Equivalence Transformations on Interorganizational Processes to Shift Communication Steps

Technical Report Version 0.91

Julius Köpke, Johann Eder

Department of Informatics-Systems,
A-9020 Klagenfurt, Austria,
{julius.koepke, johann.eder}@aau.at

Abstract. Distributed interorganizational processes can be designed by first creating a global process, which is then split into processes or views for each participant. Existing methods for automating this transformation concentrate on the control flow and neglect either the data flow or address it only partially. Even for small interorganizational processes, there is a considerably large number of potential realizations of the data flow. We analyze the problem of generating message exchanges to realize the dataflow in depth and present a solution for constructing data flows which are optimal with respect to some design objectives. The approach is based on a definition of the correctness of data flow and a complete set of transformations which preserve correctness and allow to search for an optimal solution from a generated correct solution.

1 Introduction

Interorganizational business processes face the challenge to broaden the highly successful technology of *intra*organizational processes to a fully distributed collaboration of autonomous entities retaining the advantages of intraorganizational business process management which extensively takes advantage of a central coordination. One of the major differences between centralized and distributed process management is the access to data: uniform access to a joint central data store on one hand and distributed management of data with explicit exchange of data via messages on the other hand.

In this paper we focus an a phase in the development of an interorganizational workflow where the explicit dataflow between participants is established. Starting point of our considerations is an interorganizational process definition which assumes a global data store. This model is then augmented with messages for passing data between participants such that the process model can be executed in a full distributed way, respectively projected onto the participants to define the interface of their internal process (e.g. by process views [3]). An initial process definition consists of a set of activities, control flow between the activities, assignment of the activities to participants and input and output parameters of

activities. Many approaches such as [12, 13, 26, 24, 9, 6] start with a global process definition and follow a top down or mixed strategy. A global process definition including input and output data already implicitly defines the data flow between participants. For the explicit realization of the data flow, however, there are numerous possibilities [15, 19]. Nevertheless, there are no approaches which take this multitude of solutions explicitly into account and, therefore, cannot reason about the quality of the solution. While [26, 23, 24] do not consider data flow at all, [13, 12] restrict the data flow to the distribution of decision variables. [9, 6] address data flow, however, only a single solution based on one fixed strategy is generated.

Take for example the following trivial process fragment: lets an activity A produce the parameters x and y, the succeeding activity B updates x, and the third activity C needs x and y. There are basically two solutions: (a) *transitive transfer:* the interface of B is widened to also include y (assuming that B is also admitted to see y) such that B can pass y to C, or (b) *explicit data channel* [19]: A sends y directly to C which requires additional messaging activities which are not yet included in the process definition. Now consider that B is executed conditionally. A simple solution, as proposed by [6] is that A always sends x and y to C. On the one hand, this results in additional message overhead and on the other hand C may not even be allowed to get access to the (intermediate) value of x, if B is executed. If this is the case a better solution would be to only transfer x from A to C, if B is not executed later.

One can easily see that for a given process definition as above there are numerous solutions for establishing a correct explicit data flow. We can reason about properties of a solution and define criteria, such as the number of (additional) data transfers via messages, the number of transitively passed data, etc., for choosing among the possible solutions.

The major contribution of this paper is a set of equivalence transformations on processes with explicit data flow that allow us to define the complete solution space in which we can (heuristically) search for the best solution with respect to constraints and an objective function.

The results presented here can be used for several purposes: It is possible to automatically generate the explicit data flow in interorganizational workflows, to check whether a participant with a given unchangeable process interface can be accommodated to join the interorganizational workflow, or to verify and evaluate procedures and guidelines for establishing the data flow for interorganizational processes.

2 Process Model

2.1 Basic Process Model

We follow here the approach that an interorganizational business process is defined as a process rather than as a of set of protocols between two participants. For defining the process we use block-structured workflow nets [8] supporting the

usual basic control flow patterns sequence, PAR split / join, and XOR split/join [25, 22]. We focus on block structured workflow-nets as they prevent typical flaws of unstructured business processes dealing with data [1] and are also in line with the WS-BPEL[17] standard. The process definition is extended with data definition, i.e. global variables may be defined and for each activity we denote which variables are input for this activity and which variables are output. Furthermore, we assign to each activity and each control step to one of the participants as actor. Figure 1 shows both the graphical and textual representation of our process meta mode. In our notation, $A_a(R, W)$ is an activity step where A is the label of the task to be executed by particpant a. R defines the set of input variables, W defines the set of output variables. Abstract blocks are placeholders for any sub-process (including empty ones) and are represented by their label. $SEQ(A, B)$ defines a sequence of the blocks A and B. Sequences of more than two elements can also be defined by nesting: $SEQ(A, SEQ(B, C)) \equiv SEQ(A, B, C)$. $XOR_{px,pj}(cb, A, B)$ defines a xor-block, where the xor-split is executed by participant px, the xor-join is executed by participant pj. Block A is executed, if the condition cb holds, otherwise B. $PAR_{ps,pj}(A, B)$ defines a par-split, where the split is executed by participant ps and the join is executed by participant pj. We also use a graphical representation which in analogy to the usual BPMN notation. The major difference is that we also depict which participant executes a step (subscript letters), the set of input and output variables of activity-steps and the condition of XOR splits. A communication step (also called send-receive step) is denoted by $SR_s(X, r, c)$ defines that parter s sends the content of the set of variables X to participant r, if the condition c holds.

2.2 Decision Model and Coordination

The XOR split requires special attention in interorganizational processes. There are the following possibilities: (1) The condition is not defined in the global process, or (2) the condition is defined using some global variables. In case (1) the actor of the XOR-split makes the decision and informs the other participants, if necessary. In case (2) each participant could make the decision. However, this requires that all participants receive all variables appearing in the condition to make the decision (i.e. evaluate the condition). This may result in additional communication overhead. Therefore and for providing a uniform treatment for both cases we treat case(2) like case (1): the actor of the XOR split evaluates the condition.

There exist various possibilities for the coordination of different participants:

(1) deferred constructs, where the participants are implicitly informed by the message they receive or do not receive. For an example, step B_b in Figure 2 does not need to know about the decision of the xor-split. Only when B_b is called participant b joins the process. In contrast participant a who executes the steps A_a and E_A must also be informed if E_a is not called. Otherwise, a would wait forever to be called. (See also death paths elimination in BPEL [9].)

(2) the actor sends the result of the decisions to the other participants. This allows each participant to execute each (required) xor-block locally.

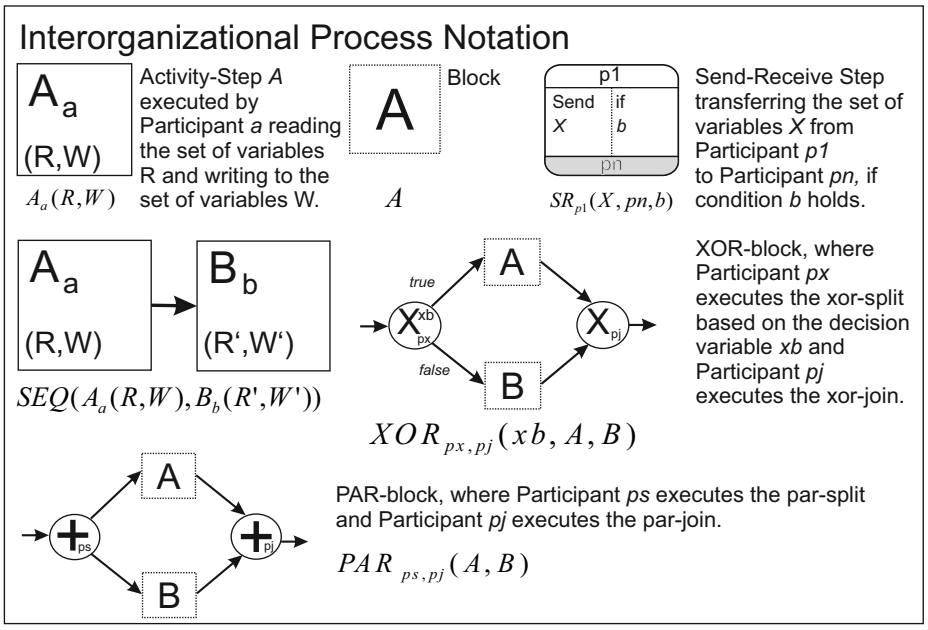

Fig. 1. Graphical Representation of the Process Model

We follow the second approach and require that for $XOR_{xs,pj}(b, A, B)$ the condition b refers only to one single boolean variable called decision variable, which is output of a preceding activity called decision step $DESC_{xs}(\{...\}, \{b\})$ with the same actor as the xor-split. This is not a restriction of the generality as this pattern can be generated automatically. For the coordination it is important that all participants take the same decisions. Since variables can in general be updated, dependent xor-gateways or send-receive steps rely on the value a decision variable had when it was written by the deciding participant. To make our life (and that of workflow designers) easier, we require that decision variables must not be updated after their corresponding xor-gateway was executed.

Data access within parallel blocks may lead to race conditions. In an intraorganizational setting this can be resolved by a transactional data store. For distributed processes we do not assume a distributed transactional data store and, therefore, do not allow parallel read-write or write-write dependencies between variables, i.e. if a variable is output in some activity it must not appear as input or output in branches parallel to this activity.

2.3 Realizing Interorganizational Data-Flow

The process definition discussed above uses global variables as if there would be a global data store like in intraorganizational processes. This means it is assumed that each activity can access the most recent value of each variable. Implicitly

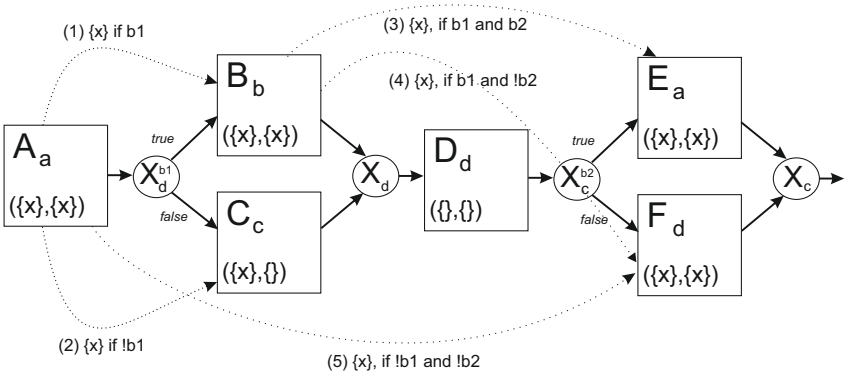

Fig. 2. Implicit Data-Flow in an interorganizational Process

this defines a data flow between the participants. For the fully distributed enactment of interorganizational processes we have to realize this implicit data flow by augmenting the process definition with explicit message exchanges which pass the content of variables between participants.

Figure 2 presents an example process using the graphical representation. The first step is represented as $A_a(\{x\}\{x\})$ in the textual representation. Therefore, step A of participant a has variable x as input and as output.

In the example there are 5 data-flow dependencies between tasks of different participants shown with dotted lines. Data-flow dependencies can be conditional. For example step E_a only needs x from B_b, if B_b and E_a are executed (if the conditions $b1$ and $b2$ hold). In order to support an interorganizational operation of the process, each data-flow between different participants needs to be implemented by messages. Now there exist multiple solutions to support the data needs of E_a. We may add a message exchange right after the execution of B_b. In this case the message is only sent if B_b is executed. However, it will always be sent, even if E_a is not executed later. The decision whether E_a is executed is made later. Therefore, it is impossible to predict whether the message needs to be sent or not. However, sending the message does not lead to an error in the data-flow. A message is sent but its contents are not consumed as they are overridden by a succeeding message. So this solution contains redundant message exchanges but it is correct. Another option is to add the message exchange sending x from b to a directly before E_a. However, in this case it must only be executed if B_b was executed. Otherwise a will get a wrong value for x. While b knows wether B_b was executed participant a does not and therefore needs to know whether a message will arrive.

To cope with this problem our process model supports the notion of conditional message exchanges, which are implemented with communication steps (also called send-receive steps). A send-receive step is denoted by $SR_{p1}(X, bn, b)$, meaning that parter $p1$ sends the content of the set of variables X to participant

bp, if the condition b holds. In the graphical notation we represent send-receive steps in analogy to BPMN choreography tasks. See the first step of $TS1a$ in Figure 3 as an example. A communication step is implemented as a sending task in the local process of the sender and as a receiving task in the local process of the receiver. In our example, we can now add the send-receive step $SR_b(\{x\}, a, b1)$ directly before E_a to solve the previously discussed problem.

2.4 Process Model Definitions

In this section we formalize our workflow model and provide the necessary definitions for the specification and verification of equivalence transformations. We define a process model here as a nested structure to emphasize that we deal with full blocked processes. this nested structure can easily be transformed into a process graph (workflow net).

Definition 1. Process Model: A process model consists of sets of participants (partners) P, variables V (including boolean decision variables D), task labels T, and a block defined recursively as follows:
Let S be a task label, R, W sets of variables, A, B blocks c a decision variable, b a boolean expression consisting of the decision variables $d_1, \ldots d_n$ then

1. $S_{p1}(R, W)$ is a block (activity step),
2. $SR_s(X, r, b)$ is a block (communication step),
3. the empty block ϵ is a block, and
4. $SEQ(A, \ldots, B)$ is a block (sequence)
5. $XOR_{p1,p2}(c, A, B)$ is a block (alternative),
6. $PAR_{p1,p2}(A, B)$ is a block (parallelism).

All $p1, p2$ are called *actors* of their respective blocks, s is the sender and r is the recipient of a communication step, R, X, c, and $d_1, \ldots d_n$ are (sets of) input variables, W and X are output variables.

In addition, a block inherits all input variables of its superordinate block. Predecessor and successor relationships are defined as usual.

The only type of blocks which may contain subordinate block and that has input variables (decision variables) are xor-blocks. Xor-Blocks have exactly one decision variables as input, which defines which branch of the xor-block is taken. All blocks which are nested in a xor-block have the decision variable as input because the corresponding participants must know which route was taken.

Definition 2. Initial Process: An *initial process* is a process that does not contain any communication steps.

Definition 3. Augmentation: An *augmentation* of an initial process P contains all the steps of P in the same topological order and some communication steps in addition.

Definition 4. Instance Type and Instantiation: Every possible instantiation I of the set of decision variables of a process P constitutes an *instance type*, P^I which is defined as a sub-model where each xor has exactly one sub-block (depending on the value of the decision variable) and only those communication steps where the condition evaluates to *true*.

We now define that such an augmented process correctly realizes the implicit data flow of an interorganizational process if for centralized and distributed executions the value of each input variable of a step originates from the output of the same activity.

Definition 5. Origin: The *origin* of an input parameter x in block a of an instance type I, $o(P^I, a, x)$, is defined as follows: If b is the closest predecessor activity step of a with x as output parameter then $o(P^I, a, x) = b$.

Definition 6. Correctness of an Initial Process: An initial process is correct, if all input parameters of all steps have a unique origin.

This correctness requirements covers the usual data flow faults like uninitialized variables and race conditions [21]. This also means that we do not allow data-flow between parallel blocks. We emphasize that due to the hierarchical definition of process models it is not possible to define an incorrect workflow net.

For the distributed execution of an augmented process we have to consider that a participant only can access the content of a variable if it was produced locally or if it was received through a communication step.

Definition 7. Distributed Origin: The *distributed origin* of the input parameter x in block a of an instance type P^I, $o^d(P^I, a, x)$ is defined as follows: Let p be the actor or sender of a and let b be the closest predecessor step of a with x as output parameter and p as actor (for activity steps) or recipient (for communication steps). If b is an activity step then $o^d(P^I, a, x) = b$, if b is a communication step $SR_s(X, p, b)$ then $o^d(P^I, a, x) = o^d(P^I, b, x)$.

Definition 8. Correct Augmentation The augmentation P of a process is correct, iff for each instantiation I of decision variables, for each input variable x of each block a: $o(P^I, a, x)$ exists and is unique and $o^d(P^I, a, x) = o(P^I, a, x)$.

3 Equivalence Transformation on Augmented Processes

There exists numerous correct augmentations of the data-flow of a process. For example all updated variables may be sent as soon as possible to all participants, they may be sent as late as possible or every data-exchange may follow the control-flow including transitive transfers. In this section we present a set of transformations on augmented processes that allow to derive all other correct augmentations.

Definition 9. Equivalence Transformation on Augmented Processes: A transformation T on the communication steps of a correctly augmented process P, resulting in a new process P' is an equivalence transformation, iff P' is also a correctly augmented process for any P and any possible application of T on P.

3.1 Equivalence Transformations on Sequences

We provide a graphical description of equivalence transformations on sequences in Figure 3 and discuss each transformation shortly in the remainder of this section. The function $ref(b)$ returns the set of all variables, referenced by the boolean expression b.

TS1a - Swap (Send-Receive / Activity): A send-receive step can be swapped with an activity, unless the activity is the destination of the send-receive step or the activity writes to some variable transmitted by the send-receive step: $SEQ(sr_{p1}(X, pn, b), a_{p2}(R, W)) \equiv SEQ(a_{p2}(R, W), sr_{p1}(X, pn, b))$, unless $(W \cap X \neq \{\}) \vee (R \cap X \neq \{\} \wedge pn = p2) \vee ref(b) \subseteq W$. The predicate $ref(b)$ returns the set of all (decision) variables that are referenced by b.)

Proof 1. *Proof that TS1a is an equivalence transformation*

Let P be a correctly augmented process and P' be a process derived from P by applying $TS1a$ once. Based on Definition 8 for every instantiation I of decision variables, for each input variable x of each block r: $o(P^I, a, x)$ exists and is unique and $o^d(P^I, a, x) = o(P^I, a, x)$. We will show that this property is preserved for P' by proving that for every instantiation I and every block r, reading any variable x: $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$ holds.
Therefore, there also exists a unique origin for $o(P'^I, r, x)$. There are the following cases for the reading block r:

1. r is located anywhere before the swapped elements:
   $SEQ(r, ..., a_{p2}(R, W), sr_{p1}(X, pn, b))$ or $SEQ(r, ..., sr_{p1}(X, pn, b), a_{p2}(R, W))$.
   The origin and distributed origin only address previous steps of r.
   Therefore, for every input variable x of r:
   $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.
2. r is the send-receive step: $r = sr_{p1}(X, pn, b)$ Input variables of r are the set X and all referenced variables of the condition $ref(b)$. From the precondition of $TS1a$ follows that $W \cap X \equiv \{\} \wedge ref(b) \nsubseteq W$. Therefore, for every input variable x of r:
   $a_{p2}(R, W) \neq o(P'^I, r, x) \wedge a_{p2}(R, W) \neq o^d(P'^I, r, x)$. Consequently, the origin is some step before $a_{p2}(R, W)$. Based on the definition of the origin and distributed origin: $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.
3. r is the activity-step: $r = a_{p2}(R, W)$. Input variables of r are defined by the set R. From the precondition of $TS1a$ follows that if $R \cap X \neq \{\}$, then pn

$\neq p2$. Therefore, for every input variable x of r:
$x \in R \wedge x \in X \Longrightarrow pn \neq p2$. $a_{p2}(R,W)$ is the direct predecessor or successor of $a_{p2}(R,W)$. Therefore, $o^d(P^I,r,x)$ cannot be realized via $sr_{p1}(X,pn,b)$ in P or P'. Consequently $o^d(P^I,r,x) = o(P^I,r,x) = o^d(P'^I,r,x) = o(P'^I,r,x)$.

4. r is any step after the swapped elements in P or P' and before another activity writing to x in P^I or P'^I.
$SEQ(a_{p2}(R,W), sr_{p1}(X,pn,b), \ldots, r)$. From the precondition of $TS1a$ follows that $W \cap X \equiv \{\}$. Therefore, for every input variable x of r:
$x \in W \Longrightarrow x \notin X$.
$x \in X \Longrightarrow x \notin W$.
if $x \in W$, and $o(P^I,r,x) = a_{p2}(R,W)$, then also $od(P^I,r,x) = a_{p2}(R,W) \wedge o(P'^I,r,x) = a_{p2}(R,W)$ (the origin is not influenced by intermediate send-receive steps). Since $sr_{p1}(X,pn,b)$ cannot send x, also $o^d(P'^I,r,x) = a_{p2}(R,W)$
if $x \in X$: $a_{p2}(R,W)$ cannot be the origin of x.
Therefore, $o^d(P^I,r,x) = o(P^I,r,x) = o^d(P'^I,r,x) = o(P'^I,r,x)$.

□

TS1b - Swap (Send-Receive - Send-Receive): Two send-receive steps in a sequence can be swapped, unless one is the destination of the other.
$SEQ(sr_{p1}(X,pn,b1), sr_{p2}(Y,pm,b2)) \equiv SEQ(sr_{p2}(Y,pm,b2), sr_{p1}(X,pn,b1))$, unless: $(pn = p2 \vee pm = p1) \wedge (X \cap Y \neq \{\} \vee ref(b1) \subseteq Y \vee ref(b2) \subseteq X)$.

Proof 2. *Proof that TS1b is an equivalence transformation*

Let P be a correctly augmented process and P' be a process derived from P by applying $TS1b$ once. Based on Definition 8 for every instantiation I of decision variables, for each input variable x of each block r: $o(P^I,a,x)$ exists and is unique and $o^d(P^I,a,x) = o(P^I,a,x)$. We will show that this property is preserved for P' by proving that for every instantiation I and every block r reading any variable x: $o^d(P^I,r,x) = o(P^I,r,x) = o^d(P'^I,r,x) = o(P'^I,r,x)$. Therefore, there also exists a unique origin for $o(P'^I,r,x)$. There are the following cases for the reading block r:

1. r is located anywhere before the swapped elements:
See Case 1 of proof of $TS1a$.
2. r is one of the send-receive steps: Input variables of r are the set X and all referenced variables of the condition $ref(b)$. From the precondition of $TS1b$ follows $(pn \neq p2 \wedge pm \neq p1) \vee (X \cap Y \equiv \{\} \wedge ref(b1) \nsubseteq Y \wedge ref(b2) \nsubseteq X)$. Since there is no other step between the send-receive steps, $o^d(P^I,r,x)$ or $o^d(P'^I,r,x)$ cannot be realized via the other send-receive step. Therefore, for every input variable x of r: $o^d(P^I,r,x) = o(P^I,r,x) = o^d(P'^I,r,x) = o(P'^I,r,x)$.
3. r is any step after the swapped elements in P or P' and before another activity writing to x in P^I or P'^I:
$SEQ(sr_{p1}(X,pn,b1), sr_{p2}(Y,pm,b2), \ldots, r)$ or

$SEQ(sr_{p2}(Y, pm, b2), sr_{p1}(X, pn, b1), \ldots, r)$
For every input variable x of r:
$x \in X \wedge pn \equiv p2 \Longrightarrow x \notin Y$
$x \in Y \wedge pm \equiv p1 \Longrightarrow x \notin X$
Therefore, $sr_{p1}(X, pn, b1)$ and $sr_{p2}(Y, pm, b2)$ are independent of each other. There cannot exist any distributed origin $o^d(P^I, r, x)$ which is based on $sr_{p1}(X, pn, b1)$ and on $sr_{p2}(Y, pm, b2)$. Therefore, $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.

□

TS2 - Change Sender: Direct sending of variables to multiple participants is equivalent to transitive sending of the same set of variables to the same participants. $SEQ(sr_{p1}(X, pn, b), sr_{pn}(X, pm, b)) \equiv SEQ(sr_{p1}(X, pn, b), sr_{p1}(X, pm, b))$

Proof 3. *Proof that TS2 is an equivalence transformation*

Let P be a correct augmentation and P' be an augmentation derived from P by applying $TS2$ once: For every instantiation I and every block r executed after $sr_{pn}(X, pm, b)$ or after $sr_{p1}(X, pm, b)$ reading any variable x: $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$. We get the following cases for r:

- r has $p1$ or pn as actor: r cannot be effected by $TS2$ since $p1$ must have the variable and pn either sends it ore gets it by $p1$ senders.

- r has pm as actor and some step s_{p1} as origin:
  For P (left hand side of $TS2$): $o^d(P^I, r, x) = o^d(P^I, sr_{pn}(X, pm, b), x) = o^d(P^I, sr_{p1}(X, pn, b), x) = s_{p1}$
  For P' (right hand side of $TS2$): $o^d(P'^I, r, x) = o^d(P'^I, sr_{p1}(X, pm, b), x) = s_{p1}$

- r has px as actor and some step s_{pz} as origin and x is transported via $p1$ and pm to px: For P: $o^d(P^I, r, x) = \ldots\ o^d(P^I, sr_{pn}(X, pm, b), x) = o^d(P^I, sr_{p1}(X, pn, b), x) \ \ldots\ = s_{pz}$
  For P': $o^d(P'^I, r, x) = \ldots\ o^d(P'^I, sr_{p1}(X, pm, b), x) \ \ldots\ = s_{pz}$

□

TS3 - Remove/Add at End: A send receive step at the end of a process is equivalent to no send-receive step at the end of the process. $SEQ(A, sr_p(X, p', b))$ $\equiv A$, when the sequence is located at the upper most level of the process.

Proof 4. *Proof that TS3 is an equivalence transformation*

Obviously there cannot exist any instance type of the process, where some block reads any variable after the end of the process. Therefore, the correctness

of the augmentation cannot be effected by removing (or adding arbitrary send-receive steps) at the end of the process. □

TS4 - Absorb/Add: A send-receive step that sends only variables written by some succeeding activity step is equivalent to only the execution of the activity-step: $SEQ(sr_p(X, pn, b), a_{pm}(R, W)) \equiv a_{pm}(R, W)$ where $X \subseteq W$, unless $R \cap X \neq \{\} \wedge pn = pm$.

Proof 5. *Proof that TS4 is an equivalence transformation*

Based on the precondition of $TS4$ follows: $pn = pm \Longrightarrow R \cap X \equiv \{\}$. Therefore, $a_{pn}(R, W)$ cannot read x, if $sr_p(X, pn, b)$ sends x to pm. Consequently only the distributed origin of variables read by blocks executed after $a_{pn}(R, W)$ can be effected by $TS4$. However, directly following from the definition of the origin and the distributed origin every succeeding step of $a_{pn}(R, W)$ has $a_{pm}(R, W)$ as its origin and its distributed origin for every variable $v \in W$ unless another step writing to v is executed after $a_{pn}(R, W)$. Therefore, the origin and the distributed origin cannot be changed by the application of the transformation.

□

TS5 - Split/Merge of Variables It is equivalent to transmit a set of variables by one single send-receive step or by two send-receive steps:
$SEQ(sr_{p1}(X, pn, b), sr_{p1}(Y, pn, b)) \equiv sr_{p1}(X \cup Y, pn, b)$

Proof 6. *Proof-Sketch that TS5 is an equivalence transformation*

The proof follows directly from the definition of the distributed origin. In order to guarantee that distributed origin and origin are equivalent for every read variable, the packaging of variables to send-receive steps has no influence as long as the condition for the execution of the communication step(s) are equivalent, which is guaranteed by the transformation.

□

TS6 - Split/Merge Conditions: Two send-receive steps in a sequence that transfer the same set of variables from the same source participant to the same target participant are equivalent to one single send-receive, which is executed if at least one of the conditions holds:
$SEQ(sr_{p1}(X, pn, b_1), sr_{p1}(X, pn, b_2)) \equiv sr_{p1}(X, pn, \{b1 \vee b2\})$

Proof 7. *Proof that TS6 is an equivalence transformation*

$SEQ(sr_{p1}(X, pn, b_1), sr_{p1}(X, pn, b_2))$ and $sr_{p1}(X, pn, \{b1 \vee b2\})$ result ins exactly the same possible instance types. Therefore, they are strictly equivalent.

□

3.2 Equivalence Transformations on XOR

We first introduce two predicates: $hasValue$ and $takesPart$. *hasValue(p1,var,pos)* returns *true*, if participant $p1$ certainly has the value of the variable *var* before

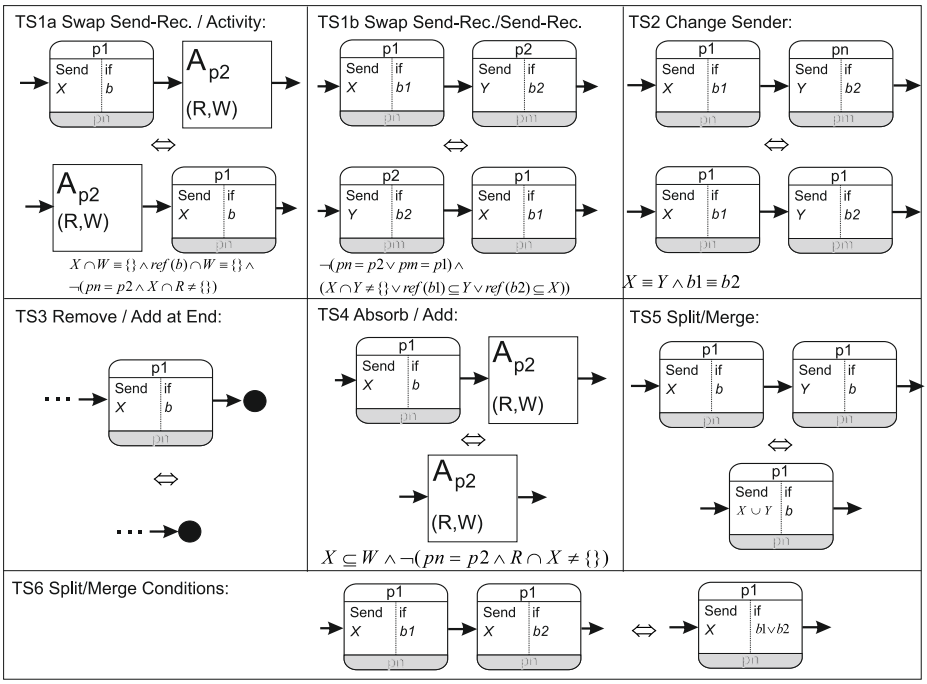

Fig. 3. Equivalence Transformations on Sequences

the execution of the block *pos. takesPart(xorBlock,participant)* returns *true*, if the participant *participant* executes any step of the xor-block *xorBlock* or an subordinate block. Figure 4 shows all equivalence transformations related to xor-blocks.

TX1 - Passing XOR-splits One send-receive step s located directly before a xor-split is equivalent to two send-receive steps with the same parameters as s, where one is located in each branch of the xor-split directly following the xor-split, if the sender and the receiver of s have the current value of the decision variable: $SEQ(SR_{p1}(X, pn, b), XOR_{p2,pj}(xb, A, B)) \equiv$
$XOR_{p2,pj}(xb, SEQ(SR_{p1}(X, pn, b)\ , A), SEQ(SR_{p1}(X, pn, b), B))$,
if $hasValue(p1, xb, XS_{p2,pj} \wedge has\ Value\ (pn, xb, XS_{p2,pj})$.

Proof 8. *Proof that TX1 is an equivalence transformation*

Let P be a correctly augmented process and P' be a process derived from P by applying $TX1$ once. Based on Definition 8 for every instantiation I of decision variables, for each input variable x of each block r: $o(P^I, a, x)$ exists and is unique and $o^d(P^I, a, x) = o(P^I, a, x)$. We will show that this property is preserved for P' by proving that for every instantiation I and every block r

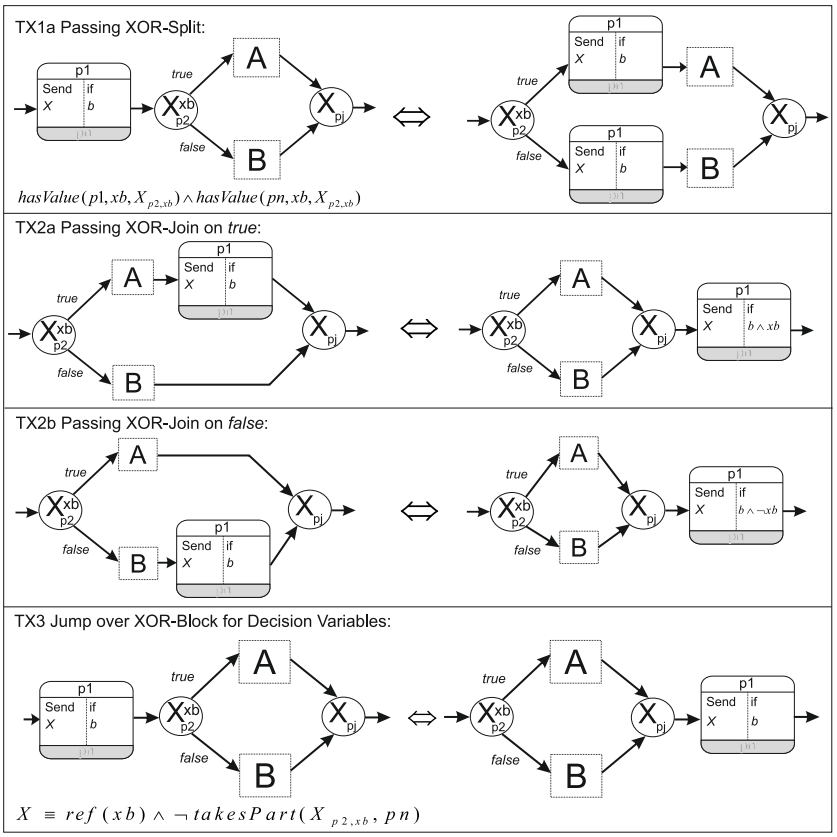

Fig. 4. Equivalence Transformations of Data-Send-Receive Steps for XOR

reading any variable x: $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$. Therefore, there also exists a unique origin for $o(P'^I, r, x)$. In analogy to the proof of $TS1a$ the potentially effected reader r reading variable x cannot exist before $XOR_{p2,pj}$. The precondition of $TX1$ requires that $hasValue(p1, xb, XS_{p2,pj} \wedge hasValue\ (pn, xb, XS_{p2,pj})$. Therefore, when $r = XOR_{p2,pj}(xb, A, B)$, r can only be a reader of x if the transmission is redundant. E.g. there must exist a different send-receive step for x, before r, which will result in the same distributed origin of x for r.

A reader with changed distributed origin may consequently only exist after the xor-split. For the xor-block $XOR_{p2,pj}(xb, A, B)$ their are exactly two different instantiations: Either the value of the decision variable xb is $false$ or $true$. The send-receive step is added to both branches and therefore, for every possible instantiation the send-receive step follows the xor-node. From the fact that the xor-block itself cannot be an origin for any step (does not write to variables) follows that no subsequent step can have a different distributed origin or origin when $SR_{p1}(X, pn, b)$ is located before or after the xor-split. □

TX2 - Passing XOR-Join *TX2a Passing XOR-Join on true:* One send-receive step s located directly before a xor-join in the *true* branch of a xor-split is equivalent to one send-receive step s' directly after the xor-join, if all parameters of s' and s are equivalent but the condition in s' is a conjunction of the one of s and the decision variable of the xor-split. $XOR_{p2,pj}(xb, SEQ(A, SR_{p1}(X, pn, b)), B) \equiv SEQ(XOR_{p2,pj}(xb, A, B), SR_{p1}(X, pn, \{b \wedge xb\})$
TX2B Passing Join on false: $XOR_{p2,pj}(xb, A, SEQ(B, SR_{p1}(X, pn, b))) \equiv SEQ(XOR_{p2,pj}(xb, A, B), SR_{p1}(X, pn, \{b \wedge \neg xb\})$

Proof 9. *Proof-Sketch that TX2 is an equivalence transformation*

1. The XOR-Join has no input variable. Therefore, it cannot be a reader.
2. The XOR-Join has no output variable. Therefore, no subsequent step can have it as an origin.
3. In every instantiation only one branch of the xor-block can be executed. Therefore, no subsequent reader can be effected by the change. □

TX3 - Jump over XOR-Block A send-receive step s, which is located directly before a xor-split is equivalent to a send-receive step directly after the corresponding xor-join, if s transmits only the decision variable of the xor-split and the target participant of s does not take part in the xor-block or any sub-block of it: $SEQ(SR_{p1}(X, pn, b), XOR_{p2,pj}(xb, A, B)) \equiv SEQ(XOR_{p2,pj}(xb, A, B), SR_{p1}(X, pn, b))$, if $\neg takesPart(XS_{p2}, pn) \wedge X \equiv \{xb\}$

Proof 10. *Proof that TX3 is an equivalence transformation*

Based on the definition of inputs of blocks, if a participant des not take part in the XOR-block, then this participant cannot have an activity which requires the decision variable as input inside the xor-block. E.g. there exists no step, inside the xor-block where the origin or distributed origin can be effected by the change.

□

TX4 - Inherit Conditions Given a send-receive step s with a condition b, which is nested into some xor-block x referencing the decision variable bx: $b \equiv b \wedge bx$, if s is located in Block A of x and $b \equiv b \wedge \neg bx$ if s is located in Block B of x.

Proof 11. *Proof-Sketch that TX4 is an equivalence transformation*

An defined in the process model every block inherits the conditions of every superordinate block implicitly. Making this explicit can consequently not change the correctness. □

TX1b - Add Send/Receive after XOR-Split Given a send-receive step s as a direct successor of a xor-split, we can add another send-receive step s' with the same parameter as s as a direct successor of the xor-split in the other branch. This is a one-way transformation. $XOR_{p2,pj}(xb, SEQ(SR_{p1}(Xpn, b), A), B) \vee XOR_{p2,pj}(xb, A, SEQ(SR_{p1}(Xpn, b), B)$
$\Longrightarrow XOR_{p2,pj}(xb, SEQ\ (SR_{p1}\ (X, pn, b), A), SEQ(SR_{p1}(X, pn, b), B))$.

Proof 12. *Proof-Sketch that TX1b is an equivalence transformation*

$TX1b$ is a shortcut for the application of $TS3$ and $TS4$ in combination with the other transformations. See the corresponding proofs. □

3.3 Equivalence Transformations on Parallel Blocks

Equivalence transformations regarding par-blocks need to consider in which branch reading or writing activities of the transmitted variables are located. We first define the predicates $hasWriter$, $hasReader$ and $inBlock$:
$hasWriter(var, Block)$ returns true if the variable var is written anywhere in the block (recursively). $hasReader(var, block, participant)$ returns true, if the variable var is read by participant $participant$ in the block (recursively). *inBlock(var, Block, participant)* is true, if $hasWriter(var, Block)$ or $hasReader$ $(var, block, participant)$. Figure 5 illustrates the transformations $TP1a$, $TP1c$, $TP2a$ as examples.

TP1: Passing PAR-Split: A communication-step, which is located directly before a par-split is equivalent to a communication-step in the first position of the branch with a consumer or a writer to every transferred variable. There are the following cases: A consumer or writer for every transmitted variable is in block A (TP1a), a consumer or writer for every transmitted variable is in block B (TP1b), a consumer for every transmitted variable is in Block A and in B (TP1c). If there is no consumer of any transmitted variable in A and B, then also TP1a and TP1b applies.

TP1a: $SEQ(SR_{p1}(X, pn, b), PAR_{ps,pj}(A, B)) \equiv PAR_{ps,pj}(SEQ(SR_{p1}(X, pn, b), A), B)$, if $\forall v \in X : (inBlock(v, A, pn) \wedge \neg inBlock(v, B, pn)) \vee (\neg inBlock(v, A, pn) \wedge \neg inBlock(v, B, pn))$

TP1b: $SEQ(SR_{p1}(X, pn, b), PAR_{ps,pj}(A, B)) \equiv PAR_{ps,pj}(A, SEQ(SR_{p1}(X, pn, b), B))$, if $\forall v \in X : (inBlock(v, B, pn) \wedge \neg inBlock(v, A, pn)) \vee (\neg inBlock(v, A, pn) \wedge \neg inBlock(v, B, pn))$

TP1c: $SEQ(SR_{p1}(X, pn, b), PAR_{ps,pj}(A, B)) \equiv PAR_{ps,pj}(SEQ(SR_{p1}(X, pn, b), A), SEQ(SR_{p1}(X, pn, b), B))$, if $\forall v \in X : (inBlock(v, A, pn) \wedge inBlock(v, B, pn))$

Proof 13. *Proof-Sketch that TP1 is an equivalence transformation*

Let P be a correctly augmented process and P' be a process derived from P by applying $TS1b$ once. Based on Definition 8 for every instantiation I of decision variables, for each input variable x of each block r: $o(P^I, a, x)$ exists and is unique and $o^d(P^I, a, x) = o(P^I, a, x)$. We will show that this property is preserved for P' by proving that for every instantiation I and every block r reading any variable x: $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$. Therefore, there also exists a unique origin for $o(P'^I, r, x)$. There are the following cases for the reading block r:

In analogy to the proof of $TS1a$, if step r is located before the par-split it cannot be effected by $TP1$. The par-split itself has no input variables. Therefore, only steps executed after the par-split may be effected by the transformation. Let pn be the participant assigned to r. We get the following cases for r reading x of pn:

1. r is in sub-block A and pn has no step reading x in sub-block B

   Following from $TP1a$, we get $\forall x \in X : (inBlock(x, A, pn) \wedge \neg inBlock(x, B, pn))$
   $\Longrightarrow SEQ(SR_{p1}(X, pn, b), PAR_{ps,pj}(A, B)) \equiv$
   $PAR_{ps,pj}(SEQ(SR_{p1}(X, pn, b), A), B)$
   Since $SEQ(SR_{p1}(X, pn, b)$ is certainly executed before r in A and the par-split has no output variable we get:
   $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.

2. r is in sub-block B and pn has no step reading x in sub-block AS and the par-split has no output variable
   Following from $TP1b$, we get $\forall x \in X : (inBlock(x, B, pn) \wedge \neg inBlock(x, A, pn))$
   $\Longrightarrow SEQ(SR_{p1}(X, pn, b), PAR_{ps,pj}(A, B)) \equiv$
   $PAR_{ps,pj}(A, SEQ(SR_{p1}(X, pn, b), B))$
   Since $SEQ(SR_{p1}(X, pn, b)$ is certainly executed before r in B we get:
   $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.

3. r is in sub-block A and pn also reads x in sub-block B with activity r' and the par-split has no output variable.
   Following from $TP1c$, we get $\forall x \in X : (inBlock(x, A, pn) \wedge inBlock(x, B, pn))$
   $Longrightarrow SEQ(SR_{p1}(X, pn, b), PAR_{ps,pj}(A, B)) \equiv$
   $PAR_{ps,pj}(SEQ(SR_{p1}(X, pn,\ b), A),\ SEQ(SR_{p1}(X, pn, b), B))$. There is no predefined ordering between r and r'. Based on Definition 6 r and r' cannot write to x. They may only read x. Therefore, in order to preserve the correctness, it is sufficient to guarantee that $SR_{p1}(X, pn, b), B)$ is certainly executed before r and before r'. Since the ordering within one branch is preserved for every possible instantiation we get:
   $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.
   $o^d(P^I, r', x) = o(P^I, r, x) = o^d(P'^I, r', x) = o(P'^I, r, x)$.

4. r is not in A and not in B but after the par-block. In this case $TP1a$ and $TP1b$ may apply. Both transformations enforce, that $SR_{p1}(X, pn, b), B)$ is certainly executed before r because the par-join node synchronizes sub-block A and sub-block B. Therefore, we get: $o^d(P^I, r, x) = o(P^I, r, x) = o^d(P'^I, r, x) = o(P'^I, r, x)$.

□

TP2: Passing PAR-Join: A send-receive step located in some branch $B1$ of a par-split directly before a par-join is equivalent to an identical send-receive directly after a par-join, if all variables transmitted by the send-receive step are read or updated in branch $B1$ and none is read or updated in the other branch, or if none of the variables is read or updated in any branch. In particular there are the cases: A reader or writer only in branch A, only in Branch B or nowhere:

TP2a: $PAR_{ps,pj}(SEQ(A, SR_{p1}(X, pn, b)), B) \equiv$
$SEQ(PAR_{ps,pj}(A, B), A, SR_{p1}(X, pn, b))$, if $\forall v \in X : (inBlock(v, A, pn) \wedge \neg inBlock(v, B, pn)) \vee (\neg inBlock(v, A, pn) \wedge \neg inBlock(v, B, pn))$

TP2b:$PAR_{ps,pj}(SEQ(SR_{p1}(X, pn, b), A), B) \equiv$
$SEQ(PAR_{ps,pj}(A, B), A, SR_{p1}(X, pn, b))$, if $\forall v \in X : (inBlock(v, B, pn) \wedge \neg inBlock(v, A, pn)) \vee (\neg inBlock(v, A, pn) \wedge \neg inBlock(v, B, pn))$

Proof 14. *Proof-Sketch that TP2 is an equivalence transformation*

The proof of $TP1$ follows from the Definition of a correct initial process 6, which forbids data-flow between parallel branches and data-flow between parallel branches and non parallel blocks thereafter. Therefore, only the following cases are relevant: 1) Data-Flow originating from subblock A of the par-block.
2) Data-Flow originating from subblock B of the par-block.
3) Data-Flow neither A nor B.

$TP2a$ handles Case 1 and Case 3, $TP2b$ handles Case 2 and Case 3. Since the par-join does not have input variables, only steps after the par-join can be effected. All Cases are handled correctly by $TP2$. □

3.4 Completeness

Theorem 1 (Completeness of the Equivalence Transformations)**.** Every correct augmentation can be created by the application of the transformations starting from any correct augmentation.

Proof 15. *Proof-Sketch*

We first define a normal form of an augmentation. An augmentation P is in normal form, when after each writer $W_p(R, W)$ a sequence of communication

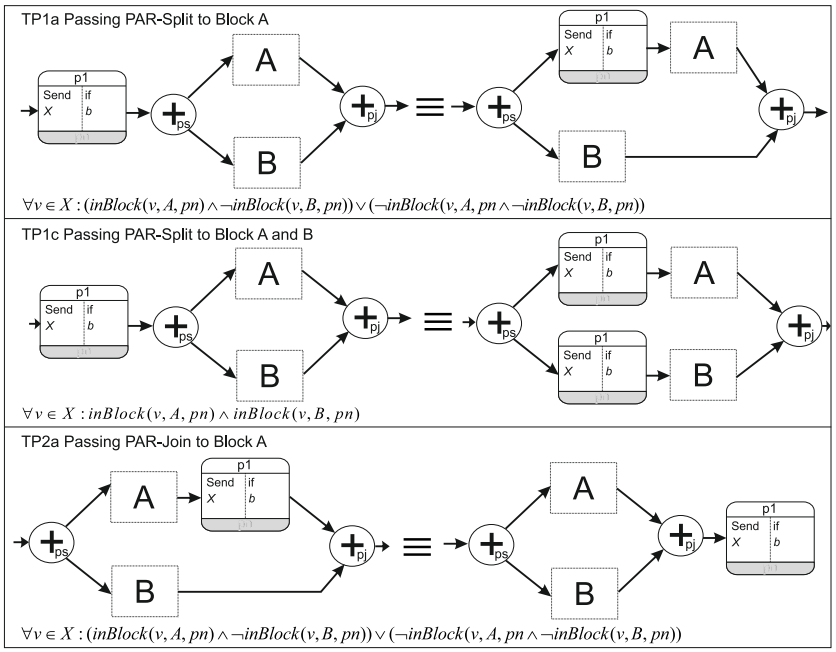

Fig. 5. Example Equivalence Transformations on PAR-Split

steps exists, which sends each $x \in W$ from p separately to all other participants and the condition of the send-receive steps is the inherited condition of the superordinate blocks.

We will now show that any correct augmentation can be transformed to this normal-form:

1. Apply $TS5$ from right to left until every communication steps sends a single variable.
2. Apply $TS6$ from right to left, until no condition contains a disjunction.
3. Apply $TS1a$ / $TS1b$ / $TX1$ / $TX2$ / $TX3$ / $TP1$ / $TP2$ / $TX4$ (from right to left) until every send-receive step is located directly before its origin (step writing to this variable).

After no further movement of any communication step is possible (ignoring changes in the order within a sequence of send-receive steps), apply $TS2$ in combination with $TS1$ to change the sender to the origin (remove transitive sends). Finally apply $TS3$ and $TS4$ to generate missing (redundant) send-receive step. Restart the process to move newly introduced steps to the position directly after the writer. Apply $TX4$ to make all inherited conditions explicit.
Finally the normal-form is reached.

The proof follows from the fact that every transformation has an inverse. □

4 Applications

The presented equivalence transformations provide a formal grounding for various applications dealing with data-flow of interorganizational processes. We will present three applications scenarios as examples here.

4.1 Optimizing Message Exchanges under Constraints:

Given an interorganizational process without communication steps we can generate the complete solution space of correct implementations of the data-flow. This allows to select the solution that best fits the needs of the participants based on objective functions and constraints. The best solutions heavily depends on the user requirements. For example a major goal could be to minimize the number of message exchanges and favour message exchanges that can be integrated into the control-flow, while accepting some potentially redundant transfers. In addition acceptable solutions can be required to meet certain constraints. For example a participant may not be allowed to receive the value of a certain variable after it was written by some other participant or a participant should not receive messages from certain other participants at all.
We have already implemented a prototype that uses best first search to find optimal solutions based on the transformations. It allows the user to define his objective function based on various parameters. Those parameters include the weighted number of send-receive steps, the weighted number of transmitted variables and the number of transfers from unknown participants in the solution. We weight send-receive steps based on their nesting level within xor-blocks and their condition. For the weighted number of send-receive steps we count only additional steps. Therefore, we do not count communication steps, which can be integrated into the control-flow. When a solution is derived, local processes can be generated for each participant by simple projection of the steps onto each participant [13]. These local processes act as interfaces for the private processes of the participants. We have conducted initial experiments with our implementation and the generated solutions are promising. Future work will address starting with a more efficient initial augmentation and the application of sophisticated heuristics and a flexible framework for modeling various constraints.

4.2 Integrating Participants with Existing Processes:

The previous scenario assumed a top-down development paradigm. However, in many cases participants already have existing processes that may not be changeable leading to a mixed approach. When participants with existing processes want to join an interorganizational process only solutions that match their (data) interfaces are applicable. Therefore, the rules can be applied both to test whether their (data) interfaces are compatible with the interorganizational process and to select the best solution based on an objective function.
This can be realized in analogy to the previous application scenario. The only difference is that we start with an interorganizational process with fixed (data)

interfaces of one or more participants. In a next step an initial augmentation is created. Then solutions can be generated. However, only those are acceptable, where the participants with existing interfaces have only send-receive steps that are equivalent to their existing interfaces. In other words solutions are generated, where the non fixed participants act as mediators for the fixed ones. An example is the following: One participant needs to receive the variables a, b, c via one single message from participant e. However, a, b and c are all last updated by different other participants. Then the equivalence transformations can be used to generate solutions where participant e collects the results of the other participants and then sends the variables with one single message.

4.3 Validation of Guidelines and Methods:

Using the equivalence definition we can also validate guidelines for designing the dataflow or procedures generating the dataflow by analyzing whether the resultuing augmented processes can be transformed to a process known to be correct (e.g. the initial processes described in Section 4.1 above).

5 Related Work

In this paper we have proposed a theory of equivalence transformations on the realization of data-flow for inter-organizational processes. As discussed in the application section the proposed rules can be used to derive local processes based on a global process. Well known approaches in this direction are the public to private approach and multi-party contracts [26, 23, 24]. These approaches address the projection of the control-flow onto different participants and define, when the control-flow of a private process correctly implements the participant's projection of the global process. Our approach complements these approaches by providing the solution space for the correct implementation of the implicit data-flow using message exchanges.
This differs from typical choreography approaches [2] either in form of interconnection modeling or in form of interaction-centric modeling which are both supported by BPMN (2) [18]. Interconnection modeling wires multiple (collaboration) models together using message exchanges. Interaction-centric modeling is supported by choreography diagrams. An advantage of the interaction-centric style is that a global view of the choreography exists, preventing typical flows of not properly aligned models. However, they still require that the message exchanges are modeled explicitly. We follow a third route. We begin with a global process which describes the goals of the choreography in terms of control- and data-flow requirements. In contrast to the previously discussed approaches, message exchanges are not part of the global process. Instead our approach allows to automatically generate an optimize the required message exchanges (choreography) between the participants. We follow a model based correctness by design approach for control- and data-flow based on process views [3]. We have addressed the data-flow perspective in this paper, while the control-flow

perspective including the proof of the correct projection and the generation of realizable local processes for any global process and any assignment of participants was addressed in our previous work [13, 12].
A recent approach addressing data in choreographies is [14]. It proposes modeling guidelines that allow to derive message contents of a given choreography automatically. It is based on a global data model which is mapped to the local ones of each participant. Since our rules allow to automatically generate optimized message exchanges (choreograhies) our output can be used as an input for [14] in order to resolve heterogeneities between the data representations of the participants. The work in [15] addresses the realization of data-flow between different participants. They propose a number of design patterns for the implementation of data dependencies and are therefore inline with our approach. However, instead of proposing common patterns we can generate the solution space of possible implementations and automatically select the best fitting one according to the users requirements.
There are numerous approaches that deal with the automatic partitioning of BPEL processes. One major aim is to find assignments of participants that result in optimal data-flow [16, 4, 28, 7]. This setting is very different from our goal, where the assignment of participants is fixed. Directly related to our approach are role based partitioning methods for executable processes such as [9–11, 6, 5]. These approaches also allow to derive processes for each participant. However, the implementation of the data-flow is based on a fixed strategy and they consequently provide only one solution. An approach focusing on privacy aspects is presented in [27]. It allows to define which participants may exchange messages and to automatically find alternative paths if certain exchanges are forbidden. In contrast, we have provided a general approach for optimizing the implementation of data-transfer between the participants based on various criteria, where privacy issues are only one possible constraint.
The correctness of the (implicit) data flow of *intra*organizational processes is addressed in works such as [20, 21]. In contrast, our approach spans the solution space for the correct realization of *inter*organizational data flow via message exchanges, taking a (correct) global process with implicit data flow as input.

6 Conclusion

There exists various implementations for the data-flow of an interorganizational process. In this paper we have provided a theory of equivalence transformations that can act as a solid foundation for numerous applications such as: Top-down development of interorganizational processes including the automatic optimization of the data-flow between different participants and the enforcement of various constraints (eg. security / access rights). Finally it allows to systematically test the compatibility of an existing process with some interorganizational process not only regarding the control-flow but also regarding the (optimized) data-flow. Our future work includes the development of equivalence transfor-

mations to additional control-flow patterns and to apply the rules for various applications.

References

1. Carlo Combi and Mauro Gambini. Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In *OTM 2009*, volume 5870 of *LNCS*, pages 42–59. Springer, 2009.
2. Gero Decker and Mathias Weske. Interaction-centric modeling of process choreographies. *INFORM SYST*, 36(2):292 – 312, 2011. Special Issue: Semantic Integration of Data, Multimedia, and Services.
3. Johann Eder, Nico Kerschbaumer, Julius Köpke, Horst Pichler, and Amirreza Tahamtan. View-based interorganizational workflows. In *CompSysTech'11*, pages 1–10. ACM, 2011.
4. Walid Fdhila, Marlon Dumas, and Claude Godart. Optimized decentralization of composite web services. In *CollaborateCom 2010*, pages 1–10. IEEE, 2010.
5. Walid Fdhila and C. Godart. Toward synchronization between decentralized orchestrations of composite web services. In *CollaborateCom 2009*, pages 1–10, 2009.
6. Walid Fdhila, U. Yildiz, and C. Godart. A flexible approach for automatic process decentralization using dependency tables. In *ICWS 2009*, pages 847–855, 2009.
7. Elio Goettelmann, Walid Fdhila, and Claude Godart. Partitioning and cloud deployment of composite web services under security constraints. In *IC2E '13*, pages 193–200, Washington, DC, USA, 2013. IEEE.
8. David Hollingsworth. The workflow reference model. 1995.
9. R. Khalaf and F. Leymann. E role-based decomposition of business processes using bpel. In *ICWS '06*, pages 770–780, 2006.
10. Rania Khalaf, Oliver Kopp, and Frank Leymann. Maintaining data dependencies across bpel process fragments. In *Service-Oriented Computing ICSOC 2007*, volume 4749 of *LNCS*, pages 207–219. 2007.
11. Rania Khalaf and Frank Leymann. Coordination for fragmented loops and scopes in a distributed business process. *INFORM SYST*, 37(6):593 – 610, 2012.
12. J. Köpke, J. Eder, and M. Künstner. Projections of abstract interorganizational business processes. In *DEXA 14*, volume 8645 of *LNCS*, pages 472–479. 2014.
13. J. Köpke, J. Eder, and M. Künstner. Top-down design of collaborating processes. In *iiWAS 14*, ACM, pages 336–345, Hanoi, Viet Nam, 2014. ACM.
14. Andreas Meyer et al. Automating data exchange in process choreographies. In *CAiSE 14*, volume 8484 of *LNCS*, pages 316–331. 2014.
15. Geert Monsieur, Monique Snoeck, and Wilfried Lemahieu. Managing data dependencies in service compositions. *J. Syst. Software*, 85(11):2604 – 2628, 2012.
16. Mangala Gowri Nanda, Satish Chandra, and Vivek Sarkar. Decentralizing execution of composite web services. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '04, pages 170–187, New York, NY, USA, 2004. ACM.
17. OASIS. OASIS Web Services Business Process Execution Language (WSBPEL) TC. Technical report, "OASIS", 2007.
18. Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Technical report, 2011.
19. Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In *ER '05*, volume 3716 of *LNCS*, pages 353–368, 2005.

20. Sherry X. Sun, J. Leon Zhao, Jay F. Nunamaker, and Olivia R. Liu Sheng. Formulating the data-flow perspective for business process management. *Information Systems Research*, 17(4):pp. 374–391, 2006.
21. Nikola Trka, WilM.P. van der Aalst, and Natalia Sidorova. Data-flow anti-patterns: Discovering data-flow errors in workflows. In Pascal van Eck, Jaap Gordijn, and Roel Wieringa, editors, *CAiSE '09*, volume 5565 of *LNCS*, pages 425–439. 2009.
22. Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.
23. Wil M. P. van der Aalst. Inheritance of interorganizational workflows: How to agree to disagree without loosing control? *IT and Management*, 4(4):345–389, 2003.
24. Wil M. P. van der Aalst, Niels Lohmann, Peter Massuthe, Christian Stahl, and Karsten Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.
25. Wil M. P. van der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.
26. Wil M. P. van der Aalst and Mathias Weske. The p2p approach to interorganizational workflows. In *CAiSE '01*, pages 140–156. Springer, 2001.
27. U. Yildiz and Claude Godart. Information flow control with decentralized service compositions. In *ICWS 2007*, pages 9–17, 2007.
28. Yanlong Zhai, Hongyi Su, and Shouyi Zhan. A data flow optimization based approach for bpel processes partition. In *ICEBE 2007*, pages 410–413, 2007.