The creation of process redesigns by selecting, transforming and replacing process parts

The Creation of Process Redesigns
by Selecting, Transforming and Replacing Process Parts

Mariska Netjes, Hajo A. Reijers, Wil M.P. van der Aalst

Eindhoven University of Technology,
PO Box 513, NL-5600 MB Eindhoven, The Netherlands
{m.netjes,h.a.reijers,w.m.p.v.d.aalst}@tue.nl

Abstract. Business Process Redesign is a process-oriented methodology to improve organizations through a continuous evaluation and redesign of their business processes. We propose the use of an evolutionary approach towards process redesign, which is based on redesign best practices. The approach is *evolutionary*, because local updates are made to an existing process. In this paper, we focus on one part of the approach: the creation of redesign alternatives. The first step in the creation of an alternative process is the selection of a process part for redesign. This is followed by a process transformation that determines an alternative for this selected part. Finally, the original process part is replaced by the transformed part, resulting in the alternative process. Using Petri net analysis techniques, the correctness of such a redesign creation is ensured.

1 Introduction

Business Process Redesign (BPR) combines a radical restructuring of a business process with a wide-scale application of information technology [23]. Although BPR originates from the nineties, most companies today still concentrate on the redesign and improvement of business processes. According to the BPTrends BPM Market Survey 2007 "the great majority of users want improved processes" [16]. A BPR initiative would benefit from the use of *best practices*. A best practice is a historical solution that seems worthwhile to replicate in another situation or setting. A list of BPR best practices is presented in [35]. Such a list allows companies to use well-performing solutions from earlier redesign efforts. A BPR best practice consists of some kind of construct or pattern that can be distinguished in the existing process, an alternative replacing the original construct and forming the redesign, and a motivation why this redesign is better. A BPR best practice supports practitioners in developing a process redesign by making evolutionary, local updates to an existing process.

Although many methods and tools are available to facilitate the redesign process (e.g. [13,22,24]), little concrete support is provided on how to create the *to-be* situation from the *as-is* [31]. Various approaches to process redesign were proposed earlier, most notably the ProcessWise methodology [18], the MIT Process Handbook [26] and the application of the change patterns described by Weber,

Rinderle-Ma and Reichert [41]. Also, a variety of tools is available, e.g. MIT's process recombinator tool [14] and the KOPeR tool by Nissen [33]. Many existing approaches and tools are limited in their application domain, while none of the approaches has succeeded to gain widespread adoption in industry.

In our *evolutionary* approach towards process redesign we take an existing process model and improve it using redesign best practices. It is evolutionary, because an existing process is taken as a starting point. This is in contrast with a revolutionary approach that starts with a clean sheet. An example of a redesign approach starting from scratch is Product-Based Workflow Design [36].

Our evolutionary approach consists of six steps, cf. Figure 1:

1 model an existing process (in terms of a model with executable semantics),
2 determine weaknesses in the process,
3 select applicable best practices,
4 create alternative models,
5 evaluate the performance of the created alternatives, and
6 choose the best alternative and implement the new process.

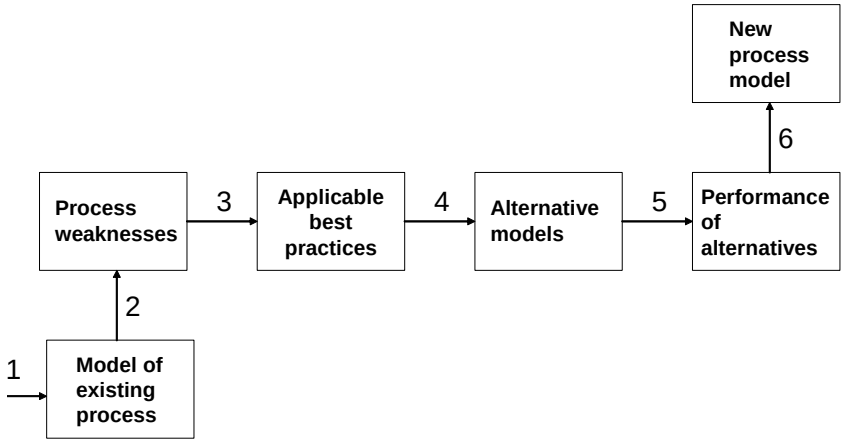

Fig. 1. Evolutionary approach towards redesign

The first three steps of Figure 1 we addressed in [30]. With regard to the modeling of an existing process, we gave a formal process definition and showed that it is not easy to spot inefficiencies in a process. For the determination of weaknesses, we presented a set of process measures, which provide a global view on the weaknesses in the process. For the selection of applicable best practices, we used and combined the set of process measures to evaluate the applicability of each best practice for the process. In the next step, the creation of alternative redesign models, the best practices are applied. This step creates the actual

redesign and results in a number of alternatives that may replace the original model. In the final steps of our approach, the performance of the various alternatives is evaluated and one redesign alternative is selected for implementation. In [9], we argue that techniques for process mining [10,11] and intelligent redesign can be used to offer better support for the (re)design and diagnosis phases of the BPM life-cycle.

In this paper, we focus on the fourth step of the evolutionary approach: *the creation of redesign alternatives*. A redesign alternative is created based on the original model by selecting a 'malfunctioning' part of the model and 'curing' it with one of the best practices. We consider the reasons for the selection of a specific part of the process and the use of a certain best practice to be out of the scope of this paper since these issues are addressed by the first three steps of the approach [30]. The contribution of this paper is twofold. First, we formally define a process model that encloses not only the control flow perspective of a process, but also the resource and data perspective. With this formal process definition we are able to model and redesign real business processes from practice. Second, we propose a method for the formally defined construction of a redesign alternative. This construction is performed in three steps. First, a process part is selected. Then, an alternative part is determined by process transformation. Finally, the original process part is replaced by the alternative part. There are different transformations for the determination of the alternative part resulting in different types of redesign alternatives.

The remainder of the paper is structured as follows. In Section 2 of this paper, we describe the problem, introduce our solution and present an example, which helps to illustrate the ideas in this paper. In Section 3, we provide the basics on graph theory, Petri nets, Workflow nets, and their properties. In Section 4, we define our notion of a process. In Section 5, we describe the three redesign steps: selection, transformation and replacement. Specific transformations are the topic of Sections 6, 7, 8 and 9. Then, we discuss related work and finally, we conclude with a summary and an outlook on future work.

2 Creation of redesigns

In this section, we explain the problem we address in this paper and introduce the proposed solution. Furthermore, we present an example that is used throughout the paper.

2.1 Problem

A redesign best practice provides directions on how the redesign should be performed. However, it only sketches the contours and does not present a redesign for a specific process. When we look at the parallelism best practice ([34], page 298), for instance, it is suggested that the redesign should have more tasks in

parallel than the original process. But it does not tell us to put tasks A, B and C of process X in parallel. In this paper, we provide a concrete method that suggests, for instance, which tasks to place in parallel by presenting an alternative parallel process model.

2.2 Creation of alternative models

The redesign best practices inspired us to come up with a redesign method that creates changes based on process transformations. This method is part of the evolutionary approach and supports step 4, *create alternative models* (see Figure 1). The method starts with the selection of a process part, step 4a. On this process part, we apply a process transformation, step 4b. A transformation performs a certain type of change and results in an alternative process part. This part replaces the original process part, thus making a local update to the process, step 4c. After the first iteration, the steps 4a, 4b and 4c, i.e., *selection*, *transformation* and *replacement*, can be performed again on the alternative process, thus creating another redesign. Each iteration results in a redesign alternative which may be used as a starting point for another local update.

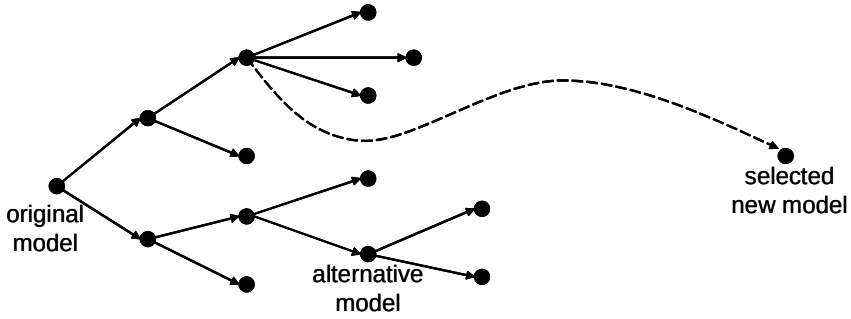

Fig. 2. Redesign alternatives tree

In Figure 2, we illustrate this iterative process with the *redesign alternatives tree*. Each node of the tree represents a process model. The root node is the original model and the start of the redesign effort, i.e., the first step of the evolutionary approach. The other nodes are alternative models and each of them results from performing step 2, 3 and 4 of the evolutionary approach. Note that an alternative model may be created from the original model (the root node) or one of the other alternative models (any other node). After the evaluation of the performance of the various alternatives (step 5), one of the alternatives is selected and implemented as the new process. This is the last step of the evolutionary approach.

A process transformation consists of operations that define how the alternative process part is derived from the selected part. We present a set of four transformations: *parallel*, *sequence*, *merge* and *unfold*. With the application of a transformation, the control flow of a process is changed. An important condition that should be fulfilled, regardless what changes are made, is that the alternative process model is correct. We discuss this correctness notion in the next section.

2.3 Example

Throughout the paper we use an example to illustrate our ideas. The example is an imaginary blood donation process which is depicted in Figure 3. In Figure 3, we already use the formal notations of the process elements, we explain their formal meaning later. In this section, we introduce the blood donation process in more general terms.

The process starts when a blood donor arrives at a donation center. He (or she) has to identify himself and his presence is registered by a desk clerk. The desk clerk instructs the donor on the procedure and provides a questionnaire. Then, the donor fills out the questionnaire and has his blood condition checked. These two tasks can be done in any order. The blood condition check consists of a check of the blood pressure and a check of the Hb value. In the meantime, one of the desk clerks has printed the labels. After this, the nurse takes blood from the donor. The nurse first takes some blood samples that will be tested in the lab and then attaches a bag that is filled with the donor blood. In the lab, the blood samples are tested for several diseases. First, a test on the hepatitis B and C virus is done by the lab analyst. Then, a lab assistant tests a blood sample on the Human T-cell Lymphotropic Virus (HTLV). Then, a blood sample is tested on the Human Immunodeficiency Virus (HIV) by a lab analyst. This HIV test consists of two separate tests on HIV and a comparison of the results. Finally, the lab assistant stores and evaluates the results of all tests and decides whether the bag of blood is save and can be used.

The blood donation process consists of 11 tasks. Two of these tasks, `Check blood condition` and `Test on HIV`, are *aggregated* tasks. An aggregated task represents a subprocess that divides the aggregated task into several smaller tasks. The aggregated task is detailed out by the subprocess and is used when a task is large or complicated to perform. In the process, tasks may have a role assigned to them. Only employees with the required role are able to perform a certain task, e.g. task `Instruct donor` may only be performed by someone with the role *desk clerk*. Furthermore, information dependencies exist between tasks. A task can only be executed when the information that is necessary for its execution is available. After the execution of a task new information may become available. For example, for the execution of task `Instruct donor` information on the patient (denoted in Figure 3 as id) is necessary. The new pieces of information that become available after the execution of task `Instruct donor` are the instruction (ins) and the questionnaire (q). Task `Answer question form` requires for its execution the availability of the information on the patient (id)

(already available after the execution of task `Register donor`), the instruction (ins) and the questionnaire (q) and cannot be executed before task `Instruct donor` is finished and the instruction (ins) and questionnaire (q) have become available.

It is possible to create an alternative process model for the blood donation process with the evolutionary approach towards process redesign. We give an example of such a redesign to illustrate the basic idea. In the first part of the current process, multiple smaller tasks are performed by a desk clerk. Before the execution of a task, a resource has to get acquainted with the case. To reduce the time that is spent to become familiar with the case, i.e., the setup time, it would be an option to combine the tasks that are performed by a desk clerk. As a result, one resource with the required role performs all tasks for one case. The positive effect of this redesign is expected to be a reduction in setup times. Another possible outcome of this redesign is that each donor only has to interact with one desk clerk. It is expected that the donor perceives a higher quality of the delivered work as a result. In the other half of the process, the lab tests are performed one after another. Unnecessary waiting times are introduced since the result of one test is not required for another test. A reduction of the throughput time is expected when (some of) the tests are executed in parallel. Next to the waiting time reduction, it is expected that there is more flexibility in a more parallel process structure. This flexibility is a result of the increase in possible orderings of the tasks. The involved resources have more freedom to schedule their work and to perform tasks in an order they prefer. Figure 4 presents the redesign alternative that follows when the introduced changes are applied to the original process. The redesign is created using the merge and parallel transformation presented in this paper.

3 Preliminaries

In this section we introduce the basic terminology and notations for graphs, Petri nets, Workflow nets and their properties.

3.1 Graphs

A graph consists of a set of nodes and a set of edges between those nodes.

Definition 1 (Graph) *A* graph $G = (N, E)$ *consist of two sets* N *and* E*.*

- N *is a non-empty finite set of* nodes*.*
- $E \subseteq N \times N$ *is a non-empty finite set of* edges*.*

Each edge has a set of two nodes associated to it, which are called its *endpoints*. One of the endpoints is designated as the *tail*, and the other endpoint is designated as the *head*. An edge is *directed from* its tail and *directed to* its head.

We define some standard graph terminology for $G = (N, E)$ [21].

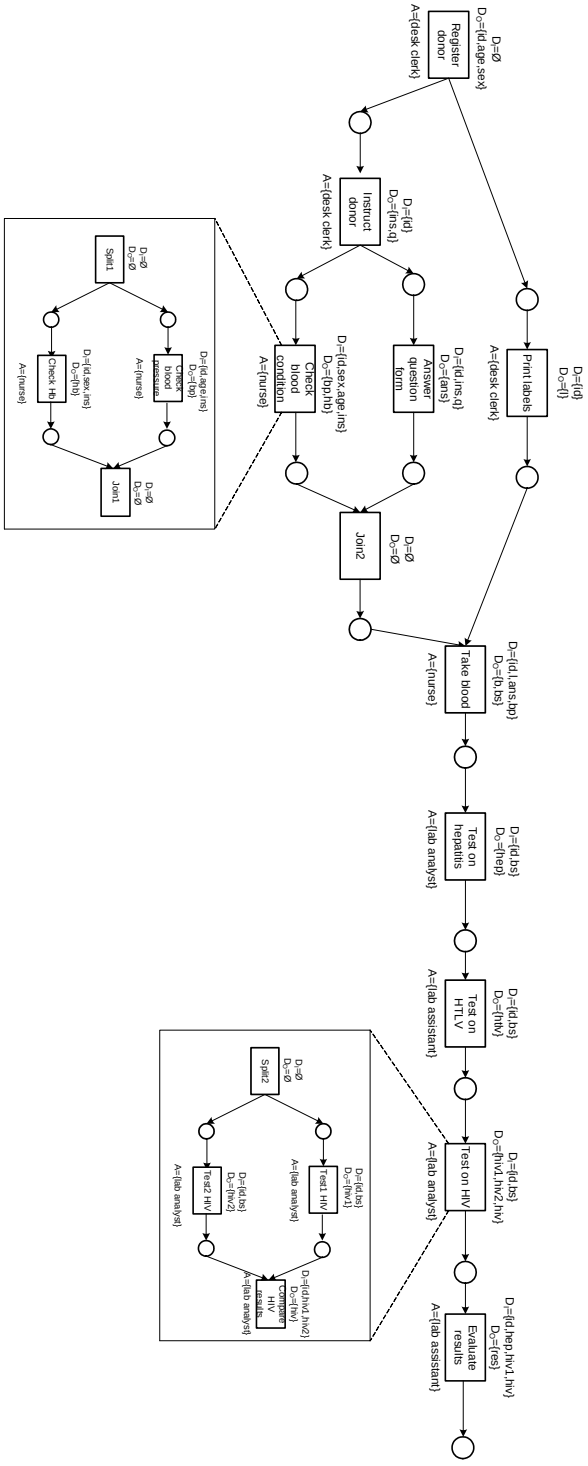

Fig. 3. Example: The blood donation process

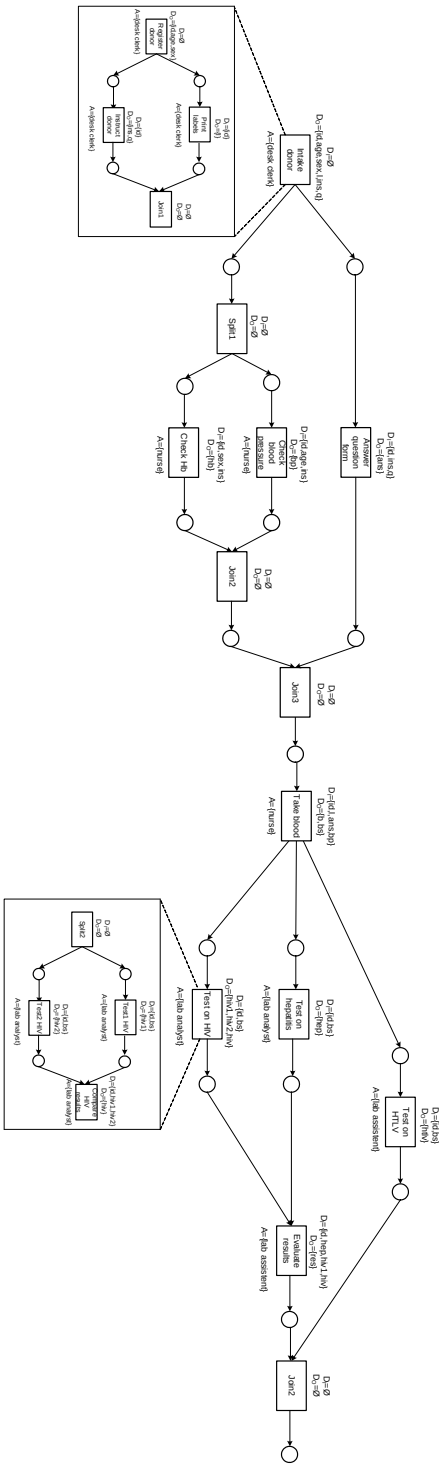

Fig. 4. Example: A possible redesign for the blood donation process

Definition 2 (Path) *A* path *in a graph $G = (N, E)$ is a nonempty sequence $n_1\ n_2...n_k$ of nodes which satisfies $(n_1, n_2), (n_2, n_3), ..., (n_{k-1}, n_k) \in E$. A path $n_1...n_k$ is said to lead from n_1 to n_k.*

Definition 3 (Circuit) *A path leading from a node x to a node y is a* circuit *if no element occurs more than once in it and $(y, x) \in E$.*

Definition 4 (Weakly connected) *A graph is* weakly connected *(or just connected) iff every pair of nodes x and y satisfies $(x, y) \in (E \cup E^{-1})^*$ [1].*

Definition 5 (Strongly connected) *A graph is* strongly connected *iff $E^* = N \times N$, i.e., for every pair of nodes x and y, there is a path leading from x to y.*

Besides weakly and strongly connected, we also define the notion of totally connected.

Definition 6 (Totally connected) *A graph is* totally connected *iff every pair of nodes x and y satisfies $(x, y) \in E^*$ or $(y, x) \in E^*$.*

Definition 7 (Acyclic) *A graph is* acyclic *iff it contains no circuits.*

A bipartite graph is a graph whose nodes can be divided into two distinct sets.

Definition 8 (Bipartite graph) *A graph is* bipartite *iff its set of nodes N can be partitioned into two sets, N_1 and N_2, in such a way, that no edge joins two nodes in the same set, i.e., the two types of nodes alternate on any path.*

3.2 Petri nets

A classical Petri net is a bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles.

Definition 9 (Petri net) *A* Petri net *is a triple (P, T, F) with:*

- *P is a non-empty finite set of* places,
- *T is a non-empty finite set of* transitions *($P \cap T = \emptyset$), and*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a set of* arcs *(flow relation).*

Elements of $P \cup T$ are called *nodes*. A node x is an *input node* of another node y iff there is a directed arc from x to y, i.e., $(x, y) \in F$. Node x is an *output node* of y iff $(y, x) \in F$. For any $x \in P \cup T$, the set $\bullet x = \{y \mid (y, x) \in F\}$ is the pre-set of x and the set $x\bullet = \{y \mid (x, y) \in F\}$ is the post-set of x. Given a set X of nodes in $P \cup T$, we define $\bullet X = \bigcup_{x \in X} \bullet x$ and $X\bullet = \bigcup_{x \in X} x\bullet$.

[1] E^* is the reflexive transitive closure of E, i.e., $(x, y) \in E^*$ if there is a (possibly empty) path from x to y in the net, and $E^{-1} = \{(y, x) | (x, y) \in E\}$.

Since Petri nets can be viewed as bipartite graphs, standard graph terminology also applies to Petri nets [19].

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as *marking*, is the distribution of tokens over places, i.e., $M \in P \rightarrow \mathbb{N}$. We will represent a state as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. We can also represent this state as follows: $p_1 + 2p_2 + p_3$. In Figure 5 a Petri net with the described state is depicted. To compare states we define a partial ordering. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

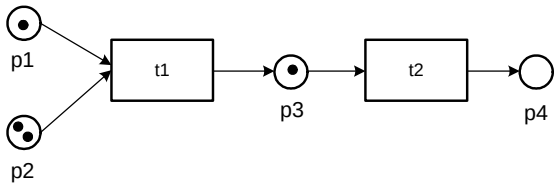

Fig. 5. A marked Petri net

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

(1) A transition t is said to be *enabled* iff each input place p of t contains at least one token.
(2) An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

In Figure 6 the result of firing t_1 in Figure 5 is shown. The resulting state is $p_2 + 2p_3$.

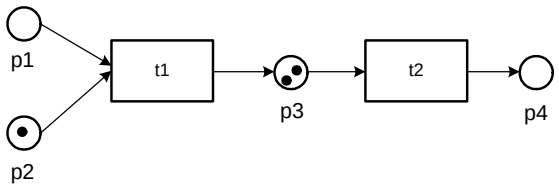

Fig. 6. The result of firing t_1

Given a Petri net (P, T, F) and a state M_1, we have the following notations:

- $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2,
- $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$,
- $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \dots t_{n-1}$ leads from state M_1 to state M_n via a (possibly empty) set of intermediate states $M_2, \dots M_{n-1}$, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \dots \xrightarrow{t_{n-1}} M_n$.

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence σ such that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*} M_1$.

We use a net system (PN, M) to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M \xrightarrow{*} M'$.

We define some standard properties for Petri nets.

Definition 10 (Live) *A Petri net (PN, M) is* live *iff, for every reachable state M' and every transition $t \in T$ there is a state M'' reachable from M' which enables t.*

Definition 11 (Bounded, safe) *A Petri net (PN, M) is* bounded *iff for each place $p \in P$ there is a natural number n such that for every reachable state the number of tokens in p is less than n. The net is safe iff for each place the maximum number of tokens does not exceed 1.*

We also define a structural property.

Definition 12 (Marked graph) *A Petri net (P, T, F) is a* marked graph *iff each place has at most one input transition and at most one output transition, i.e., for all $p \in P$: $| \bullet p | \leq 1$ and $|p \bullet| \leq 1$.*

The structural properties defined for graphs also apply to Petri nets. See [19] for a more elaborate introduction to these standard notions.

3.3 WF-nets

A Petri net that models the control-flow dimension of a workflow, is called a *Workflow net* (WF-net). The WF-net is defined by Van der Aalst [1]. In a WF-net the transitions correspond to tasks. Places correspond to pre- and post-conditions of these tasks. It should be noted that a WF-net specifies the dynamic behavior of a single *case* (or *process instance*) in isolation.

Definition 13 (WF-net [1]) *A Petri net $PN = (P, T, F)$ is a* WF-net *(Workflow net) if and only if:*

- *There is one* source place *$i \in P$ such that $\bullet i = \emptyset$.*
- *There is one* sink place *$o \in P$ such that $o\bullet = \emptyset$.*
- *Every node $x \in P \cup T$ is on a path from i to o.*

A WF-net has one input place (i) and one output place (o) because any case handled by the process represented by the WF-net is created when it enters the process and is deleted once it is completely handled. The third requirement in Definition 13 has been added to avoid "dangling tasks and/or conditions", i.e., tasks and conditions which do not contribute to the processing of cases.

The number of tasks that are carried out and the order in which tasks are performed may vary from case to case. A WF-net determines how cases are routed over the various tasks. Van der Aalst and Van Hee [7] recognize four basic constructions for the routing of cases. In a *sequential routing construct* tasks have to be carried out one after another. A *parallel routing construct* is necessary if more than one task can be executed at the same time or in any order. A *selective routing construct* is used when there exists differences in routing between individual cases. With the last form of routing, the *iterative routing construct*, it is possible to repeat the execution of a task, resulting in multiple executions of the task for one case.

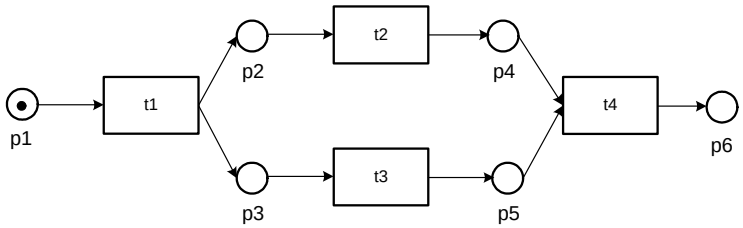

Fig. 7. Parallel routing

For the last three routing constructs it may be necessary to include routing tasks. A routing task is a task that is solely added for routing purposes without providing any additional value to the case execution. In Figure 7, we give an example of the use of routing tasks to model a parallel routing construct. In the example, the tasks $t1$ and $t4$ are routing tasks. A routing task like $t1$ is called an AND-split, because the work is being split up over various tasks that are executed in parallel. Tasks, like $t4$, that join various parallel routes, are called AND-join. Routing tasks are also used for the selective and the iterative routing constructs. Then, a routing task is used to select one of a number of tasks and is called an XOR-split. A routing task may also be used to join several routes of which only one route has been followed and such a routing task is called an XOR-join.

Given the definition of a WF-net it is easy to derive the following properties [4].

Proposition 1 (Properties of WF-nets [8]). *Let $PN = (P, T, F)$ be a Petri net.*

- *If PN is a WF-net with source place i, then for any place* $p \in P$: $\bullet p \neq \emptyset$ *or* $p = i$, *i.e., i is the only source place.*
- *If PN is a WF-net with sink place o, then for any place* $p \in P$: $p\bullet \neq \emptyset$ *or* $p = o$, *i.e., o is the only sink place.*
- *If PN is a WF-net and we add a transition* t^* *to PN which connects sink place o with source place i, i.e.,* $\bullet t^* = \{o\}$ *and* $t^*\bullet = \{i\}$, *then the resulting Petri net is strongly connected.*

3.4 Soundness

In this section we summarize some of the basic results for WF-nets presented in [1,2,4].

The three requirements stated in Definition 13 can be verified statically, i.e., they only relate to the structure of the Petri net. However, there is another requirement which should be satisfied:

> *For any case, the procedure will terminate eventually and the moment the procedure terminates there is a token in place o and all the other places are empty.*

Moreover, there should be no dead tasks, i.e., it should be possible to execute an arbitrary transition by following the appropriate route through the WF-net. These requirements correspond to the so-called *soundness property* [2].

Definition 14 (Soundness [2]) *A procedure modeled by a WF-net* $PN = (P, T, F)$ *is sound if and only if:*

(i) *For every state M reachable from state i, there exists a firing sequence leading from state M to state o, i.e.,* $\forall_M (i \xrightarrow{*} M) \Rightarrow (M \xrightarrow{*} o)$ [2],
(ii) *State o is the only state reachable from state i with at least one token in place o, i.e.,* $\forall_M (i \xrightarrow{*} M \ \wedge \ M \geq o) \Rightarrow (M = o)$, *and*
(iii) *There are no dead transitions in* (PN, i), *i.e.,* $\forall_{t \in T} \ \exists_{M,M'} \ i \xrightarrow{*} M \xrightarrow{t} M'$.

Note that the soundness property relates to the dynamics of a WF-net. The first requirement in Definition 14 states that starting from the initial state (state i), it is always possible to reach the state with one token in place o (state o). Clearly, there should not be an infinite loop in the WF-net. Since the WF-net is used in the context of Workflow Management (WFM) [7], all choices are made (implicitly or explicitly) by applications, humans or external actors which do not introduce infinite loops.

Given a WF-net $PN = (P, T, F)$, we want to decide whether PN is sound. In [1], Van der Aalst has shown that soundness corresponds to liveness and boundedness. To link soundness to liveness and boundedness, we define an extended

[2] Note that there is an overloading of notation: the symbol i is used to denote both the *place* i and the *state* with only one token in place i (see Section 3.2).

net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$. $\overline{PN}$ is the Petri net obtained by adding an extra transition t^* which connects o and i. The extended Petri net $\overline{PN} = (\overline{P}, \overline{T}, \overline{F})$ is defined as follows: $\overline{P} = P$, $\overline{T} = T \cup \{t^*\}$, and $\overline{F} = F \cup \{(o, t^*), (t^*, i)\}$. In the remainder we call such an extended net the *short-circuited* net of PN. The short-circuited net allows for the formulation of the following theorem.

Theorem 1. *A WF-net PN is sound if and only if* $(\overline{PN}, i)$ *is live and bounded.*

Proof. See [1]. □

This theorem shows that standard Petri-net-based analysis techniques can be used to verify soundness in polynomial time.

Sometimes we require a WF-net to be safe, i.e., no marking reachable from (PN, i) marks a place twice. Although safeness is defined with respect to some initial marking, we extend it to WF-nets and simply state that a WF-net is safe or not given initial marking i.

4 Process Definition

The starting point of a redesign effort is a process that is currently being used in an organization. In this section we provide a formal process definition to describe a real process. A process has certain characteristics, like its structure, and our formal notion of these characteristics is addressed in the first half of this section. Several properties of the formal process definition are discussed in the second half.

4.1 Process characteristics

In our formal process definition we distinguish between the *process structure* and *process information*. Both are necessary to model a realistic business process and to create redesign alternatives that are applicable in practice. First, we focus on the process structure for which we introduce the notion of a *SISO-net*. A SISO-net, as given in Definition 15, is a generalized WF-net with a single input (SI) and a single output (SO).

Definition 15 (SISO-net) *A Petri net* (P, T, F) *is a SISO-net if and only if there is a unique source node* i *and a unique sink node* o *such that:*

- $\{n \in P \cup T \mid \bullet n = \emptyset\} = \{i\}$,
- $\{n \in P \cup T \mid n\bullet = \emptyset\} = \{o\}$,
- $\forall_{n \in P \cup T} : (i, n) \in F^* \wedge (n, o) \in F^*$.

Note that a WF-net is also a SISO-net, but that transition-bordered SISO-nets are not WF-nets.

With the SISO-net we describe the process structure. Next to the process structure or control flow, we introduce process information. Process information consists of *data dependencies* and *roles*. When a data dependency exists between two transitions, this means that one transition requires information for its execution that becomes available with the execution of the other transition. For each transition, the data elements that should be available before it can be executed, are called its input data elements. The data elements that become available after its execution are called its output data elements. For more information on the use of data dependencies in process modeling we refer to [38]. When a role is assigned to a transition, the transition has to be executed by a resource that has the required role. A transition does not always have an assigned role, for instance, if the task is solely added to the process for routing purposes. For more information on the use of resources and roles in process modeling we refer to [7]. With the use of data dependencies and roles, the data and resource perspectives are included in our process definition. Process information is defined as an *annotation* of the SISO-net, i.e., an annotated SISO-net is a SISO-net enriched with process information.

Definition 16 (Annotated SISO-net) *The 8-tuple* $S = (P, T, F, D, D_I, D_O, R, A)$ *is an annotated SISO-net if:*

- (P, T, F) *is a SISO-net,*
- D *is a finite set of data elements,*
- $D_I \in T \rightarrow \mathcal{P}(D)$ [3] *relates transitions to sets of input data elements, i.e., these elements should be available before a transition can be executed,*
- $D_O \in T \rightarrow \mathcal{P}(D)$ *relates transitions to sets of output data elements, i.e., these elements become available when a transition has been executed,*
- R *is a finite set of roles, and*
- $A \in T \not\rightarrow R$ [4] *assigns an optional role to each transition.*

We introduce some shorthand notations for the annotated SISO-net $S = (P, T, F, D, D_I, D_O, R, A)$:

- $\pi_P(S) = P$,
- $\pi_T(S) = T$,
- $\pi_F(S) = F$,
- $\pi_D(S) = D$,
- $\pi_{D_I}(S) = D_I$,
- $\pi_{D_O}(S) = D_O$,
- $\pi_R(S) = R$, and
- $\pi_A(S) = A$.

Next to the annotated SISO-net, we introduce a *layered annotated SISO-net*. The notion of a layered annotated SISO-net is needed because it is possible for an annotated SISO-net to contain *aggregated transitions*. An aggregated transition

[3] $\mathcal{P}(X)$ is the powerset of X, i.e., $Y \in \mathcal{P}(X)$ if and only if $Y \subseteq X$.
[4] A is a partial function, i.e., the domain of A is a subset of T.

is linked with a subprocess that divides the aggregated transition into several smaller pieces of work. Such a sub process is used for more complicated or larger transitions to give more insight on what should exactly be done. Aggregated transitions are executed just like any other task, that is, when a resource starts the execution of an aggregated task, (s)he finishes the complete underlying sub process at once. This way, the ACID properties (atomicity, consistency, isolation, durability) ([7], page 166) also hold for an aggregated transition.

A layered annotated SISO-net has a two layered-process structure: 1) the upper layer is an annotated SISO-net, including aggregated transitions, 2) the lower layer contains an annotated SISO-net for each of the aggregated transitions. We restrict ourselves to two layers, because a multiple-layered process model may be difficult to understand. A layered annotated SISO-net contains a finite set of annotated SISO-nets. The places and transitions forming these individual annotated SISO-nets are required to be disjoint, i.e., a place or transition can only be a member of one annotated SISO-net. We require a lower layer annotated SISO-net to start and end with a transition because the annotated SISO-net is, just like its associated aggregated transition, preceded and followed by a place. To make the aggregated transition executable by one role, all transitions in a lower layer annotated SISO-net are executed by the same role. The SISO-net assignment function assigns to each aggregated transition its associated, annotated SISO-net. A transition and its assigned annotated SISO-net have the same annotation. Input data elements that are not created by a transition in the assigned annotated SISO-net are the input data elements of the transition. All output data elements in the assigned annotated SISO-net are output data elements of the transition. The role assigned to the transitions in the assigned annotated SISO-net is also assigned to the transition.

Definition 17 (Layered annotated SISO-net) *$LS = (S, SS, map)$ is a layered annotated SISO-net with:*

- *$S = (P, T, F, D, D_I, D_O, R, A)$ is an annotated SISO-net,*
- *SS is a finite set of annotated SISO-nets, such that:*
  - *i_s is the source node of $s \in SS$,*
  - *o_s is the sink node of $s \in SS$,*
  - *$\forall_{s \in SS} : i_s, o_s \in \pi_T(s)$,*
  - *$\forall_{s,s' \in SS} : s \neq s' \Rightarrow (\pi_P(s) \cap \pi_P(s') = \emptyset) \wedge (\pi_T(s) \cap \pi_T(s') = \emptyset)$, and*
- *$map \in T \not\rightarrow SS$ assigns an annotated SISO-net to each aggregated transition such that:*
  - *the assignment is injective and surjective,*
  - $\forall_{t \in dom(map)} : D_I(t) = \left(\bigcup_{t' \in \pi_T(map(t))} \pi_{D_I}(map(t))(t') \right) \setminus \left(\bigcup_{t' \in \pi_T(map(t))} \pi_{D_O}(map(t))(t') \right)$,
  - $\forall_{t \in dom(map)} : D_O(t) = \bigcup_{t' \in \pi_T(map(t))} \pi_{D_O}(map(t))(t')$,

- $\forall_{t \in dom(map) \cap dom(A)} : \forall_{t' \in \pi_T(map(t))} : A(t) = \pi_A(map(t))(t')$.

We introduce some shorthand notations for a layered annotated SISO-net $LS = (S, SS, map)$:

- $\pi_S(LS) = S$,
- $\pi_{SS}(LS) = SS$, and
- $\pi_{map}(LS) = map$.

4.2 Process properties

The annotated SISO-net has one source node and one sink node and we give two operations to find this in- and output of the annotated SISO-net.

Definition 18 (In, out) *Let S be an annotated SISO-net. Operation $\mathbf{in}$ returns the source node of S, i.e., $\mathbf{in}(S) = i$, with $i \in \pi_P(S) \cup \pi_T(S)$ and $\bullet i = \emptyset$. Operation $\mathbf{out}$ returns the sink node of S, i.e., $\mathbf{out}(S) = o$, with $o \in \pi_P(S) \cup \pi_T(S)$ and $o\bullet = \emptyset$.*

For SISO-nets we define properties similar to a WF-net. Before doing so, Definition 19 presents for a given annotated SISO-net the associated WF-net.

Definition 19 (Associated WF-net) *Let S be an annotated SISO-net and i_S and o_S two "fresh" identifiers, i.e., $\{i_S, o_S\} \cap (\pi_P(S) \cup \pi_T(S)) = \emptyset$, then $\widetilde{S} = (P, T, F)$ is the WF-net associated with S with:*

$$P = \pi_P(S) \cup \begin{cases} \emptyset & \text{if} \quad \mathbf{in}(S) \in \pi_P(S) \wedge \mathbf{out}(S) \in \pi_P(S), \\ \{i_S\} & \text{if} \quad \mathbf{in}(S) \notin \pi_P(S) \wedge \mathbf{out}(S) \in \pi_P(S), \\ \{o_S\} & \text{if} \quad \mathbf{in}(S) \in \pi_P(S) \wedge \mathbf{out}(S) \notin \pi_P(S), \\ \{i_S, o_S\} & \text{if} \quad \mathbf{in}(S) \notin \pi_P(S) \wedge \mathbf{out}(S) \notin \pi_P(S). \end{cases}$$

$T = \pi_T(S)$,

$$F = \pi_F(S) \cup \begin{cases} \emptyset & \text{if} \quad \mathbf{in}(S) \in \pi_P(S) \wedge \mathbf{out}(S) \in \pi_P(S), \\ \{(i_S, \mathbf{in}(S))\} & \text{if} \quad \mathbf{in}(S) \notin \pi_P(S) \wedge \mathbf{out}(S) \in \pi_P(S), \\ \{(\mathbf{out}(S), o_S)\} & \text{if} \quad \mathbf{in}(S) \in \pi_P(S) \wedge \mathbf{out}(S) \notin \pi_P(S), \\ \{(i_S, \mathbf{in}(S)), (\mathbf{out}(S), o_S)\} & \text{if} \quad \mathbf{in}(S) \notin \pi_P(S) \wedge \mathbf{out}(S) \notin \pi_P(S). \end{cases}$$

Definition 20 defines safeness and soundness for an annotated SISO-net. The basic idea is that these properties apply to the associated WF-net.

Definition 20 (Safe and sound annotated SISO-net) *An annotated SISO-net S is safe and sound if and only if the associated WF-net $\widetilde{S}$ is safe and sound.*

Definition 21 defines safeness and soundness for a layered annotated SISO-net.

Definition 21 (Safe and sound layered annotated SISO-net) *A layered annotated SISO-net LS is safe and sound if $\pi_S(LS)$ is safe and sound and for all $s \in \pi_{SS}(LS)$, s is safe and sound.*

In the next section we describe the creation of a redesign for a layered annotated SISO-net by selecting, transforming and replacing process parts.

5 Selection, Transformation, Replacement

When conducting a process redesign with the evolutionary approach, part of an existing process model needs to be changed. This change is performed in three steps:

(1) *Selection*: the process part that should be changed is selected from the process,
(2) *Transformation*: an alternative for the selected part is created,
(3) *Replacement*: in the process the selected part is replaced by the alternative part.

The result is a redesign alternative for the original process. In this section we elaborate on the three redesign steps.

5.1 Selection

The first step of the redesign effort is the selection of the part of the process that should be changed. Figure 8 illustrates the selection of a process part.

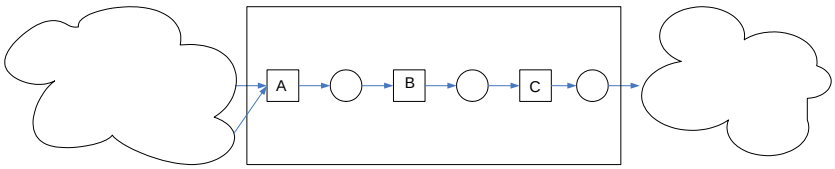

Fig. 8. The selection of a process part

We use the notion of a *component* to define the selected process part. A component may be seen as a selected part of a SISO-net with a clear start and end, i.e., a component is a sub net satisfying properties similar to a SISO-net. Definition 22 gives the component for an annotated SISO-net. The definition is derived from Van der Aalst and Bisgaard Lassen [8], who introduced the notion of a component in a similar setting.

Definition 22 (Component) *Let S be an annotated SISO-net, then C is a component in S if and only if:*

- $C \subseteq \pi_P(S) \cup \pi_T(S)$,
- *there are source and sink nodes* $i_C, o_C \in C$ *such that:*
  - $i_C \neq o_C$,
  - $\bullet(C \setminus \{i_C\}) \subseteq C \setminus \{o_C\}$,
  - $(C \setminus \{o_C\})\bullet \subseteq C \setminus \{i_C\}$, *and*
  - $(o_C, i_C) \notin \pi_F(S)$.

Note that any component contains at least one transition and one place. We only consider non-trivial components, i.e., components with more than one transition. The following definition provides the projection of a net on one of its components.

Definition 23 (Projection) *Let S be an annotated SISO-net and C a component in S. The projection of S on C, $S||_C$, is then defined as $S||_C = (P, T, F, D, D_I, D_O, R, A)$ with:*

- $P = \pi_P(S) \cap C$ *is the set of places,*
- $T = \pi_T(S) \cap C$ *is the set of transitions,*
- $F = \pi_F(S) \cap (C \times C)$ *is the flow relation,*
- $D = \bigcup_{t \in T} \pi_{D_I}(S)(t) \cup \pi_{D_O}(S)(t)$ *is the set of data elements,*
- $D_I \in T \rightarrow \mathcal{P}(D)$ *is the set of input data elements such that* $\forall_{t \in T} : D_I(t) = \pi_{D_I}(S)(t)$,
- $D_O \in T \rightarrow \mathcal{P}(D)$ *is the set of output data elements such that* $\forall_{t \in T} : D_O(t) = \pi_{D_O}(S)(t)$,
- $R = \bigcup_{t \in T} \pi_A(S)(t)$ *is the set of roles, and*
- $A \in T \not\rightarrow R$ *is the role assignment such that* $dom(A) = dom(\pi_A(S)) \cap C$ *and* $\forall_{t \in dom(A)} : A(t) = \pi_A(S)(t)$.

Note that the shorthand notations for an annotated SISO-net, see the shorthand notations following Definition 16, can also be used for $S||_C$, the projection of S on C.

The next theorem not only shows that an annotated SISO-net projected on a component results in an annotated SISO-net, but that, provided the initial annotated SISO-net is safe and sound, the projection is also safe and sound. Later on, this result is used to prove the compositional nature of safe and sound annotated SISO-nets.

Theorem 2. *Let S be a safe and sound annotated SISO-net and C a component of S with source node i_C and sink node o_C, then the projection $S||_C$ is a safe and sound annotated SISO-net.*

Proof. First, we prove that $S||_C$ is an annotated SISO-net. If $i_C \in C$ is the source node of C, it has become the unique source node of $S||_C$. Similarly, o_C has become the unique sink node of $S||_C$. Each node $x \in C \setminus \{i_C\}$ must be on a

path from i_C. Otherwise x would be a source node or there is another source node $y \in C \setminus \{i_C\}$ such that there is a path from y to x. Both are impossible which follows from the definition of a component (Definition 22). A similar argument may be given for a path from x to o_C. Therefore, $S||_C$ is an annotated SISO-net.

Second, we prove that the annotated SISO-net $S||_C$ is safe and sound. We start with soundness. Assume i_C and o_C are places. Since S is sound, it is possible to mark i_C in S when starting in (S, i) because otherwise transitions in $i_C\bullet$ would be dead. Because S is sound, place o_C will become marked in the process of reaching (S, o) from (S, i_C). This is the only possible way to remove tokens from $S||_C$ as there are no other places in $S||_C$ that can transfer tokens to the rest of S. Also, if o_C becomes marked, all other places in $S||_C$ are empty. If that would not be the case, o_C could be marked again by internal firings of $S||_C$ (not safe) or a token would be stuck in $S||_C$ (not sound). Therefore, $S||_C$ is sound. If i_C and/or o_C are not places, similar arguments apply. $S||_C$ is also safe because if this would not be the case S would also not be safe. □

Soundness and safeness are desirable properties. Theorem 2 shows that these desirable properties are propagated to any projection of the net on a component. The component is the input for the next redesign step: the transformation.

5.2 Transformation

At this point, we are able to select a process part from the original process. We can use this result for the second step of process redesign: the transformation. In the transformation, an alternative process part is created for the selected process part, i.e., the component. Figure 9 shows an example of a transformation. The

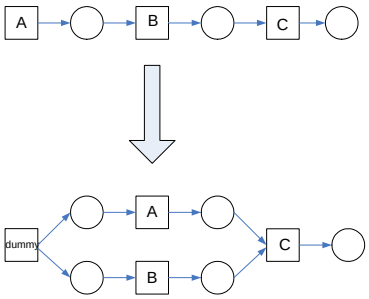

Fig. 9. A transformation: the selected component is transformed

transformation specifies the type of change that is performed. We introduce several transformations that each change the selected component in a different way:

– The *unfold* transformation splits aggregated tasks into several smaller tasks and is discussed in Section 6,

– The *parallel* transformation places tasks that do not depend on one another in parallel and is discussed in Section 7,
– The *sequence* transformation puts a total ordering on tasks and is discussed in Section 8, and
– The *merge* transformation combines multiple tasks into one aggregated task and is discussed in Section 9.

Note that these transformations are just four examples. Our approach also allows for other transformations and is therefore extendible.

5.3 Replacement

After the selection of a component and the creation of an alternative for this selected part, an alternative process is constructed. First, the selected component is removed from the process by disconnecting its source and sink node from the surrounding process. This is illustrated with Figure 10. Then, the alternative part replaces the removed component and is connected to the surrounding process. Figure 11 illustrates the replacement with the alternative process part.

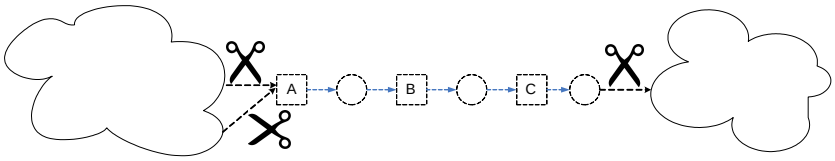

Fig. 10. The replacement of a component...

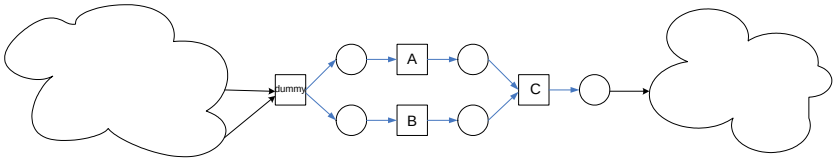

Fig. 11. ... with an alternative process part

Definition 24 (Replace) *Let* $LS_1 = (S_1, SS_1, map_1)$ *and* $LS_2 = (S_2, SS_2, map_2)$ *be two layered annotated SISO-nets, and* C *a non-trivial component in* S_1 *with source node* i_C *and sink node* o_C*.* S_2 *is such that* $\texttt{in}(S_2)$ *is a place if and only if* i_C *is a place and* $\texttt{out}(S_2)$ *is a place if and only if* o_C *is a place. Operation* `replace` *substitutes* $S_1||_C$ *in* LS_1 *with* S_2 *resulting in* $\texttt{replace}(LS_1, C, LS_2) = (S_3, SS_3, map_3)$ *with:*

- $S_3 = (P_3, T_3, F_3, D_3, D_{I_3}, D_{O_3}, R_3, A_3)$ *is an annotated SISO-net with:*
  - $P_3 = (\pi_P(S_1) \setminus C) \cup \pi_P(S_2)$ *is the set of places* [5]*,*
  - $T_3 = (\pi_T(S_1) \setminus C) \cup \pi_T(S_2)$ *is the set of transitions,*
  - $F_3 = (\pi_F(S_1) \cap ((P_3 \times T_3) \cup (T_3 \times P_3))) \cup$ $\pi_F(S_2) \cup$ $\{(n, \texttt{in}(S_2)) \mid (n, i_C) \in \pi_F(S_1)\} \cup$ $\{(\texttt{out}(S_2), n) \mid (o_C, n) \in \pi_F(S_1)\}$ *is the flow relation,*
  - $D_3 = \left(\bigcup_{t \in \pi_T(S_1) \setminus C} \pi_{D_I}(S_1)(t) \cup \pi_{D_O}(S_1)(t) \right) \cup \pi_D(S_2)$ *is the set of data elements,*
  - $D_{I_3} \in T_3 \to \mathcal{P}(D_3)$ *is the set of input data elements such that:*
    - $\forall_{t \in \pi_T(S_1) \setminus C} : D_{I_3}(t) = \pi_{D_I}(S_1)(t)$*, and*
    - $\forall_{t \in \pi_T(S_2)} : D_{I_3}(t) = \pi_{D_I}(S_2)(t)$*,*
  - $D_{O_3} \in T_3 \to \mathcal{P}(D_3)$ *is the set of output data elements such that:*
    - $\forall_{t \in \pi_T(S_1) \setminus C} : D_{O_3}(t) = \pi_{D_O}(S_1)(t)$*, and*
    - $\forall_{t \in \pi_T(S_2)} : D_{O_3}(t) = \pi_{D_O}(S_2)(t)$*,*
  - $R_3 = \left(\bigcup_{t \in \pi_T(S_1) \setminus C} \pi_A(S_1)(t) \right) \cup \pi_R(S_2)$ *is the set of roles, and*
  - $A_3 \in T_3 \not\to R_3$ *is the role assignment such that:*
    - $dom(A_3) = (dom(\pi_A(S_1)) \setminus C) \cup dom(\pi_A(S_2))$*,*
    - $\forall_{t \in dom(\pi_A(S_1)) \setminus C} : A_3(t) = \pi_A(S_1)(t)$*, and*
    - $\forall_{t \in dom(\pi_A(S_2))} : A_3(t) = \pi_A(S_2)(t)$*,*
- $SS_3 = \{map_1(t) \mid t \in dom(map_1) \setminus C\} \cup SS_2$ *is the set of annotated SISO-nets, and*
- $map_3 \in T_3 \not\to SS_3$ *is the assignment function such that:*
  - $dom(map_3) = (dom(map_1) \setminus C) \cup dom(map_2)$*,*
  - $\forall_{t \in dom(map_1) \setminus C} : map_3(t) = map_1(t)$*, and*
  - $\forall_{t \in dom(map_2)} : map_3(t) = map_2(t)$*.*

The next theorem shows that the result of the replacement is again a safe and sound layered annotated SISO-net.

Theorem 3. *Let* $LS_1 = (S_1, SS_1, map_1)$ *and* $LS_2 = (S_2, SS_2, map_2)$ *be two layered annotated SISO-nets, and* C *a non-trivial component in* S_1*. If* $LS_3 = \texttt{replace}(LS_1, C, LS_2)$*, then:*

- LS_3 *is a layered annotated SISO-net.*
- *If* LS_1 *is safe and sound and* LS_2 *is safe and sound, then* LS_3 *is safe and sound.*

Proof. First, we prove that LS_3 is a layered annotated SISO-net. If source node i and sink node o of LS_1 are not involved in the replacement, i.e., these nodes are not incorporated in $S_1||_C$, then replacing $S_1||_C$ by S_2 does not remove source i

[5] We assume there are no "name clashes".

or sink o of LS_1. If source node i (sink node o) is incorporated in $S_1||_C$, then this node is replaced by the source (sink) node of S_2. Replacing does not introduce any new source or sink nodes and all nodes are on a path from i to o.

Next, we prove that if LS_1 and LS_2 are safe and sound, LS_3 is safe and sound. Theorem 2 already showed that if LS_1 is safe and sound, $S_1||_C$ is safe and sound. We follow the same line of reasoning as in the proof of Theorem 3 in [8] where it is shown that a safe and sound (sub) WF-net behaves like a transition which may postpone the production of tokens and that in a safe and sound WF-net such a sub net can be replaced by a transition without changing dynamic properties such as safeness and soundness. In our case, component C in LS_1 becomes "active" is one of its transitions fires. Then there may be several internal steps in C executed in parallel with the rest of LS_1 but eventually o_C (if o_C is a place) or the places in $o_C\bullet$ (if o_C is a transition) get marked. Since LS_1 is safe, C can be activated only once. Similarly, an activation of LS_2 in LS_3 will lead to a marking of o_2 (if o_2 is a place) or the places in $o_2\bullet$ (if o_2 is a transition) in LS_2. Hence, if one abstracts from the internal states of $S_1||_C$ and LS_2, LS_1 and LS_3 have identical behaviors and clearly LS_3 is safe and sound. In other words: since both $S_1||_C$ and LS_2 behave like a transition which may postpone the production of tokens, we can replace $S_1||_C$ with LS_2 without changing dynamic properties such as safeness and soundness. Furthermore, in the proof of Theorem 3 in [4] it is shown that a WF-net, resulting from the replacement of a transition from a safe and sound WF-net with another safe and sound (sub) WF-net, is safe and sound. Also here, the sub net behaves like a transition which may postpone the production of tokens and a transition can be replaced by the sub net without changing dynamic properties such as safeness and soundness. □

The result of the replacement may contain routing transitions (Section 3.3), that have become superfluous. An AND-split routing transition is superfluous when it is preceded by exactly one place and this place is preceded by exactly one transition that does not have other outputs (the two transitions are in a sequence). An AND-join routing transition is superfluous when it is followed by exactly one place and this place is followed by exactly one transition that does not have other inputs. The routing transition and the other transition may be merged (thus removing the routing transition) with the *aggregation* of transitions. This is one of the soundness preserving transformation rules described by Van der Aalst [1]. The aggregation of an AND-split and the aggregation of an AND-join is illustrated in Figure 12. In [1], Van der Aalst formalizes the opposite of aggregation, i.e., the division of one transition into two transitions. Here, we include a similar formalization of the aggregation of two transitions into one transition. We add the restriction that there is only an aggregation possible if at least one of the transitions is a routing transition.

Definition 25 (Aggregation) *Let $S = (P, T, F, D, D_I, D_O, R, A)$ be an annotated SISO-net. This net can be transformed into an annotated SISO-net $S' = (P', T', F', D, D'_I, D'_O, R, A')$ iff there exist $t_1, t_2 \in T$, $p \in P$ and $t_3 \in T'$ with*

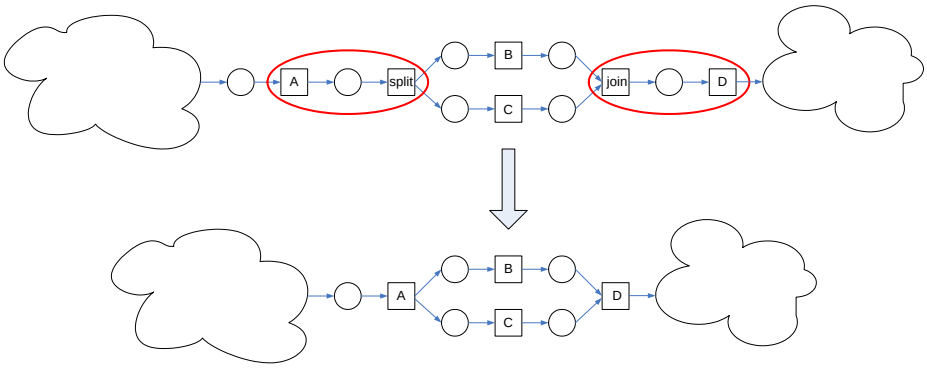

Fig. 12. The aggregation of an AND-split and the aggregation of an AND-join

$t_1\bullet = \bullet t_2 = \{p\}$, $\bullet p = \{t_1\}$, $p\bullet = \{t_2\}$ *and* $(D_I(t_1) = D_O(t_1) = \emptyset) \vee (D_I(t_2) = D_O(t_2) = \emptyset)$ *such that:*

- $P' = P \setminus \{p\}$ *is the set of places,*
- $T' = (T \setminus \{t_1, t_2\}) \cup \{t_3\}$ *is the set of transitions with* $\{t_3\} \cap T = \emptyset$*,*
- $F' = \{(x, y) \in F \mid (x \neq t_1) \wedge (x \neq t_2) \wedge (y \neq t_1) \wedge (y \neq t_2)\} \cup \{(x, t_3) \in (P' \times T') \mid (x, t_1) \in F\} \cup \{(t_3, y) \in (T' \times P') \mid (t_2, y) \in F\}$ *is the flow relation,*
- $D'_I \in T' \rightarrow \mathcal{P}(D)$ *is the set of input data elements such that:*
  - $\forall_{t \in T' \setminus \{t_3\}} : D'_I(t) = D_I(t)$*, and*
  - $D'_I(t_3) = D_I(t_1) \cup D_I(t_2)$*,*
- $D'_O \in T' \rightarrow \mathcal{P}(D)$ *is the set of output data elements such that:*
  - $\forall_{t \in T' \setminus \{t_3\}} : D'_O(t) = D_O(t)$*, and*
  - $D'_O(t_3) = D_O(t_1) \cup D_O(t_2)$*,*
- $A' \in T' \not\rightarrow R$ *is the role assignment such that* [6]*:*
  - $$dom(A') = \begin{cases} dom(A) & \text{if} \quad \{t_1, t_2\} \cap dom(A) = \emptyset, \\ (dom(A) \cap T') \cup \{t_3\} & \text{if} \quad \{t_1, t_2\} \cap dom(A) \neq \emptyset. \end{cases}$$
  - $\forall_{t \in dom(A) \setminus \{t_1, t_2\}} : A'(t) = A(t)$,
  - $$A'(t_3) = \begin{cases} A(t_1) & \text{if} \quad t_1 \in dom(A), \\ A(t_2) & \text{if} \quad t_2 \in dom(A). \end{cases}$$

It is easy to see that if we take a sound annotated SISO-net and we apply the aggregation rule, then the resulting net is still an annotated SISO-net. Moreover, the resulting annotated SISO-net is sound. Van der Aalst [1] shows that this is the case for the application of any of his soundness preserving transformation rules to a WF-net.

[6] Note that either t_1, t_2 or t_1 and t_2 are an AND-split / -join and that an AND-split or -join task does not have a role assigned to it.

Theorem 4. *The aggregation rule preserves soundness, i.e., if an annotated SISO-net is sound, then the annotated SISO-net transformed by the aggregation rule is also sound.*

Proof. See [1]. □

The aggregation of transitions has to be performed for all annotated SISO-nets present in a layered annotated SISO-net until all superfluous AND-splits and AND-joins have been removed.

In this section we addressed the creation of a redesign alternative by 1) selecting a process part, 2) transforming it into its alternative part and, 3) making a replacement. In the next sections we focus on the process transformations and describe several transformations in detail. In the next section we discuss the unfold transformation.

6 Unfold Transformation

With the unfolding of tasks, aggregated tasks (which are at the upper layer of the process) are split up into several smaller tasks. Unfolding may result in a higher run-time flexibility, because the scheduling and execution of some smaller tasks allows for more possibilities than the scheduling and execution of one large task. Furthermore, a task that is too large may be unworkable and dividing it into smaller tasks could enhance the quality realized by the involved resource(s). A drawback of smaller tasks are the longer set-up times, i.e., the time that a resource spends to become familiar with a case [34]. The unfolding of tasks is done with the unfold transformation.

We discuss the unfold transformation in the following order: First, a component is selected. Then, the operation for the unfolding of a component is presented. Next, the selected component is replaced by the alternative unfolded process part. Finally, a redesign for the blood donation process example is created with the unfold transformation.

6.1 Selection

As a starting point for the unfold transformation a component (Definition 22) is selected from a process. The selected component is the input for the redesign of a process with the unfold transformation.

6.2 Transformation

The unfold transformation is performed by replacing the aggregated transitions present in the component by their associated lower layer annotated SISO-nets. For the layered annotated SISO-net $LS_1 = (S_1, SS_1, map_1)$ the component is defined on the annotated SISO-net S_1. As a result, the aggregated transitions

are included in the component, since they are located in the upper layer of LS_1. The lower layer annotated SISO-nets are not included in the component, but can be obtained from LS_1 through the aggregated transitions. Note that a lower layer annotated SISO-net always starts and ends with a transition. This simplifies the unfolding considerably, because the lower layer net can simply replace the aggregated transition without violating the bipartite nature of an annotated SISO-net.

Definition 26 (Unfold) *Let $LS_1 = (S_1, SS_1, map_1)$ be a layered annotated SISO-net and C a component of S_1 with projection $S_1||_C$. Operation* **unfold** *changes component C into a layered annotated SISO-net*
unfold$(LS_1, C) = (S_2, SS_2, map_2)$ *with:*

- $S_2 = (P_2, T_2, F_2, D_2, D_{I_2}, D_{O_2}, R_2, A_2)$ *is an annotated SISO-net with:*
  - $P_2 = \pi_P(S_1||_C) \cup \bigcup_{t \in dom(map_1) \cap C} \pi_P(map_1(t))$ *is the set of places,*
  - $T_2 = \{t \in \pi_T(S_1||_C) \mid t \notin dom(map_1)\} \cup \bigcup_{t \in dom(map_1) \cap C} \pi_T(map_1(t))$ *is the set of transitions,*
  - $F_2 = \{(x,y) \in \pi_F(S_1||_C) \mid \{x,y\} \cap dom(map_1) = \emptyset\} \cup \bigcup_{t \in dom(map_1) \cap C} \pi_F(map_1(t)) \cup$
    $\{(p,t) \in (P_2 \times T_2) \mid \exists_{t' \in dom(map_1) \cap C} : p \in {\bullet t'} \wedge t = \mathbf{in}(map_1(t'))\} \cup$
    $\{(t,p) \in (T_2 \times P_2) \mid \exists_{t' \in dom(map_1) \cap C} : t = \mathbf{out}(map_1(t')) \wedge p \in t'{\bullet}\}$
    is the flow relation,
  - $D_2 = \pi_D(S_1||_C)$ *is a set of data elements,*
  - $D_{I_2} \in T_2 \to \mathcal{P}(D_2)$ *is the set of input data elements such that:*
    - $\forall_{t \in \pi_T(S_1||_C) \setminus dom(map_1)} : D_{I_2}(t) = \pi_{D_I}(S_1)(t)$, *and*
    - $\forall_{t' \in dom(map_1) \cap C} : \forall_{t \in \pi_T(map_1(t'))} : D_{I_2}(t) = \pi_{D_I}(map_1(t'))(t)$,
  - $D_{O_2} \in T_2 \to \mathcal{P}(D_2)$ *is the set of output data elements such that:*
    - $\forall_{t \in \pi_T(S_1||_C) \setminus dom(map_1)} : D_{O_2}(t) = \pi_{D_O}(S_1)(t)$, *and*
    - $\forall_{t' \in dom(map_1) \cap C} : \forall_{t \in \pi_T(map_1(t'))} : D_{O_2}(t) = \pi_{D_O}(map_1(t'))(t)$,
  - $R_2 = \pi_R(S_1||_C)$ *is a set of roles, and*
  - $A_2 \in T_2 \not\to R_2$ *is the role assignment such that:*
    - $dom(A_2) = dom(\pi_A(S_1||_C)) \setminus dom(map_1) \cup \bigcup_{t \in dom(map_1) \cap C} dom(\pi_A(map_1(t)))$,
    - $\forall_{t \in dom(\pi_A(S_1||_C)) \setminus dom(map_1)} : A_2(t) = \pi_A(S_1)(t)$, *and*
    - $\forall_{t' \in dom(map_1) \cap C} : \forall_{t \in \pi_T(map_1(t'))} : A_2(t) = \pi_A(map_1(t'))(t)$,

- $SS_2 = \emptyset$ *is an empty set of annotated SISO-nets, and*
- $map_2 \in T_2 \not\rightarrow SS_2$ *is the assignment function such that* $dom(map_2) = \emptyset$.

Note that the result of the unfold operation is a layered annotated SISO-net with an empty set of lower layer SISO-nets since the lower layer SISO-nets are moved to the upper layer.

With Theorem 5 we show that the resulting net is indeed a layered annotated SISO-net. Moreover, Theorem 5 shows that this layered annotated SISO-net is safe and sound.

Theorem 5. *Let* $LS_1 = (S_1, SS_1, map_1)$ *be a layered annotated SISO-net,* C *a component of* S_1 *with projection* $S_1||_C$*, and* $LS_2 = \texttt{unfold}(LS_1, C)$*, then:*

- LS_2 *is a layered annotated SISO-net.*
- *If* $S_1||_C$ *is safe and sound and the annotated SISO-nets associated with the aggregated transitions in* $S_1||_C$*, i.e., the set* $SS_C = \{map_1(t) \mid t \in dom(map_1) \cap C\}$*, are safe and sound, then* LS_2 *is safe and sound.*

Proof. First, we prove that LS_2 is a layered annotated SISO-net. When the source and sink node of $S_1||_C$ are not involved in the unfolding, i.e., these nodes are not aggregated transitions, then these nodes become the source and sink node of LS_2. If the source or sink node is an aggregated transition, then this node is replaced by the source (sink) node of the associated annotated SISO-net. Hence, no new source or sink nodes are introduced. Further, we note that all annotated SISO-nets associated with an aggregated transition are transition bordered (see Definition 17), i.e., such an annotated SISO-net starts and ends with a transition. With the replacement of an aggregated transition with a transition bordered annotated SISO-net, all nodes remain on a path from source to sink node.

Second, we prove that the layered annotated SISO-net LS_2 is safe and sound. In [4], it is shown that if a transition in a safe and sound WF-net is replaced by a safe and sound sub WF-net the resulting WF-net is safe and sound. We follow a similar line of reasoning as the proof in [4] to prove that LS_2 is a safe and sound layered annotated SISO-net. We have to prove that if $S_1||_C$ and all $s \in SS_C$ are safe and sound, LS_2 is safe and sound. LS_2 is safe because every reachable state corresponds to a combination of a safe state of $S_1||_C$ and a safe state of any of the $s \in SS_C$. The annotated SISO-net $s \in SS_C$, associated with the aggregated transition t, i.e., $map_1(t) = s$, behaves like a transition which may postpone the production of tokens for $t\bullet$. Thus, when abstracting from the internal states of s, t and s have identical behaviors. Unfolding t, i.e., replacing t with s in $S_1||_C$, does not change other parts of $S_1||_C$. So, because $S_1||_C$ and all $s \in SS_C$ are safe and sound, LS_2 is also safe and sound. □

The next step is the replacement of the selected component with the alternative process part that has been created with the unfold transformation.

6.3 Replacement

The layered annotated SISO-net LS_2 is the alternative process part that results from the application of the unfold transformation for the component C which

has been selected from the layered annotated SISO-net LS_1. The redesign is finished with the replacement of $S_1||_C$ in LS_1 with LS_2. This replacement is performed by operation `replace`(LS_1, C, LS_2) (Definition 24). In Theorem 3 it is stated that the resulting net LS_3 is a layered annotated SISO-net and that LS_3 is safe and sound if LS_1 and LS_2 are both safe and sound layered annotated SISO-nets. For this matter, we require the original process LS_1 to be a safe and sound layered annotated SISO-net. From Theorem 5 it follows that the alternative process part LS_2 is safe and sound. The resulting net LS_3 may contain routing transitions (AND-splits and -joins), that are superfluous. These routing transitions can be removed using the aggregation rule given in Definition 25. The end result is a redesign alternative for the initial process.

We illustrate the three redesign steps with the blood donation process example.

6.4 Example

Suppose that the employees working in the blood donation process (Figure 3) have indicated that the task `Test on HIV` is too large to perform as one task. Since the task is too large it seems logical to divide it in smaller pieces of work with the unfold transformation. The creation of a redesign alternative with the unfold transformation starts with the selection of a component from the blood donation process. A component including the task `Test on HIV` is selected. This selection is depicted with a circle around the selected process part in Figure 13. After the application of the unfold transformation, the lower layer net associated with the aggregated task `Test on HIV` is incorporated in the redesign alternative while the aggregated task has been removed. The resulting redesign alternative is depicted in Figure 14. The positive effect of this redesign is expected to be a higher job satisfaction for the employees and a better quality of the delivered work. Also, more run-time flexibility of the process is expected, because the scheduling and execution of several tasks offers more options to handle the work than the scheduling and execution of one task.

In the next section we present a second transformation: the parallel transformation.

7 Parallel Transformation

In a process, tasks may be executed one after another while no data dependencies exist between these tasks. Flow relations between such tasks may cause unnecessary delays in the process execution. A task may be waiting for another task to finish, while the task does not need the output data of that task for its execution. As an alternative, tasks without data dependencies between one another may be executed in parallel. Parallel tasks may be executed in any order or simultaneously. The obvious benefit would be a reduction in throughput time. A possible

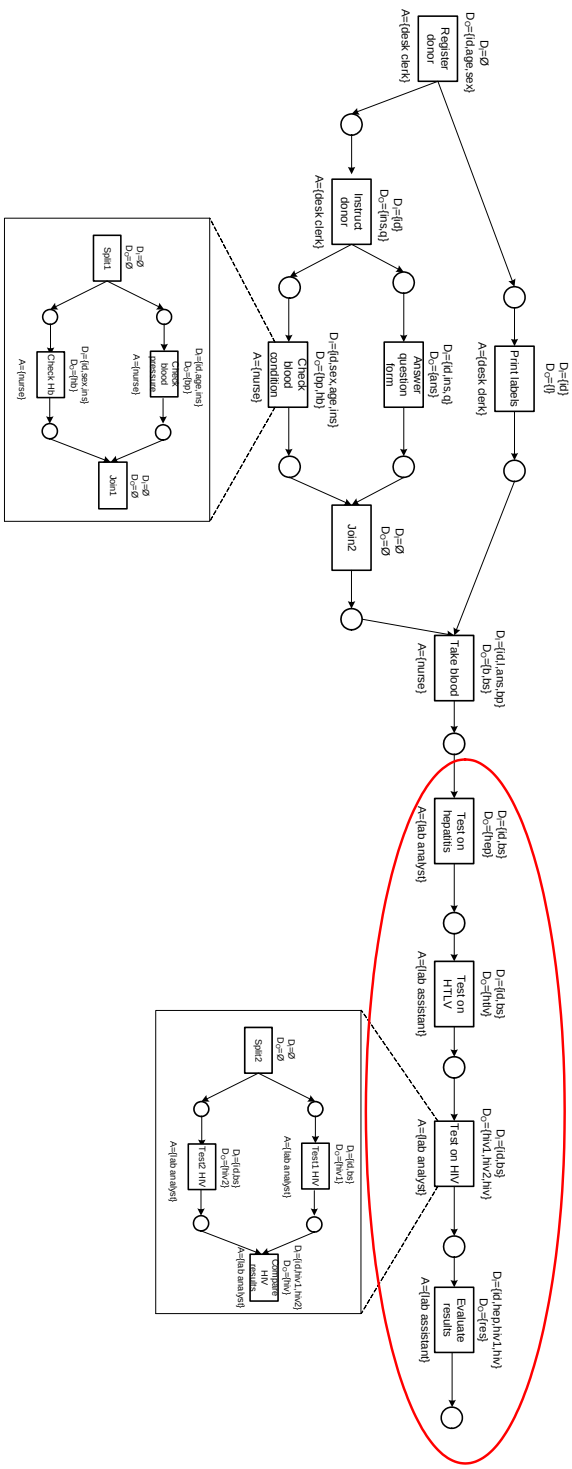

Fig. 13. Component selection

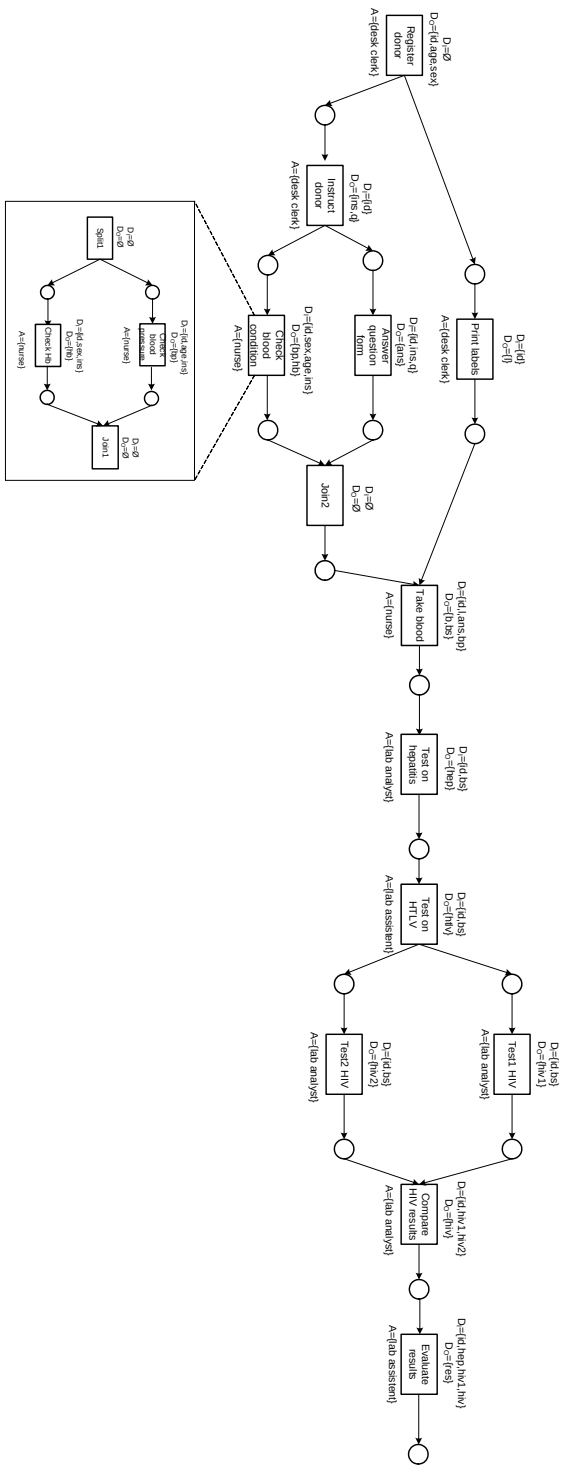

Fig. 14. Alternative redesign after unfolding

negative effect is a loss of quality because processes with concurrent behavior are more complex, which may introduce errors [34]. More parallel behavior is introduced by applying the parallel transformation.

We discuss the parallel transformation in the following order: The first sub section presents the component selection. In the second sub section, the parallel transformation and the operations used during the parallelization of a component are given. In the third sub section, the replacement of the selected component by the alternative parallel process part is presented. The final sub section handles the creation of a redesign alternative with the parallel transformation for the blood donation process example.

7.1 Selection

As a starting point for creating a redesign with the parallel transformation a component (Definition 22) is selected from a process. For the layered annotated SISO-net $LS_1 = (S_1, SS_1, map_1)$ the component is defined on the annotated SISO-net S_1. The component is explicitly defined on S_1 instead of on LS_1 to ensure that the parallel transformation is not applied on any of the nets in SS_1. The reason for leaving SS_1 untouched, is that the nets in SS_1 are likely to be constructed intentionally as they are by another transformation. Such a transformation, resulting in the addition of a net to SS_1, is, for instance, the merge transformation which is discussed in Section 9.

A potential problem when performing the parallel transformation is the parallelization of a selective routing construct (see Section 3.3). Typically, in a selective routing construct, one task out of a number of tasks is selected for execution. Putting tasks from a selective routing construct in parallel would interfere with the desired behavior. To circumvent this matter, we require the component to be a marked graph (MG). Marked graphs allow for the representation of parallel routing constructs but not of selective routing constructs. Furthermore, we require the component to be acyclic. This is necessary for the creation of sound redesign alternatives.

Definition 27 (Acyclic MG component) *C is an acyclic MG component in a layered annotated SISO-net $LS = (S, SS, map)$ if and only if:*

- *C is a component in S,*
- *$S||_C$ is a marked graph (cf. Definition 12), and*
- *$S||_C$ is acyclic (cf. Definition 7).*

With the parallel transformation we put tasks that do not depend on one another in parallel. The parallelization is achieved by first removing all relations between the tasks followed by adding only necessary relations between tasks that have a data dependency. As a result, tasks that do not depend on one another are performed in parallel.

Data elements should be distributed over the acyclic MG component in such a way that the data elements necessary for the execution of a task are available when the task becomes enabled. An intuitive distinction between input and output data elements can be made with the *operations* that a task performs on the data element. When a task *reads* a data element, the data element is input data. When a task performs a *write* operation on a data element, the data element is output data. There are two types of write operation: creating the initial value and overwriting the existing value [38]. In this paper, we only consider the creation of the initial value because it is critical for the data perspective. Since the initial value for a data element can only be created once, a data element can only be an output data element once. Furthermore, a task that creates a certain data element as its output has to be placed before the task(s) that read this data element as an input. Note that because we restrict ourselves to acyclic marked graphs it is automatically enforced that the tasks are also executed in this order.

Definition 28 (Proper component) *C is a proper component in a layered annotated SISO-net $LS = (S, SS, map)$ if and only if:*

- *C is an acyclic MG component in S,*
- $\forall_{t_1,t_2 \in \pi_T(S||_C)} : \pi_{D_O}(S||_C)(t_1) \cap \pi_{D_O}(S||_C)(t_2) \neq \emptyset \Rightarrow t_1 = t_2$,
- $\forall_{t_1,t_2 \in \pi_T(S||_C)} : \pi_{D_O}(S||_C)(t_1) \cap \pi_{D_I}(S||_C)(t_2) \neq \emptyset \Rightarrow (t_1, t_2) \in (\pi_F(S||_C))^*$, *and*
- $\forall_{t_2 \in \pi_T(S||_C)} \forall_{d \in \pi_{D_I}(S||_C)(t_2)} : \exists_{t_1 \in \pi_T(S||_C)} : (t_1, t_2) \in (\pi_F(S||_C))^* \ \wedge$ $d \in \pi_{D_O}(S||_C)(t_1)$.

As an input for the parallel transformation a proper component is selected.

7.2 Transformation

With the parallel transformation several non-dependent tasks, i.e., tasks with a disjoint set of data elements, are placed in parallel. The parallel transformation starts with the translation of a proper component into a graph. For reasons of efficiency the focus during the transformation is on the *graph structure*, i.e., places are disregarded because of the absence of selective routing constructs. The result of the parallel transformation is an annotated SISO-net with the maximum parallel process structure.

As the first step of the transformation, the transitions of a component are translated to the nodes of a graph. While translating, transitions that do not have any output data elements, i.e. for which $D_O(t) = \emptyset$, are removed because the execution of such a transition does not provide any value for the process. Note that this means that routing transitions are also removed. Edges between the nodes reflect the data dependencies.

Definition 29 (Parallel) *Let $LS = (S, SS, map)$ be a layered annotated SISO-net and C a proper component of S. Operation* `parallel` *changes C into the graph* `parallel`$(LS, C) = (N, E)$ *with:*

- $N = \{n \in \pi_T(S||_C) \mid \pi_{D_O}(S)(n) \neq \emptyset\}$ *is the set of nodes,*
- $E = \{(n_1, n_2) \in N \times N \mid (\pi_{D_O}(S)(n_1) \cap \pi_{D_I}(S)(n_2)) \neq \emptyset\}$ *is the set of edges.*

Note that the graph resulting from Definition 29 may contain only one node. From here on, we only consider non-trivial graphs, i.e., graphs with more than one node. The graph has to be further transformed into a SISO-net. A SISO-net has one source node and one sink node, while the graph may have multiple start or end nodes. The operations `start` and `end` return the start and end nodes of a graph.

Definition 30 (Start, end) *Operation* $\texttt{start}(G)$ *and operation* $\texttt{end}(G)$ *respectively return the start and end nodes of graph* $G = (N, E)$*. Formally:*

- $\texttt{start}\ (G) = \{n \in N \mid \forall_{n' \in N} : (n', n) \notin E\}$*, and*
- $\texttt{end}\ (G) = \{n \in N \mid \forall_{n' \in N} : (n, n') \notin E\}$.

Using the `start` and `end` operations, it is possible to determine the number of start and end nodes of a graph. Definition 31 transforms a graph into a graph with a single start node and a single end node.

Definition 31 (Siso) *Operation* `siso` *changes graph* $G = (N, E)$ *into the graph* $\texttt{siso}(G) = (N', E')$ *with* [7]*:*

$$N' = N \ \cup \begin{cases} \{i_{split}\} & \text{if} \quad |\texttt{start}(G)| > 1, \\ \{o_{join}\} & \text{if} \quad |\texttt{end}(G)| > 1, \end{cases}$$

$$E' = E \cup \begin{cases} \{(i_{split}, n) \in (N' \times N') \mid n \in \texttt{start}(G)\} & \text{if} \quad |\texttt{start}(G)| > 1, \\ \{(n, o_{join}) \in (N' \times N') \mid n \in \texttt{end}(G)\} & \text{if} \quad |\texttt{end}(G)| > 1. \end{cases}$$

After the execution of operation `siso` the resulting graph starts and ends with one single node.

With the performance of the `parallel` operation more relations may be introduced between the nodes of the graph then strictly necessary. When the paths (a,b), (b,c) and (a,c) exist in a graph, the direct path (a,c) becomes superfluous. Operation `reduce` removes all superfluous relations from a graph (i.e., performs a transitive reduction).

Definition 32 (Reduce) *Graph* $G' = (N, E')$ *is the reduced graph of* $G = (N, E)$ [8]*, if and only if:*

- $(E')^* = E^*$,
- $\forall_{(n_1,n_2) \in E'} : (E' \setminus \{(n_1, n_2)\})^* \neq E^*$*, and*
- $E' \subseteq E$.

[7] i_{split} and o_{join} are two "fresh" identifiers, i.e., $\{i_{split}, o_{join}\} \cap N = \emptyset$.

[8] The transitive reduction of an acyclic graph results in one unique reduced graph [12].

Operation $\texttt{reduce}(G)$ *gives the reduced graph* G'*.*

The graph that results from the application of the operations `parallel`, `siso` and `reduce` on the component C has certain properties.

Theorem 6. *Let* $LS = (S, SS, map)$ *be a layered annotated SISO-net,* C *a proper component of* S*, and* $G = \texttt{reduce}(\texttt{siso}(\texttt{parallel}(LS, C)))$*, then:*

- $\texttt{start}(G)$ *is a singleton.*
- $\texttt{end}(G)$ *is a singleton.*
- *All nodes in* G *are on a path from* $\texttt{start}(G)$ *to* $\texttt{end}(G)$*.*
- G *is acyclic.*

Proof. First, we prove that $\texttt{start}(G)$ is a singleton. Operation `siso` adds a source node when the graph resulting from $\texttt{parallel}(LS, C)$ has multiple start nodes. These start nodes are connected to the added source node and become its successors. The added source node is the only source node in the resulting graph. Operation `reduce` only removes superfluous relations keeping the source node in place. If the graph resulting from $\texttt{parallel}(LS, C)$ already has a single start node this remains the only source node. A similar argument can be given to show that $\texttt{end}(G)$ is a singleton.
Second, we prove that all nodes in G are on a path from $\texttt{start}(G)$ to $\texttt{end}(G)$. Assume there is a node x in G that is not on a path from $\texttt{start}(G)$ to $\texttt{end}(G)$. Then, this node or one of its predecessors is a start node of G. Since there is also the source node defined by $\texttt{start}(G)$, there must be multiple start nodes in G. However, this is not possible because with operation `siso` we made sure there is one unique source node defined by $\texttt{start}(G)$. So, there must be a path from $\texttt{start}(G)$ to node x. With operation `reduce` relations are removed, but a relation between two nodes is only removed if there remains a path between the nodes. A similar argument can be given to show that there is a path from node x to $\texttt{end}(G)$. So, all nodes in G are on a path from $\texttt{start}(G)$ to $\texttt{end}(G)$.
Finally, we prove that G is acyclic. The input for operation `parallel` is a proper component C, i.e., C is an acyclic marked graph and for all tasks in C it holds that if the task creates a certain data element as its output, it is placed before the task(s) that requires this data element as an input. Since operation `parallel` only adds a relation between two tasks that have a data dependency, no cycles are introduced. Operation `siso` adds relations between the source (sink) node and the start (end) nodes. This does not introduce cycles. Operation `reduce` only removes relations. So, G is acyclic. □

The graph that results from the application of the operations `parallel`, `siso` and `reduce` on the component C is transformed into a SISO-net by translating all nodes in the graph to transitions. Places and corresponding flow relations are added between these transitions to create a SISO-net. The resulting SISO-net starts and ends with a transition while the component may start and/or end with a place. For the replacement of the component by the alternative SISO-net, the SISO-net should start and end with the same type of node as the component.

If the source node of the component is a place, an additional place, called i_S, has to be added and connected to the start node of the SISO-net. If the sink node of the component is a place, an additional place, called o_S, has to be added and connected to the end node of the SISO-net. In Definition 33 this translation from graph to SISO-net is given.

Definition 33 (Net) *Let $LS = (S, SS, map)$ be a layered annotated SISO-net, C a proper component of S with a single input node i_C and a single output node o_C and $G = (N, E) = \texttt{reduce}(\texttt{siso}(\texttt{parallel}(LS, C)))$. Operation $\texttt{net}$ translates G into a SISO-net $\texttt{net}(G) = (P, T, F)$, with i_S and o_S as two "fresh" identifiers, i.e., $\{i_S, o_S\} \cap N = \emptyset$, and:*

$P = \{p_{(n,n')} \mid (n, n') \in E\} \cup$

$$\begin{cases} \emptyset & \text{if} \quad i_C \notin \pi_P(S||_C) \wedge o_C \notin \pi_P(S||_C), \\ \{i_S\} & \text{if} \quad i_C \in \pi_P(S||_C) \wedge o_C \notin \pi_P(S||_C), \\ \{o_S\} & \text{if} \quad i_C \notin \pi_P(S||_C) \wedge o_C \in \pi_P(S||_C), \\ \{i_S, o_S\} & \text{if} \quad i_C \in \pi_P(S||_C) \wedge o_C \in \pi_P(S||_C), \end{cases}$$

$T = N,$

$F = \{(n, p_{(n,n')}) \mid (n, n') \in E\} \ \cup \ \{(p_{(n,n')}, n') \mid (n, n') \in E\} \cup$

$$\begin{cases} \emptyset & \text{if} \quad i_C \notin \pi_P(S||_C) \wedge o_C \notin \pi_P(S||_C), \\ \{(i_S, \texttt{start}(G))\} & \text{if} \quad i_C \in \pi_P(S||_C) \wedge o_C \notin \pi_P(S||_C), \\ \{(\texttt{end}(G), o_S)\} & \text{if} \quad i_C \notin \pi_P(S||_C) \wedge o_C \in \pi_P(S||_C), \\ \{(i_S, \texttt{start}(G)), (\texttt{end}(G), o_S)\} & \text{if} \quad i_C \in \pi_P(S||_C) \wedge o_C \in \pi_P(S||_C). \end{cases}$$

As a final step, the annotation is restored. With operation `annotate` the transitions in the resulting net that are also in the original component are annotated with the original annotation. Note that operation `siso` may have added routing transitions. Such a routing transition t is annotated with $D_I(t) = \emptyset$, $D_O(t) = \emptyset$ and has no role assignment.

Definition 34 (Annotate) *Let $LS = (S, SS, map)$ be a layered annotated SISO-net, C a proper component of S, and $(P, T, F) = \texttt{net}(\texttt{reduce}(\texttt{siso}(\texttt{parallel}(LS, C))))$. Operation $\texttt{annotate}$ changes (P, T, F) into an annotated SISO-net $(P, T, F, D, D_I, D_O, R, A)$ with:*

- *$D = \pi_D(S||_C)$ is the set of data elements,*
- *$D_I \in T \to \mathcal{P}(D)$ is the set of input data elements such that:*
  - *$\forall_{t \in T \cap \pi_T(S||_C)} : D_I(t) = \pi_{D_I}(S)(t)$, and*
  - *$\forall_{t \in T \setminus \pi_T(S||_C)} : D_I(t) = \emptyset$,*
- *$D_O \in T \to \mathcal{P}(D)$ is the set of output data elements such that:*
  - *$\forall_{t \in T \cap \pi_T(S||_C)} : D_O(t) = \pi_{D_O}(S)(t)$, and*
  - *$\forall_{t \in T \setminus \pi_T(S||_C)} : D_O(t) = \emptyset$,*
- *$R = \pi_R(S||_C)$ is the set of roles, and*

– $A \in T \not\rightarrow R$ *is the role assignment such that:*
  - $dom(A) = dom(\pi_A(S||_C))$*, and*
  - $\forall_{dom(A)} : A(t) = \pi_A(S)(t)$.

With Theorem 7 we show that the net resulting from the operations `parallel`, `siso`, `reduce`, `net`, and `annotate` is indeed an annotated SISO-net. Moreover, Theorem 7 shows that this annotated SISO-net is safe and sound. The result is used, later on, to show that the replacement of the component (selected from a layered annotated SISO-net) by the net results in a layered annotated SISO-net which is safe and sound.

Theorem 7. *Let* $LS = (S, SS, map)$ *be a layered annotated SISO-net and* C *a proper component of* S*. The annotated SISO-net* $S' = \texttt{annotate}(\texttt{net}(\texttt{reduce}(\texttt{siso}(\texttt{parallel}(LS, C)))))$ *is a safe and sound annotated SISO-net.*

Proof. First, we prove that S' is an annotated SISO-net. S' is the result of the translation of the graph, resulting from the operations `reduce`, `siso`, and `parallel` performed on C, to an annotated SISO-net. In this translation, only nodes are added between the nodes of the graph. There is just one source node and just one sink node in the graph (conform Theorem 6) and no new source and sink nodes are introduced. All nodes are on a path from the source node to the sink node in the resulting annotated SISO-net. This is proven for the resulting graph in Theorem 6. With the translation of nodes to transitions and the addition of places between transitions the paths remain the same. So, all nodes are on a path from source node to sink node. Hence, S' is an annotated SISO-net.

Second, we prove that the annotated SISO-net S' is safe and sound. Assume the source node $\texttt{in}(S')$ and the sink node $\texttt{out}(S')$ are places and $\texttt{in}(S')$ is initially marked with one token. For safeness the maximum number of tokens in a place does not exceed 1. Each place in S' has one ingoing and one outgoing transition, so only one transition can consume the token present in $\texttt{in}(S')$ and produce one token in its output place. The net is acyclic since the graph is acyclic (conform Theorem 6), so transitions can only fire once and thus S' is safe. For the proof of soundness, we create the short-circuited net $\overline{S'}$ by adding transition t^*. Transition t^* connects the sink place, $\texttt{out}(S')$, of S' with the source place, $\texttt{in}(S')$, of S'. S' is sound if and only if the short-circuited system with initial marking $\texttt{in}(S')$, $(\overline{S'},\texttt{in}(S'))$, is live and bounded (Theorem 1). In a safe net all places are bounded to 1, so $(\overline{S'},\texttt{in}(S'))$ is bounded. Remains to prove that $(\overline{S'},\texttt{in}(S'))$ is live. Each place in $\overline{S'}$ has at most one ingoing and one outgoing transition, so $\overline{S'}$ is a marked graph. It follows that the short-circuited system $(\overline{S'},\texttt{in}(S'))$ is a T-system. The annotated SISO-net S' is acyclic, so there are no circuits in S' and all nodes in S' are on a path from $\texttt{in}(S')$ to $\texttt{out}(S')$. From this, it follows that the short-circuited system contains one circuit that includes all transitions. For the proof of liveness we use Theorem 3.15 from [19]: a T-system is live iff every circuit is initially marked. $(\overline{S'},\texttt{in}(S'))$ contains one circuit and this circuit is initially marked. So, $(\overline{S'},\texttt{in}(S'))$ is live. The same arguments apply if $\texttt{in}(S')$ and $\texttt{out}(S')$ are not places. Hence, S' is a safe and sound annotated SISO-net. □

The next step is the replacement of the selected component with the alternative process part that has been created with the parallel transformation.

7.3 Replacement

The output of the parallel transformation is an annotated SISO-net. As a last step, before it is possible to perform the replacement, a layered annotated SISO-net has to be created from this output. The annotated SISO-net contains the transitions from the proper component, only the number of routing transitions may have changed. So, all aggregated transitions are still present in the annotated SISO-net and operation `layer` relates the aggregated transitions to the lower layer annotated SISO-nets.

Definition 35 (Layer) *Let $LS_1 = (S_1, SS_1, map_1)$ be a layered annotated SISO-net and C a proper component of S_1. Operation* `layer` *translates the annotated SISO-net $S_2 =$* `annotate`$(\texttt{net}(\texttt{reduce}(\texttt{siso}(\texttt{parallel}(LS_1, C)))))$*, into a layered annotated SISO-net* `layer`$(S_1) = (S_2, SS_2, map_2)$ *with:*

- $SS_2 = \{map(t) \mid t \in dom(map_1) \cap C\}$ *is a set of annotated SISO-nets,*
- $map_2 \in \pi_T(S_2) \not\rightarrow SS_2$ *is the assignment function such that:*
  - $dom(map_2) = dom(map_1) \cap C$*, and*
  - $\forall_{t \in dom(map_2)} : map_2(t) = map_1(t)$.

The layered annotated SISO-net LS_2, the result of `layer(annotate(net(reduce(siso(parallel(`LS, C`))))))`, is the alternative process part that has to replace the component C which has been selected from the layered annotated SISO-net LS_1. The redesign is finished with the replacement of $S_1||_C$ in LS_1 with LS_2. This replacement is performed by operation `replace`(LS_1, C, LS_2) (Definition 24). In Theorem 3 it is stated that the resulting net LS_3 is a layered annotated SISO-net and that LS_3 is safe and sound if LS_1 and LS_2 are both safe and sound layered annotated SISO-nets. For this matter, we require the original process LS_1 to be a safe and sound layered annotated SISO-net. From Theorem 7 it follows that the alternative process part LS_2 is safe and sound. The resulting net LS_3 may contain routing transitions (AND-splits and -joins), that are superfluous. These routing transitions can be removed using the aggregation rule given in Definition 25. The end result is a redesign alternative for the initial process.

We illustrate the three redesign steps with the blood donation process example.

7.4 Example

Suppose that the blood donation center, that is responsible for the execution of the blood donation process (Figure 3), wants to shorten the throughput time of the blood donation process. A reduction in throughput time is one of the

benefits of the application of the parallel transformation. The creation of a redesign alternative with the parallel transformation starts with the selection of a component from the blood donation process. In the current process (Figure 3), the lab tests are performed one after another. Therefore, the process part that handles the tests is selected and this selection is depicted with a circle around the selected process part in Figure 15. The redesign alternative is depicted in Figure 16. After the application of the parallel transformation, the three tests are placed in parallel and can be executed simultaneously or in any order. Next to the expected reduction of the throughput time, the additional benefit would be that the involved employees would have the freedom to perform the tests in the order they prefer.

In the next section we present the sequence transformation.

8 Sequence Transformation

Employees and clients may perceive a sequential process as a simpler process than processes with concurrent behavior, because the order of the tasks is fixed. Also, the synchronization that is required after the parallel execution of tasks is not necessary in sequential processes yielding a higher quality of the output [34]. A sequence of tasks is created by applying the sequence transformation. With this transformation it is not intended to create a sequential process with the "best" ordering of the tasks because what is "best" depends on the redesign goal. Sometimes it is better postpone a task if the result is not required for immediately following tasks. Perhaps its execution may not be necessary, which saves costs. Other times, tasks check various conditions that must be satisfied to deliver a positive end result. When the task with the highest chance on rejecting the case and the lowest effort is performed first, costs are saved because not all checks have to be performed for all cases. In another situation, a task may be moved close to a similar task to diminish set-up times [34].

We discuss the sequence transformation in the following order: First, a component is selected. Then, the sequence transformation and the operations used when sequencing a component are presented. Next, the selected component is replaced by the alternative sequential process part. Finally, a redesign for the blood donation process example is created with the sequence transformation.

8.1 Selection

As a starting point for creating a redesign with the sequence transformation a component (Definition 22) is selected from a process. Remember that a component only encloses part of the upper layer of a process without considering the lower layer nets.

A potential problem when performing the sequence transformation is the presence of a selective routing construct (see Section 3.3). Tasks that are placed

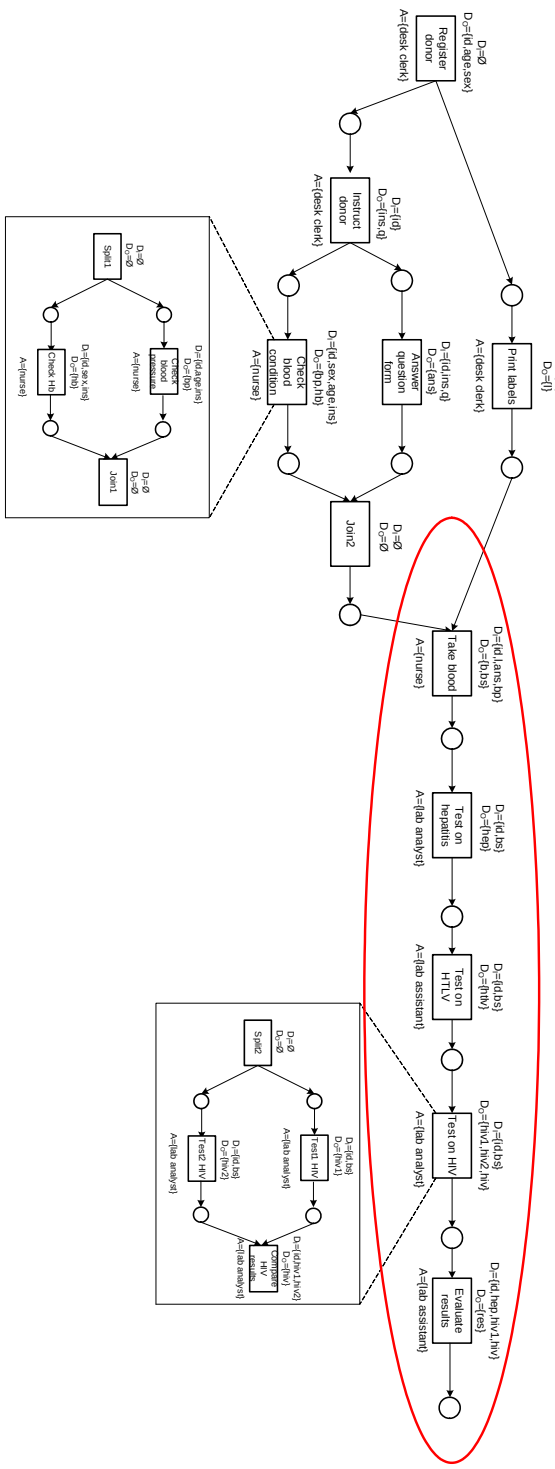

Fig. 15. Component selection

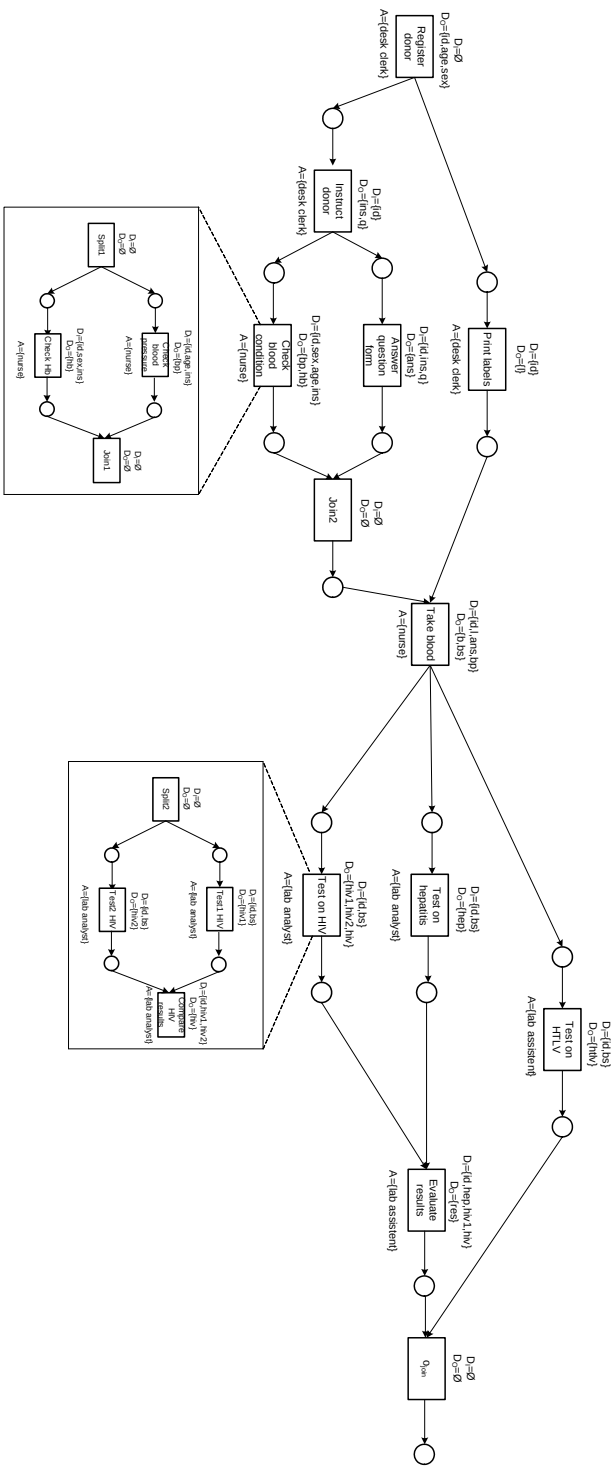

Fig. 16. Alternative redesign after parallelization

in a sequence are all performed, while in a selective routing construct only a selection of the tasks is performed. For this reason we omit any selective routing constructs from the input and select only marked graph (MG) components. We also require the component to be acyclic. This is necessary for the creation of sound redesign alternatives. See Definition 27 for the definition of an acyclic MG component.

The selected acyclic MG component is the input for the redesign of a process with the sequence transformation.

8.2 Sequence transformation

An acyclic MG component is the input for the sequence transformation and an annotated SISO-net is its output. During the transformation the focus is on the graph structure for reasons of efficiency, i.e., places are disregarded. The result of the sequence transformation is an annotated SISO-net with all the tasks in the net placed in a sequence.

As a preparatory step, the component C is translated into a graph. To this end, the transitions of the component are translated to the nodes of the graph, but transitions that do not have any output data elements are removed altogether. Flow relations are translated to edges between nodes that are directly connected (through a place) in the component. Furthermore, edges are added between nodes that are connected in the component through transitions that do not have any output data elements (and the corresponding places).

Definition 36 (Graph) *Let $LS = (S, SS, map)$ be a layered annotated SISO-net and C an acyclic MG component of S. Operation* `graph` *translates C into a graph* $\texttt{graph}(LS, C) = (N, E)$ *with:*

- $N = \{n \in \pi_T(S||_C) \mid \pi_{D_O}(S)(n) \neq \emptyset\}$ *is the set of nodes,*
- $E = \{(n_1, n_2) \in (N \times N) \mid (n_1, n_2) \in (\pi_F(S))^*\}$ [9] *is the set of edges.*

Note that the graph resulting from the application of operation `graph` to the component may contain only one node. We only consider non-trivial graphs, i.e., graphs with more than one node. The resulting graph potentially includes many redundant relations between the nodes. This is not an issue, because the graph is only used as an input for the `sequence` operation. Using operation `sequence` a sequence is created from the graph. First, all possible sequential graphs are determined. Some of these graphs have orderings that result in a reduction of time, others may reduce the costs. For the decision which graph to take we use an injective function that assigns a unique number to each of the possible graphs. Then, the sequential graph with the lowest number is selected.

[9] An edge is added between a pair of nodes if there is a (possibly empty) path from the first node to the second node in the original net.

Definition 37 (Sequence) *Let $LS = (S, SS, map)$ be a layered annotated SISO-net, C an acyclic MG component of S and $G = (N, E) =$ `graph`(LS, C). Operation* `sequence` *is a function such that a set of graphs SG is derived from the graph G followed by the selection of one graph $G' = (N, E')$ from SG. Let $f : SG \to \mathbb{N}$ be an injective function. Then:*

- *SG is the set of graphs such that $(N, E'') \in SG$ if and only if:*
  - *$E'' \subseteq N \times N$,*
  - *$|E''| = |N| - 1$,*
  - *$E \subseteq (E'')^*$,*
  - *(N, E'') is totally connected,*
- *$G' = (N, E')$ is the graph selected from SG such that:*
  - *$G' \in SG$, and*
  - *$\forall_{G'' \in SG} : f(G') \leq f(G'')$.*

Operation `sequence`(G) *returns the graph G'.*

With Theorem 8 we show that operation `sequence` always returns a graph.

Theorem 8. *Let $LS = (S, SS, map)$ be a layered annotated SISO-net, C an acyclic MG component of S, and $G =$ `graph`(LS, C). Then:* `sequence`(G) *exists.*

Proof. SG, the largest set of graphs derived from G, is the intermediate result used to determine `sequence`(G). We have to prove that SG contains at least one graph, say G'. SG is derived from G, the result of `graph`(LS, C). It follows from Definition 36 that G contains at least two transitions. Assume $G = (N, E)$ with $N = \{n_1, n_2\}$, i.e., we assume that there are only two nodes. From the first requirement stated in Definition 37, it follows that $E' = \{(n_1, n_2)\}$ or $E' = \{(n_2, n_1)\}$. The second requirement states that if $(n_1, n_2) \in E$ then $E' = \{(n_1, n_2)\}$ and if $(n_2, n_1) \in E$ then $E' = \{(n_2, n_1)\}$. So, if the two nodes are ordered in G then SG contains one graph. Otherwise, there are two possible orderings of the nodes and SG contains two graphs. The resulting graph(s) is/are totally connected, i.e., requirement three is also fulfilled. Hence, SG contains at least one graph when `graph`(LS, C) contains two nodes. The same line of reasoning can be followed to prove that SG contains at least one graph starting with a graph `graph`(LS, C) containing more than two nodes. □

The graph that results from the application of the operations `graph` and `sequence` [10] on the component C has certain properties.

Theorem 9. *Let $LS = (S, SS, map)$ be a layered annotated SISO-net, C an acyclic MG component of S, and $G =$ `sequence`(`graph`(LS, C)).*
Then:

- `start`(G) *is a singleton.*

[10] Note that the operations `siso` and `reduce` are not applied. The operation `siso` (Definition 31) is not performed, because the graph is totally connected. The operation `reduce` is not performed, because a sequence has no superfluous relations.

- *end*(G) *is a singleton.*
- *All nodes in* G *are on a path from* *start*(G) *to* *end*(G)*.*
- G *is acyclic.*

Proof. First, we prove that `start`(G) is a singleton. All nodes present in `graph`(LS, C) are totally connected by the `sequence` operation. The resulting sequence starts with one node, which is denoted as `start`(G), and this is the only (unique) source node. A similar argument can be given to show that `end`(G) is a singleton.
Then, we prove that all nodes in G are on a path from `start`(G) to `end`(G). Assume there is a node x in G that is not on a path from `start`(G) to `end`(G). Then, this node or one of its predecessors is a start node of G. Since there is also the source node defined by `start`(G), there must be multiple start nodes in G. However, this is not possible because with operation `sequence` we made sure all nodes are totally connected. So, there must be a path from `start`(G) to node x. A similar argument can be given to show that there is a path from node x to `end`(G). So, all nodes in G are on a path from `start`(G) to `end`(G).
Finally, we have to prove that G is acyclic. One of the requirements in operation `sequence` is that the resulting graph is acyclic. □

Then, the graph resulting from the operations `graph` and `sequence` is transformed into an annotated SISO-net with operation `net` (Definition 33) and operation `annotate` (Definition 34). With Theorem 10 we show that the net resulting from these operations is indeed an annotated SISO-net. Moreover, Theorem 10 shows that this annotated SISO-net is safe and sound. The result is used, later on, to show that the replacement of the component (selected from a layered annotated SISO-net) by the net results in a layered annotated SISO-net which is safe and sound.

Theorem 10. *Let* $LS = (S, SS, map)$ *be a layered annotated SISO-net and* C *an acyclic MG component of* S*. The annotated SISO-net* $S' =$ *annotate*(*net* (*sequence*(*graph*(LS, C)))) *is a safe and sound annotated SISO-net.*

Proof. The proof that S' is a safe and sound SISO-net is similar to the proof provided for Theorem 7. □

The next step is the replacement of the selected component with the alternative process part that has been created with the sequence transformation.

8.3 Replacement

The output of the sequence transformation is an annotated SISO-net. As a last step, before it is possible to perform the replacement, a layered annotated SISO-net has to be created from this output. This is done with an operation similar to operation `layer` (Definition 35). Only this time, the input parameter S_2 in Definition 35 is changed to $S_2 =$`annotate`(`net` (`sequence`(`graph`(LS, C)))).

The resulting layered annotated SISO-net, called LS_2, is the alternative process part that has to replace the component C which has been selected from the layered annotated SISO-net LS_1. The redesign is finished with the replacement of $S_1||_C$ in LS_1 with LS_2. This replacement is performed by operation $\texttt{replace}(LS_1, C, LS_2)$ (Definition 24). In Theorem 3 it is stated that the resulting net LS_3 is a layered annotated SISO-net and that LS_3 is safe and sound if LS_1 and LS_2 are both safe and sound layered annotated SISO-nets. For this matter, we require the original process LS_1 to be a safe and sound layered annotated SISO-net. From Theorem 10 it follows that the alternative process part LS_2 is safe and sound. The resulting net LS_3 may contain routing transitions (AND-splits and -joins), that are superfluous. These routing transitions can be removed using the aggregation rule given in Definition 25. The end result is a redesign alternative for the initial process.

We illustrate the three redesign steps with the blood donation process example.

8.4 Example

Suppose that the donation center, that is responsible for the execution of the blood donation process (Figure 3), has received complaints from the donors about the instruction. At the instruction step in the process, the donor is instructed on the next tasks which are placed in parallel and can be executed in a random order. A solution may be to perform the involved task in a fixed order, i.e., apply the sequence transformation. For the donor it is also important that the throughput time of the donation process is as short as possible. Therefore, the result of the sequence transformation has to be the sequence with the minimum throughput time. The creation of a redesign alternative with the sequence transformation starts with the selection of a component from the blood donation process. A process part, including the instruction and the other involved tasks, is selected and this selection is depicted with a circle around the selected process part in Figure 17. Before the sequence transformation is performed, the redesigner unfolds the selected component with the unfold operation (Definition 26). The unfolded process part is depicted in Figure 18. In general, the expected positive effect of the sequence transformation is a higher quality as perceived by the donor because (s)he is instructed with the specific order of the tasks. Next to this, the synchronization task `Join2` becomes unnecessary and together with this task any time, costs or errors caused by the synchronization are removed. More specific is the redesign goal to obtain the sequence with the lowest throughput time. Such is sequence is created with a function that selects the sequence with the minimum throughput time when performing operation `sequence`. From Figure 18 it follows that the task `Instruct donor` has to be placed before the other tasks. After the instruction, the donor has to wait for the availability of the nurse for the tasks `Check blood pressure` and `Check Hb`. The donor could use this time to fill out the questionnaire, so task `Answer question form` is placed after `Instruct donor`. The tasks `Check blood pressure` and `Check Hb`

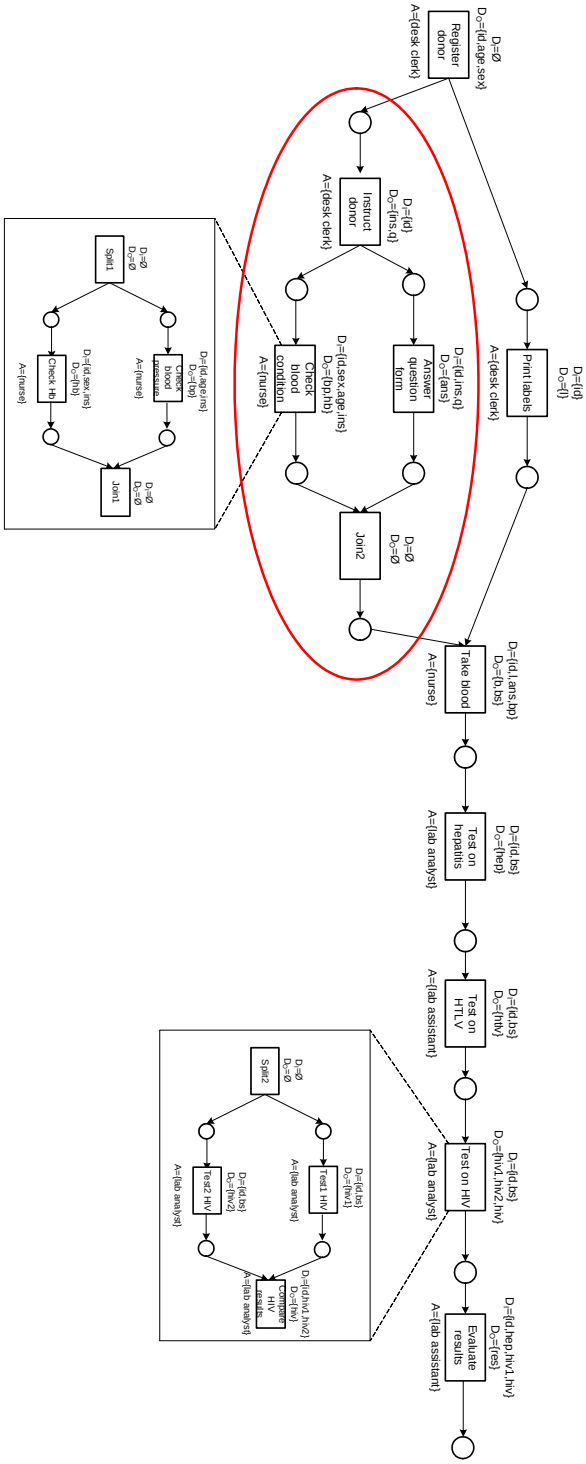

Fig. 17. Component selection

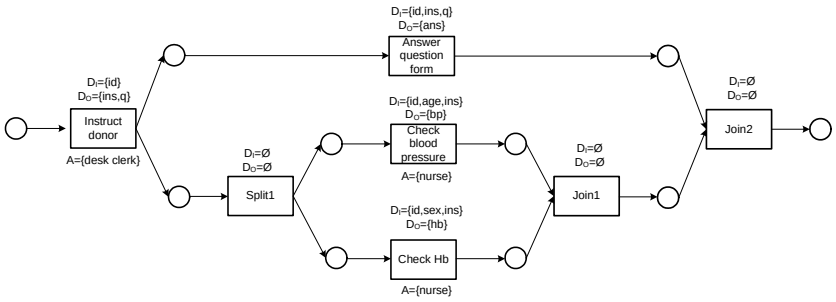

Fig. 18. Unfolded process part

are placed together because they are both performed by the same role. When a nurse performs both tasks for the same donor at once this will save setup time. The redesign alternative resulting from the application of the sequence transformation with the minimum throughput time function is depicted in Figure 19.

In the next section we present the merge transformation.

9 Merge Transformation

With the merging of tasks, multiple tasks are combined into one aggregated task. Combining tasks may result in the reduction of setup times, i.e., only one resource has to get acquainted with the case before the execution of the aggregated task. An additional incentive to merge tasks is the expected positive effect on the quality of the delivered work due to fewer hand-overs of the work between resources [34]. An aggregated task is introduced by applying the merge transformation.

We discuss the merge transformation in the following order: First, the selection of a component and the selection of several tasks from the component by means of a task cluster is discussed. Then, the merge transformation and the operations used during the merge are presented. Next, the selected component is replaced by the alternative merged process part. Finally, a redesign for the blood donation process example is created with the merge transformation.

9.1 Selection

As a starting point for the merge transformation, a component is selected from a process. A potential problem when performing the merge transformation is the presence of a selective routing construct (see Section 3.3). If only part of the selective routing construct is included in the combination of tasks, then the behavior of the process is changed and it is no longer possible to make the intended selection. For this reason we omit any selective routing constructs from the input and select only marked graph (MG) components. We also require the

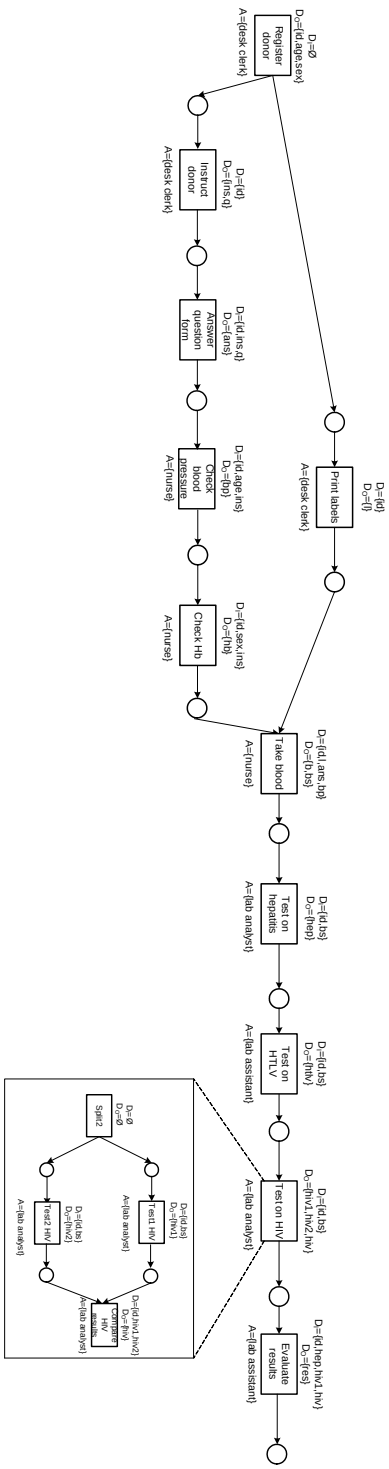

Fig. 19. Alternative redesign after sequencing

component to be acyclic. This is necessary for the creation of sound redesign alternatives. See Definition 27 for the definition of an acyclic MG component.

Unlike the other transformations, the merge transformation is not applied on the component. Suppose we want to apply the merge transformation to the blood donation process, see Figure 3. It is not possible to select a component from the process where all tasks in the component have the same role. Nevertheless, there is a group of tasks with the same role, namely the tasks `Register donor`, `Instruct donor`, and `Print labels`, that could be merged. For a wider application of the merge transformation, we select a group of tasks from the selected component. We introduce the notion of a task cluster to describe the group of tasks that may be combined into an aggregated task. The merge transformation replaces the task cluster with an aggregated task making the task cluster a sub net of the aggregated task.

Before we present the definition of the task cluster, we focus on the requirements that the task cluster needs to fulfill. A task cluster is a Petri net containing a sub set of the component nodes. All tasks in the task cluster have the same role. All tasks in the task cluster need to be weakly connected (Definition 4) to create a meaningful sub net and to integrate the aggregated task in a correct way. The task cluster does not contain any aggregated tasks to keep the two layered-process structure. The task cluster starts and ends with a task to make a straight forward connection between the aggregated task, that replaces the selected task cluster, and the places surrounding the task cluster. The task cluster has a set of input transitions that receive their inputs from one or more places outside the task cluster. Similarly, there exists a set of output transitions. All input transitions are located before any of the output transitions to prevent the introduction of loops. In Definition 38, the task cluster is defined according to the described requirements.

Definition 38 (Task cluster) *Let $LS = (S, SS, map)$ be a layered annotated SISO-net and C an acyclic MG component in S. The Petri net (P, T, F) is a task cluster in C if and only if there exists a $C' \subseteq C$ such that:*

- $P = \pi_P(S) \cap C'$,
- $T = \pi_T(S) \cap C'$,
- $F = \pi_F(S) \cap (C' \times C')$,
- $\forall_{(x,y)\in\pi_F(S)} : (x \in P \Rightarrow y \in T) \wedge (y \in P \Rightarrow x \in T)$,
- $\forall_{t_1,t_2\in T\cap dom(\pi_A(S))} : \pi_A(S)(t_1) = \pi_A(S)(t_2)$,
- (P, T, F) *is weakly connected,*
- $T \cap dom(map) = \emptyset$,
- $(\bullet P \cup P\bullet) \subseteq T$,
- $T_i = \{t \in T \mid \exists_{p\in\pi_P(S)\setminus P} : (p, t) \in \pi_F(S)\}$,
- $T_o = \{t \in T \mid \exists_{p\in\pi_P(S)\setminus P} : (t, p) \in \pi_F(S)\}$, *and*
- $\forall_{t_1\in T_i}\forall_{t_2\in T_o} : (t_2, t_1) \in F^* \Rightarrow t_1 = t_2$.

Note that a task cluster may contain only one node. We only consider non-trivial task clusters, i.e., task clusters with more than one node. An acyclic MG

component including one non-trivial task cluster is the input for a redesign with the merge transformation.

9.2 Transformation

With the merge transformation, the task cluster is removed from the component and replaced by an (aggregated) transition. The task cluster, then, becomes the sub net describing the execution of the aggregated transition in more detail. Since the task cluster is not an annotated SISO-net, it should be transformed into an annotated SISO-net before it can be incorporated as a sub net. The task cluster may have multiple start or end nodes, so the first step is to transform the task cluster into a SISO-net. To this end, we define the operation `sisoPN` that performs a similar operation as operation `siso` (Definition 31). Since we may have to add places and transitions, instead of nodes, operation `siso` cannot be applied. Note, however, that Petri nets are bipartite graphs, so the operations `start` and `end` can also be applied to Petri nets.

Definition 39 (SisoPN) *Operation* $\mathtt{sisoPN}$ *changes a Petri net* $PN = (P, T, F)$ *into a SISO-net* $\mathtt{sisoPN}(PN) = (P', T', F')$ *with* [11]*:*

$$P' = P \cup \begin{cases} \{p_t \mid t \in \mathtt{start}(PN)\} & \text{if} \quad |\mathtt{start}(PN)| > 1, \\ \{p_t \mid t \in \mathtt{end}(PN)\} & \text{if} \quad |\mathtt{end}(PN)| > 1, \end{cases}$$

$$T' = T \ \cup \begin{cases} \{i_{split}\} & \text{if} \quad |\mathtt{start}(PN)| > 1, \\ \{o_{join}\} & \text{if} \quad |\mathtt{end}(PN)| > 1, \end{cases}$$

$$F' = F \cup \begin{cases} \{(p_t, t) \in (P' \times T') \mid t \in \mathtt{start}(PN)\} & \text{if} \quad |\mathtt{start}(PN)| > 1, \\ \{(i_{split}, p_t) \in (T' \times P') \mid t \in \mathtt{start}(PN)\} & \text{if} \quad |\mathtt{start}(PN)| > 1, \\ \{(t, p_t) \in (T' \times P') \mid t \in \mathtt{end}(PN)\} & \text{if} \quad |\mathtt{end}(PN)| > 1, \\ \{(p_t, o_{join}) \in (P' \times T') \mid t \in \mathtt{end}(PN)\} & \text{if} \quad |\mathtt{end}(PN)| > 1. \end{cases}$$

The SISO-net resulting from the application of operation `sisoPN` to the task cluster is transformed into an annotated SISO-net with operation `annotate` (Definition 34).

The `merge` operation creates a layered annotated SISO-net from the component. The places, transitions and relations that are captured in the task cluster are not included and a new transition is added. The annotated SISO-net that is the result of $\mathtt{annotate}(\mathtt{sisoPN}(PN))$ is included as a sub net and is associated with the new transition, making this transition an aggregated transition. All transitions, except for the new transition, have the same annotation as they had in the original layered annotated SISO-net. The new aggregated transition has to be annotated in accordance with its associated sub net. The set of input data elements of the aggregated transition is not the same as the union of all input data elements related to the transitions in the sub net. It is a sub set

[11] p_t is a "fresh" identifier for places, i.e., $\{p_t\} \cap P = \emptyset$, and i_{split} and o_{join} are two "fresh" identifiers for transitions, i.e., $\{i_{split}, o_{join}\} \cap T = \emptyset$.

of these input data elements, because there may be one or more transitions in the sub net that are dependent on a transition in the sub net. Then, the input for a certain transition becomes available while the sub net, i.e., the aggregated transition, is executed. Such input data elements can not be input data elements of the aggregated transition, because the required input is not available before the execution of the aggregated transition is started. All output data elements of transitions that are included in the sub net form the set of output data elements of the aggregated transition. The assigned role of the aggregated transition is the same as the role that is assigned to the transitions in the sub net.

Definition 40 (Merge) *Let* $LS_1 = (S_1, SS_1, map_1)$ *be a layered annotated SISO-net,* C *an acyclic MG component of* S_1*,* PN *a task cluster in* C*, and* $S' = \texttt{annotate}(\texttt{sisoPN}(PN)) = (P, T, F, D, D_I, D_O, R, A)$*. Operation* `merge` *replaces* PN *in* C *with a new transition* t' [12] *and adds* S' *as a sub net of* t'*,* $\texttt{merge}(LS_1, C, PN) = (S_2, SS_2, map_2)$ *with:*

- $S_2 = (P_2, T_2, F_2, D_2, D_{I_2}, D_{O_2}, R_2, A_2)$ *is an annotated SISO-net with:*
  - $P_2 = \pi_P(S_1||_C) \setminus P$ *is the set of places,*
  - $T_2 = (\pi_T(S_1||_C) \setminus T) \cup \{t'\}$ *is the set of transitions,*
  - $F_2 = (\pi_F(S_1||_C) \cap ((P_2 \times T_2) \cup (T_2 \times P_2))) \cup \{(p, t') \in (P_2 \times T_2) \mid p\bullet \cap T \neq \emptyset\} \cup \{(t', p) \in (T_2 \times P_2) \mid \bullet p \cap T \neq \emptyset\}$ *is the flow relation,*
  - $D_2 = \pi_D(S_1||_C)$ *is a set of data elements,*
  - $D_{I_2} \in T_2 \rightarrow \mathcal{P}(D_2)$ *is the set of input data elements such that:*
    - $\forall_{t \in T_2 \setminus \{t'\}} : D_{I_2}(t) = \pi_{D_I}(S_1)(t)$*, and*
    - $D_{I_2}(t') = \left(\bigcup_{t \in T} D_I(t)\right) \setminus \left(\bigcup_{t \in T} D_O(t)\right)$,
  - $D_{O_2} \in T_2 \rightarrow \mathcal{P}(D_2)$ *is the set of output data elements such that:*
    - $\forall_{t \in T_2 \setminus \{t'\}} : D_{O_2}(t) = \pi_{D_O}(S_1)(t)$*, and*
    - $D_{O_2}(t') = \bigcup_{t \in T} D_O(t)$,
  - $R_2 = \pi_R(S_1||_C)$ *is a set of roles, and*
  - $A_2 \in T_2 \not\rightarrow R_2$ *is the role assignment such that:*
    - $dom(A_2) = (dom(\pi_A(S_1||_C)) \setminus T) \cup \{t'\}$,
    - $\forall_{t \in dom(\pi_A(S_1||_C)) \setminus T} : A_2(t) = \pi_A(S_1)(t)$*, and*
    - $A_2(t') = \bigcup_{t \in T \cap dom(A)} A(t)$,
- $SS_2 = \{S'\}$ *is a set of annotated SISO-nets, and*
- $map_2 \in T_2 \not\rightarrow SS_2$ *is the assignment function, such that:*
  - $dom(map_2) = \{t'\}$*, and*
  - $map_2(t') = \{S'\}$.

The layered annotated SISO-net that results from $\texttt{merge}(LS_1, C, PN)$ may contain more places (and relations) between the transitions in the upper layer of the layered annotated SISO-net than strictly necessary. We distinguish two types of redundant places: parallel places and implicit places. In Figure 20, we

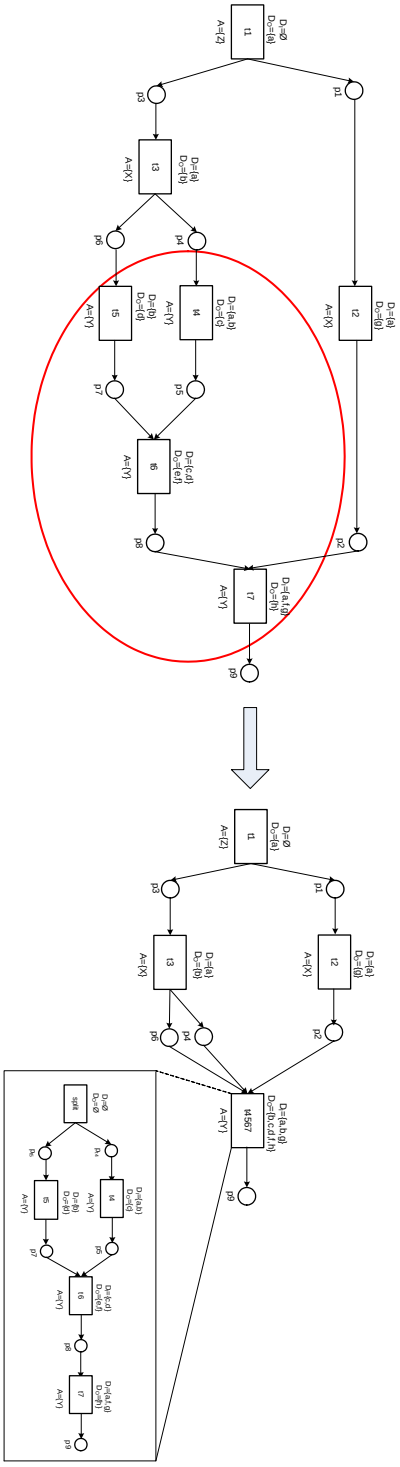

Fig. 20. Example of parallel places

give an example of parallel places. In the redesign, presented at the righthand side of the example, the transitions *t3* and *t4567* have two places connecting them, namely place *p4* and place *p6*. This construct is the starting point for one of Murata's reduction rules, namely the fusion of parallel places [29]. Parallel places, like *p4* and *p6*, can be fused and with the application of this reduction rule safeness and soundness are preserved [29].

In Figure 21, we give an example of a merge resulting in a redesign with implicit places. In the redesign, presented at the righthand side of the example,

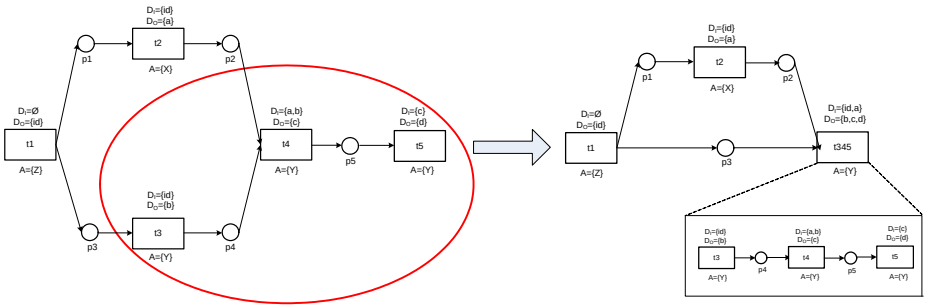

Fig. 21. Example of implicit places

the place *p3* connecting transition *t1* with transition *t345* is redundant because transition *t1* is never directly followed by transition *t345*. Place *p3* is a so-called implicit place [3]. An implicit place is a place which always contains enough tokens to allow for the enabling of the transitions connected to it [3,15]. In the example, place *p3* always contains a token when transition *t2* puts a token in place *p2* thus allowing for the enabling of transition *t345*. Implicit places, like *p3*, can be removed without changing the behavior of the process, because it does not restrict the set of possible firing sequences [3].

The layered annotated SISO-net resulting from the application of the operation `merge` has certain properties.

Theorem 11. *Let* $LS_1 = (S_1, SS_1, map_1)$ *be a layered annotated SISO-net,* C *an acyclic MG component of* S_1*,* PN *a task cluster in* C*,*
$S' = \texttt{annotate}(\texttt{sisoPN}(PN)) = (P, T, F, D, D_I, D_O, R, A)$*, and*
$(S_2, SS_2, map_2) = \texttt{merge}(LS_1, C, PN)$*, then:*

- $\texttt{start}(S_2)$ *and* $\texttt{start}(s)$*, for all* $s \in SS_2$*, are singletons,*
- $\texttt{end}(S_2)$ *and* $\texttt{end}(s)$*, for all* $s \in SS_2$*, are singletons,*
- *All nodes in* S_2 *are on a path from* $\texttt{start}(S_2)$ *to* $\texttt{end}(S_2)$ *and all nodes in* s*, for all* $s \in SS_2$*, are on a path from* $\texttt{start}(s)$ *to* $\texttt{end}(s)$*, and*

[12] t' is a "fresh" identifier for a transition, i.e., $\{t'\} \cap \pi_T(S_1) = \emptyset$.

– *S_2 and all $s \in SS_2$ are acyclic.*

Proof. First, we prove that $\texttt{start}(S_2)$ is a singleton. A component starts with one single node (Definition 22). Suppose the start node of the component is not included in the task cluster. Then, it is clear that this node becomes the only start node of S_2. Otherwise, a new single node (that is, the new transition) becomes the single start node of S_2. A similar argument can be given to show that $\texttt{end}(S_2)$ is a singleton.
Then, we prove that $\texttt{start}(s)$, for all $s \in SS_2$, is a singleton. Operation `sisoPN` adds a source node when the task cluster has multiple start nodes. These start nodes are connected to the added source node and become its successors. The added source node is the only source node in the resulting SISO-net. So, $\texttt{start}(s)$, for all $s \in SS_2$, is a singleton. A similar argument can be given to show that $\texttt{end}(s)$, for all $s \in SS_2$, is a singleton.
Next, we have to show that all nodes in S_2 are on a path from $\texttt{start}(S_2)$ to $\texttt{end}(S_2)$ and all nodes in s, for all $s \in SS_2$, are on a path from $\texttt{start}(s)$ to $\texttt{end}(s)$. This is similar to the proof provided for Theorem 6.
Finally, we prove that S_2 and all $s \in SS_2$ are acyclic. We start with S_2. The input for the merge transformation is a component C with its projection $S||_C$ being acyclic. A task cluster is selected within C such that the task cluster is weakly connected and that all inputs of the task cluster are located before any of the outputs of the task cluster. These requirements ensure that all input places of the task cluster connect the tasks in the component, that are preceding the task cluster, with the new task (that replaces the task cluster). Similarly, all output places connect the new transition with tasks in the component that are following the new task. Hence, there are no circuits introduced and S_2 is acyclic.
Remains to prove that all $s \in SS_2$ are acyclic. A task cluster within C only included relations that are present in C resulting in an acyclic task cluster. An annotated SISO-net is created from the task cluster. Operation `sisoPN` adds relations between the source (sink) node and the start (end) nodes. This does not introduce circuits. So, all $s \in SS_2$ are acyclic. □

With Theorem 12 we show that the result of the merge transformation is indeed a layered annotated SISO-net. Moreover, Theorem 12 shows that this layered annotated SISO-net is safe and sound. The result is used, later on, to show that the replacement of the component (selected from a layered annotated SISO-net) by the net results in a layered annotated SISO-net which is safe and sound.

Theorem 12. *Let $LS_1 = (S_1, SS_1, map_1)$ be a layered annotated SISO-net, C an acyclic MG component of S_1, PN a task cluster in C, and*
$S' = \texttt{annotate}(\texttt{sisoPN}(PN)) = (P, T, F, D, D_I, D_O, R, A)$, then
$(S_2, SS_2, map_2) = \texttt{merge}(LS_1, C, PN)$ is a safe and sound layered annotated SISO-net.

Proof. For all annotated SISO-nets present in (S_2, SS_2, map_2) the proof is similar to the proof provided for Theorem 7. □

The next step is the replacement of the selected component with the alternative process part that has been created with the merge transformation.

9.3 Replacement

The layered annotated SISO-net LS_2 is the alternative process part that results from the application of the merge transformation on the acyclic MG component C which includes task cluster PN and which has been selected from the layered annotated SISO-net LS_1. The redesign is finished with the replacement of $S_1||_C$ in LS_1 with LS_2. This replacement is performed by operation $\texttt{replace}(LS_1, C, LS_2)$ (Definition 24). In Theorem 3 it is stated that the resulting net LS_3 is a layered annotated SISO-net and that LS_3 is safe and sound if LS_1 and LS_2 are both safe and sound layered annotated SISO-nets. For this matter, we require the original process LS_1 to be a safe and sound layered annotated SISO-net. From Theorem 12 it follows that the alternative process part LS_2 is safe and sound. The resulting net LS_3 may contain routing transitions (AND-splits and -joins), that are superfluous. These routing transitions can be removed using the aggregation rule given in Definition 25. The end result is a redesign alternative for the initial process.

We illustrate the three redesign steps with our running example, the blood donation process.

9.4 Example

Suppose that the donation center, that is responsible for the execution of the blood donation process (Figure 3), would like to improve the quality of the process, as perceived by the donors. Part of the perceived quality is determined by the interaction between the donors and the employees of the donation center. In the current blood donation process, the donor has to interact two times with a desk clerk (tasks `Register donor` and `Instruct donor`). It could be that a donor has to interact with two different desk clerks. It is expected that the donor perceives a higher quality of the delivered work, when (s)he has to interact with one desk clerk. A possible solution would be that one task handles both interactions between a donor and a desk clerk. This solution is created with the merge transformation. The creation of a redesign alternative with the merge transformation starts with the selection of a component and the selection of a task cluster from the blood donation process. A process part, including the first half of the process, is selected and this selection is depicted with a circle around the selected process part in Figure 17. Within this component, a task cluster is selected. Next to the tasks `Register donor` and `Instruct donor`, the task `Print labels` is included in the task cluster. This way, all tasks that are performed by the desk clerk are combined, so a desk clerk can handle all actions for one donor at once. The redesign alternative resulting from the merge transformation is depicted in Figure 23. The aggregated task corresponding to the task cluster with tasks `Register donor`, `Print labels`, and `Instruct donor` received the

name `Intake donor`. Next to a higher perceived quality, a reduction in setup times is expected as a benefit, since the setup time is only spent once before the execution of the aggregated task.

With the merge transformation we have presented our fourth, and last, transformation. This brings us to the end of the formal part of this working paper. In the next section we present related work.

10 Related work

Various structured approaches to process redesign were proposed earlier, most notably the ProcessWise methodology [18] and the MIT Process Handbook [26]. Also, a variety of tools is available, e.g. MIT's process recombinator tool [14], a number of tools that apply case-based reasoning [25,28], and the KOPeR tool by Nissen [33]. Many existing approaches and tools are limited in their application domain, while none of the approaches has succeeded to gain widespread adoption in industry. We have provided a more extensive literature review on this topic in [31].

Nissen's work [32,33] is most related to our evolutionary approach. The main contribution of his work, implemented by the KOPeR tool, is the construction of a set of measures that, applied to processes, would at first help to diagnose pathologies. The pathologies are mapped to matching transformations that are then applied to the processes in order to improve their performances. Nissen's transformations are redesign heuristics that are presented to the user. The user, however, has to construct the redesigns him/herself. Furthermore, Nissen used a rather simplistic process model without a formal base. Our formal process definition, however, allows for the modeling of realistic, complex business processes. In earlier work [30], we used Nissen's idea to come up with a similar approach. In this approach, we defined a set of process measures, which provide a global view on the characteristics of the process. Actual values for the process measures may reveal weaknesses in the process. Identified weaknesses are removed by the application of one or more redesign best practices. We, however, did not describe in [30] how the actual process change is made while applying the best practices. This topic is the contribution of this report by means of the process transformations. With the presented transformations, it is possible to apply the task composition, resequencing, parallelism, and triage best practice.

The redesign of processes with as outcome a more parallel, sequential, composed or decomposed process is frequently performed. Limam Mansar and Reijers [27], for instance, identify a "top ten" list of most popular redesign best practices which include task (de)composition, resequencing and parallelism. According to Reijers [34], task composition is the most cited redesign best practice. Van der Aalst [5] provides concrete quantitative guidelines for the redesign of processes with task composition and parallelism to obtain the 'optimal' process with respect to the utilization of resources and throughput time. Quantitative

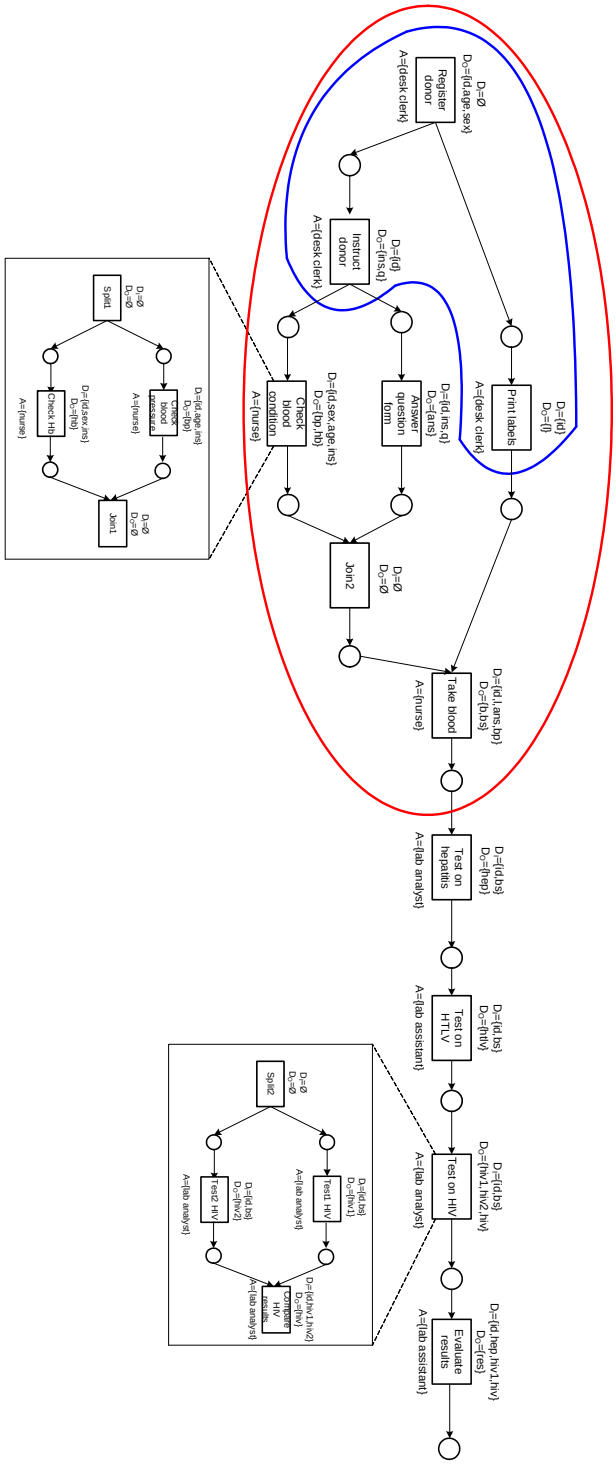

Fig. 22. Component and task cluster selection

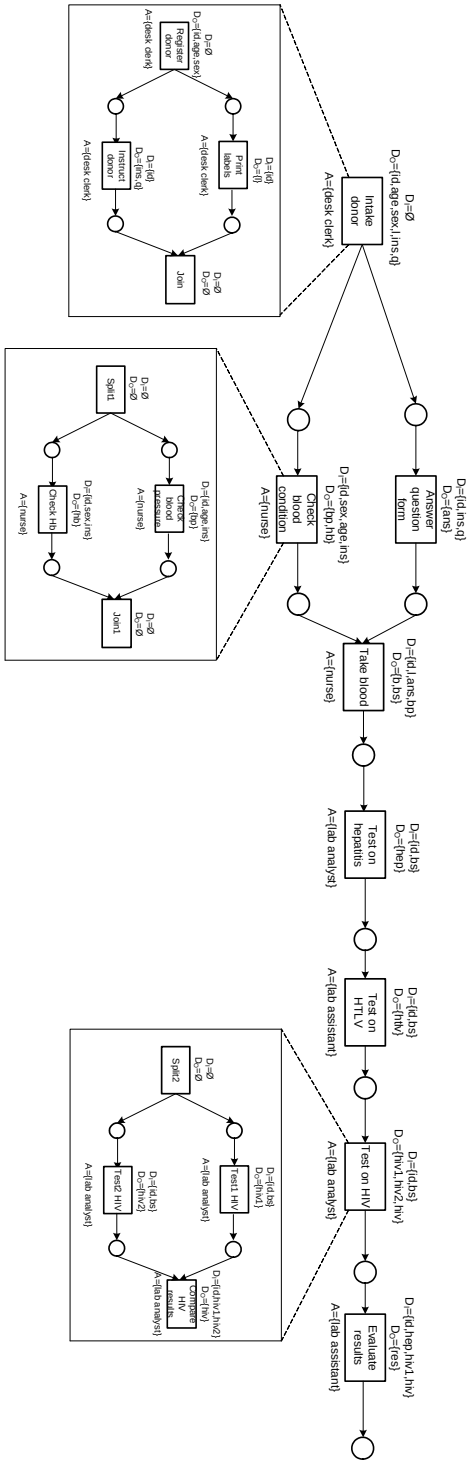

Fig. 23. Alternative redesign after merging

support for task composition and parallelism is also provided by Buzacott [17] and for task composition by Dewan, Seidmann and Walter [20] and Seidmann and Sundararajan [37].

A comprehensive set of change patterns and change support features is proposed by Weber, Rinderle-Ma and Reichert [41]. The set of change patterns includes a "parallelize activities", an "add control dependency" (i.e., sequence), an "extract sub process" (i.e., compose), and "inline sub process" (i.e., decompose) pattern. Formal semantics are provided for the patterns. When using, for instance, the "parallelize activities" pattern the user has to decide him/herself which activities have to be placed in parallel in the changed model. Our transformations not only result in a changed model, they also propose the changes to the user. Furthermore, our changes are more extensive than the changes Weber *et al.* perform. The "inline sub process" pattern, for example, only inlines one sub net at the time, while our unfold transformation unfolds as many sub nets as necessary. The parallel and sequence transformation also have a more elaborate result. Weber *et al.* use the patterns and support features to evaluate how well approaches and systems deal with process change.

The creation of changes in (specific subclasses of) Petri nets while preserving soundness and other desirable properties has been researched extensively [1,4,8,19,29,39,40]. Part of that work focuses on the reduction of nets to facilitate analysis. Murata [29] proposes reduction rules that create a simpler Petri net, while preserving the system properties to be analyzed. Desel and Esparza [19] present a method for the analysis of well-formedness of a free-choice net using reduction rules. These rules may be applied in a reversed manner resulting in synthesis rules. Vanhatalo, Volzer and Leymann [40] propose to decompose a workflow graph into a tree of single-entry-single-exit (SESE) fragments and to check soundness of each fragment in isolation. Van der Aalst [4] uses the hierarchical (de)composition of WF-nets to establish the correctness of a workflow by analyzing each of the subflows separately. Our safe and sound replacement of the component with the alternative process part is based on this hierarchical (de)composition.

Another part of the related work on soundness preserving changes aims at the modification of nets. In our evolutionary approach towards redesign we use and extend this part of the related work. Suzuki and Murata [39] discuss the replacement of a transition by a sub net and vice versa, that is refinement and abstraction of a net. We perform similar operations with the unfold and merge transformation. Van der Aalst [1] presents eight basic soundness preserving transformation rules for WF-nets. A rule presents the substitution of a transition with either two sequential tasks, two conditional tasks, two tasks in parallel, or an iterative task or the opposite transformations. Our transformations are an extension of these rules. Van der Aalst and Bisgaard Lassen [8] use the notion of a component and replace such a component by a single transition. A component is a selected part of a WF-net that has a clear start and end. We use a similar notion, also called component, when conducting process redesign.

Van der Aalst and Bisgaard Lassen use the component to translate WF-nets to BPEL by mapping components onto BPEL constructs, thus generating readable and compact BPEL code.

Literature on process change mainly focuses on the control flow perspective. In our work, we also include the data and resource perspective by means of the annotation with data dependencies and roles. Sun, Leon Zhao, Nunamaker and Liu Sheng [38] provide a data-flow perspective for detecting and correcting errors caused by missing, redundant, or conflicting data. Their data-flow framework consists of the data-flow specification and the data-flow analysis. Their method to specify the data flow in a workflow system is comprehensive and intuitive. They describe different scenarios that may result in data-flow errors which are used to formally define data-flow verification rules. The work of Sun *et al.* is complementary to our work. We require the necessary dependencies to be present, but with the data-flow analysis we could check for this. Also, we may have redundant or conflicting data in our process definition which we may detect and correct with the approach of Sun *et al.*

11 Summary and outlook

Business Process Redesign (BPR) is a popular methodology for companies to boost the performance of their business processes. Although literature on BPR is available in abundance, little concrete support on how to get from *as-is* towards *to-be* is available. With our evolutionary approach towards process redesign we aim to fill this gap between *as-is* and *to-be*. Part of the approach is a concrete method for the creation of the actual alternative redesign. In this working paper, we present this method which consists of three steps: 1) the *selection* of a process part, 2) the *transformation* of this process part into an alternative part and, 3) the *replacement* of the original process part with the alternative part. The result is an alternative redesign for the process. The actual change in the process is made with process transformations. So far, four process transformations are defined: *parallel*, *sequence*, *merge* and *unfold*. In our definitions of the transformations we choose to aim at *extreme* changes, for instance, we put as many tasks as possible in parallel. Of course, depending on the specific process the redesign effort is made for, more conservative changes may be preferable. But with some adaptations of the definitions it is also possible to create such redesigns.

The process transformations are performed on a process that covers the control flow perspective, the data perspective and the resource perspective. The notion of data dependencies between tasks has been introduced to model the data perspective. A similar notion can, however, be used to model other forms of dependencies between tasks, like logical constraints or a specific ordering of tasks. With the notion of roles, the resource perspective has been modeled. Instead of labeling a task with a role, tasks can be labeled differently with, for instance, departments, applications or geographical locations. Dependencies and

labels can, therefore, be specified according to the process information that is available within the organization and the specific redesign goals identified by the organization. Therefore, the presented process transformations give much more redesign possibilities than their number may suggest. In addition, the use of more or other process attributes may also lead to new process transformations. The current set of process transformations is not exhaustive and serves as a starting point for process redesign.

Next to the extension of the set of process transformations, future research should focus on the selection. On the one hand, there is the selection of the process part that should be changed. On the other hand, there is the selection of the process transformation that should be performed on the selected process part. These selection issues are, of course, related to one another. In [30], we present global process measures and relate the outcome of the measures to the selection of applicable best practices. A possible direction would be to come up with a similar method that focuses on process parts (local measures) and process transformations.

Another part of the future work is the fifth step of the evolutionary approach: *the evaluation of redesign alternatives*. Evaluation of process models can be done by simulating the model or (for simple processes) with analytical techniques. For such an evaluation, performance data (time, cost and quality indicators) are required. These data may be collected in event logs which are derived from the execution of the actual process. Log-based extension of a process model with a new aspect or perspective (e.g., enriching the model with performance data) is part of the process mining research [10,11].

Next to the further development of the theory of evolutionary process improvement, we work on the development of a tool that provides concrete support to practitioners. The ProM tool [6] is used as a framework to experiment with process mining and redesign techniques. We envision as the ultimate goal of our research the delivery of an automated redesign tool. This tool would support all steps of the approach in an "intelligent" way. By this, we mean that the tool not only automates the various steps of the approach, but also interacts with the redesigner. The redesigner is able to indicate which performance dimensions (time, costs, quality) should be improved, whether certain process characteristics should perhaps not be changed, and which promising alternatives should be combined in constructing the best alternative. Our approach is a solution that should primarily help redesign novices in finding process alternatives based on the redesign best practices. Secondly, more experienced redesigners are supported in the creation and evaluation of such alternatives in a structured and less time-consuming manner.

Acknowledgement

This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Dutch Ministry of Economic Affairs.

References

1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. *Systems Analysis - Modelling - Simulation*, 34(3):335–367, 1999.
4. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
5. W.M.P. van der Aalst. Reengineering Knock-out Processes. *Decision Support Systems*, 30(4):451–468, 2001.
6. W.M.P. van der Aalst, B.F. van Dongen, C.W. Gunther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. ProM 4.0: Comprehensive Support for Real Process Analysis. In J. Kleijn and A. Yakovlev, editors, *Petri Nets and Other Models of Concurrency*, volume 4546 of *Lecture Notes in Computer Science*, pages 484–494. Springer-Verlag, Berlin, 2007.
7. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2004.
8. W.M.P. van der Aalst and K. Bisgaard Lassen. Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation. *Information and Software Technology*, 50(3):131–159, 2008.
9. W.M.P. van der Aalst, M. Netjes, and H.A. Reijers. Chapter 4: Supporting the Full BPM Life-Cycle Using Process Mining and Intelligent Redesign. In K. Siau, editor, *Contemporary Issues in Database Design and Information Systems Development*, pages 100–132. IGI Global, Hershey, USA, 2007.
10. W.M.P. van der Aalst, H.A. Reijers, A.J.M.M. Weijters, B.F. van Dongen, A.K. Alves de Medeiros, M. Song, and H.M.W. Verbeek. Business Process Mining: An Industrial Application. *Information Systems*, 32(1):713–732, 2007.
11. W.M.P. van der Aalst and A.J.M.M. Weijters, editors. *Process Mining*, Special Issue of Computers in Industry, Volume 53, Number 3. Elsevier Science Publishers, Amsterdam, 2004.
12. A.V. Aho, M.R. Garey, and J.D. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.
13. M. Al-Mashari and M. Zairi. BPR implementation process: An analysis of key success and failure factors. *Business Process Management Journal*, 5(1):87–112, 1999.

14. A. Bernstein, M. Klein, and T.W. Malone. The process recombinator: a tool for generating new business process ideas. *International Conference on Information Systems*, pages 178–192, 1999.
15. G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 360–376. Springer-Verlag, Berlin, 1987.
16. BPTrends. BPTrends BPM Market Survey 2007. http://www.bptrends.com, 2007.
17. J.A. Buzacott. Commonalities in Reengineered Business Processes: Models and Issues. *Management Science*, 42(5):768–782, 1996.
18. P. Calvert. An Advanced BPR Methodology with Integrated, Computer-based Tools. In B.C. Glasson, I.T. Hawryszkiewycz, B.A. Underwood, and R.A. Weber, editors, *Business Process Re-engineering: Information Systems Opportunities and Challenges*, pages 161–170. Elsevier, 1994.
19. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
20. R. Dewan, A. Seidmann, and Z. Walter. Workflow Optimization through Task Redesign in Business Information Processes. In *Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences (HICSS '98) Volume 1*, pages 240–252. IEEE Computer Society, Washington DC, USA, 1998.
21. J.L. Gross and J. Yellen. *Handbook of Graph Theory*. CRC Press, Boca Raton, 2004.
22. V. Grover, S. Jeong, W. Kettinger, and J. Teng. The implementation of Business Process Reengineering. *Journal of Management Information Systems*, 12(1):109–144, 1995.
23. M. Hammer and J. Champy. *Reengineering the corporation: a manifesto for business revolution*. Harper Business Editions, New York, 1993.
24. W. Kettinger, J. Teng, and J. Guha. Business process change: A study of methodologies, techniques, and tools. *MIS Quarterly*, 21(1):55–80, 1997.
25. S. Ku and Y.H. Suh. An Investigation of the K-tree Search Algorithm for Efficient Case Representation and Retrieval. *Expert Systems with Applications*, 11(4):571–581, 1996.
26. T.W. Malone, K. Crowston, and G.A. Herman. *Organizing Business Knowledge: The MIT Process Handbook*. MIT Press, 2003.
27. S. Limam Mansar and H.A. Reijers. Best Practices in Business Process Redesign: validation of a redesign framework. *Computers in Industry*, 56(5):457–471, 2005.
28. D.M. Min, J.R. Kim, W.C. Kim, D. Min, and S. Ku. IBRC: Intelligent Bank Reengineering System. *Decision Support Systems*, 18(1):97–105, 1996.
29. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
30. M. Netjes, S. Limam Mansar, H.A. Reijers, and W.M.P. van der Aalst. An Evolutionary Approach for Business Process Redesign: Towards an Intelligent System. In J. Cardoso, J. Cordeiro, and J. Filipe, editors, *Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007)*. INSTICC, Setubal, 2007.
31. M. Netjes, I. Vanderfeesten, and H.A. Reijers. "Intelligent" Tools for Workflow Process Redesign: A Research Agenda. In C. Bussler and A. Haller, editors, *Business Process Management Workshops: BPM 2005*, volume 3812 of *Lecture Notes in Computer Science*, pages 444–453. Springer-Verlag, Berlin, 2006.

32. M. Nissen. Knowledge-Based Organizational Process Redesign: Using Process Flow Measures to Transform Procurement, PhD. Dissertation. University of Southern California, downloadable at http://web.nps.navy.mil/ menissen/, 1996.
33. M. Nissen. Redesigning Reengineering through Measurement-Driven Inference. *MIS Quarterly*, 22(4):509–534, 1998.
34. H.A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*, volume 2617 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 2003.
35. H.A. Reijers and S. Limam Mansar. Best Practices in Business Process Redesign: An Overview and Qualitative Evaluation of Successful Redesign Heuristics. *Omega: The International Journal of Management Science*, 33(4):283–306, 2005.
36. H.A. Reijers, S. Limam Mansar, and W.M.P. van der Aalst. Product-based Workflow Design. *Journal of Management Information systems*, 20(1):229–262, 2003.
37. A. Seidmann and A. Sundararajan. The Effects of Task and Information Asymmetry on Business Process Redesign. *International Journal of Production Economics*, 50(2), 1997.
38. S.X. Sun, J. Leon Zhao, J.F. Nunamaker, and O.R Liu Sheng. Formulating the Data-Flow Perspective for Business Process Management. *Information Systems Research*, 17(4):374–391, 2006.
39. I. Suzuki and T. Murata. A method for stepwise refinements and abstractions of Petri nets. *Journal of Computer and Systems Sciences*, 27(1):51–76, 1983.
40. J. Vanhatalo, H. Volzer, and F. Leymann. Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In B. Kramer, K.J. Lin, and P. Narasimhan, editors, *International Conference on Service-Oriented Computing: ICSOC 2007*, volume 4749 of *Lecture Notes in Computer Science*, pages 43–55. Springer-Verlag, Berlin, 2007.
41. B. Weber, S.B. Rinderle-Ma, and M.U. Reichert. Change Support in Process-Aware Information Systems - A Pattern-Based Analysis. Technical Report TR-CTIT-07-76, ISSN 1381-3625, http://eprints.eemcs.utwente.nl/11331, 2007.