Equivalence Transformations for the Design of Interorganizational Data-Flow

Julius Köpke and Johann Eder(✉)

Department of Informatics-Systems, Alpen-Adria Universität, 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, a key technology to facilitate interorganizational cooperation and e-business, face the challenge to retain the advantages of *intra*organizational business process management for the design of enterprise information systems - which extensively takes advantage of a central coordination - while expanding the technology to fully distributed collaborations of autonomous entities. One of the major differences between centralized and distributed process management is the access to data: uniform access to a joint central data store versus distributed management of data with explicit exchange of data via messages.

We focus on a phase in the development of an interorganizational workflow where the explicit dataflow between participants is established. Starting point of our considerations is a 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 fully distributed way, respectively projected onto the participants to define the interface of their internal process (e.g. by process views [3,4]). An initial process definition consists of a set of activities, the control flow between them, assignment of the activities to participants, and input and output parameters of activities. Many approaches such as [7,10,14,15,26,28] 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

J. Zdravkovic et al. (Eds.): CAiSE 2015, LNCS 9097, pp. 367–381, 2015.
DOI: 10.1007/978-3-319-19069-3_23

explicit realization of the data flow, however, there are numerous possibilities [17,21]. 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 [25,26,28] do not consider data flow at all, [14,15] restrict the data flow to the distribution of decision variables. [7,10] address data flow, however, only a single solution based on one fixed strategy is generated.

Take for example the following trivial process chunk: 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* [21]: 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 [7] 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: 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 set of protocols between two participants. For defining the process we use block-structured workflow nets [9] supporting the usual basic control flow patterns sequence, par split / join, and xor split/join [24,27]. 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 [19] 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 its input and its output. Furthermore, we assign to each activity and each control step one of the participants as actor.

In our notation, $A_a(R, W)$ is an activity step where A is the label of the task to be executed by participant a. R defines the set of input variables, W defines the set of output variables. Abstract blocks are placeholders for any subprocess (including empty ones) and are represented by their label. $SEQ(A, B)$ defines a sequence of the blocks A and B. Sequences 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 show which the participant executing a step as subscript, the set of input and output variables of activity-steps, and the condition of xor-splits. See Fig. 1, 2, 3, 4 for examples. A communication step (also called send-receive step) is denoted by $SR_s(X, pn, b)$. It defines that participant s sends the content of the set of variables X to participant pn, 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 Fig. 2 as an example for $SR_{p1}(X, pn, b)$. 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.

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 are several 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 Fig. 1 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 [10].)

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

We follow the second approach and require that for any $XOR_{ps,pj}(b, A, B)$ the condition b refers only to one single boolean variable called decision variable,

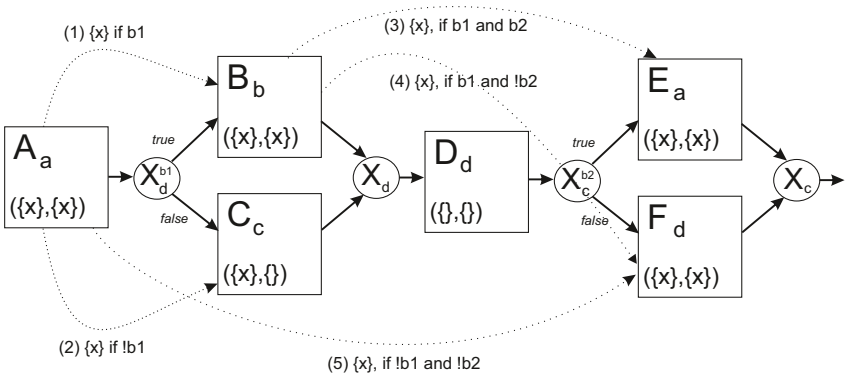

Fig. 1. Implicit data flow in an interorganizational process

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 only be written once.

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 of 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 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.

Fig. 1 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. 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 overwritten 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 whether 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 (see Sect. 2.3) where the condition is a boolean expression consisting of decision variables. 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

A process model consists of sets of participants 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, c a decision variable, b a boolean expression consisting of the decision variables $d_1, \ldots d_n$ then $S_{p1}(R, W)$ is a block (activity step), $SR_s(X, r, b)$ is a block (communication step), the empty block is a block, and if A and B are blocks, then $SEQ(A, \ldots, B)$, $XOR_{p1,p2}(c, A, B)$, $PAR_{p1,p2}(A, B)$ are blocks. 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.

An *initial process* does not contain communication steps. An *augmentation* of an initial process P contains all the steps of P in the same topological order and some communication steps in addition. Each 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 of P where each xor retains exactly one sub-block (depending on the value of the decision variable) and only those communication steps where the condition is *true* while the other sub-block is empty.

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.

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 $o(P^I, a, x) = b$.

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 [23]. 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 steps.

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 1. 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 Transformations 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. We present a set of transformations on augmented processes that allow to derive all other correct augmentations.

3.1 Equivalence Transformations on Sequences

We provide a graphical description of equivalence transformations on sequences in Fig. 2 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 c can be swapped with an activity a, unless a is the destination of c or a writes to some variable transmitted or referenced by c: $SEQ(SR_{p1}(X, pn, b), a_{p2}(R, W)) \equiv SEQ(a_{p2}(R, W), SR_{p1}(X, pn, b))$, unless $(W \cap X \neq \{\}) \vee (R \cap X \neq \{\} \wedge pn = p2) \vee ref(b) \subseteq W)$

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: $(pn = p2 \vee pm = p1) \wedge (X \cap Y \neq \{\} \vee ref(b1) \subseteq Y \vee ref(b2) \subseteq X)$.

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

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.

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: $SEQ(SR_{p1}(X, pn, b), a_{p2}(R, W)) \equiv a_{p2}(R, W)$ where $X \subseteq W$, unless $R \cap X \neq \{\} \wedge pn = p2$.

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: $SEQ(SR_{p1}(X, pn, b), SR_{p1}(Y, pn, b)) \equiv SR_{p1}(X \cup Y, pn, b)$

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: $SEQ(SR_{p1}(X, pn, b_1), SR_{p1}(X, pn, b_2)) \equiv SR_{p1}(X, pn, \{b1 \vee b2\})$

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 the execution of the block *pos*. *takesPart(xorBlock,participant)* returns *true*, if the participant *participant* participates in any step of the xor-block *xorBlock* (recursively). Fig. 3 shows all equivalence transformations on 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 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 $hasValue(p1, xb, XS_{p2,pj} \wedge hasValue(pn, xb, XS_{p2,pj})$.

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. $XOR_{p2,pj}(xb, SEQ(A, SR_{p1}(X, pn, b)), B) \equiv SEQ(XOR_{p2,pj}(xb, A, B), SR_{p1}(X, pn, \{b \wedge xb\})$

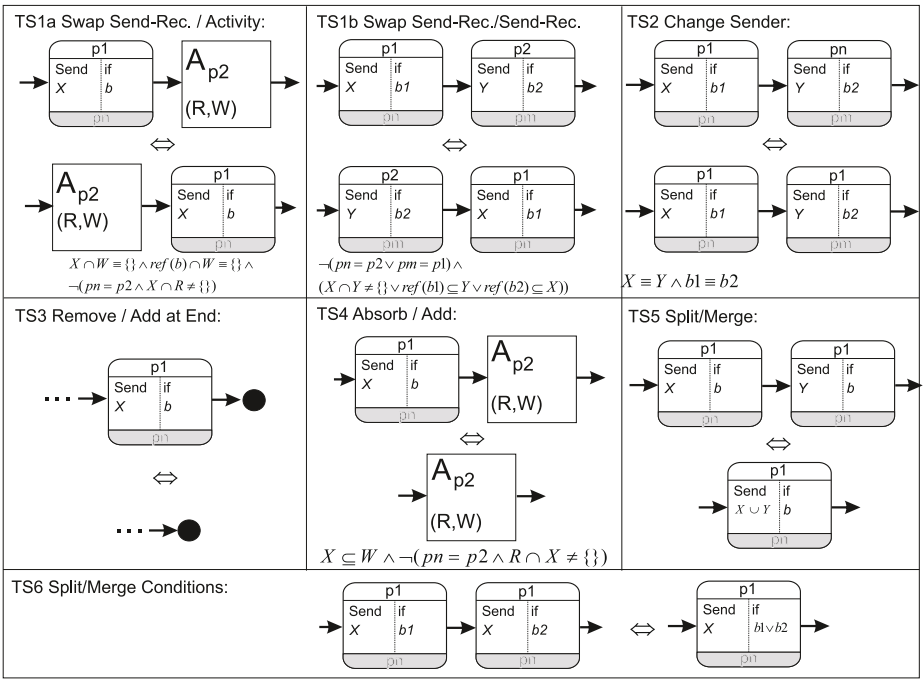

Fig. 2. Equivalence transformations on sequences

TX2b Passing Join on false: $XOR_{p2,pj}(xb, A, SEQ(B, SR_{p1}(X, pn, b))) \equiv SEQ(XOR_{p2,pj}(xb, A, B), SR_{p1}(X, pn, \{b \wedge \neg xb\})$

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 $\neg takesPart(XS_{p2}, pn) \wedge X \equiv \{xb\}$

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 in Block A of x and $b \equiv b \wedge \neg bx$ if s is in Block B of x.

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. $XOR_{p2,pj}(xb, SEQ(SR_{p1}(Xpn, b), A), B) \vee XOR_{p2,pj}(xb, A, SEQ(SR_{p1}(Xpn, b), B)$
$\Longrightarrow XOR_{p2,pj}(xb, SEQ(SR_{p1}(X, pn, b), A), SEQ(SR_{p1}(X, pn, b), B))$.

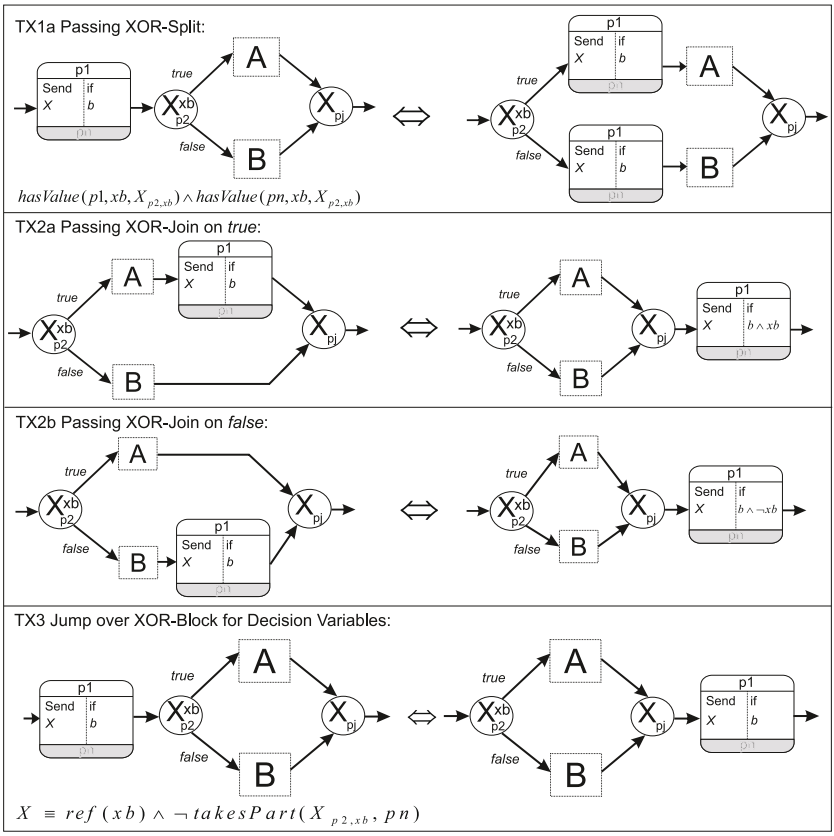

Fig. 3. Equivalence transformations for XOR

3.3 Equivalence Transformations on Parallel Blocks

Equivalence transformations on 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 *inB*:
$hasWriter(var, Block)$ 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). *inB(var, Block, participant)* is *true*, if $hasWriter(var, Block)$ or *hasReader* $(var, block, participant)$. Fig. 4 illustrates $TP1a$, $TP1c$, $TP2a$ as examples.

TP1: Passing PAR-Split: A send-receive-step, which is located directly before a par-split is equivalent to a send-receive 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 $\forall v \in X : (inB(v, A, pn) \wedge \neg inB(v, B, pn)) \vee (\neg inB(v\ , A, pn) \wedge \neg inB(v, B, pn))$

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 $\forall v \in X : (inB(v, B, pn) \wedge \neg inB(v, A, pn)) \vee (\neg inB(v, A\ , pn) \wedge \neg inB(v, B, pn))$

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 $\forall v \in X : (inB(v, A, pn) \wedge inB(v, B, pn))$

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 $\forall v \in X : (inB(v, A, pn) \wedge \neg inB(v, B, pn)) \vee (\neg inB(v, A, pn) \wedge \neg inB(v, B, pn))$

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 $\forall v \in X : (inB(v, B, pn) \wedge \neg inB(v, A, pn)) \vee (\neg inB(v, A, pn) \wedge \neg inB(v, B, pn))$

3.4 Correctness and Completeness

Theorem 1 (Correctness of the Equivalence Transformations)**.** Any application of any of the transformations on a correct augmentation of a process P (see Definition 1) P leads to another correct augmentation of the process P.

We prove the correctness of each transformation by showing that the transformation does not change origin and d-origin of all input variables of each block (details in [13]).

Theorem 2 (Completeness of the set of equivalence transformations)**.** Every correct augmentation of a process P can be created by the application of the transformations starting from any correct augmentation of the process P.

We define a normal form for augmentations of a process and show that, if an augmentation cannot be transformed to this normal form it is incorrect. The theorem then follows from the fact that each transformation has an inverse (details in [13]).

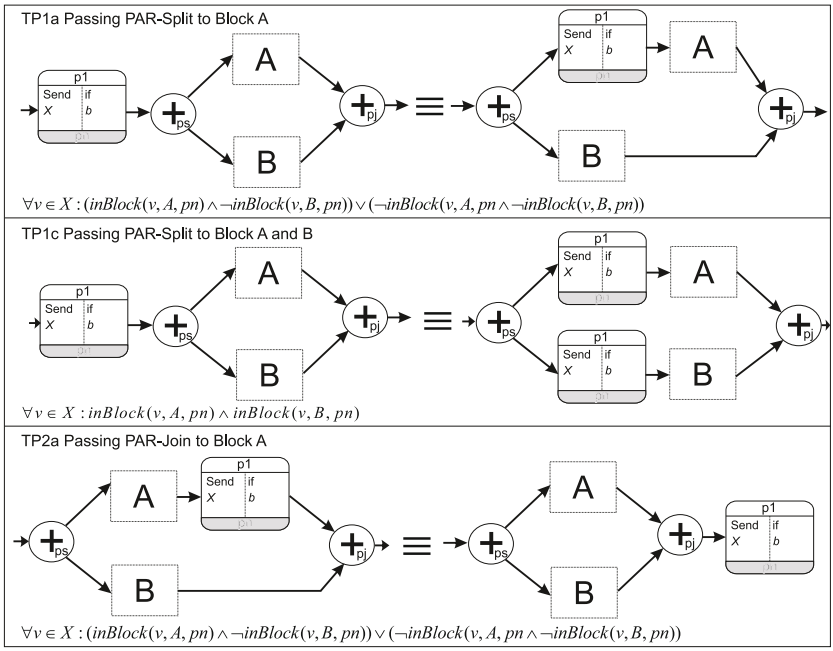

Fig. 4. Example equivalence transformations on PAR-split

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 user requirements. For example a major goal could be to achieve simple processes with minimal number of send-receive steps and favor message exchanges that can be integrated into the control-flow, while accepting some potentially redundant transfers. In another scenario minimizing the transferred data at runtime could be the major goal, when communications costs are high. For any of the previous examples additional constraints may exist. For example a participant may not be allowed to receive the value of a certain variable at all or after some specific step or a participant should not receive messages from a predefined set of other participants.

We have implemented a proof-of-concept prototype that uses best first search to find optimal solutions based on the transformations starting from an initial augmentation where all values are broadcasted to all participants after each modification. It allows the user to define an objective function based on various parameters including the weighted number of send-receive steps, the weighted number of transmitted variables and the number of transfers from unknown participants. The weight of send-receive steps is based on their nesting level within xor-blocks and their conditions. For the weighted number of send-receive 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 [15]. 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 resulting 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

We propose a set of equivalence transformations on the realization of the data flow of inter-organizational processes to derive (interfaces to) local processes from a global process definition. Approaches like the public to private approach and multi-party contracts [25,26,28] - address the projection of the control-flow onto different participants and the correct implementation of control-flow in the private process. Our approach complements these approaches for the correct implementation of the data flow specification using message exchanges.

Typical choreography approaches [2] either in form of interconnection modeling or in form of interaction-centric modeling are both supported by BPMN (2) [20]. 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 flaws of not properly aligned models. However, they still require that the message exchanges are modeled explicitly. We follow a different strategy. We begin with a global process describing the goals of the choreography in terms of control-and data flow requirements - message exchanges are not part of the global process. Instead our approach allows to automatically generate and optimize the required message exchanges (choreography) between the participants. We address the data flow perspective here, while the correct projection of the control flow is described in [14,15].

A recent approach addressing data in choreographies is [16]. 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 (choreographies) our output can be used as an input for [16] in order to resolve heterogeneities between the data representations of the participants. [17] proposes a set of design patterns for the implementation of data flows satisfying data dependencies. Instead of proposing a fixed set of common patterns we allow to automatically select the best solution according to the users requirements.

Numerous approaches deal with the automatic partitioning of BPEL processes with the aim to find assignments of participants that result in optimal data flow [5,8,18,30]. 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 [6,7,10–12]. These approaches also allow to derive processes for each participant. However, the implementation of the data flow is based on a fixed strategy and consequently provide only one solution is provided. An approach focusing on privacy aspects [29] 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-transfers based on various criteria where privacy issues and constraints - among those also privacy constraints.

The correctness of the (implicit) data flow of *intra*organizational processes is addressed in works such as [22,23]. 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

Interorganizational business process management - a promising techniques to foster collaboration and e-business - still requires research and development, in particular in architecture, design and implementation techniques. There exists various implementations for the data flow of an interorganizational process. In this paper we have provided a comprehensive set of equivalence transformations that can act as a solid foundation for several 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 (e.g. security / access rights), or the validation of methods and procedures for designing interorganizational processes with data flow. 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.

References

1. Combi, C., Gambini, M.: Flaws in the flow: the weakness of unstructured business process modeling languages dealing with data. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2009, Part I. LNCS, vol. 5870, pp. 42–59. Springer, Heidelberg (2009)
2. Decker, G., Weske, M.: Interaction-centric modeling of process choreographies. Inform. Syst. **36**(2), 292–312 (2011)
3. Eder, J., Kerschbaumer, N., Köpke, J., Pichler, H., Tahamtan, A.: View-based interorganizational workflows. CompSysTech 2011, pp 1–10. ACM (2011)
4. Eder, J., Tahamtan, A.: Temporal consistency of view based interorganizational workflows. In: Kaschek, R., Kop, C., Steinberger, C., Fliedl, G. (eds.) Information Systems and E-business Technologies. LNBIP, vol. 5, pp. 96–107. Springer, Heidelberg (2008)
5. Fdhila, W., Dumas, M., Godart, C.: Optimized decentralization of composite web services. In: CollaborateCom 2010, pp. 1–10. IEEE (2010)
6. Fdhila, W., Godart, C.: Toward synchronization between decentralized orchestrations of composite web services. In: CollaborateCom 2009, pp. 1–10. IEEE (2009)
7. Fdhila, W., Yildiz, U., Godart, C.: A flexible approach for automatic process decentralization using dependency tables. In: ICWS 2009, pp. 847–855 (2009)
8. Goettelmann, E., Fdhila, W., Godart, C.: Partitioning and cloud deployment of composite web services under security constraints. In: IC2E 2013, pp. 193–200. IEEE (2013)
9. Hollingsworth, D.: The workflow reference model (1995)
10. Khalaf, R., Leymann, F.: Role-based decomposition of business processes using bpel. In: ICWS 2006, pp. 770–780 (2006)
11. Khalaf, R., Kopp, O., Leymann, F.: Maintaining data dependencies across BPEL process fragments. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 207–219. Springer, Heidelberg (2007)

12. Khalaf, R., Leymann, F.: Coordination for fragmented loops and scopes in a distributed business process. Inform. Syst. **37**(6), 593–610 (2012)
13. Köpke, J., Eder, J.: Equivalence Transformations on Interorganizational Processes to Shift Communication Steps Technical report, AAU Klagenfurt (2014). http://isys.uni-klu.ac.at/PDF/2014-EQTrans.pdf
14. Köpke, J., Eder, J., Künstner, M.: Projections of abstract interorganizational business processes. In: Decker, H., Lhotská, L., Link, S., Spies, M., Wagner, R.R. (eds.) DEXA 2014, Part II. LNCS, vol. 8645, pp. 472–479. Springer, Heidelberg (2014)
15. Köpke, J., Eder, J., Künstner, M.: Top-down design of collaborating processes. In: iiWAS 2014. ACM (2014)
16. Meyer, A., Pufahl, L., Batoulis, K., Kruse, S., Lindhauer, T., Stoff, T., Fahland, D., Weske, M.: Automating data exchange in process choreographies. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 316–331. Springer, Heidelberg (2014)
17. Monsieur, G., Snoeck, M., Lemahieu, W.: Managing data dependencies in service compositions. J. Syst. Software **85**(11), 2604–2628 (2012)
18. Nanda, M.G., Chandra, S., Sarkar, V.: Decentralizing execution of composite web services. In: Proc. 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, pp. 170–187. ACM press (2004)
19. OASIS. OASIS Web Services Business Process Execution Language (WSBPEL) TC. Technical report, "OASIS" (2007)
20. Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Technical report (2011)
21. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow data patterns: identification, representation and tool support. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 353–368. Springer, Heidelberg (2005)
22. Sun, S.X., Zhao, J.L., Nunamaker, J.F., Sheng, O.R.L.: Formulating the data-flow perspective for business process management. Information Systems Research **17**(4), 374–391 (2006)
23. Trčka, N., van der Aalst, W.M.P., Sidorova, N.: Data-flow anti-patterns: discovering data-flow errors in workflows. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 425–439. Springer, Heidelberg (2009)
24. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, Pierre, Balbo, Gianfranco (eds.) ICATPN 1997. LNCS, vol. 1248. Springer, Heidelberg (1997)
25. van der Aalst, W.M.P.: Inheritance of interorganizational workflows: How to agree to disagree without loosing control? IT and Management **4**(4), 345–389 (2003)
26. van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty contracts: Agreeing and implementing interorganizational processes. Comput. J. **53**(1), 90–106 (2010)
27. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)
28. van der Aalst, W.M.P., Weske, M.: The P2P approach to interorganizational workflows. In: Dittrich, K.R., Geppert, A., Norrie, M. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 140–156. Springer, Heidelberg (2001)
29. Yildiz, U., Godart, C.: Information flow control with decentralized service compositions. ICWS **2007**, 9–17 (2007)
30. Zhai, Y., Hongyi, S., Zhan, S.: A data flow optimization based approach for BPEL processes partition. In: ICEBE 2007, pp. 410–413 (2007)