 in Fig. 2 may be replaced by any of the three other oWFNs M_1, M_2, and M_3 without changing the set of strategies of M. For technical details we refer to the technical report [9].

3.2 Accordance-Preserving Transformation Rules

The inheritance-preserving transformation rules presented in the last section are limited in the sense that they do not allow to change the order of messages. In the following, we present six *accordance-preserving transformation rules*. Five of these rules preserve accordance in both directions and one rule preserves accordance only in one direction. Although these transformation rules are powerful, they are not complete, meaning they do not cover all possible service implementations. Given an oWFN N, each transformation rule specifies a pattern N' of N (see Def. 8) which can be replaced by another oWFN N'' yielding an implementation of N. Theorem 2 justifies that this replacement does not violate the overall contract. As a formal definition of all transformation rules would not add any value to the paper and is also impossible due to the page limit, the rules are only informally described and illustrated by help of some figures. For the formalization of all rules including their correctness proofs we refer to the technical report [9].

The first of the rules is depicted in Fig. 3(a) and specifies that a sequence of sending events can be merged and the events can be sent simultaneously. Rule 1 preserves accordance in both directions. Thus, we can derive that a sequence of sending events can also be reordered or can be sent concurrently. Reordering of sending events and executing sending events concurrently preserve accordance in both directions. The same holds for a sequence of receiving events. The corresponding rule (Rule 2) is, however, not depicted in the paper.

A generalization of the two previous rules is specified by Rule 3 (see Fig. 3(b)). A sequence of receiving events followed by a sequence of sending events can be executed simultaneously while preserving accordance in both directions.



(a) Rule 1: $\mathit{Strat}(N_1) = \mathit{Strat}(N_2)$. (b) Rule 3: $\mathit{Strat}(N_3) = \mathit{Strat}(N_4)$.

Fig. 3. Rule 1 and Rule 3.

From Rules 1–3 we can derive that every oWFN pattern that has a transition connected to more than one interface place can be transformed into an equivalent oWFN pattern which has only transitions connected to a single interface place.

In the following, without loss of generality, we therefore restrict ourselves to patterns where each transition is connected to at most one interface place.

So far, we excluded the possibility that a sending event is followed by a receiving event. Rule 4, depicted in Fig. 4, specifies that first sending and then receiving a message can also be executed concurrently and vice versa. Rule 4 preserves accordance in both directions, too.

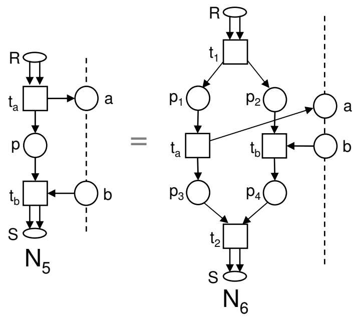


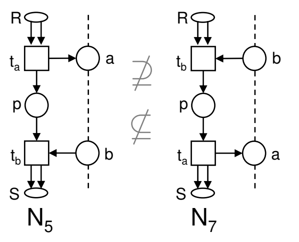
Fig. 4. Rule 4: $Strat(N_5) = Strat(N_6)$.

Figure 5(a) shows that first sending and then receiving cannot be reordered in general: N_7 does not accord with N_5 and N_5 does not accord with N_7. The oWFN depicted in Fig. 5(b) is a strategy for N_5 but no strategy for N_7. The oWFN depicted in Fig. 5(c), in contrast, is a strategy for N_7 but not for N_5.

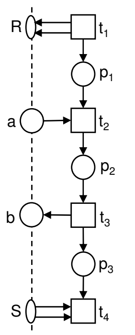
From this antipattern follows that first sending and then receiving (cf. N_5) cannot be transformed into an oWFN that sends and receives simultaneously, because we could transform the latter net into N_7 by applying Rule 3. Consequently, first sending then receiving does not accord to sending and receiving simultaneously and vice versa.

Rule 5 specifies how an alternative branch can be added to an oWFN pattern N_8 depicted on the left hand side of Fig. 6. The pattern N_8 first receives a and then enters either the left or the right branch. In the left (right) branch, message b (c) is sent, and then message d (e) is received . The pattern N_8 can be transformed into N_9 by adding an alternative branch. In this branch, d is received, and then a message f is sent. Afterwards, this branch can be arbitrary; that is, there can be any continuation (including direct continuation in S) of this net illustrated by the frame. Rule 5 preserves accordance in one direction only.

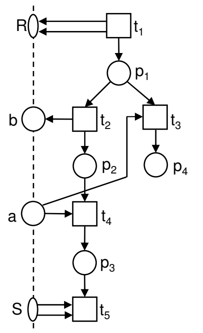
The intuition behind the next transformation rule (Rule 6) is the possibility to add (remove) "dead code" to (from) a service. To motivate this transformation rule, consider a party that wants to reuse an existing service in the contract. This service may provide functionality to other parties not involved in the current contract. Technically, in the first step, this party makes internal all interface places of this service that are not used, and in the second step, it looks for transformation rules justifying the service to be a valid private view. Rule 6 is



(a) Antipattern: *Strat*(N_5) $\neq$ *Strat*(N_7).



(b) Strategy for N_5 but not for N_7.



(c) Strategy for N_7 but not for N_5.

Fig. 5. Counterexamples.

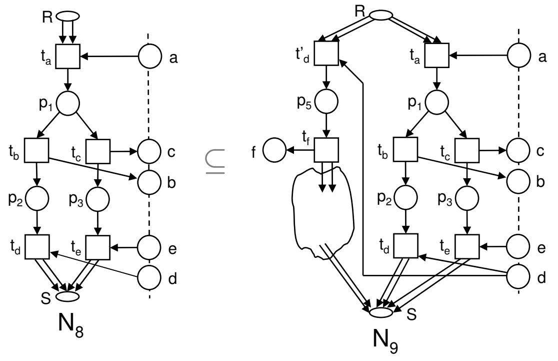


Fig. 6. Rule 5 (adding an alternative branch): *Strat*(N_8) $\subseteq$ *Strat*(N_9).

depicted in Fig. 7. N_{10} receives a, then sends b, and then it can behave arbitrarily. The oWFN pattern N_{11} results from adding an alternative branch to N_{10}. This branch can be entered if place c is marked. Afterwards, the branch may behave arbitrarily. In the end, both branches are synchronized in S. However, c is an example of an internal place with empty preset (it is a former interface place). Thus, transition t_c will never be enabled. Rule 6 preserves accordance in both directions, meaning neither adding nor deleting "dead code" will change the set of strategies for N_{10} and N_{11}.

The six transformation rules presented in this section reflect the crucial impact of the order of sending and receiving messages. The first two rules show that sequences of sending events and sequences of receiving events can be executed simultaneously while preserving accordance in both directions. This was our motivation to consider only oWFNs where each transition is connected to at most one interface place. Transforming first-send-then-receive into send-and-receive-

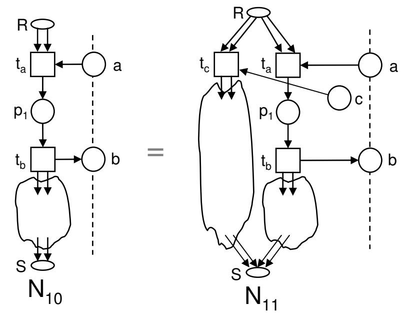


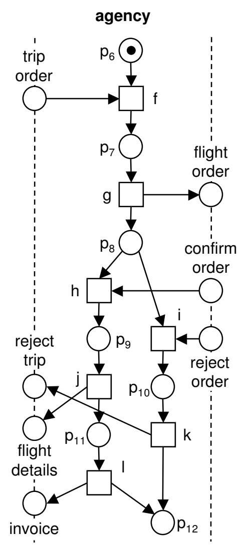
Fig. 7. Rule 6 (adding "dead code"): $\mathit{Strat}(N_{10}) = \mathit{Strat}(N_{11})$.

concurrently preserves accordance in both directions (Fig. 4). However, first-send-then-receive cannot be transformed into send-and-receive-simultaneously and vice versa. Consider first-receive-then-send next. It can be transformed into receive-and-send-simultaneously (Fig. 3(b)) while preserving accordance in both directions, but it cannot be transformed into receive-and-send-concurrently and vice versa.

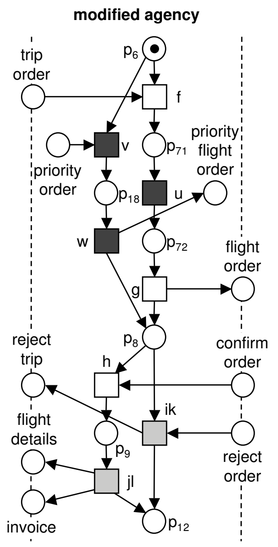
4 Case Study

In this section, we demonstrate how accordance-preserving transformation rules can be applied to derive a private view of the agency from its respective public view, taken from the running example in Fig. 1(a). On first sight, the modified agency depicted in Fig. 8(b) and the (original) agency (depicted again in Fig. 8(a) to ease the comparison) are not very similar. We will now show that the modified agency was derived from the original agency by applying the transformation rules defined in Sect. 3:

– A new transition u (newly-added transitions are depicted in dark gray) is inserted in-between the reception of the trip order and the sending of the flight order. This transition explicitly models the preparation of the flight order from the trip order sent by the traveler. The addition is justified by rule pattern M_3 (cf. Fig. 2(d)) which preserves projection inheritance and thus accordance.
– The rejection message from the airline is instantly routed to the traveler. Rule 3 justifies the merging of transition i and k to the single transition ik (transitions created by merging transitions of the public view are depicted in light gray).
– Messages flight details and invoice can be sent simultaneously to the traveler by transition jl. The merging of transition j and l is justified by Rule 1.
– Finally, a new branch was added to the agency, starting with transition v. Intuitively, this branch models additional behavior that is available when the modified agency service is running in a different environment. When priority



(a) public view



(b) private view

Fig. 8. The public view (a) and a private view (b) of the agency of Fig. 1(a).

order is an input place for messages sent from a (modified) traveler service and priority flight order is an output place for messages sent to a (modified) airline service, the newly-added branch can be triggered by messages. Thus, the modified agency can be reused in a different contract. However, the places priority order and priority flight order are not exposed as interface places, and as the place priority order is not marked, the branch is dead. Therefore, the addition is justified by Rule 6.

As all applied rules are accordance-preserving, the modified agency (cf. Fig. 8(b)) is a correct private view of the agency (cf. Fig. 8(a)), and thus accords with the running example contract (cf. Fig. 1(a)).

5 Conclusion

An interorganizational process couples interacting processes handled by different parties. In this context, a contract serves as an agreement of these parties to a public description of the overall process. Each of the parties implements its part of the contract. These parts correspond to services and are termed "views". Each party has a public view (the part of the process it is responsible for) and the private view (the process actually implemented). The notion of accordance relates these two views and serves as a local correctness criterion. Accordance is particularly suitable for interactions based on asynchronous message passing and the local criterion ensures the correctness of the overall process even if parties do not exactly behave as specified.

In this paper, we presented six *transformation rules* to derive a private view from a public view. As transformation rules preserve accordance between the public and the private view, the private view is correct by construction. Accordance guarantees that the overall process will always terminate properly; that is, the overall process cannot run into a deadlock or livelock. We showed that some of the rules preserve accordance in both directions while other preserve accordance only in one direction. We discussed that the notion of accordance generalizes the notion of projection inheritance [1, 2, 7, 8].

In ongoing work, we look for other correctness criteria than weak termination. Moreover, we want to relate the notion of accordance to other equivalence notions described in literature. Furthermore, an accordance check that returns which rules have to be applied to derive one service from the other one seems to be of practical relevance.

Acknowledgments The authors would like to thank Arjan J. Mooij for pointing out an error in a preliminary version of this paper.

References

1. Aalst, W. van der: Inheritance of Interorganizational Workflows: How to agree to disagree without loosing control? Information Technology and Management Journal **4**(4) (2003) 345–389
2. Aalst, W. van der, Weske, M.: The P2P approach to Interorganizational Workflows. In Dittrich, K., Geppert, A., Norrie, M., eds.: Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01). Volume 2068 of Lecture Notes in Computer Science., Springer (2001) 140–156
3. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: Proceedings of the IEEE International Conference on Web Services (ICWS 2007), IEEE Computer Society (2007) 296–303
4. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer, Berlin, Heidelberg, New York, Tokyo (1985)
5. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1**(3) (2005) 35–43
6. Aalst, W. van der: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
7. Aalst, W. van der, Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. Theor. Comput. Sci. **270**(1-2) (2002) 125–203
8. Basten, T., Aalst, W. van der: Inheritance of Behavior. Journal of Logic and Algebraic Programming **47**(2) (2001) 47–145
9. Aalst, W. van der, Massuthe, P., Stahl, C., Wolf, K.: Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. Informatik-Berichte 213, Humboldt-Universität zu Berlin (2007)
10. Papazoglou, M.: Agent-oriented technology in support of e-business. Commun. ACM **44**(4) (2001) 71–77
11. Glabbeek, R. van, Weijland, W.: Branching Time and Abstraction in Bisimulation Semantics. Journal of the ACM **43**(3) (1996) 555–600