Top-Down Design of Collaborating Processes

Julius Köpke, Johann Eder, and Markus Künstner
Department of Informatics-Systems, Alpen-Adria Universität Klagenfurt
Klagenfurt, Austria
{julius.koepke, johann.eder, markus.kuenstner}@aau.at

ABSTRACT

Interorganizational business processes aim to integrate local processes to support the seamless cooperation of organizations. Process view approaches are an adequate method balancing the communication requirements for enabling collaboration and the required privacy hiding internals of private business processes. We focus on a top-down scenario for the development of interorganizational processes, where first an abstract global process is designed. Then each step of the global process is assigned to one of the partners and a local process is generated for each partner as a view on the global process. Finally, the partners implement or adopt their processes based on their views. We present an algorithm for the fully automatic generation of views for any block-structured input process with arbitrary partner assignments, provide a method for merging the partner's views to reconstruct the global process and prove the correctness of the view generation method.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Workflow management*

Keywords

Process View Generation, Process Partitioning, Interorganizational Business Processes

1. INTRODUCTION

Technical support for the interorganizational cooperation between different organizations requires the integration of business processes. Web technologies [11, 28, 2] and SOA-based protocols [1] are a good basis for the implementation of such interorganizational processes which can be realized in form of choreographies or orchestrations [23]. We specifically address choreographies that allow a fully distributed operation without the need for a central coordinator.

iiWAS '14, December 04 - 06 2014, Hanoi, Viet Nam

ACM 978-1-4503-3001-5/14/12 ...$15.00.
http://dx.doi.org/10.1145/2684200.2684282.

In this paper we focus on the top down part of forming cooperating processes: transforming some given global process to a set of collaborating distributed processes that realize the interorganizational process which currently is still a tedious and error-prone mostly manual task. We aim in automating this step with a model-based view approach.

Process views [6, 24, 8, 2, 21, 26, 7] allow to represent the externally observable behavior of business processes and to balance the request for privacy and loose coupling between processes with the communication demands for collaboration. Most approaches for process views such as [8, 2, 21, 24, 26] follow a bottom up or - in analogy to the role of views in federated databases [25] - *global as view* approach. A view is derived from a private process definition, which is actually instantiated and executed in a process engine at runtime. The integration of such cooperating views constitutes an interorganizational business process.

Views can also be used the other way round - to distribute the steps of a global process definition. In this top-down or *local as view* approach, first, a global interorganizational process is defined as an abstract process. The activities contained in this process definition are then distributed onto the involved partners. Since the process definition is abstract, it means that none of the steps are executed *globally*, since there is no global or central component. Each step which is defined in the global process is executed by one of the participating processes.

The projection of the global process onto a particular participant defines a view, i.e. a (local) workflow derived from the abstract global process. Such a local process specifies the obligations of a particular partner (execution of steps defined in the global process) and the externally observable behavior of the private processes. So a local process is also a view on the private process which is actually executed by a partner. The local process hides the parts of the private process which are confidential or not relevant for the collaboration.

The p2p approach presented in [32, 29] is an example for such a top-down modeling approach. As the construction of views on a global process by projection requires the introduction of communication steps and the distribution of data, in particular data needed for control flow decisions it is quite different from the construction of views on local processes which only abstract from the process by deletion and aggregation of steps [8, 2, 21, 24, 26].

In this paper, we present an automatic process partitioning algorithm that can be used in a top-down development approach. Starting with a global process each step of the

process is assigned to one of the partners. In a next step the global process with partner assignments is partitioned fully automatically into views for each partner. Finally, the partners can use the generated views to create new or adopt existing processes according to their views.

We have already presented the general idea of the projection approach in a short paper [19]. Here we present the following novel contributions:

- A complete partitioning algorithm (Section 4).
- An algorithm to merge a set of generated views to reconstruct the global process (Section 5.1).
- A formal proof that the partitioning algorithm returns views that correctly specify the distributed execution of the input process (Section 5).

2. PROCESS MODEL

We specify the process definitions we consider in this approach with a meta-model. It is based on the capabilities of block structured workflow nets (called full blocked workflows by the WfmC [12] supporting sequence, PAR split / join, XOR split/join, and LOOP split/join as control flow patterns [31, 27]. We focus on block structured workflow-nets as they prevent typical flaws of unstructured business processes dealing with data [3] and are also in line with the WS-BPEL[22] standard. The meta-model in Figure 1 shows the essential components of this model in a simple way. The representation of algorithms in the following sections is based on this meta-model. Block structured workflow-nets can be represented either as graphs or as hierarchies of (abstract) activities. The meta-model captures both representations - the transformation between these two representations is straightforward. Therefore, we use the appropriate representation in our different algorithms.
A process is defined by activity-declarations, variable declarations and the control flow between activities expressed by steps that refer to activities (activity-steps) or that define the control flow in terms of sequences, XOR-, Parallel-, and Loop- constructs. We also introduce abstract steps, which represent any abstract activity as black boxes.

A step is associated with partners to define which partner is responsible for the execution. For activity-steps, one partner can be assigned. For the control steps PAR, XOR, and $LOOPS$ two potentially different partners ($firstPartner$) and ($lastPartner$) can be assigned that are responsible for the split (decision or parallel invocation) and the join (synchronization) respectively. PAR, XOR are limited to two sub blocks (branches) which does not restrict the expressiveness.

In this meta-model we abstract from data with the exception of decision variables. $XOR-$ and $LOOP-$ blocks are typically associated with a condition which is a Boolean expression based on variables. In interorganizational P2P workflows, these expressions are not viable since they would require that all variables in the condition expressions are synchronized between all the partners effected by the flow decision. Therefore, we restrict the control flow decisions to single Boolean variables. This approach can also be interpreted as one partner executes an expression and communicates the result to the involved partners. This restriction does not reduce the generality of our model since any input process with a complex condition in an XOR split based on several variables can always be transformed to a process where the variables or the result of an evaluation of a subexpression are communicated to one partner who evaluates the condition and writes the results of the Boolean expressions to decision variables. The passing of data (other than decision variables) between partners is out of scope for this paper.

For $XOR-$ and $LOOP-$ blocks the $firstPartner$ is responsible for the decision. Therefore, he must be able to compute the value of the decision variable. Sequences, PAR, and XOR have two sub blocks, Loops have one sub block. For $XOR-$blocks the sub block $stepA$ is executed if the decision variable is $true$, otherwise $stepB$. In case of loops the sub block $stepA$ is executed, if the decision variable is $true$, otherwise the loop is not entered or terminated. The sub block $stepA$ is the first block of a sequence, $stepB$ is the second one.

For all steps the method $getPartners()$ returns a list of all partners that are (recursively) involved in the step. Communication steps ($sendSteps$ and $receiveSteps$) realize the control flow and pass decision variables between partners asynchronously.

3. PROJECTING A GLOBAL PROCESS TO VIEWS FOR EACH PARTNER

As in [29, 32] we propose the following procedure for defining interorganizational workflows: (1) define a global (abstract) workflow, (2) assign each step in the workflow to a partner (3) partition the process. (4) Create private processes based on the generated views. The global process is executed by the distributed execution of the private processes.

While the work in [29, 32, 30] concentrates on the question whether a private process correctly implements a (public) view (is in accordance with) our aim is to provide a partitioning procedure that does not impose any restrictions on the global process and on the partner assignments. Additional design rationales are: (1) the resulting views should be as simple as possible, and (b) unnecessary message exchange should be avoided. E.g. a partner should only know about control structures and receive only messages that are absolutely required. In the following, we discuss our partitioning method by examples for each control structure. Our approach operates in two phases: First, the global process is augmented with communication steps that realize the interorganizational control flow and variable passing. In a next step, views for the partners are created by projection.

Sequence Blocks:.

In Figure 2 an example for the partitioning of a sequence is shown. The global process contains the steps D_A, F_B and G_C in a sequence. D is defined to be executed by A, B executes F and C executes G. In order to support a distributed execution the global process is first augmented with pairs of send- and receive- steps whenever a step is followed by another step, which is assigned to a different partner. In particular in the example send- and receive-steps are inserted between D_A and F_B and between F_B and G_C in the augmented global process. The corresponding views for each partner are shown on the right. They are created by simply projecting only steps that are assigned to the specific target partner.

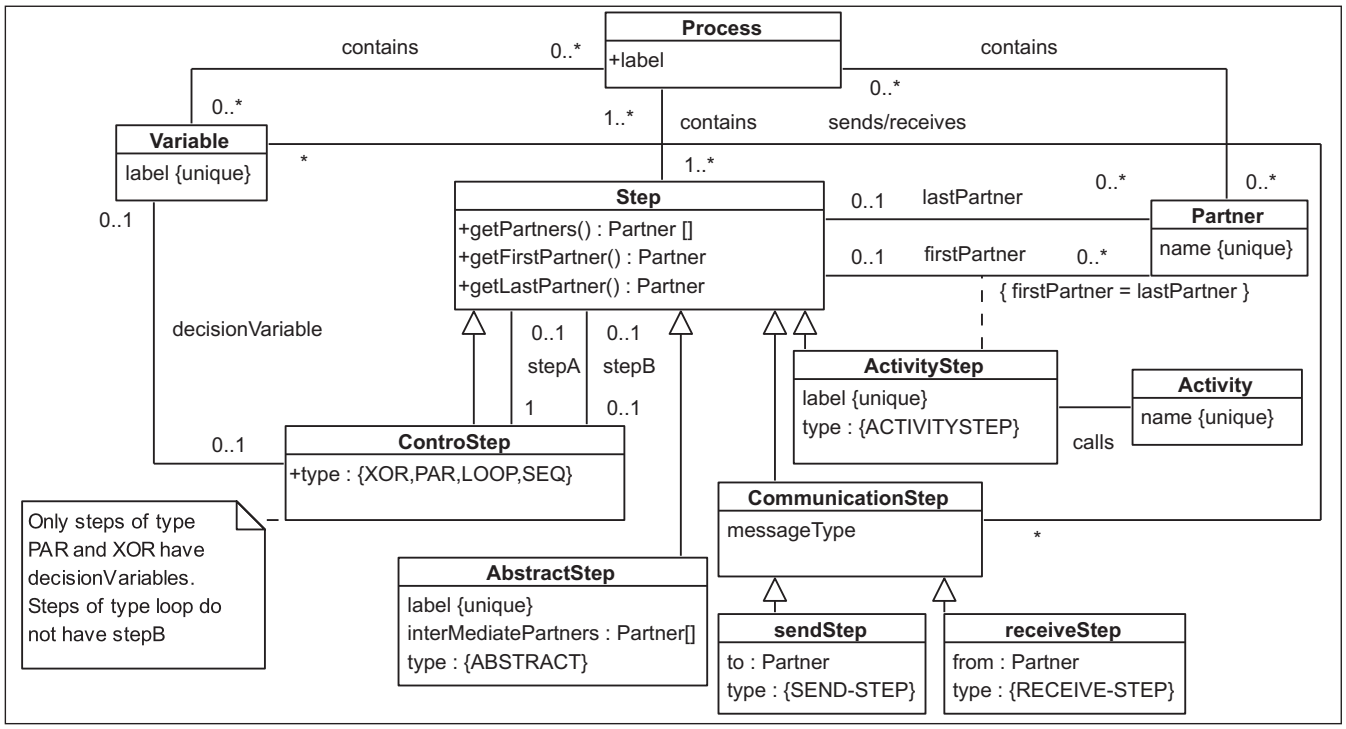

Figure 1: Workflow Meta-Model

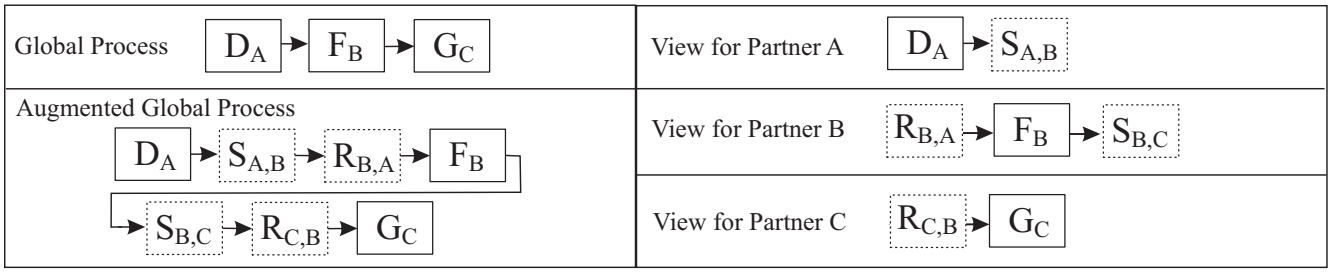

Figure 2: Partitioning of Sequences

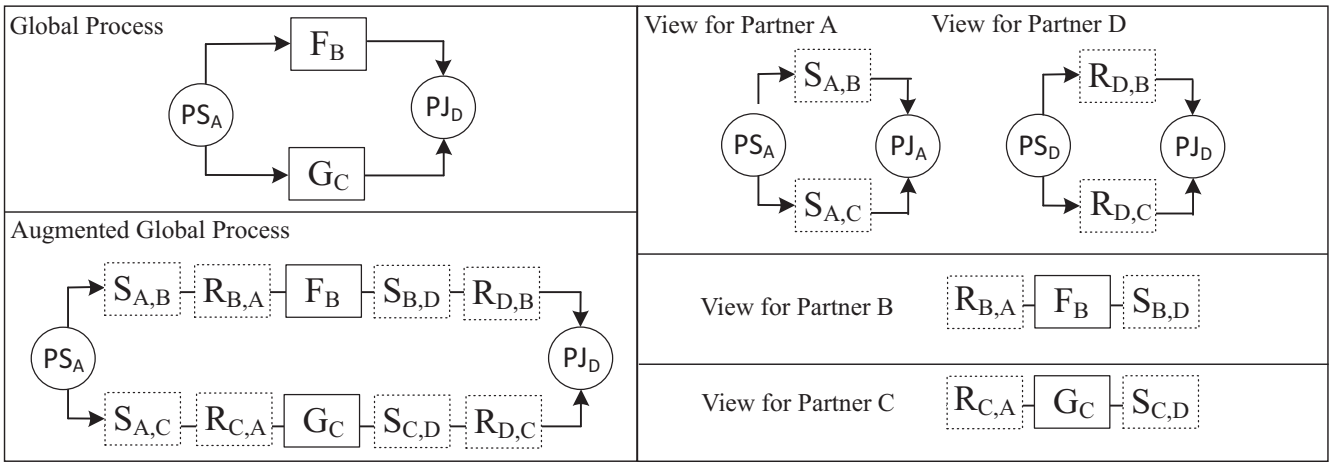

Figure 3: Partitioning of Parallel Blocks

Parallel Blocks:.

An example for the augmentation and partitioning of a parallel block is shown in Figure 3. Partner A is responsible for the parallel split. The partners B and C execute their tasks F and G in parallel. Partner D is responsible for the synchronization. The process is augmented by send- and receive- steps between partner A and B, and A and C and finally between B and D and C and D. While the parallel split and join steps remain in the views of partner A and D, partners B and C do not need to know about the parallel execution and get only sequences in their views.

XOR-Blocks:.

XOR-blocks are based on some decision variable. The owner of the XOR-split takes the decision. All partners who need to execute the XOR split themselves are informed about the decision by a message transporting the decision variable. In the example in Figure 4 partner A is responsible for the XOR split and distributes the decision variable to B and D.

Partner C does not need to know about the decision because he does not need to have an XOR-block in his view. After the XOR-split, the control flow must be forwarded. This is realized by a send- and receive- steps between A and C. There is no such communication required between A and B because B can proceed as soon as the value of the decision variable was received.

The XOR join is executed by partner D. Therefore, like for parallel join nodes the last partner in each branch needs to pass the control flow to the join partner. However, the join partner requires the decision variable from partner A to know whether the incoming message will arrive from B or from C.

In the example partner C has a sequence of steps in his view rather than an XOR-Block. This simplification is possible because C takes exclusively part in one branch of the XOR-block (and in no other step before, after, or parallel to this XOR-block). Therefore, C does not need to know about the XOR-block at all. Such a simplification is not possible for partner B because B must know whether step F needs to be executed before step H or not.

LOOP-Blocks:.

Loops are partitioned into views in analogy to XOR-blocks. The first partner in the loop (assigned to the loop-split node) is responsible for the decision. Therefore, the first partner distributes the decision variable to the partners that take part in the loop. An example is shown in Figure 5. As for $XOR-$blocks, the first partner in the loop body (B) does not need to be explicitly called by A because he receives the decision variable from A and can directly execute the loop split and the activity step. After each iteration of the loop, the current version of the decision variable is distributed to all relevant partners by a sequence of send- and receive- steps that are added directly before the loop-join.

4. AUTOMATIC GENERATION OF VIEWS

After we have introduced the general idea of the partitioning process we will now present an algorithm that realizes the discussed methods fully automatically. As in the examples, the approach is based on two phases: Augmentation and projection.

4.1 Augmentation

The automatic augmentation procedure is shown in Algorithm 1. It directly operates on the hierarchic process representation as discussed in Section 2. It is called with the root block of a process ($inBlock$) and recursively traverses the process tree. Depending on the type of the input block ($inBlock$) the different augmentation patterns introduced in Section 3 are applied.

In particular, in case of sequences send- and receive- steps are inserted between two subsequent elements $< e1, e2 >$, if $e1.lastPartner \neq e2.firstPartner$ and if the steps are not communication steps (see Figure 2). In case of $XOR-$ and $LOOP-$blocks the required decision variables are distributed to the dependent partners and the split and join steps are augmented with necessary communication steps (see Figure 4 and Figure 5). Either communication steps for decision variables or communication steps for the control flow are inserted. Parallel blocks are handled by adding communication steps for split and join steps as shown in Figure 3. Finally, loop blocks are addressed by adding communication steps for decision variables and for the control flow as shown in Figure 5.

4.2 Projection

After the global process is augmented with send- and receive- steps views for each partner are created by simply projecting the steps to the specific partners. In order to project the augmented global process to some partner p, the following steps are performed: (1) Blocks that do not contain any direct or indirect partner assignment to p are removed. (2) Control steps (XOR, LOOP, PAR), where p takes part in are rewritten to p. Thus, the first and the last partner (split and join node in graph representation) are assigned to p. (3) A special case are abstract steps that may contain multiple partners. For such blocks, the block itself is projected to p by removing all partner assignments (*firstPartner, lastPartner, interMediatePartners*) that are not equivalent to p. All these operations can easily be realized in a simple traversal of the view.

Due to space limitations we do not provide the corresponding algorithm here and refer to the technical report [18] for all details.

The corresponding procedure *projectProcess()* is shown in algorithm 2. It is called with an augmented global workflow *inBlock* and a partner p. The output of the procedure is the view for partner p.

4.3 Cleanup

After the views are created for each partner, some cleanup steps are executed using an additional traversal of the views. The following cleaning actions are performed for each view:

(1) If a $PAR - Block$ has one empty branch ($StepA$ or $StepB$ is null) in the view, then the $PAR-Block$ is replaced by a sequence.

(2) If an $XOR - Block$ has one empty branch ($StepA$ or $StepB$ is null) and the first partner of the corresponding XOR-Block in the global process is not assigned to the partner of the view (the partner of the view is not responsible for the split), and the partner of the view is not assigned to any other block that is executed before or after the XOR-block, then the $XOR - block$ is replaced by a sequence.

(3) Sequences that are containing only one element are eliminated. The details of the cleanup algorithm are dis-

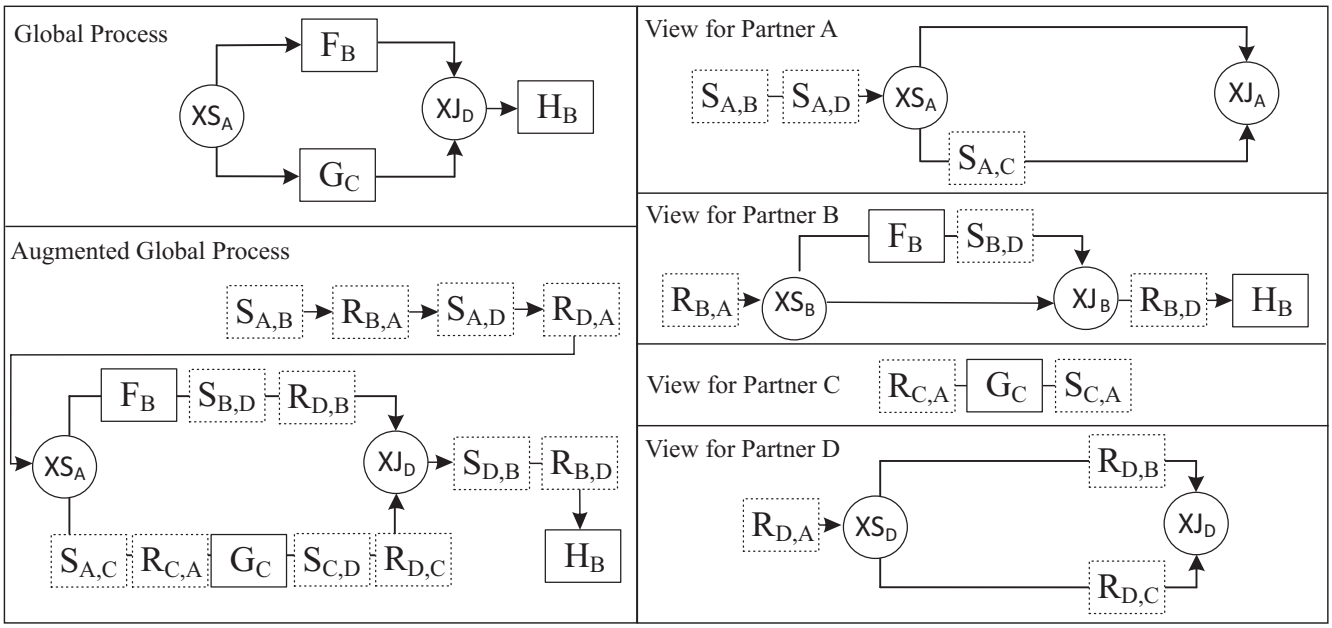

Figure 4: Partitioning of XOR-Blocks

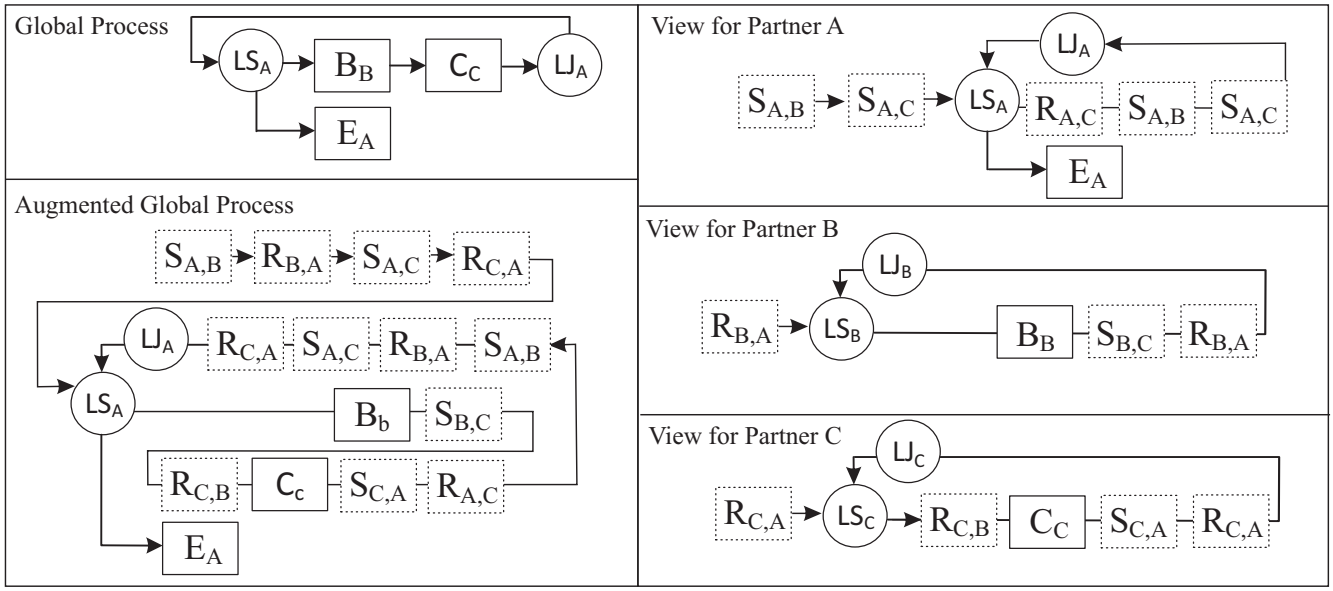

Figure 5: Partitioning of Loop-Blocks

cussed in [18].

5. EVALUATION

A partitioning is correct, if the global process and the set of communicating local processes are behaviorally equivalent, i.e. if the distributed execution of the ensemble of communicating process generated by the partitioning algorithm admits exactly the same traces of activity invocations as a centralized execution of the global process would. A sufficient condition for this behavioral equivalence is when a correct merging of the set of derived processes is identical to the global process. We will present a method for merging views in Sect. 5.1 and provide the correctness proof of our partitioning approach in Sect. 5.2.

5.1 Simple Merge of Views

A set of views $V = v_{p1} \ldots v_{pn}$ of n partners that adhere to some global process G can be merged to reconstruct the global process. We can assume that the generated views are structurally equivalent in the sense that no equivalence transformations [4, 5] of the views need to be applied in order to merge them. We will discuss a simple process merge method that exploits this property and operates on

Algorithm 1 Augmentation of a Global Process

```
Procedure augmentProcess
Input: RootBlockOfGlobalProcess inBlock
if (inBlock.type = ActivityStep or inBlock.type =
Abstract) then
    return // Base Case
end if
if (inBlock.type = 'SEQ' or inBlock.type = 'XOR' or
inBlock.type = 'PAR') then
    augmentProcess(inBlock.StepA);
    augmentProcess(inBlock.StepB);
end if
// Add send-/receive pairs for control flow
// if succeeding partners of non Communication Steps
// are different.
if (inBlock.type = 'SEQ') then
    if ((inBlock.StepA.lastPartner !=
    inBlock.StepB.firstPartner) and
    !(stepOf(inBlock.StepA.lastPartner).type =
    CommunicationStep and
    stepOf(inBlock.StepB.firstPartner).type =
    CommunicationStep)) then
        addSendReceive(inBlock)
    end if
end if
if (inBlock.type = 'XOR') then
    for all (partner ∈ inBlock.getPartners()) do
        // Add communication steps for control flow if
        dec. vars. are not sent.
        if partnerOnlyInOneBranch(partner,inBlock)
        then
            augmentSplit(inBlock)
        end if
    end for
    augmentJoin(inBlock)
    // Distribute dec. vars. to relevant partners.
    for all (partner ∈ inBlock.getPartners()) do
        if (partner ≠ inBlock.firstPartner and
        (partnerInBothBranches(partner,inBlock) or
        partner = inBlock.lastPartner or
        (partnerOnlyInOneBranch(partner,inBlock) and
        !partnerOnlyInThisBlock(partner,inBlock))))
        then
            addConditionMessages(inBlock,partner)
        end if
    end for
end if
if (inBlock.type = 'PAR') then
    augmentSplit(inBlock); augmentJoin(inBlock);
end if
if (inBlock.type = 'LOOP') then
    for all (partner ∈ {inBlock.getPartners()}) do
        if (partner ≠ inBlock.firstPartner) then
            addConditionMessages(inBlock,partner)
        // Distribute decision variables
        end if
    end for
    augmentSplit(inBlock);
    augmentProcess(inBlock.StepA);
end if
```

Algorithm 2 Create a View for a Specific Partner

```
Procedure projectProcess
Input: RootBlockOfAugmentedGlobalProcess inBlock,
Partner p
if ({p} ∩ inBlock.getPartners() == {}) then
    hide(inStep)
    return
end if
if (inBlock.type = 'Simple'}) then
    return
end if
if (inBlock.type = 'Abstract'}) then
    projectAbstractStep(inBlock,p)
    return
end if
if (inBlock.type = 'SEQ') then
    projectProcess(elem.StepA,p)
    projectProcess(elem.StepB,p)
end if
if (inBlock.type ∈ {XOR, AND, LOOP}) then
    projectBlock(inBlock,p)
    projectProcess(inBlock.stepA,p)
    if (inBlock.type ≠ LOOP) then
        projectProcess(inBlock.stepB,p)
    end if
end if
```

the graph representation of the views here. Each view $v \in V$ can be represented as a graph (V, E) (in particular a directed, attributed, acyclic graph). Each node in v has a type ∈ {XOR-Split, XOR-Join, PAR-Split, PAR-Join, LOOP-Split, LOOP-Join, ACTIVITY-STEP, Send-Step, Receive-Step, AbstractStep}. The nodes of each view are connected by the set of directed edges $v.E$. Outgoing edges of XOR-Split and LOOP-Split nodes are annotated with $true$ and $false$. All further properties of the graph representation are equivalent to those discussed in the meta-model of Section 2. The merge algorithm makes the following assumptions on the input views: For all send- and receive- steps the triple *(sendNode.to, receiveNode.from, messageType)* is always unique and activity- or abstract steps with the same label in different views define the same step in the global process.

5.1.1 Observations of the properties of a single view

We will now discuss some properties of a single view V_{px} of some partner px with regard to the merged global process V_{union}.

1. Any edge $(n1, n2) \in V_{px}.E$ where $n1.type = send \wedge n2.type \neq send \wedge (n1.sendsVar \neq n2.decisionVariable$ cannot be part of V_{union}.

2. Any edge $(n1, n2) \in V_{px}.E$ is impossible in V_{union} if $n2.type = abstract \wedge n2.first \neq px$ or $n1.type = abstract \wedge n1.last \neq px$

A send-step is connected to some receive step of another process during merging. If it is additionally connected to some local step this results in unwanted token multiplication in the merged global process. However, this behavior does not happen in the distributed execution of the private

process because the processes are synchronized with a succeeding receive step. Token multiplication / parallelism is only allowed for the distribution of decision variables (1). Incoming control flow to abstract steps can only exist to the first partner; outgoing control flow can only exist for the last partner of an abstract step. All other incoming and outgoing control flows of abstract steps are only valid in a specific view (2).
Therefore, we first remove all edges that match properties (1) or (2) before we proceed with our merge method.

5.1.2 Merging of prepared views

After the views are prepared based on the observations of Section 5.1.1 they can be merged to obtain the global process. Due to space limitations, we cannot provide the merging algorithm here and refer the reader to [18] for all details. View merging basically operates in three stages: First a merged graph that contains all the edges and all the nodes of all input views is created. In a next step matching send- (s) and receive nodes (r) where ($s.to = r.from$ *and* $s.messageType = r.messageType$) are merged. In particular, whenever a match is found send- and- receive nodes are eliminated and the previous node of the send- and the succeeding node of the receive-node are directly connected. Any additional edges of the send-node are moved from the send- node to the previous node. We preserve the annotations of edges (relevant for XOR and $LOOP-$ blocks) by adding the annotations of ($s.prev$, s) and of ($r, r.next$) to ($s.prev, r.next$).
In a next step all abstract nodes with the same label are merged and all *XOR-Split*, *Loop-Split* and *Loop-Join* nodes are merged, when they have the same previous node. Finally, unconnected edges and nodes are removed from the graph.

5.1.3 Correctness of merged views

The merging method as discussed in the previous section may produce a global process that is not behaviorally equivalent to the distributed execution of the input views. By splitting the input views into pieces during preparation and reconnecting them by matching send- and receive- steps the order of steps in the views is not guaranteed to be preserved in the merged result. Additionally, the condition for the predecessor relation between any two steps may be different in the the merged result and the views. E.g. In some view a step a may be followed by a step b in all cases but in the merged result b is only executed under a specific condition.

Definition 1. A merge-result is correct for the set of input views V, iff:

$\forall\ v \in V : \forall\ (n1,n2) \in prec(v)\ \exists\ (g1,g2) \in prec(V_{union})$ such that $n1 = g1 \wedge n2 = g2 \wedge condPrec(n1, n2, v) = condPrec(g1, g2, V_{union})$

$prec(v)$ defines the predecessor relation between all activity- or abstract steps in the view/process v. $condPrec(n1, n2, v)$ returns the set of conditions under which $n2$ is a predecessor of $n1$ in the view/process wf.

The algorithms for the automatic evaluation of the correctness property of a merge according to this definition is provided in [18].

5.2 Proving the Correctness of the Partitioning Method

Definition 2. Correctness of a Partitioning:

A partitioning $P(p) = \{v_{p1}, ..., v_{pn}\}$ of a process p for n partners is correct, iff: The representation of p as a graph is structurally equivalent to the correct merge of the views in graph representation:
$toGraph(p) \equiv merge(\{toGraph(v_{p1}), ..., toGraph(v_{pn})\}) \wedge correct(merge(\{toGraph(v_{p1}), ..., toGraph(v_{pn})\}), P(p))$.

This narrow definition of equivalence is applicable because we know that our algorithms will not require applying any equivalence transformations [4] such as changing the structure of the hierarchy before the graphs are merged or compared.

Theorem 1 (Correct Partitioning)**.** Any global process is correctly partitioned into views for k partners using the presented partitioning approach.

Proof 1. We prove the theorem by induction over the nesting depth of the global process. A process W of nesting depth 0 can only be one activity-step and the partitioning is equal to the process and is trivially correct.

We assume that processes of nesting levels up to n are partitioned correctly and show that under this assumption the processes of nesting level $n+1$ are correctly partitioned. A process of nesting level $n+1$ can be a *XOR-, LOOP-, SEQUENCE-, PAR-* Block with a sub block of nesting level n.

We use abstract steps as placeholders for the sub blocks. This is possible because they have exactly the same properties as any other block: They are defined by a first partner, a last partners and an optional set of intermediate partners (see Figure 1).

We prove by exhaustive analysis of cases, i.e. for all four types of control steps and all possible assignment configurations of partners to steps that the partitioning is correct in the following way: We generate a global process, partition the process substituting the abstract step with its projections (which are correct according to the induction assumption) and merge the generated views and compare the result with the global process.

In Figure 6 the generic cases for nesting level n+1 for sequences, parallel and loop blocks are shown. Each generic case is annotated with a set of slots #1, ..., #n that are placeholders for partners. For example, a loop has 5 slots that can be filled with up to 5 different partners. Slots #1,#2,#4,#5 are single valued slots (they must be filled with exactly one partner), while the slot #{3} can be filled with sets of partners including the empty set. The possible partner assignments range from all slots $(1, 1, \{1\}, 1, 1)$ are filled with the same partner to each slot has a different partner $(1, 2, \{3\}, 4, 5)$. We include multi-valued slots with the empty set and any set of partner of size 1. We did not test sets bigger than one because they do not constitute an additional assignment configuration for the partitioning algorithm.

While sequences, $PAR-$ and $LOOP-$ blocks are not context dependent (the partitioning only depends on the current block), the augmentation and projection of $XOR-$blocks is context-dependent. An XOR-block is transformed into a sequence in the view of a partner p and the corresponding de-

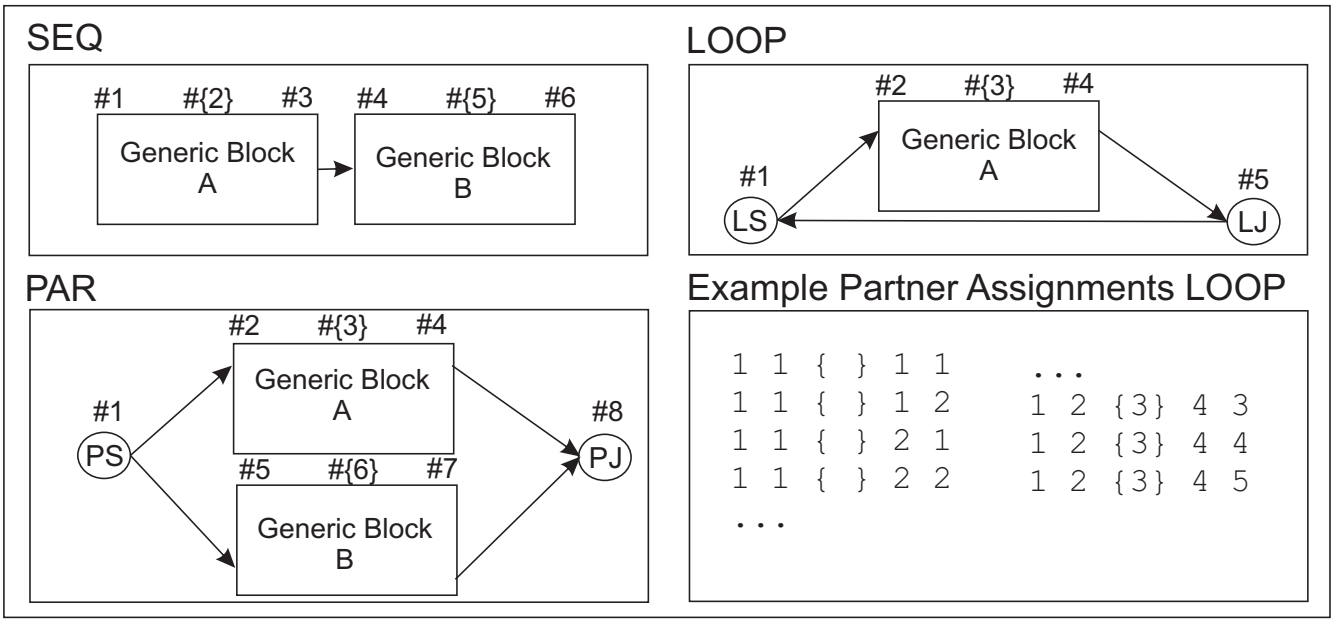

Figure 6: Generic Cases for Sequence-, Parallel- and Loop- blocks

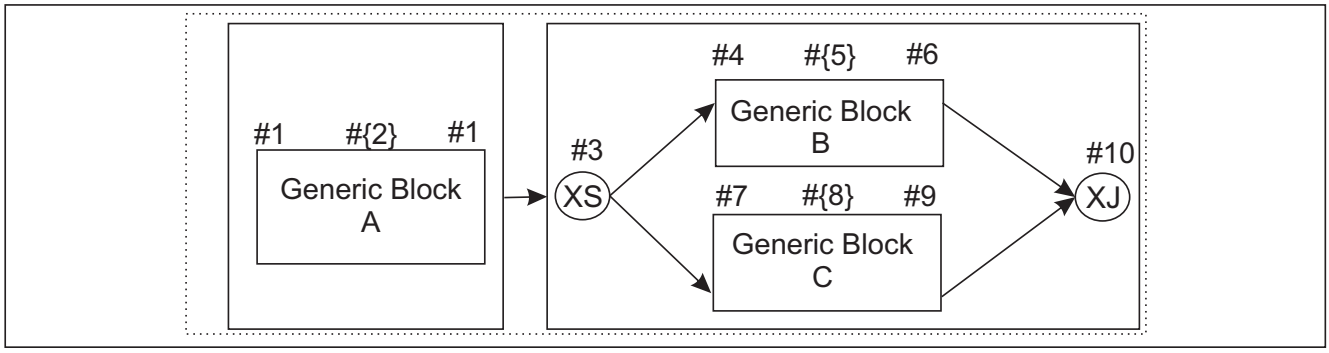

Figure 7: Generic Cases for $XOR-$Blocks

cision variable is not sent to p, if p does not take part in the other branch of the XOR-block and if p does not take part in any other block that is not a parent of the XOR block. See line 24 in Alg. 1 and item 2 in section 4.3. While the condition that a partner does not occur in both branches of an XOR *(partnerInBothBranches(partner,inBlock))* is local to the current block, the condition that a partner only takes part in this block *(partnerOnlyInThisBlock(partner,inBlock))* is dependent on the context. The two possible non-local context instantiations are that the partner either takes part in another block or that the partner does not take part in another block. Both non-local context instantiations must be covered by the analysis of $XOR-$blocks. According to the augmentation and projection algorithm, it makes no difference in what block type outside of the current XOR-block the partner in question p takes part. Therefore, it is sufficient to analyze all possible cases, when an XOR-block is nested into a sequence of two steps to simulate all possible context instantiations. The case of an XOR nested in a sequence including its 10 slots for partner assignments is shown in Figure 7.

We have implemented all required algorithms and could show that our partitioning algorithm correctly partitions all cases. □

6. RELATED WORK

Workflow View mechanisms typically allow to hide and aggregate elements of a process in order to provide a good balance between the information that needs to be shared for the cooperation and privacy concerns of the partners. Most process view approaches such as [8, 2, 21, 24, 26] allow to define views on private processes which is especially useful for outsourcing or producer/consumer scenarios. In contrast to the aforementioned view approaches we have presented a top-down approach that allows to derive views from a global process. This scenario is also addressed by the p2p approach to interorganizational workflows [32] and more recently with multi party contracts [30]. Both approaches are based on extensions of petri-nets and therefore abstract from the data perspective using indeterminism. This is especially problematic, when decisions need to be synchronized be-

tween different partners. In contrast to our approach the partitioning procedure presented in [32] does not guarantee that the resulting partitions are valid for any assignments of partners. Instead, they define that a partitioning is only correct if all resulting partitions are valid interorganizational workflow nets. This restriction was partly addressed in [30], where open workflow nets are used for modeling. However, a valid partitioning is still restricted: Interface places between partners must always be bilateral. This results in problems for possible partitions. For example, the outgoing flow of an XOR (a place with two outgoing transitions) cannot relate to different partners. The same holds for XOR joins. Therefore, it is up to the designer to create a global process that can be partitioned correctly based on the requirements. We do not need to impose such restrictions and we operate on deterministic models. We achieve this by an augmentation phase that automatically injects the required communication steps for decisions and control flow between the partners. This allows us to guarantee that any full-blocked process can be partitioned automatically based on any arbitrary partner assignment.

Another field of related research is the top-down interaction modeling of choreographies including data (message contents) which is addressed in approaches such as [17, 20]. However, they have a different scope: In interaction modeling, the entities of concern for projection are limited to messages while ignoring tasks. In our case not message-exchanges but processes are partitioned and the message-exchanges are generated automatically.

An alternative top-down process partitioning method for global processes was presented in [13]. It translates XOR and LOOP constructs in the global process to deferred choice / deferred loop constructs in the views. This is an elegant solution that minimizes message exchanges in some cases. However, it does not comply with non-local constraints, which can result in wrong projections. Another problem is that the approach requires broadcasting messages for the termination of loops. Both problems are solved by our approach.

Also directly related to our approach are partitioning methods for executable BPEL processes, which are based on the assignment of tasks to partners. Partitioning BPEL processes has specific requirements such as the distributed elimination of death paths. Approaches for their partitioning are discussed in the work of Khalaf et al [15, 14, 16]. The authors of [10, 9] claim that the work of Khalaf et al. does not provide a complete solution for the partitioning of BPEL processes and only addresses certain specific aspects of BPEL. Instead, they propose to define the partitioning on a higher level of abstraction using general block-structured processes in their own approach. However, in [10] loops are not addressed and their extension in [9] addresses loops but lacks a formal description and implementation. In contrast, our method provides a completely automatic partitioning method that also addresses loops and we could prove that our algorithms produce correct views for any block-structured input process with any assignment of partners. None of the other works stated above [15, 14, 16, 10, 9] have proven the correctness of their solutions.

7. CONCLUSIONS

We presented a novel process partitioning method for the top down development of distributed interorganizational processes. The general idea is that the cooperating partners first agree on a global business process. In a next step, each step of the global process is assigned to one of the partners. Finally, our partitioning algorithm is used to automatically derive local process definitions as views on the global process for each partner. The views contain the (abstract) tasks that should be realized by the partners and the required control structures and communication steps that specify the distributed execution (choreography) of the global process. The partners then have to map their local process to their private processes which are then actually executed. In contrast to existing approaches, we can guarantee that every full-blocked process is correctly partitioned for any arbitrary partner assignment and we generate deterministic processes by explicitly addressing the distribution of decision variables. The partitioning algorithm is an essential module in designing P2P processes. A view derived for a particular partner can be used as skeleton for the private process. Such a top-down view can also be used for checking the compatibility of a private process or of the public interface of the process with the interorganizational process.

8. REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, Berlin, Germany, October 2003.

[2] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, Feb. 2006.

[3] C. Combi and M. Gambini. Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In *On the Move to Meaningful Internet Systems: OTM 2009*, volume 5870 of *LNCS*, pages 42–59. Springer, 2009.

[4] J. Eder and W. Gruber. A meta model for structured workflows supporting workflow transformations. In *Proceedings of ADBIS 02*, pages 326–339, London, UK, UK, 2002. Springer.

[5] J. Eder, W. Gruber, and H. Pichler. Transforming workflow graphs. *Interoperability of Enterprise Software and Applications*, pages 203–214, 2006.

[6] J. Eder, N. Kerschbaumer, J. Köpke, H. Pichler, and A. Tahamtan. View-based interorganizational workflows. In *Proc. 12th Int. Conf. Computer Syst. and Tech. (CompSysTech'11)*, pages 1–10. ACM, 2011.

[7] J. Eder and A. Tahamtan. Temporal consistency of view based interorganizational workflows. In *Proc. Information Systems and e-Business Technologies, UNISCON 2008,*, volume 5 of *Lecture Notes in Business Information Processing*, pages 96–107. Springer, 2008.

[8] R. Eshuis and P. Grefen. Constructing customized process views. *Data Knowl. Eng.*, 64(2):419–438, Feb. 2008.

[9] W. Fdhila and C. Godart. Toward synchronization between decentralized orchestrations of composite web services. In *Collaborative Computing: Networking,*

Applications and Worksharing, 2009. CollaborateCom 2009. 5th International Conference on, pages 1–10, Nov 2009.

[10] W. Fdhila, U. Yildiz, and C. Godart. A flexible approach for automatic process decentralization using dependency tables. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 847–855, July 2009.

[11] H. Groiss and J. Eder. Workflow systems for inter-organizational business processes. *ACM SIGGroup Bulletin*, 18:23–26, 1997.

[12] D. Hollingsworth. The workflow reference model. 1995.

[13] N. Kerschbaumer. *View-Based Interorganizational Workflows*. Phd-thesis, Alpen Adria Universitaet Klagenfurt, Universitaetsstrasse 65-67, 9020 Klagenfurt, November 2011.

[14] R. Khalaf, O. Kopp, and F. Leymann. Maintaining data dependencies across bpel process fragments. In *Service-Oriented Computing - ICSOC 2007*, volume 4749 of *LNCS*, pages 207–219. 2007.

[15] R. Khalaf and F. Leymann. E role-based decomposition of business processes using bpel. In *Proceedings of ICWS '06*, pages 770–780, Sept 2006.

[16] R. Khalaf and F. Leymann. Coordination for fragmented loops and scopes in a distributed business process. *Information Systems*, 37(6):593 – 610, 2012.

[17] D. Knuplesch, R. Pryss, and M. Reichert. Data-aware interaction in distributed and collaborative workflows: Modeling, semantics, correctness. In *CollaborateCom*, pages 223–232. IEEE, 2012.

[18] J. Köpke, J. Eder, and M. Künstner. Implementing projections of abstract interorganizational business processes. Technical report, Universität Klagenfurt - ISYS, 2014. http://isys.uni-klu.ac.at/PDF/2014-Impl-Process-Partitioning.pdf.

[19] J. Köpke, J. Eder, and M. Künstner. Projections of abstract interorganizational business processes. In H. Decker, L. Lhotska, S. Link, M. Spies, and R. R. Wagner, editors, *Database and Expert Systems Applications*, volume 8645 of *Lecture Notes in Computer Science*, pages 472–479. Springer International Publishing, 2014.

[20] H. Nguyen, P. Poizat, and F. Zaidi. Automatic skeleton generation for data-aware service choreographies. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, pages 320–329, Nov 2013.

[21] A. Norta and R. Eshuis. Specification and verification of harmonized business-process collaborations. *Information Systems Frontiers*, 12(4):457–479, Sept. 2010.

[22] OASIS. OASIS Web Services Business Process Execution Language (WSBPEL) TC. Technical report, "OASIS", Apr. 2007.

[23] C. Peltz. Web services orchestration and choreography. *IEEE Computer*, 36(10):46–52, October 2003.

[24] M. Reichert, J. Kolb, R. Bobrik, and T. Bauer. Enabling personalized visualization of large business processes through parameterizable views. In *Proc. 27th Annu. ACM Symp. Applied Comput. (SAC'06)*, pages 1653–1660. ACM, 2012.

[25] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3):183–236, 1990.

[26] S. Smirnov, H. Reijers, M. Weske, and T. Nugteren. Business process model abstraction: a definition, catalog, and survey. *DISTRIB PARALLEL DAT*, 30(1):63–99, 2012.

[27] W. M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, volume 1248 of *LNCS*, pages 407–426. Springer, 1997.

[28] W. M. P. van der Aalst. Process-oriented architectures for electronic commerce and interorganizational workflow. *Information Systems*, 24(8):115–126, 1999.

[29] W. 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.

[30] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. Multiparty contracts: Agreeing and implementing interorganizational processes. *Comput. J.*, 53(1):90–106, 2010.

[31] W. 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, July 2003.

[32] W. M. P. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. In *Advanced Information Systems Engineering*, pages 140–156. Springer, 2001.