Alignment of choreography changes in BPEL processes

Andreas Wombacher
University of Twente,
7500 AE Enschede, The Netherlands
a.wombacher@utwente.nl

Abstract

Choreography changes performed by one party may affect other parties. The changes and the implication for other parties can be determined. However, the required changes on the orchestration are difficult to determine since a choreography is an abstraction of the orchestration and thus information is lost. In this paper an approach is proposed to enrich the orchestration with the observed changes using a syntactical representation of the orchestration language and semantic invariants of the orchestration language.

1. Introduction

A service composition combines several services each with an internal state. These stateful services may be subject to change over a period of time. A stateful service can be described by its bilateral observable behavior, i.e., choreography per business partner, and its implemented behavior, i.e., orchestration. A choreography contains information about potential message exchanges and their sequences, while an orchestration specifies when a certain message sequence is used. Thus, business critical information is maintained in the orchestration and not shared with business partners, i.e. other services, in a choreography. While a choreography can be derived from an orchestration [16, 15], the construction of an orchestration based on choreographies will always miss business critical information, like e.g. specification of the data flow as well as decision criteria in loops and choices. However, if a service changes, the change will be observable in a choreography only. Further, a change in a choreography may require other services also to adapt their choreographies and accordingly their orchestration.

The issue is how to integrate the information about changes on the level of choreographies into an existing orchestration and explain these changes to the service developer. Currently, a developer of a service has to manually identify what the choreography changes are and how to integrate them in an existing orchestration. Approaches of aligning choreography and orchestration are based on a formal model for orchestration which represents either orchestration language syntax or its semantics. The mainly XML based syntactical representations have the same formal representation for each language construct. However, message sequence oriented semantic representations differ significantly per language construct. Further, operations performed on a semantic representation loose fast the direct correlation to syntactic elements, especially if minimization operations are performed. The approach presented in this paper provides a formal method to semantically represent choreographies and their changes, a syntactical representation of orchestrations, and a set of axioms/invariants to superimpose a semantic representation of an orchestration on its syntactical representation.

An example of a loop invariant is that a loop starts and stops at the same point. Syntactically a loop is represented e.g. by an XML tag. The formal model including the XML tag may be modified in such a way that the tag is contained in the formal model, but the loop does not have anymore the same start and end point. Thus, the semantic sequence indicates that there is no loop while the syntactic loop tag indicates a loop. Using the invariant to connect XML tags to model properties enables to identify syntactic and semantic mismatches and to resolve them. This introduction of invariants and the corresponding alignment of syntactical and semantical representation of an orchestration enables an approach to operationalize the propagation of choreography changes in an existing orchestration by re-using as much as possible existing structures in the orchestration. Applying the proposed approach recommends changes in the existing orchestration to a service developer and therefore reduces the required manual integration of service maintenance drastically.

The approach is outlined in Fig 1. Here three services are involved, where the service is implemented as a BPEL process. The BPEL specification represents the orchestration and will be represented in a formal model called Nested Word Automata (NWA)[4] with superimposed invariants to

978-0-7695-3811-2/09 $26.00 © 2009 IEEE
DOI 10.1109/SCC.2009.11

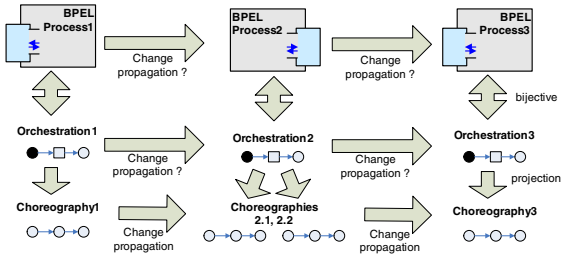

Figure 1. Approach overview

support semantic and syntactic alignment. From the orchestration the choreography can be derived via a projection represented in a formal model called annotated Finite State Automata (aFSA)[15]. If e.g. orchestration1 is changed then this may effect choreography1. The changed choreography is provided to service2 which can derive additive and subtractive changes from the new choreography and the previous choreography 2.1. These changes will now be integrated into orchestration2, from which the new BPEL process can be constructed by applying the BPEL semantic invariants. Further, the choreographies 2.1 and 2.2 can be derived and changes can be propagated further to service 3 and maybe also back to service 1. Thus, the presented approach is the first step needed for enabling service behavior negotiation.

2. Related Work

Choreography matchmaking is limited to checking properties on choreographies. All approaches [5, 1, 7] have a projection from orchestration to choreography. In DYCHOR [11] changes in choreographies are matched to orchestration changes by comparing execution sequences and identifying where to extend the current orchestration. This works only for simple changes, like e.g. adding a new message, but does not work for e.g. removing a loop as discussed in this paper.

Alignment of choreographies and orchestrations is partly addressed in workflow inheritance [12, 2], where operations are added to orchestrations without changing choreographies. However, this approach does not account for already existing orchestrations, which makes re-creating complete orchestrations a labor intensive task.

Another way of creating an orchestration via process mining [13], where an orchestration is derived from log data of process instances. Process mining techniques are solely based on log data and therefore do not take choreography information nor existing orchestrations into account. Not considering existing orchestrations means not using available information which complicates deriving orchestrations. Process variant mining [9] facilitates an existing orchestration and a set of variations of the orchestration to produce a new orchestration with the smallest weighted distance to the set of orchestration variants. However, in this approach the orchestration is adapted based on a set of orchestrations instead of a set of choreographies. Thus, the targeted issue to create a new orchestration based on an existing one and changed choreographies is not addressed.

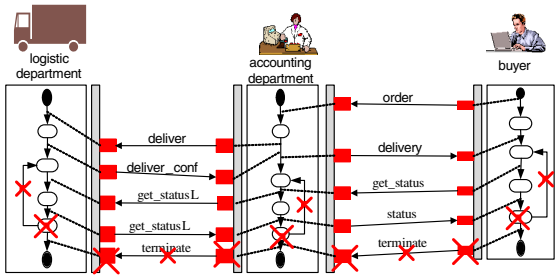

Figure 2. Example scenario

3. Formalization

The presented approach will be introduced along an example procurement scenario depicted in Fig 2. The buyer starts the process by sending an order to the accounting department, which checks the order and forwards it to the logistics department (*deliver* message). The logistics department confirms the order via a *deliver confirmation* message, which is forwarded to the buyer as a *delivery* message. The buyer afterwards can track the progress of the order using a tracking system (*get_status* and *status* messages between accounting and buyer, and *get_statusL* messages between accounting and logistics). The buyer can repeat parcel tracking many times. After the order has been received the buyer has to terminate the process by sending a *terminate* message which is forwarded to the logistics as a *terminateL* message.

Now the logistics department changes its services, allowing parcel tracking exactly once per order. As a consequence, there is no need for the *terminateL* message anymore. These changes only remove elements from the BPEL process as represented by crossed out activities in Fig 2. These subtractive changes are propagated to the accounting and further to the buyer. In the course of the paper formal choreography and orchestration models are introduced and the change propagation for accounting is illustrated.

3.1. Choreography

In this paper annotated Finite State Automata (aFSA) [16] are used as a formalism for choreographies. An aFSA

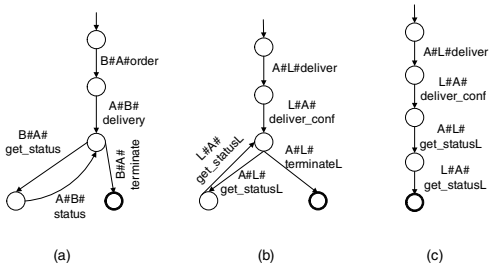

Figure 3. (a) Public Accounting choreography for buyer (b) Subtractive change initiated by Logistics (c) Additive change initiated by Logistics

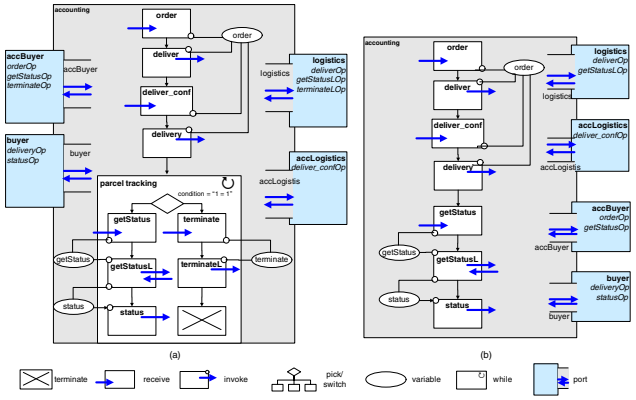

Figure 4. Accounting BPEL process: (a) before change (b) after change

is an extension of classical Finite State Automata. Due to lack of space and the fact that annotations are not relevant for the example used in this paper, the definition of classical finite state automata is used for simplicity.[1] The graphical representation of a Finite State Automaton (FSA) is based on states being represented as circles and transitions as arcs (annotated with labels). Final states are depicted as states with thick line. Transitions are labelled where a label $B\#A\#msg0$ indicates that party B sends message $msg0$ to party A (see, e.g. Fig. 3). The standard Finite State Automaton (FSA) [8] is defined as follows:

Definition 1 ***(Finite State Automaton (FSA))***
A Finite State Automaton A is represented as a tuple $A = (\tilde{Q}, \tilde{\Sigma}, \tilde{\delta}, \tilde{q}_0, \tilde{F})$ *where* $\tilde{Q}$ *is a finite set of states,* $\tilde{\Sigma}$ *is a finite set of messages,* $\tilde{\delta} : \tilde{Q} \times \tilde{\Sigma} \times \tilde{Q}$ *represents labelled transitions,* $\tilde{q}_0 \in \tilde{Q}$ *is a start state, and* $\tilde{F} \subseteq \tilde{Q}$ *constitutes a set of final states.*

The choreography of the accounting process with the buyer is depicted as an FSA in Fig 3a). The choreography of the accounting process with the logistics equals the FSA depicted in Fig 3b). This FSA also represents the subtractive change choreography initiated by logistics. The additive change choreography between logistics and accounting is depicted in Fig 3c). A subtractive/additive change describes message sequences discarded from/added to the original choreography. The DYCHOR approach provides a method to derive additive and subtractive changes [11].

3.2. Orchestration

In this paper an orchestration is specified as a BPEL process, which is an XML document with a specific semantics for individual XML tags. The accounting BPEL process is depicted in 4a). It represents the same messages as contained in the choreographies (see Fig 3a and b). The original BPEL process consists of a *sequence* for setting up the order and a *while* loop for repeated order tracking. In the loop based on the message sent by the buyer (*pick* activity) the process is either terminated or a status is requested from the logistics and its value is returned to the buyer.

There are different ways of formally representing XML documents, like e.g. hedge automata [6] or push-down automata [8]. In this paper Nested Word Automata (NWA) [4] are used since NWA support best the integration with FSA on an operational level. The alphabet used in an NWA is $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_i$ consisting of three disjoint sets: XML start tags Σ_c, XML end tags Σ_r, and BPEL activities Σ_i which includes exchanged messages.

A nested word automaton is defined as follows [4]:

Definition 2 ***(Nested Word Automaton (NWA))***
A nested word automaton over Σ *is* $NWA = (Q, q_0, Q_f, P, p_0, P_f, \langle \delta_c, \delta_r, \delta_i \rangle)$ *where:*

- Q*: finite set of states;*
- $q_0 \in Q$*: initial state*
- $Q_f \subseteq Q$*: set of final states*
- P*: finite set of hierarchical states*
- $p_0 \in P$*: initial hierarchical state*
- $P_f \subseteq P$*: set of final hierarchical states*
- $\delta_i \subseteq Q \times \Sigma \times Q$*: internal transition with source state, label and target state*
- $\delta_c \subseteq Q \times \Sigma \times Q \times P$*: call transition with source state, start tag representing nesting, target state and hierarchical state*
- $\delta_r \subseteq Q \times P \times \Sigma \times Q$*: return transition with source state, source hierarchical state of the nesting start, end*

[1] aFSA can differentiate between optional and mandatory changes, thus provide an additional expressiveness, which does not effect the main contribution of this paper (see Sect 1) of aligning syntactic and semantic orchestration models by using invariants.

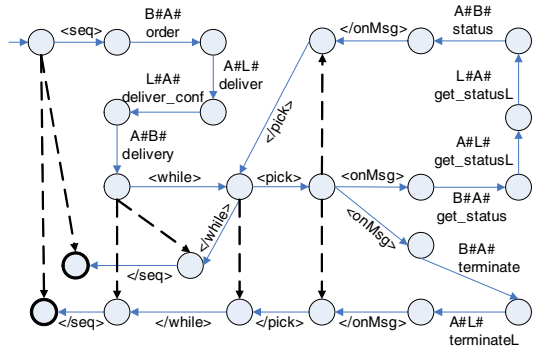

Figure 5. Accounting private process as NWA

tag closing nesting and target state

Regular nested word languages over Σ are closed under union and intersection [3, 4].

The graphical representation of a NWA is similar to the one of a FSA, see e.g. Fig 5. Again states are represented as circles, where final states are represented as states with a thick line. Transitions are represented as arrows labelled with messages or BPEL tags. In case of a start tag, an outgoing transition with a solid line indicates the Q state change, while the corresponding outgoing dotted line indicates the hierarchical state change P. The hierarchical state is consumed by an end tag represented by an incoming dotted arrow. Internal transitions are represented like FSA transitions.

3.3. Orchestration language

The supported subset of BPEL elements are *sequence*, *while*, *pick*, *switch*, *terminate*, *receive*, *reply*, *invoke*, and *assign*. NWA provides a formalism to represent XML documents syntactically, i.e. as call and return transitions. The semantics of BPEL tags are represented in how the states are connected by transitions, like e.g. a loop starts and stops at the same state. The relation to the syntax is that there is a $\langle while \rangle$ call transition resulting in the start state and there is a return transition $\langle /while \rangle$ leaving the stop state of the loop. These relations between syntactical representation and semantic are defined as an axiom/invariant on an NWA representing a BPEL process.

The loop example is formally described as
$\delta_1, \ldots, \delta_n \wedge n \geq 1 \wedge 1 \leq j \leq n-1 \wedge$
$source(\delta_j + 1) = target(\delta_j) \wedge source(\delta_1) = target(\delta_n)$
$\Leftrightarrow$
$(q, \langle while \rangle, source(\delta_1), p) \in \delta_c \wedge$
$(source(\delta_1), p, \langle /while \rangle, q') \in \delta_r$

where the upper part of the logical expression describes the semantics, i.e. a loop starts and stops at the same state, and the lower part describes the existence of corresponding start and end tags. If parts of the invariant are violated then the *while* construct is not required anymore. E.g. if $n < 1$ or $source(\delta_1) \neq target(\delta_n)$ then the $\langle while \rangle$ and $\langle /while \rangle$ tags can be removed. The tags must be added, if a loop structure is detected and no corresponding tags exist. This expression is called an invariant, since this relation is specific for the BPEL language and independent of the specific BPEL process.

The invariant of a choice semantics, like a *pick* activity, is that there are two or more alternatives which can be selected. Each alternative represents a message sequence. All alternatives join again at a certain state. The syntax of a choice is that there is a $\langle pick \rangle$ call transition resulting in the decision state and that there is a $\langle /pick \rangle$ return transition leaving the join state. Further, each alternative message sequence is started with an $\langle onMsg \rangle$ call transition and completed with a $\langle /onMsg \rangle$ return transition. The branching can formally described as
$\delta_{1,1}, \ldots, \delta_{1,n_1} \wedge \delta_{2,1}, \ldots, \delta_{2,n_2} \wedge n_1, n_2 \geq 3 \wedge \wedge k \in \{1, 2\}$
$source(\delta_{k,j+1}) = target(\delta_{k,j}) \wedge 1 \leq j \leq n_k - 1 \wedge$
$source(\delta_{1,1}) = source(\delta_{2,1}) \wedge$
$target(\delta_{1,n_1}) = target(\delta_{2,n_2})$
$\Leftrightarrow$
$(q, \langle pick \rangle, source(\delta_{1,1}), p) \in \delta_c \wedge$
$(target(\delta_{1,n_1}), p, \langle /pick \rangle, q') \in \delta_r \wedge$
$label(\delta_{1,1}) = label(\delta_{2,1}) = \langle onMsg \rangle \wedge$
$label(\delta_{1,n_1}) = label(\delta_{2,n_2}) = \langle /onMsg \rangle$

Again, if parts of the invariant are violated, like e.g. $n_1, n_2 \leq 2$, then tags $\langle onMsg \rangle$ and $\langle /onMsg \rangle$ are removed. Further, if there is only one trace left then remove tags $\langle onMsg \rangle$ and $\langle pick \rangle$. The semantics of other constructs are defined similarly.

Using these invariants, the NWA (see Fig 5) can be constructed by parsing the BPEL process (see Fig 4a) and creating the corresponding states and transitions. The creation of the NWA is not the focus of this paper, but a description of a comparable algorithm can be found in [17]. The creation of accounting's choreography for logistics is a projection of accounting's NWA where all transitions not being message exchanges related to logistics are re-labelled as empty ε transitions. Again, this is not further outlined due to lack of space.

4. Change propagation

Given the NWA (see Fig 5) representing the BPEL process (see Fig 4a) and the subtractive and additive changes depicted in Fig 3b) and c), the proposed approach is illustrated. The basic idea is to use automata operations to

- propagate subtractive changes, i.e., removing message

sequences from the NWA orchestration model,
- propagate additive changes, i.e., adding message sequences to the NWA orchestration model,
- adapt the resulting NWA to conform to the BPEL language invariants (see Sect 3.3) by adding and removing BPEL tags according to the semantics represented by the NWA.

The result of the sketched approach is a modified BPEL process where additive and subtractive choreography changes are reflected with orchestration changes. This is possible since, the NWA representing orchestrations contains the syntactical structure and the semantics of the BPEL process, while the FSA contains the unstructured semantics of choreographies. Performing insertion and removal of message sequences on the NWA preserves the NWA syntactic structure. However, the underlying message sequences may change (see Sect 1). Thus, using BPEL invariants allows to adapt the syntactical structure to reflect the message sequence semantics.

Based on this approach the service developer can faster adapt the BPEL process by completing data flow and resolving degrees of freedom in control flows introduced by additive changes. In the following the individual steps are explained and the required automata operations are introduced.

4.1. Subtractive Change

Applying a subtractive change means removing message sequences specified in the subtractive change FSA Sub from the orchestration NWA NW. Thus, the intersection of the NW and the complement of the subtractive change Sub are calculated.

The subtractive change FSA Sub has an alphabet $\tilde{\Sigma}$ of messages occurring only in bilateral communication. Additional messages used in interactions with other services may be contained in the internal alphabet Σ_i of NWA, but are not contained in the FSA alphabet $\tilde{\Sigma}$. Therefore, the FSA has to be extended by the additional messages $\Sigma_i \setminus \tilde{\Sigma}$. The extension uses automaton operation shuffle product $\&$ with a simple automaton allowing arbitrary combinations of additional messages $(\Sigma_i \setminus \tilde{\Sigma})^*$. The shuffle product is based on two FSAs resulting in a new FSA, which is constructed by arbitrarily interleaving message sequences of the two FSAs, while keeping the original message sequence ordering per FSA [10].

The complement of the FSA Sub requires a complete FSA, i.e., a FSA with an outgoing transition for each state and for each element of the alphabet $\tilde{\Sigma}$. A complete FSA can be constructed by adding all missing transitions to a newly introduced cancellation state [10]. The complement of a complete FSA A is a FSA A' with the same structure as A but a set of final states $\tilde{F}'$ defined as $\tilde{F}' = \tilde{Q} \setminus \tilde{F}$.

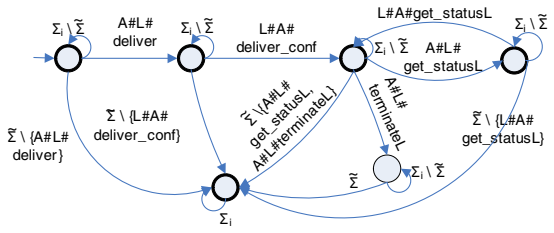

Figure 6. Complemented subtractive change

Applying these operations on the subtractive change FSA Sub depicted in Fig 3b) results in the FSA depicted in Fig 6 specifying the message sequences to preserve.

Since changes and the orchestration are represented in different automata models, specialized union and intersection definitions are required. Intersection and union are based on cross-product construction of states and transitions of FSA and NWA. Since FSA do not have a notion of call and return transition, the cross-product of transitions is based on the cross-product of FSA transitions with internal transitions of NWA.

Definition 3 ***(Intersection of NWA and FSA)*** *Let $NW = (Q, q_0, Q_f, P, p_0, P_f, \langle \delta_c, \delta_r, \delta_i \rangle)$ be an NWA over an alphabet Σ and $A = (\tilde{\Sigma}, \tilde{Q}, \tilde{q}_0, \tilde{\delta}, \tilde{F})$ an FSA with $\Sigma = \tilde{\Sigma}$. The intersection $NW' = NW \cap A$ is a NWA with $NW' = (Q', q'_0, Q'_f, P, p_0, P_f, \langle \delta'_c, \delta'_r, \delta'_i \rangle)$ where:*

- $Q' = Q \times \tilde{Q}$
- $q'_0 = (q_0, \tilde{q}_0)$,
- $Q'_f = Q_f \times \tilde{F}$,
- *Internal* $\delta'_i = \{((q_{11}, q_{21}), \alpha, (q_{12}, q_{22}))|(q_{11}, \alpha, q_{12}) \in \delta_i, (q_{21}, \alpha, q_{22}) \in \tilde{\delta}\}$
- *Call* $\delta'_c = \{((q_{11}, \tilde{q}), \alpha, (q_{12}, \tilde{q}), p)|(q_{11}, \alpha, q_{12}, p) \in \delta_c, \tilde{q} \in \tilde{Q}\}$
- *Return* $\delta'_r = \{((q_{11}, \tilde{q}), p, \alpha, (q_{12}, \tilde{q}))|(q_{11}, p, \alpha, q_{12}) \in \delta_r, \tilde{q} \in \tilde{Q}, \tilde{q}' \in \tilde{Q}\}$

The union of NWA and FSA is also based on a cross product with the set of final states $Q'_f = (Q_f \times \tilde{Q}) \cup (Q \times \tilde{F})$.

Calculating the intersection of the accounting NWA NW (see Fig 5) and the complemented subtractive change FSA Sub (see Fig 6) results in the NWA $NW'' = NW \bigcap_j (\overline{Sub_j} \& (\Sigma_i \setminus \tilde{\Sigma})^*)$. Considering only message sequences of NW'' which result in a final hierarchical state, NW'' has the same graphical representation depicted in Fig 5 except that the set of final states Q_f is empty. Thus, the NWA does not has any final state. This means the intermediate result after applying the subtractive change does not contain any accepted message sequences anymore.

This is intuitively correct since all message sequences which can be constructed by the original BPEL process end

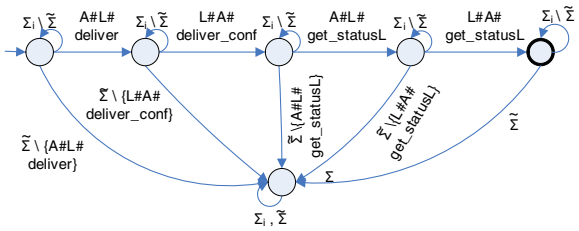

Figure 7. Completed additive change

with a *terminate* and *terminateL* message. Since these messages are removed, all sequences of the original BPEL process are removed. Next the additive changes are applied.

4.2. Additive Change

Applying the additive change means adding message sequences specified in the additive change FSA Add to the outcome of the subtractive change propagation NW''. Thus, the union of the NW'' extended by new messages and the complete additive change Add are calculated. The issue with additive changes is to keep as much of the original BPEL structure as possible to keep manual edits minimal.

Applying the union operation requires that the two automata Add and NW'' are complete. A complete FSA Add can be constructed by adding all missing transitions to a newly introduced cancellation state [10]. The FSA Add has to be completed for $\tilde{\Sigma}$ assuming that the internal alphabet Σ_i of NWA NW'' does not contain messages which are not contained in $\tilde{\Sigma}$ and are relevant for the bilateral communication. The FSA Add is extended as in the subtractive case by messages unrelated to the bilateral communication by using the shuffle product with $complete(Add)\&(\Sigma_i \setminus \tilde{\Sigma})^*$. Applying these operations on the additive change FSA Add (see Fig 3c) results in FSA depicted in Fig 7 representing the message sequences to be added.

The additive change Add may contain messages in $\tilde{\Sigma}$ which are not contained in the internal alphabet Σ_i of NWA NW'' or occur in a different order than in NWA NW''. Therefore, NWA NW'' is completed ($\tilde{\Sigma} - complete(NW'')$) with regard to the alphabet $\tilde{\Sigma}$ of additive change FSA Add. The issue with this completion is that the relation of messages in the additive change FSA Add and messages used in other bilateral communication, like e.g. with the buyer, are not specified. The design decision of the $\tilde{\Sigma} - complete()$ operation is to introduce as less as possible combinations of message sequences and keep as much structure as possible from the original NWA NW. I am convinced that the service developer identifies the degree of freedom and prefers this over a potentially exponential enumeration of message sequences in a BPEL process. Therefore, the $\tilde{\Sigma} - complete()$ operation introduces cancellation states per nesting level.

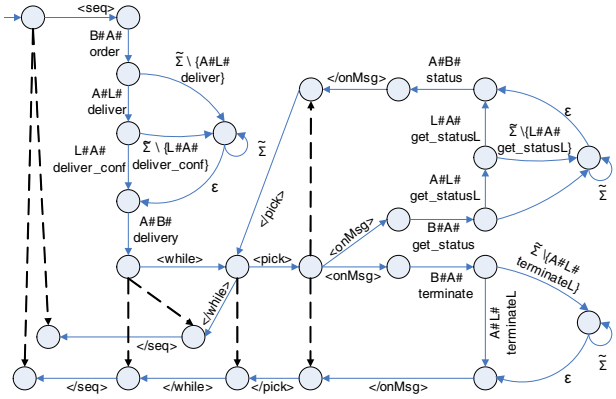

Figure 8. $\tilde{\Sigma}$-Completed NWA of NWA NW''

A nesting level is a cluster of internal transitions per nesting, i.e., a set of internal transitions reachable without additional call or return transitions. The clustering can be represented based on an equivalence relation.

Definition 4 ***(Nesting Relation)***
A nesting relation on a nested word automaton $NW = (Q, q_0, Q_f, P, p_0, P_f, \langle \delta_c, \delta_r, \delta_i \rangle)$ *over* Σ *can be defined as*

$$\delta_0 \equiv \delta_n \Leftrightarrow \exists_{j=0\ldots n-1} \delta_j \in \delta_i . target(\delta_j) = source(\delta_{j+1}) \vee source(\delta_j) = target(\delta_{j+1})$$

The nesting relation is an equivalence relation and can be used to factorize internal transitions into equivalence classes, i.e., set of internal transitions used at the same nesting level. The factorization is denoted as $\delta_j / \equiv$ with $\delta_j / \equiv = \{\delta_k \in \delta_i \mid \delta_j \equiv \delta_k\}$. Applying this factorization to the example in Fig 5 results in the following factorization, where transitions are represented by their labels for brevity.

$$\begin{aligned}
\Delta_1 &= [A\#L\#deliver] \equiv \\
&= \{B\#A\#order, A\#L\#deliver, \\
&\quad L\#A\#deliver_conf, A\#B\#delivery\} \\
\Delta_2 &= [A\#L\#terminateL] \equiv \\
&= \{B\#A\#terminate, A\#L\#terminateL\} \\
\Delta_3 &= [A\#L\#get_statusL] \equiv \\
&= \{B\#A\#get_status, A\#L\#get_statusL, \\
&\quad L\#A\#get_statusL, A\#B\#status\}
\end{aligned}$$

Based on this nesting relation the $\tilde{\Sigma} - complete()$ operation can be defined as follows:

Definition 5 ***($\tilde{\Sigma}$-completion of NWA)***
Let $NW = (Q, q_0, Q_f, P, p_0, P_f, \langle \delta_c, \delta_r, \delta_i \rangle)$ *be a NWA over an alphabet* Σ *and let* $\tilde{\Sigma}$ *be an alphabet with* $\tilde{\Sigma} \subseteq \Sigma_i$.

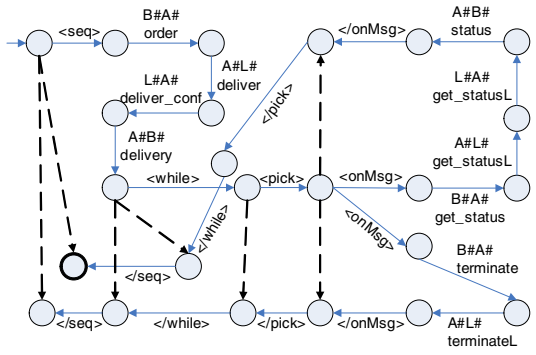

Figure 9. Accounting NWA after applied subtractive and additive change

Then $NW' = \tilde{\Sigma} - complete(NW)$ *is a NWA over* Σ *with* $NW' = (Q', q_0, Q_f, P, p_0, P_f, \delta_r, \delta'_i))$ *where* $\{\Delta_1, \ldots, \Delta_n\} = \delta_i / \equiv$ *being the n equivalence classes for the internal transitions* $Q' = Q \bigcup_{k=1}^{n} \{q_{c,k}\}$ *and*

$$\begin{aligned} \delta'_i = \quad & \bigcup_{k=1}^{n} \{ \quad (q_1, \alpha, q_{c,k}), (q_{c,k}, \varepsilon, q_2) | (q_1, \beta, q_2) \in \Delta_k, \\ & \beta \in \tilde{\Sigma}', \alpha = \tilde{\Sigma} \setminus \{\alpha' \mid (q_1, \alpha', q) \in \Delta_k\} \\ & \bigcup_{k=1}^{n} \{ \quad (q_{c,k}, \tilde{\Sigma}, q_{c,k})\} \cup \delta_i \end{aligned}$$

Based on this definition of $\tilde{\Sigma} - complete()$ operation, a new message sequence may nevertheless be represented by several alternative message sequences in the resulting BPEL process, however, the number of permutations is significantly lower than in case of a classical completion operation.

Applying the $\tilde{\Sigma} - complete()$ operation with $\tilde{\Sigma}$ being the alphabet of the additive change FSA Add on the NWA NW'' (see Fig 5 without final states) results in NWA $\tilde{\Sigma} - complete(NW'')$ depicted in Fig 8.

The application of the additive change FSA to the NWA NW'' results in NWA NW' with

$$\begin{aligned} NW' = \quad & \tilde{\Sigma} - complete(NW'') \cup \\ & \bigcup_j (complete(Add_j) \& (\Sigma_i \setminus \tilde{\Sigma})^*) \end{aligned}$$

Thus, the union of the completed additive change FSA Add (see Fig 7) and the completed NWA NW'' (see Fig 8) results in NWA NW' depicted in Fig 9. This is the automaton containing all message sequences after applying subtractive and additive changes.

4.3. Orchestration Language

The NWA NW' in Fig 9 can be represented as a valid XML document. However, the transitions of the original NWA has been altered due to applying several automata operations. Therefore, it has to be checked whether the existing XML tags have a matching semantics in the message sequences or whether there are message sequences, which require additional XML tags. This check is performed using the BPEL invariants (see Sect 1 and 3.3). In addition, changes performed on the BPEL process are identified and documented. Therefore, several operations are performed on the derived NWA NW': first internal activities to be removed are detected, then new messages added are detected, and finally an alignment of the NWA according to BPEL language invariants is performed. The steps are informally explained below and illustrated using the example.

- Check all possible transitions whether they can reach a hierarchical state p_0, thus a state without nesting, but not being a final state in Q_f: Report internal transitions which can not reach a final state as transitions to be deleted. Further, replace the transitions with the empty word ε for further processing. In the example two transitions *B#A#terminate* and *A#L#terminateL* are replaced by $varepsilon$ and reported as deletions to the service developer.
- Check whether there are transitions with a new message label $\tilde{\Sigma} \setminus \Sigma$ and report these transitions as additions. In the example there are no such transitions.
- Check for each language construct whether the invariant is still fulfilled. If not, the language construct has to be replaced by an empty word. With regard to the example the following actions have to be taken:
  - One branch of the *pick* activity does not contain any non empty internal transitions anymore, thus these transitions can be replaced by empty words. Replace transitions $\langle onMsg \rangle$ and $\langle /onMsg \rangle$ by ε and report the removal to the service developer.
  - The *pick* activity does now contain only a single branch and therefore is not meaningful anymore. Replace transitions $\langle pick \rangle, \langle onMsg \rangle$, $\langle pick/ \rangle, \langle /onMsg \rangle$ with nested transitions *B#A#get_status* by ε and report the removal to the service developer.
  - The *while* activity invariant is no longer fulfilled since the message sequence nested by the *while* activity does not start and end in the same state anymore. Therefore, replace transitions $\langle while \rangle$ and $\langle /while \rangle$ by ε and report the removal to the service developer.
- Check for each sequence of transitions whether language invariants are fulfilled. If this is the case add the language construct. With regard to the example no new nestings have been detected.

After the recommendations have been evaluated by the service developer, all recommended delete operations on activities of the BPEL process have been approved. Thus the resulting NWA is depicted in Fig 10, which corresponds to the resulting BPEL process depicted in Fig 4b).

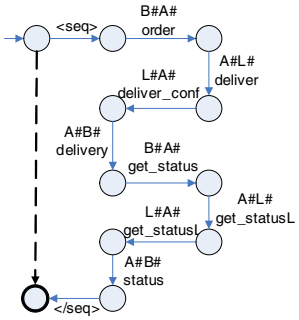

Figure 10. Consistent reduced Accounting NWA after applied changes

4.4. Further Propagation

The new BPEL process (see Fig 4b) and the corresponding NWA (see Fig 10) do not contain a loop and the *B#A#terminate* message anymore, which are currently still expected by the buyer BPEL process. Thus, the next steps are to re-calculate new additive and subtractive changes with the buyer and to repeat the complete procedure for the buyer BPEL process. It might be that changes on the accounting BPEL process are only accepted if also the logistics accepts changes to his orchestration. Thus, the interaction with logistics must be checked again after propagation.

A detailed description on propagating changes and identifying needs to propagate changes are described in the DYCHOR approach [11]. Details about a protocol for decentralized decision making on whether further propagation of changes is required can be found at [14].

5. Discussion, Conclusion and Future Work

The aFSA and NWA as presented in this paper are implemented in Java. The aFSA and the basis for the NWA implementation are described in [16, 17]. The scenarios used in Sect. 4 have been implemented by Java calls representing operations on aFSA and NWA objects. The resulting automata are then visualized using DOT graphing tool. The graphs used in this paper are modified and reduced to make them easier to read.

The presented approach provides a means to propagate changes on the choreography level to changes on orchestration level. The main contribution of this approach is to use a formal model for representing syntactic properties of orchestrations and apply a notion of semantic invariants to align the orchestration model in a syntactic and semantic way. The presented work is partly implemented.

In future work other application domains for superimposing semantic invariants on XML language models will be investigated.

References

[1] W. Aalst. Interorganizational workflows: An approach based on message sequence charts and petri nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.

[2] W. Aalst. Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. *Electronic Commerce Research*, 2(3):195–231, 2002.

[3] R. Alur and P. Madhusudan. Adding nesting structure to words. In O. H. Ibarra and Z. Dang, editors, *Developments in Language Theory*, LNCS 4036 , 1–13, 2006.

[4] R. Alur and P. Madhusudan. Adding nesting structure to words. *Journal of the ACM*, page 45, 2009.

[5] I. Chebbi, S. Tata, and S. Dustdar. The view-based approach to dynamic interorganizational workflow cooperation. TR TUV-1841-2004-23, TU Vienna, 2004.

[6] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007.

[7] P. Grefen. Towards dynamic interorganizational business process management. In *Proc. of the 15th Int'l WS on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE'06)*, 2006.

[8] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2001.

[9] C. Li, M. U. Reichert, and A. Wombacher. Discovering reference process models by mining process variants. In *ICWS '08*, 45–53, 2008.

[10] O. Matz, A. Miller, A. Podtthoff, W. Thomas, and E. Valkema. Report on the program AMoRE. Technical Report 9507, Christian-Albrechts Universtaet, 1995.

[11] S. Rinderle, A. Wombacher, and M. Reichert. Evolution of process choreographies in DYCHOR. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, LNCS 4275, 273–290, 2006.

[12] W. van der Aalst and T. Basten. Inheritance of workflows: An approach to tackling problems related to change. *Theoret. Comp. Science*, 270(1-2):125–203, 2002.

[13] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. *Knowledge and Data Engineering, IEEE Transactions on*, 16(9):1128–1142, 2004.

[14] A. Wombacher. Decentralized decision making protocol for service composition. In *ICWS*, 2005.

[15] A. Wombacher. *Decentralized establishment of consistent, multi-lateral collaborations*. PhD thesis, 2005.

[16] A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold:. Matchmaking for business processes based on choreographies. *IJWS*, 1(4):14–32, 2004.

[17] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming BPEL into annotated deterministic finite state automata enabling process annotated service discovery. In *ICWS*, 316–323, 2004.