Evolution of Process Choreographies in DYCHOR

Stefanie Rinderle[1], Andreas Wombacher[2], and Manfred Reichert[2]

[1] Dept. DBIS, University of Ulm, Germany
stefanie.rinderle@uni-ulm.de

[2] Informaton Systems Group, University of Twente, The Netherlands
{a.wombacher, m.u.reichert}@ewi.utwente.nl

Abstract. Process-aware information systems have to be frequently adapted due to business process changes. One important challenge not adequately addressed so far concerns the evolution of process choreographies, i.e., the change of interactions between partner processes in a cross-organizational setting. If respective modifications are applied in an uncontrolled manner, inconsistencies or errors might occur in the sequel. In particular, modifications of private processes performed by a single party may affect the implementation of the private processes of partners as well. In this paper we present the DYCHOR (DYnamic CHOReographies) framework which allows process engineers to detect how changes of private processes may affect related public views and - if so - how they can be propagated to the public and private processes of partners. In particular, DYCHOR exploits the semantics of the applied changes in order to automatically determine the adaptations necessary for the partner processes. Altogether our framework provides an important contribution towards the realization of adaptive, cross-organizational processes.

1 Introduction

The economic success of an enterprise more and more depends on its ability to flexibly and quickly react on changes at the market, the development, or the manufacturing side. For this reason companies are developing a growing interest in improving the efficiency and quality of their internal business processes and in optimizing their interactions with business partners and customers. Recently, we have seen an increasing adoption of business process automation technologies by enterprises as well as emerging standards for business process orchestration and choreography in order to meet these goals. Respective technologies enable the definition, execution, and monitoring of the operational processes of an enterprise. In connection with Web service technology, in addition, the benefits of business process automation and optimization from within a single enterprise can be transferred to cross-organizational business processes (*process choreographies*) as well. The next step within this evolution will be the emergence of the agile enterprise being able to rapidly set up new processes and to quickly adapt existing ones to changes in its environment.

One important challenge not adequately addressed so far concerns the evolution of process choreographies, i.e., the controlled change of the interactions between partner processes in a cross-organizational setting. If one party changes its process in an uncontrolled manner, inconsistencies or errors regarding these interactions might occur. Generally, the partners involved in a process choreography exchange messages via their public processes, which can be considered as special views on their private processes (i.e., the process orchestrations). If one of these partners has to change the implementation of his private process (e.g., to adapt it to new laws or optimized processes) the challenging question arises whether this change affects the interactions with partner processes and their implementation as well. Obviously, as long as a modified business process is not part of a process choreography, change effects can be kept local. The same applies if changes of a private process have no impact on related public views.

In general, however, we cannot always assume this. The modification of a private process may not only influence corresponding public processes, but also the public and private processes of its partners. For this reason, it is indispensable for any IT infrastructure to provide adequate methods for (automatically) *propagating* changes of a private process to the partner processes (if required). This important issue has not been considered by current approaches so far. As a consequence adaptations of process choreographies have turned out to be both costly and error-prone. Note that the handling of respective changes is not trivial since we must be able to precisely state which effects on partner processes result after adaptating a (private) process. In any case we need precise and formal statements about this in order to avoid implementation holes later on.

In this paper we deal with these challenges and present our DYCHOR approach which allows for the controlled evolution of process choreographies. We show how changes of a private process may affect related public views and - if so - how they can be propagated to the public/private processes of partners as well. To be able to precisely state whether change propagations to partner processes become necessary we introduce a formal model based on annotated Finite State Automata. We further exploit the semantics of the applied change operations in order to derive necessary adaptations automatically. Due to the autonomy of partners, however, private partner processes cannot be adapted automatically to changes of a process choreography. DYCHOR allows for the comprehensive assistance of users in accomplishing this task in a correct and effective manner. In this paper we restrict our considerations to structural changes (e.g., the insertion or deletion of process activities). Other adaptations of process models (e.g., the change of transition conditions) require a similar approach, but are outside the scope of this paper. We do also not address dynamic changes (i.e., the migration of running choreographies to respective changes at the type level) in this paper. Dynamic adaptations of choreographies and process instances, however, constitute an important part of our change framework [1,2].

Sect. 2 introduces an application scenario which we use throughout the paper in order to illustrate basic concepts of our framework. In Sect. 3 we discuss basic issues related to process choreographies and interactions between partner processes. In particular, we introduce our formal model and show how it can be used to automatically generate public processes out of private ones. This provides the basis for dealing with process changes. Sect. 4 presents a classification of changes and Sect. 5 provides methods for propagating changes on behalf of selected scenarios. Sect. 6 sketches implementation issues and Sect. 7 discusses related work. We close with a summary and an outlook in Sect. 8.

2 Practical Scenario

Further discussions are based on a simple procurement process within a virtual enterprise (cf. Fig. 1). It comprises a buyer, an accounting department, and a logistics department. The accounting department approves an order (*order* message) sent by a buyer and forwards it to the logistics department (*deliver* message) to deliver requested goods. The logistics department confirms the receipt (*deliver_conf* message) to the accounting department, which forwards this message (extended by the expected delivery date and the parcel tracking number) to the buyer (*delivery* message). The buyer can do parcel tracking (*get_status* and *status* messages) of the shipped goods. Corresponding messages are forwarded by the accounting department to the logistics department.

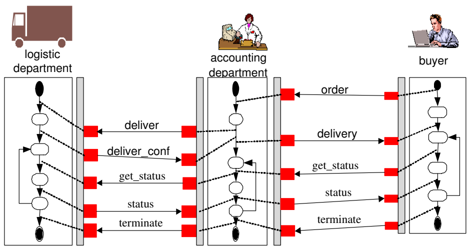

Fig. 1. Example Overview

This scenario represents a *process choreography*, i.e., a conversation between partner processes. More precisely, the participating partners exchange messages via their *public* processes, which constitute special views on *private processes* [3]. We describe the private process of the accounting department in more detail denoting it according to the BPEL specification [4]. To keep the example simple, we abstract from the structure of the exchanged messages and use simplified message names. Concrete message structures could be, for example, taken from the RosettaNet Partner Interface Processes (PIPs) 3A4 (Request Purchase Order), 3A7 (Notify of Purchase Order Update), and 3B2 (Notify of Advanced Shipment) [5].

Regarding Web services, for example, messages are exchanged by invoking operations at the respective partner sites. A Web service may comprise one or more operations (grouped within porttypes) which can be specified using WSDL.

Each operation then represents a potential message exchange between partners. If an operation contains only one single input message, it is considered to be asynchronous, otherwise the operation is synchronous. Regarding our example all operations are asynchronous except the synchronous *getStatusOP* operation provided by the *logistics* service.

We base the description of private processes on such porttype definitions (i.e., Web service specifications) by directly referring to them. In the following, private processes are denoted in BPEL [4] and are therefore specified in terms of tasks (named *activities* in the BPEL terminology) representing basic pieces of work to be performed by potentially nested services. The control flow of a BPEL process constrains possible execution orders of its activities and is based on constructs for selective (*switch* and *pick* activities), sequential (*sequence* activity), and parallel (*flow* activity) execution. In addition, a BPEL process defines the data flow between activities (variable handling and *assign* activity for mapping data between messages) regardless of their concrete implementation. Based on this understanding, the process model of one partner includes activities realizing its interaction with the other partners. These interactions are represented by exchanging messages (*receive*, *reply*, *invoke*, and *pick* activities in BPEL).

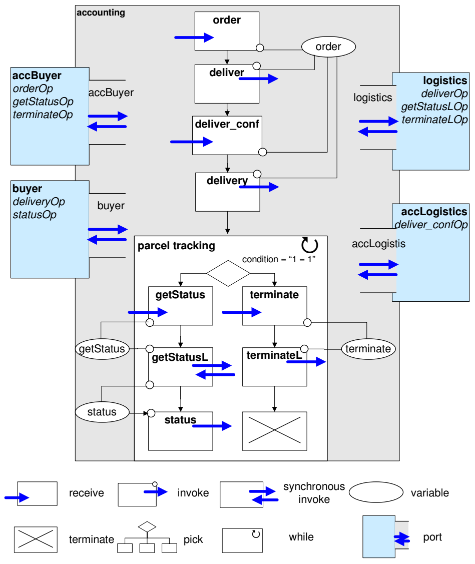

Fig. 2. Accounting BPEL Private Process

The BPEL specification of the accounting department private process is depicted in Fig. 2. The partnerLink definition associates a partner name to a bilateral interaction between two roles. The association of roles to concrete parties and operations is done in the partnerLinkType definition contained in the related WSDL file. The process starts by receiving an *order* message sent by the buyer, which is forwarded to the logistics department via a *deliver* message.

The logistics department answers asynchronously with a *deliver_conf* message. The accounting department process receives this message and forwards it to the buyer via a *delivery* message. Since the buyer is allowed to do parcel tracking arbitrarily often, this step is embedded in a non-terminating loop within the accounting process. More precisely, the accounting department may receive a *get_status* message sent by the buyer, which is followed by a synchronous invocation of the logistics *get_statusL* operation (representing two messages) and the reporting of the respective status back to the buyer (via a *status* message). Alternatively, it must be possible to terminate accounting as well as logistics process at some point in time. For this, a *termination* message can be initiated by the buyer; this message is then send to the accounting department process, which forwards it to the logistics process. After this both processes are terminated.

As a second example consider the private process of the buyer, which is depicted in Fig. 3. – We omit further details and focus on the bilateral interaction between the accounting and buyer process in the following.

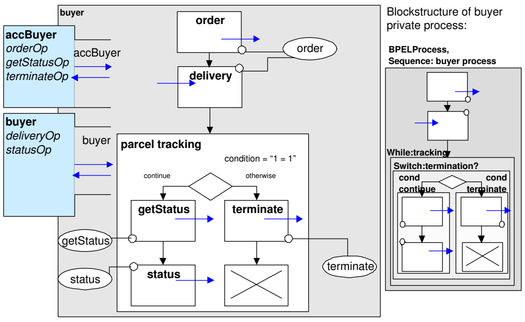

Fig. 3. Buyer BPEL Private Process

3 Process Choreographies

We discuss basic issues related to the evolution of choreographies between partner processes. We show how this can be supported in a (semi-)automated way.

3.1 Overview

For several reasons business processes steadily evolve. Thus process-oriented information systems have to be continuously adapted as well. As long as the modified processes are not part of a process choreography, change effects can be kept local. The same applies if changes of a private process have no impact on related public processes. In general, however, we cannot always assume this. Regarding process choreographies the modification of a private process may not only influence related public processes, but also the public and private processes of partners. As an example take an activity inserted into a private process and invoking an external operation of a partner process (by sending a corresponding message to it). If the partner process is not adapted accordingly (e.g., by inserting a receive activity processing the message sent) the execution of the modified process choreography could fail. Thus it is crucial to provide adequate methods to (automatically) *propagate* changes of a private process to partner processes.

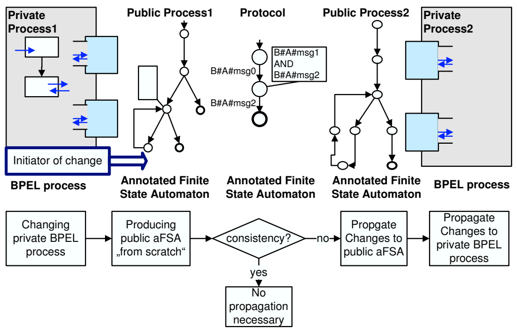
Private Process1
Public Process1
Protocol
Public Process2
Private Process2
B#A#msg0
B#A#msg1 AND B#A#msg2
B#A#msg2
Initiator of change
BPEL process
Annotated Finite State Automaton
Annotated Finite State Automaton
Annotated Finite State Automaton
BPEL process
Changing private BPEL process
Producing public aFSA „from scratch“
consistency?
no
yes
No propagation necessary
Propgate Changes to public aFSA
Propagate Changes to private BPEL process

Fig. 4. DYCHOR Approach

Fig. 4 depicts our DYCHOR approach for the evolution of choreographies. Assume that private process 1 (left side) is modified and therefore is regarded as initiator of the following choreography change (if necessary). Then, at first, the public view on this process is recreated in order to reflect changes that might affect the interactions with partner processes. If this results in a modification of public process 1 (and only then) we further check whether adaptations of public process 2 (right side of figure) become necessary as well. This is accomplished by calculating the consistency of the two public processes, i.e., the guarantee of a deadlock free execution of the interaction. In case of inconsistency the change of public process 1 has to be propagated to public process 2[1]; otherwise the execution of the process choreography will fail. DYCHOR exploits semantics of the applied changes in order to automatically adapt public process 2 in such a case. After having performed respective modifications the adaptation of private process 2 also becomes necessary. However, due to the autonomy of the partners and due to the privacy of the mission critical business decisions (represented in the private process), an automatic adaptation of private processes is generally not desired. Nevertheless, the system should assist process engineers in accomplishing this task by suggesting respective adaptations of private process 2.

3.2 Formal Model

The sketched approach (i.e., the correct propagation of private process changes) requires a formal model for representing public processes. Different approaches have been proposed in literature, which can be classified according to their underlying communication model: The models suggested in [6] and [7], for example,

[1] A general correctness criterion for this is provided in Section 4.2.

support asynchronous communication. By contrast synchronous communication is supported by [8]. Since Web services often use synchronous communication based on the HTTP protocol, in the following we apply the annotated Finite State Automata model as introduced in [8].

DYCHOR uses annotated Finite State Automata (aFSA) to represent message sequences that can be handled by a public process. Transitions of such an aFSA are labeled, whereas a label $A\#B\#msg$ indicates that party A sends message msg to party B (see, for example, the left aFSA in Fig. 5). Furthermore, aFSAs can differentiate between mandatory and optional messages. This is achieved by annotating states with logical expressions. In the right aFSA from Fig. 5, for example, the depicted conjunctive annotation expresses that both messages $B\#A\#msg1$ and $B\#A\#msg2$, which may be sent by party B, have to be supported by a trading partner. Thus the messages are mandatory. Obviously, the aFSAs of two interacting public processes must meet certain constraints in order to ensure correct execution of the respective process choreography. We formalize this and summarize basic aFSA characteristics necessary for the further understanding. Hence, we introduce the definition of formulas[2] used in the annotations, before introducing the aFSA.

Definition 1 ***(Definition of Formulas)***
The syntax of the supported logical formulas is given as follows: (i) the constants true and $false$ are formulas, (ii) the variables $v \in \Sigma$ are formulas, where Sigma is a finite set of messages, (iii) if ϕ is a formula, so is $\neg\phi$, (iv) if ϕ and ψ are formulas, so is $\phi \wedge \psi$ and $\phi \vee \psi$. – The set of all formulas is defined as E.

Based on the set of formulas E the standard Finite State Automaton (FSA) [10] is extended as follows:

Definition 2 ***(annotated Finite State Automaton (aFSA))***
An annotated Finite State Automaton A is represented as a tuple $A = (Q, \Sigma, \Delta, q_0, F, QA)$ where Q is a finite set of states, Σ is a finite set of messages, $\Delta : Q \times \Sigma \times Q$ represents labeled transitions, $q_0 \in Q$ is a start state, $F \subseteq Q$ constitutes a set of final states, and $QA : Q \times E$ is a finite relation of states and logical terms within the set E of formulas.

The graphical representation of an annotated Finite State Automaton (aFSA) is based on the usual representation of FSA. States are represented as circles and transitions as arcs (annotated with labels). Final states are depicted as states with thick line. In addition to FSA, an aFSA can have state annotations (denoted as squares connected to the respective states). Fig. 5 shows two aFSA examples: Transitions are labeled whereas a label represents a message exchanged between party A and party B.

In our DYCHOR framework, a public process (in terms of aFSA models) can be automatically derived from the specification of a private one. In [11], for a subset of BPEL, we have provided respective mapping rules. Based on the given

[2] The logical formulas are specified adapting the definition in [9].

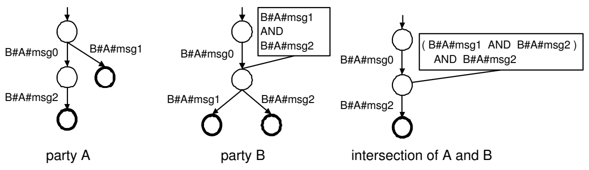

Fig. 5. aFSA Representation

aFSA definition, intersection and emptiness operations can be defined (cf. [8]), which are quite similar to the ones of standard FSA.

Definition 3 ***(Intersection of two aFSAs)***
Let $A_1 = (Q_1, \Sigma_1, \Delta_1, q_{10}, F_1, QA_1)$ *and* $A_2 = (Q_2, \Sigma_2, \Delta_2, q_{20}, F_2, QA_2)$ *be two aFSA. The intersection* $A := A_1 \cap A_2$ *of these automata is given by* $A = (Q, \Sigma, \Delta, q_0, F, QA)$, *with:* $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_1 \cap \Sigma_2$, $q_0 = (q_{10}, q_{20})$, $F = F_1 \times F_2$, $\Delta = \{((q_{11}, q_{21}), \alpha, (q_{12}, q_{22})) | \beta \in \{\alpha, \varepsilon\}, (q_{11}, \beta, q_{12}) \in \Delta_1, (q_{21}, \beta, q_{22}) \in \Delta_2\}$, *and* $QA = \bigcup_{(q_1,e_1) \in QA_1, (q_2,e_2) \in QA_2} \{((q_1, q_2), e_1 \wedge e_2)\}$

In particular, the intersection of two aFSAs is based on the usual cross product construction of automata intersection, where state annotations are combined by conjunction. Fig. 5 illustrates the intersection applied on party A and B. Note that the resulting aFSA only contains those transitions that can be processed by both automata. The annotation in the intersection automaton is the conjunction of the annotation contained in party B and the default annotation of party A, that is, $B\#A\#msg2$, resulting in $(B\#A\#msg1 \wedge B\#A\#msg2) \wedge B\#A\#msg2$.

Based on the intersection automaton, it can be checked whether the accepted language is empty. Emptiness means that the a set of message exchanges exists, where all associated mandatory transitions are supported by a trading partner's aFSA. Again this emptiness test is based on standard automaton emptiness test, where it is checked whether the automaton contains a single path to a final state. Regarding aFSAs this emptiness test has to be extended by requiring that all transitions of a conjunction associated to a single state are available in the automaton and a final state can be reached following each of these transitions. As a consequence, two automata are consistent, if their intersection is non-empty; i.e., there is at least one path from the start to a final state, where each formula annotated to a state on this path evaluates to *true*. A variable becomes *true*, if there is a transition labeled equally to the variable from the current state to another state where the annotation evaluates to *true*. Finally the automaton is non-empty, if the annotation of the start state is *true*.

For the above example the intersection automaton for parties A and B is depicted in Fig. 5. This aFSA is empty since it does not contain the mandatory transition labeled $B\#A\#msg1$: The variable $B\#A\#msg2$ of the annotation evaluates to *true* since there is a path to a final state. By contrast the variable $B\#A\#msg1$ is evaluated to $false$ because there is no such transition available at that state providing a path to a final state.

The non-emptiness of the intersection of two automata guarantees for the absence of deadlock with respect to the execution of these two automata. This property can be derived due to the differentiation between mandatory and optional messages in an automaton. Deadlock freeness is also called *consistency*. If consistency is defined between two parties then we call it *bilateral consistency*.

3.3 Public Process Generation

We assume the private process being specified with BPEL. We sketch how BPEL "blocks" from a private process have to be mapped to states of the related public process (represented by an aFSA). As we will see later, this mapping is useful when changing process choreographies. In this context it is not worth applying the mapping on the originator side of a change. However, when propagating changes of a public process to its underlying private process the mapping can be used to determine the blocks in the private process to be modified.

The mapping is illustrated on behalf of the buyer process. It is based on a depth first traversal of the BPEL structure where each block represents a part of the automaton. As a consequence of the strict nesting of a BPEL document, the names of the blocks are associated with a particular state of the resulting automaton model. Regarding the private and public part of the buyer process (cf. Fig. 3 + 6 a) we obtain the mapping shown in Table 1. It represents the relation between the state numbers (aFSA of the public process) and the BPEL block names (BPEL specification of private process) of the private and the public process. Note that a single state in the public process may be assigned to several BPEL elements since, in general, not all elements have an effect on the public process. As a consequence, the required modifications can be limited to the first block mentioned due to the depth first traversal of the private process.

Table 1. Buyer Mapping Table

State Number	BPEL Block Name
1	BPELProcess, Sequence:buyer process
2	Sequence:buyer process
3	Sequence:buyer process, While:tracking, Switch:termination?, Sequence:cond continue, Sequence:cond terminate
4	Sequence:cond continue
5	Sequence:cond terminate

3.4 View Generation

As a basis for bilateral consistency checking, it has to be ensured that the processes to be compared are representing the bilateral message exchanges only.

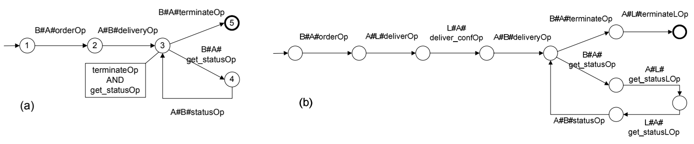

Fig. 6. (a) Buyer Public Process (b)Accounting Public Process

Deriving the bilateral view of a public process is illustrated next on behalf of the accounting process. The accounting private process (cf. Fig. 2) can be transformed in a public process (cf. Fig. 6 b).

The view $\tau_P(wf)$ of party P on the public process wf is generated by relabeling all transitions not related to P. E.g., in the buyer view $\tau_{Buyer}(Acc)$ of the accounting process, messages exchanged with Logistics are relabeled with the empty word ε. Effected messages are *A#L#deliverOp*, *L#A#deliver_confOp*, *A#L#terminateLOp*, *A#L#get_statusOp*, and *L#A#get_statusOp*. The minimized Buyer view of the Accounting public process is shown in Fig. 7a. Applying the same method for Logistics results in the automaton depicted in Fig. 7b.

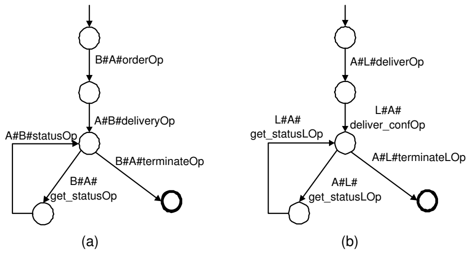

Fig. 7. Accounting Public Process: (a) Buyer View (b) Logistics View

4 Process Choreography Evolution

DYCHOR classifies process changes in two dimensions: The first one (*change framework*) specifies whether the change adds message sequences to an automaton (*additive* change) or removes messages from it (*subtractive* change). The second dimension (*change propagation*) indicates whether a process change has effects on trading partners or not, i.e., whether the protocol the trading partners agreed has to be modified (*variant* change) or not (*invariant* change).

4.1 Change Framework

We give a definition for the difference between two aFSAs, which is used to characterize two basic kinds of change operations on public processes.

Definition 4 ***(Difference of two aFSA)***
Let $A_1 = (Q_1, \Sigma_1, \Delta_1, q_{10}, F_1, QA_1)$ *and* $A_2 = (Q_2, \Sigma_2, \Delta_2, q_{20}, F_2, QA_2)$ *be two*

aFSA. The difference $A := A_1 \setminus A_2$ of these two aFSA is given by
$A = (Q, \Sigma, \Delta, q_0, F, QA_1)$ *with:* $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_1 \cap \Sigma_2$, $q_0 = (q_{10}, q_{20})$,
$F = F_1 \times (Q_2 \setminus F_2)$, $\Delta = \{((q_{11}, q_{21}), \alpha, (q_{12}, q_{22}))|\beta \in \{\alpha, \varepsilon\},$
$(q_{11}, \beta, q_{12}) \in \Delta_1, (q_{21}, \beta, q_{22}) \in \Delta_2\}$

This definition requires that the automata are complete; i.e, for every state there exists an outgoing transition for each element of the alphabet Σ. In this paper we focus on additive and subtractive changes and their application to aFSAs. Based on such basic change operations more complex changes can be defined. Our framework considers other operations (e.g., to shift process activities) as well as complex changes (defined by applying a set of basic changes operations). Their treatment, however, is outside the scope of the paper. Based on the difference operator we can give a formal definition for additive/subtractive changes:

Definition 5 ***(Additive / Subtractive Change Operations)***
Let A be the aFSA of a public process and let δ be a change operation which transforms A into another aFSA A'. Then:

- *δ is an additive change operation* $:\Longleftrightarrow A' \setminus A \neq \emptyset$
- *δ is a subtractive change operation* $:\Longleftrightarrow A \setminus A' \neq \emptyset$

Based on this definition additive (subtractive) changes of an aFSA correspond to the addition (deletion) of potential message sequences to (from) this aFSA. Note that this does not relate to the structural complexity of the respective private or public processes.

4.2 Propagation Criterion and Invariant Changes

Let A and B be the aFSAs of two public partner processes and let A $\cap$ B $\neq \emptyset$ be the protocol (choreography) between them. If A is changed to A' (by applying change operation δ) the challenging question is whether δ has to be propagated to B or not. Intuitively, no propagation is needed if the protocols before and after applying δ are equivalent. Formally:

$$A \cap B \equiv A' \cap B \Longleftrightarrow (A \setminus A') \cap B = \emptyset \wedge (A' \setminus A) \cap B = \emptyset$$

This constraint, however, is too restrictive since we can also ignore options that are completely under the control (i.e., are to be decided) by the party having performed the change. More precisely, no propagation is needed if $A' \cap B \neq \emptyset$ (assuming that A and B have been bilaterally consistent before the change).

Definition 6 *(Variant and Invariant Changes)*
Let A and B be the aFSAs of two public processes which are consistent, i.e., $A \cap B \neq \emptyset$. Let δ be a change operation which transforms A into another aFSA A'. Then:

- *δ is an invariant change* $:\Longleftrightarrow A' \cap B \neq \emptyset$
- *δ is a variant change* $:\Longleftrightarrow A' \cap B = \emptyset$

The aFSA B expresses all options it considers as being mandatory for the respective public process. Thus if public process A' has been changed in a way such that these options are no longer met, change propagation becomes necessary. Accordingly we can state that changes are invariant (i.e., no change propagation is needed) if the intersection between A' and B does not become empty. Note that this can apply for both additive and subtractive changes.

In summary, if the changed public process A' is still consistent with the public process B of a partner it is considered as being invariant and no further actions are needed. By contrast if A' and B turn out to be inconsistent, additional actions become necessary in order to guarantee the successful execution of the processes. How corresponding actions look like is discussed in the following section.

5 Selected Evolution Scenarios

In the following, we provide methods for the propagation of additive changes to partner processes. Due to lack of space we omit a discussion of further changes here (for details on, for example, subtractive changes see [12]).

5.1 Invariant Additive Change

At first we consider invariant additive changes. For example, assume that the accounting process wants to provide an additional order message format to buyers. This change can be realized by adding an alternative activity (order_2) to the accounting process which then receives and processes respective messages *B#A#order_2Op* (cf. Fig. 8).

Since the added message *B#A#order_2Op* is received by the accounting workflow, the buyer view on the respective public process changes (cf. Fig. 9a). However, from the viewpoint of the buyer this change does not require an immediate treatment and propagation to its public and private process. The reason is that the intersection automaton (cf. Fig. 9b) of the modified public view of the buyer on the accounting process and the buyer's current public process (cf Fig. 6) is non-empty. Thus, no change propagation is required.

Invariant subtractive changes can be handled accordingly and are therefore not further treated in this paper. Generally, when adding received messages to a process or removing sent messages from it we can obtain invariant changes.

5.2 Variant Additive Changes

The formal basis for variant additive changes is provided by Def. 5 and Def. 6: Let A and B be the aFSAs of two public processes and let A $\cap$ B $\neq \emptyset$ be the protocol between them. Let further δ be a change operation transforming A into A'. Then: δ is called *variant additive change* if the following constraint holds: $A' \setminus A \neq \emptyset \wedge A' \cap B = \emptyset$.

According to Def. 6 change propagation to B and the related private process become necessary now. We illustrate this scenario by an example. Assume that

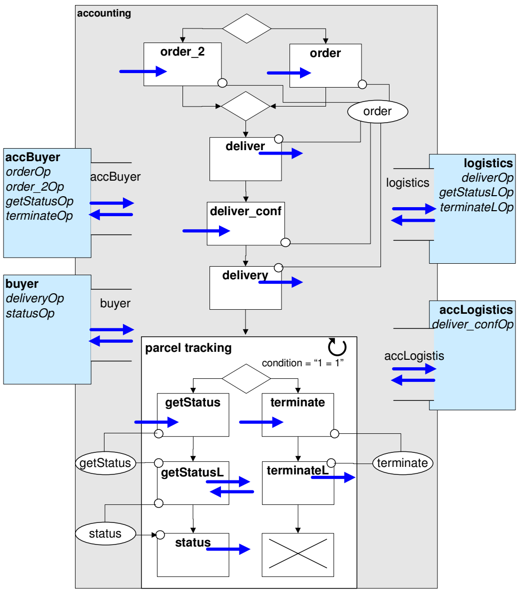

Fig. 8. Invariant Change of Accounting Private BPEL Process

the accounting private process shall be extended with the option to cancel orders (e.g., due to a product being out of stock). This change can be accomplished by adding a respective decision node and an activity to send the cancel message to the buyer ($A\#B\#cancelOp$) – the result is depicted in Fig. 10.

Next we derive the new version of the accounting public process and apply the buyer view on it (cf. Fig. 11a). Then we calculate the intersection of this automaton with the one of the buyer public process (cf. Fig. 6) which results in the automaton shown in Fig. 11b). Note that this automaton is empty since there exists no transition labeled $A\#B\#cancelOp$ on any path to a final state. This makes the annotation containing this message invalid and therefore results in an empty automaton. As a consequence, the variant change of the accounting process has to be propagated to the buyer process.

We now sketch the steps necessary for propagating addiditve changes to the opponent's private/public process:

1. Recalculate the opponent's public view on the new public process of the change originator and determine the newly inserted sequence (i.e., the messages potentially exchanged with the opponent's public process).
2. Calculate the union of the opponent's current public process and the newly introduced message sequence (cf. Step 1.) The resulting aFSA provides the basis for potential adaptations of the opponent's public process.
3. Based on the outcome of Step 2 we can derive those regions of the opponent's private process where adaptations may have to be performed.

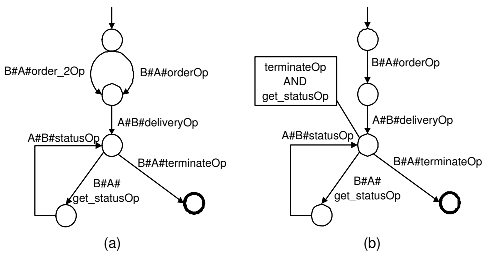

Fig. 9. (a) Public Buyer View on Accounting Process After Invariant Change (b) Intersection of a) with Buyer Public Process

4. Perform the necessary changes of the opponent's private process.
5. Recalculate the opponent's public process. If it is consistent with the public process of the change originator we are finished. Otherwise, go back to the previous step and repeat it with a modified set of changes.

We explain the different steps along our example:

ad 1) We determine the changes of the buyer view on the accounting public process A'. Based on this we calculate potential adaptations of the buyer public process B. More precisely we determine $A'' := \tau_{Buyer}(A') \setminus B$ (cf. Fig. 11c). In general, we have to consider the difference $\tau_{Buyer}(A') \setminus (\tau_{Buyer}(A) \cap B)$. However, since only those message sequences must be added to B which have not been contained before, derivation of $\tau_{Buyer}(A') \setminus B$ is sufficient.

ad 2) We calculate the union of the described difference (cf. Step 1) with the original buyer public process. Based on this we can derive potential changes of the buyer public process. – The union of two aFSAs can be created using the complement and intersection operator in accordance to the deMorgan law: $A \cup B \equiv \overline{\overline{A} \cap \overline{B}}$; thus, $B' := A'' \cup B$ (cf. Fig. 11d).

ad 3) We apply the potential changes to the buyer public process. The regions to be adapted in the corresponding private process can then be derived from the states that have been modified for the buyer public process. For this we use the mapping that was created when generating the buyer public process out of the corresponding private one. Note that observable states can be mapped to a particular process region and that non-present transitions provide a hint on what is missing exactly.

In order to derive the states which have been changed when transforming aFSA B to aFSA B' the difference automaton is traversed parallel to the original public process (comparable to bisimulation). In particular, the first state where the difference automaton contains a transition which is not contained in the original public process, indicates the state where a new transition has been added. The missing transition indicates which message has to be additionally considered by the private process. With regard to the Buyer public process, this is the case for state no. 2 in the original public process as depicted in Fig. 6.

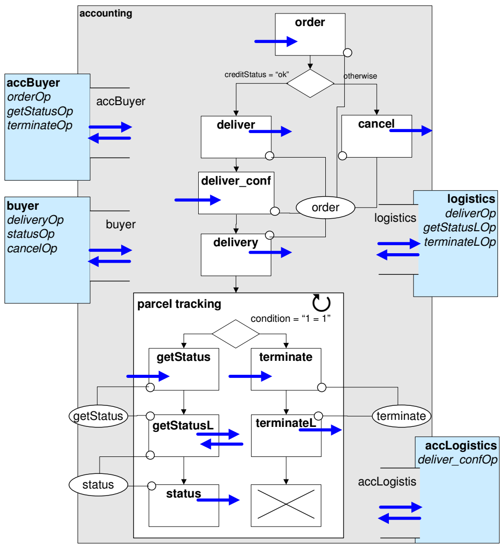

Fig. 10. Additive Change of Accounting Private BPEL Process

From the mapping table we can derive that the change in the Buyer private process is related to the block specified by the sequence activity labeled "buyer process". The receive delivery activity contained in the sequence has to be changed to a pick activity allowing to receive either the delivery or the cancel message. Information to be added can be derived from the difference automaton depicted in Fig. 11c). Fig. 12 shows the resulting Buyer private process.

ad 4 and 5) Finally, we perform the change of the private process accordingly and recalculate the public process on it. After this we check whether intersection of the changed buyer public process and the buyer view of the accounting public process is non-empty, i.e. whether related aFSAs are bilaterally consistent.

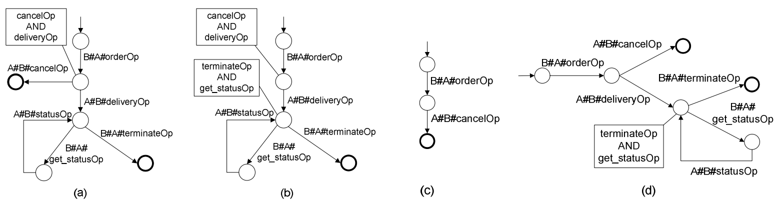

Fig. 11. Accounting Process: (a) Public (Buyer) View (b) Intersection of a) with Buyer Public View (c) Difference at Buyer View of Accounting Public Process (minimized) (d) New Buyer Public Workflow (minimized)

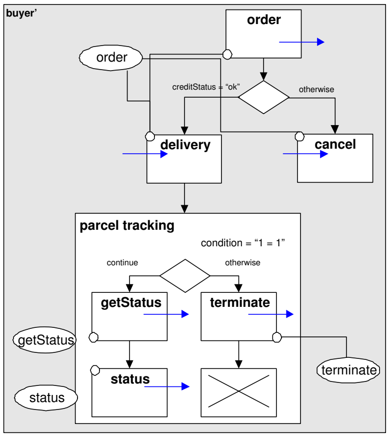

Fig. 12. Buyer Process after Propagation of Additive Changes

6 Implementation Issues

We have implemented the core of the presented approach in a proof-of-concept prototype [8]. Further, a partial mapping from BPEL (private processes) to annotated Finite State Automata (public processes) has been realized [11] and been used for implementing service discovery [13]. The extension of classical UDDI proposed in this context uses BPEL specifications of public processes and bilateral consistency to improve the precision of service discovery results. Finally, we have proposed a protocol to derive a potential cross-organizational process (i.e., a potential service composition in Web Service notation) in a decentralized way resulting in a set of services as a basis for consistency checking [14].

These building blocks can be used for setting up the concrete change framework of DYCHOR. As indicated the only information which has to be exchanged between partners is about the changes applied to public processes. The difference calculation as well as the necessary adaptations of the own public and private processes can be accomplished locally. Finally, decentralized consistency checking can be applied to guarantee the successful introduction of the changes and the consistency of the changed choreography.

7 Related Work

Handling of changes is known in software engineering as refactoring like e.g. [15], where the aim is to propagate changes without altering the behavior of the application. These approaches propagate changes on a centrally available source base and do not know different abstractions layers comparable to choreographies and orchestrations as discussed in this paper.

Checking consistency of a cross-organizational process can be based on the set of potential execution sequences. A straightforward approach is to check consistency on a centralized process representation, which has to be split into several public processes afterwards. This principle was applied to different process description formalism in the past, like Workflow Nets (WF Nets) [3], guarded Finite State Automata [16], Colored Place/Transition Nets [17], and Statecharts [18]. However, these top-down approaches are based on centralized consistency checking, which is different to the DYCHOR approach described in this paper.

By contrast, the bottom-up approach of constructing the cross-organizational process out of several public processes has not been investigated in sufficient detail so far. Respective proposals have been made, for example, in [6,16,7]. However, they require centralized decision making and are also not constructive; i.e., they only specify criteria for various notions of consistency but do not provide an approach to adapt public processes in a way making the overall cross-organizational process consistent. In addition, these approaches neither address synchronous communication nor allow for decentralized consistency checking.

Issues related to dynamic workflow change have been investigated in detail (e.g., [19,20,1]). Respective approaches address ad-hoc changes of single process instances as well as process schema evolution (i.e., controlled change of process types and propagation of these modifications to already running process instances [20,1]). However, these approaches focus on the adaptation of process orchestrations, i.e., process instances controlled by a single endpoint. By contrast, issues related to changes of process choreographies have been neglected so far. What can be learned from approaches dealing with dynamic changes of process orchestrations is the idea of controlled change propagation. These approaches aim at propagating process type changes to running process instances without causing inconsistencies or errors. Similarly, we have provided an approach for the controlled propagation of the changes of private processes within a choreography to the choreography itself and the respective partner processes.

8 Conclusion and Future Work

In this paper we have presented the DYCHOR framework for introducing changes to private processes, for recalculating related public views automatically, and for propagating resulting modifications to partner processes if required. We have provided a formal model and precise criteria allowing us to automatically decide which adaptations become necessary due to changes of private partner processes. The treatment of different change scenarios adds to the completeness of our approach. Finally, we have implemented the basic mechanisms presented in this paper in a proof-of-concept prototype. We will analyze how to adopt the implemented approach within a real application scenario.

Another challenging issue is the treatment of running process instances (participating in a choreography) when changing private and public process models. In particular, for long-running choreographies, the propagation of choreography changes to already running instances is highly desirable. In future work we will address this issue by elaborating different strategies. These strategies range from managing change propagation by a central coordinator to completely decentralized solutions (e.g., optimistic vs. pessimistic instance migrations). The ultimate challenge will be to address the problem of concurrent process choreography and process instance changes, i.e., how to propagate process choreography changes to process instances which have already been subject to ad-hoc changes themselves. Meeting these challenges will be key for future service-oriented infrastructures, ultimately resulting in highly adaptive process choreographies.

References

1. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems – a survey. DKE **50** (2004) 9–34
2. Rinderle, S., Reichert, M., Dadam, P.: Flexible support of team processes by adaptive workflow systems. Distributed and Parallel Databases **16** (2004) 91–116
3. Aalst, W., Weske, M.: The P2P approach to interorganizational workflows. In: Proc. CAiSE'06, Interlaken, Switzerland (2001)
4. Andrews et al., T.: Bpel4ws v 1.1 (2003)
5. RosettaNet: RosettaNet home page. http://www.rosettanet.org (2004)
6. Aalst, W.: Interorganizational workflows: An approach based on message sequence charts and petri nets. Systems Analysis - Modelling - Simulation **34** (1999) 335–367
7. Kindler, E., Martens, A., Reisig, W.: Inter-operability of workflow applications: Local criteria for global soundness. In: Business Process Management, Models, Techniques, and Empirical Studies, Springer-Verlag (2000) 235–253
8. Wombacher, A., Fankhauser, P., Mahleko, B., Neuhold:, E.: Matchmaking for business processes based on choreographies. IJWS **1** (2004) 14–32
9. Chomicki, J., Saake, G., eds.: Logics for Database and Information Systems. Kluwer (1998)
10. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison Wesley (2001)
11. Wombacher, A., Fankhauser, P., Neuhold, E.: Transforming BPEL into annotated deterministic finite state automata enabling process annotated service discovery. In: Proc. of Intl. Conf. on Web Services (ICWS). (2004) 316–323
12. Rinderle, S., Wombacher, A., Reichert, M.: On the controlled evolution of process choreographies. Technical Report TR-CTIT-05-47, University of Twente (2005)
13. Wombacher, A., Mahleko, B., Neuhold, E.: IPSI-PF: A business process matchmaking engine. In: Proc. of Conf. on Electronic Commerce (CEC). (2004) 137–145
14. Wombacher, A.: Decentralized decision making protocol for service composition. In: Proc IEEE Int Conf on Web Services (ICWS). (2005) (accepted for publication).
15. Mens, T., Tourwe, T.: A survey of software refactoring. IEEE Transactions on Software Engineering **30** (2004) 126–139
16. Fu, X., Bultan, T., Su, J.: Realizability of conversation protocols with message contents. In: Proc. IEEE Intl. Conf. on Web Services (ICWS). (2004) 96–103
17. Yi, X., Kochut, K.J.: Process composition of web services with complex conversation protocols. In: Proc. Conf. on Design, Analysis, and Simulation of Distributed Systems Symposium at Adavanced Simulation Technology. (2004) 141–148
18. Wodtke, D., Weikum, G.: A formal foundation for distributed workflow execution based on state charts. In: Proc. ICDT'06. (1997) 230–246
19. v.d. Aalst, W., Basten, T.: Inheritance of workflows: An approach to tackling problems related to change. Theoret. Comp. Science **270** (2002) 125–203
20. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. DKE **24** (1998) 211–238