On the Formal Generation of Process Redesigns

Mariska Netjes, Hajo A. Reijers, and 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 (BPR) is a process oriented methodology to improve organizations. Although literature on BPR is available in abundance, little concrete support on how to get from *as is* towards *to be* is available [12]. We propose the use of an evolutionary redesign approach that is based on BPR best practices to fill this gap. The approach is *evolutionary* in nature, because local updates are made to an existing process. In this paper we focus on one part of the approach: the generation of redesign alternatives. The first step in the generation 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 generation is ensured.

Keywords: Business Process Redesign, process modeling, Business Process Management, workflows, best practices.

1 Introduction

Business Process Redesign (BPR) combines a radical restructuring of a business process with a wide-scale application of information technology [8]. Although many methods and tools are available to facilitate a BPR effort (e.g. [5,7,9]), little concrete support is provided on how to create the *to be* situation from the *as is* [12]. 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 would like to mention the approach and the tool that are most related to our work. Weber, Rinderle-Ma and Reichert [17] provide a comprehensive set of change patterns with formal semantics. The KOPeR tool [13] supports the detection of malfunctioning process parts and suggests the type of change that has to be made.

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 [15]. Such a list allows companies to use well-performing solutions from earlier redesign efforts. A BPR best practice supports practitioners in developing a process redesign by making evolutionary, local updates to an existing process. With our *evolutionary* approach to process redesign we aim to fill the gap between *as is* and *to be* by taking an existing process model and improving it using the BPR best practices (from [15]). It is evolutionary, because an existing process is taken as a starting point.

D. Ardagna et al. (Eds.): BPM 2008 Workshops, LNBIP 17, pp. 224–235, 2009.

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

1. model an existing process,
2. determine weaknesses in the process,
3. select applicable best practices,
4. generate alternative models,
5. evaluate the performance of the generated alternatives, and
6. choose the best alternative and implement the new process.

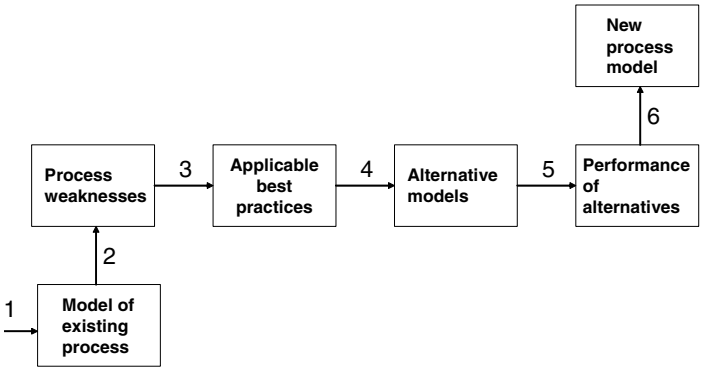

Fig. 1. Evolutionary approach towards redesign

The first three steps of Figure 1 we addressed in [10]. With regard to the modeling of an existing process (step 1), we gave a formal process definition and showed that it is not straightforward to spot inefficiencies in a process. For the determination of weaknesses (step 2), 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 (step 3), 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 generation of alternative redesign models (step 4), the best practices are applied. This step performs the actual change on part of the process, thus making a local update to the process. Then, the process can be redesigned again resulting in another update. Each iteration results in a redesign alternative which may be used as a starting point for another local update. In such a way, a tree of alternative process models is generated with the original process model as its root node.

In the final steps of our approach, the performance of the various alternatives is evaluated (step 5) and one redesign alternative is selected as the best alternative (step 6). 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 [4].

In this paper, we focus on the fourth step of the evolutionary approach: *the generation of redesign alternatives*. An 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 a specific best practice to be out of the scope of this paper since these issues are addressed by the first three steps of the approach [10]. Our contribution with this paper is a formally defined method for the generation of a process redesign. This generation 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.

In Section 2 of this paper, we define our notion of a process. In Section 3, we discuss the three steps of the redesign generation: selection, transformation and replacement and illustrate these with a transformation called *parallel*. In Section 4, we describe three other transformations: *sequence*, *unfold* and *merge*. Finally, we conclude and give an outlook on future work.

2 Process Definition

The starting point of a redesign effort with our approach is a process that is currently being used in an organization. A process has certain characteristics, like its structure, and several properties.

2.1 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 generate process redesigns 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 1, is a generalized WorkFlow net (WF-net) [1] with a single input (SI) and a single output (SO). Note that a WF-net is also a SISO-net, but that, for example, transition-bordered SISO-nets are not WF-nets.

Definition 1 (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:*

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

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 *dependencies* and *labels* that present the process information in a generic way. Dependencies exists between transitions, i.e., one transition depends on another. For each transition the dependencies, that should be satisfied before it can be executed, are defined as its input dependencies. The dependencies that are fulfilled after its execution are its output dependencies. Each transition may have a label assigned to it representing the type of

the transition. 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. An example of an annotated SISO-net can be found in Figure 2.

Definition 2 (Annotated SISO-net). *The 8-tuple* $S = (P,T,F,D,D_I,D_O,L,A)$ *is an annotated SISO-net with:*

- (P,T,F) *is a SISO-net,*
- D *is a finite set of dependencies,*
- $D_I \in T \rightarrow \mathcal{P}(D)$ [1] *relates transitions to sets of input dependencies, i.e., these dependencies should be satisfied before a transition can become enabled,*
- $D_O \in T \rightarrow \mathcal{P}(D)$ *relates transitions to sets of output dependencies, i.e., these dependencies are fulfilled when a transition fires,*
- L *is a finite set of labels, and*
- $A \in T \not\rightarrow L$ [2] *assigns an optional label to each transition.*

Dependencies and *labels* may be specified according to the process information that is available within the organization and the specific redesign goals identified by the organization. Data elements are a specific way to represent the dependencies between transitions. The dependencies, then, model the data perspective of the process. Sun, Leon Zhao, Nunamaker and Liu Sheng describe the data perspective in more detail and provide an approach similar to ours for the detection and correction of data errors [16]. Other specific dependencies between transitions may be the ordering between transitions in the original process or business dependencies identified by the organization. Roles represent a specific means to label transitions. The labels, then, model the resource perspective of the process. Other types of labeling, like departments, applications or geographical locations, may be used in a similar way.

2.2 Process Properties

The single input (SI) and the single output (SO) are important nodes in the annotated SISO-net and we give two operations to find the in- and output of the SISO-net.

Definition 3 (Input, output). *Let* $S = (P,T,F,D,D_I,D_O,L,A)$ *be an annotated SISO-net with source node i and sink node o. Operation* `in` *returns the source node of S, i.e.,* $\texttt{in}(S) = i$*, with* $i \in P \cup T$*. Operation* `out` *returns the sink node of S, i.e.,* $\texttt{out}(S) = o$*, with* $o \in P \cup T$*.*

A SISO-net has properties similar to a WF-net. A WF-net is constructed from a SISO-net by adding a place to the begin (the end) of the SISO-net when it starts (ends) with a transition. Definition 4 gives some properties of a SISO-net, namely safeness and soundness. The basic idea is that these properties are based on the corresponding WF-net [1].

Definition 4 (Safe and sound annotated SISO-net). *An annotated SISO-net S is defined to be safe and sound if and only if the associated WF-net is safe and sound.*

In the next section we describe the generation of a process redesign for an annotated SISO-net.

[1] $\mathcal{P}(X)$ is the powerset of X, i.e., $Y \in \mathcal{P}(X)$ if and only if $Y \subseteq X$.

[2] A is a partial function, i.e., the domain of A is a subset of T.

3 Selection, Transformation, and Replacement

When conducting a process redesign with the evolutionary approach, part of an existing process model is changed. This change is performed in three steps: selection, transformation and replacement.

3.1 Selection

The first step of the generation of a process redesign is the selection of the part of the process that should be changed. Figure 2 shows an example of a selected process part. The process part is selected from a blood donation process. This process described the donation of blood by donors in a donation center. In the first part of the process, the intake and condition check of the donor is performed. In the second part, that is selected, blood is taken from a donor and tested for several diseases. The donation center wants to shorten the duration of the lab tests.

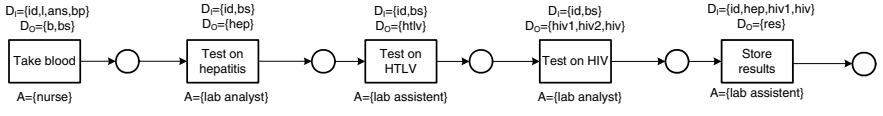

Fig. 2. Example: A selected process part

For the selection we use the notion of a *component*. A component can be seen as a selected part of the annotated SISO-net with a clear start and end, i.e., a component is a sub net satisfying properties similar to an annotated SISO-net. Our definition of a component is derived from van der Aalst and Bisgaard Lassen [3] who introduced the notion of a component.

Definition 5 (Component). *Let $S = (P,T,F,D,D_I,D_O,L,A)$ be an annotated SISO-net. Then C is a component in S if and only if:*

- $C \subseteq P \cup T$,
- *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*
  - $(o_C, i_C) \notin F$.

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 the net on the component.

Definition 6 (Projection). *Let $S = (P,T,F,D,D_I,D_O,L,A)$ be an annotated SISO-net and let C be a component in S. The projection of S on C, $S||_C$, is then defined as $S||_C = (P',T',F',D',D'_I,D'_O,L',A')$ with:*

- $P' = P \cap C$ *is the set of places,*
- $T' = T \cap C$ *is the set of transitions,*

- $F' = F \cap (C \times C)$ *is the flow relation,*
- $D' = \bigcup_{t \in T'} D_I(t) \cup D_O(t)$ *is the set of dependencies,*
- $D'_I \in T' \to \mathcal{P}(D')$ *such that* $\forall_{t \in T'} : D'_I(t) = D_I(t)$ *is the set of input dependencies,*
- $D'_O \in T' \to \mathcal{P}(D')$ *such that* $\forall_{t \in T'} : D'_O(t) = D_O(t)$ *is the set of output dependencies,*
- $L' = \bigcup_{t \in T'} A(t)$ *is the set of labels, and*
- $A' \in T' \not\to L'$ *such that* $dom(A') = dom(A) \cap C$, *and* $\forall_{t \in dom(A')} : A'(t) = A(t)$ *is the label assignment.*

The next theorem is used to prove the compositional nature of safe and sound annotated SISO-nets.

Theorem 1. *Let S be a safe and sound annotated SISO-net and let C be a component of S. The projection* $S||_C$ *is a safe and sound annotated SISO-net.*

Proof. See [11].

Soundness and safeness are desirable properties. Theorem 1 shows that these desirable properties are propagated to any component in the net. The component is the input for the next step: the transformation.

3.2 Transformation

In the transformation, an alternative process part is generated for the selected process part. The transformation is the actual change that is made to the process. The transformations presented in this paper are extensions of the basic soundness preserving transformation rules invented by van der Aalst [1]. Due to page limitations we discuss only one transformation in detail: the *parallel transformation* and describe the other transformations briefly in the next section.

In an existing process, tasks may be required to be executed one after another while no real dependencies exist between these tasks. This gives unnecessary delays in the process execution, because a task may be waiting for another task to finish, while this task is not depending on the results of that task. Often, it is a better option to perform tasks without dependencies between one another in parallel. Parallel tasks may be executed in any order or simultaneously. The obvious benefit would be a reduction in throughput time. A possible negative effect is a loss of quality because processes with concurrent behavior are more complex [14].

With the parallel transformation we put tasks that do not depend on one another in parallel. The parallelization is achieved by putting relations between dependent tasks. This places tasks that do not depend on one another in parallel. Requirements for the annotation of the selected process part are necessary because relations are added based on the dependencies between tasks. These dependencies should be distributed over the process in such a way that the input dependencies for each task are fulfilled before the task becomes enabled. This means that each dependency has to be fulfilled, i.e., is an output dependency, before it can be used as an input dependency. That is, a task that fulfills an output dependency is placed before the transition(s) that require this

dependency as an input. Note that this means that a dependency has to be an output dependency of one task in the process, but that it does not imply that it has to be an input dependency of any task. The requirements for the use of dependencies also imply that a task can not have the same dependency both as an input and as an output. Also note that the component has to be an acyclic marked graph (see [6] for an introduction to these standard notions) to enforce that all input dependencies are fulfilled before a transition can become enabled.

The parallel transformation is started with a component which projection is an acyclic marked graph fulfilling the stated requirements for the annotation. Its result 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 solely serve as routing transitions are removed. Edges between the nodes reflect the dependencies and places are not necessary and therefore omitted.

Definition 7 (Parallel). *Let S be an annotated SISO-net and let C be a component in S with its projection $S||_C = (P,T,F,D,D_I,D_O,L,A)$ being an acyclic marked graph. Operation* `parallel` *changes C into the graph* `parallel`$(S,C) = (N,E)$ *with:*

- $N = \{n \in T \mid D_I(n) \neq \emptyset \vee D_O(n) \neq \emptyset\}$[3] *is the set of nodes,*
- $E = \{(n_1,n_2) \in N \times N \mid (D_O(n_1) \cap D_I(n_2)) \neq \emptyset\}$ *is the set of edges,*

The graph has to be further transformed into an annotated SISO-net. First, a source (or sink) node is added and connected to the start (or end) nodes if the graph starts (or ends) with more than one node. Then, superfluous relations are removed from the graph. The nodes in the graph are translated into the transitions of an SISO-net and places and corresponding arcs are added between these transitions. Finally, the annotation is restored. The result is a safe and sound annotated SISO-net. The complete formalization of the parallel transformation can be found in [11].

Figure 3 shows the alternative process part generated with the parallel transformation for the selected process part in Figure 2. In the example it can be seen that, for instance, transition *Test on hepatitis* and transition *Test on HIV* do not share any dependencies and are placed in parallel. Transition *Store results*, however, depends on these two transitions, and is therefore placed after the two transitions. It is also interesting to note that the output of transition *Test on HTLV* is not used in any way. Real life processes may contain such inconsistencies and Sun *et al.* provide an approach to detect and correct such errors [16].

3.3 Replacement

After the selection of a process part and the creation of an alternative for this selected part with process transformation, an alternative process is constructed. The selected component is replaced by the created alternative process part.

[3] Routing transitions, i.e., transitions for which holds $D_I(t) = D_O(t) = \emptyset$, are not considered and henceforth removed.

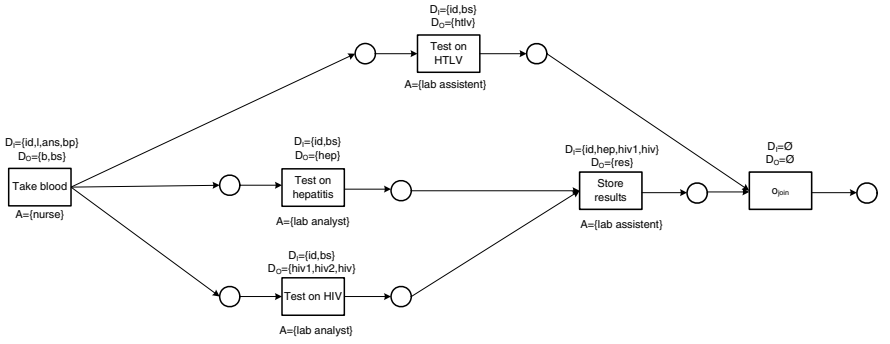

Fig. 3. Example: a transformed process part

Definition 8 (Replace). *Let* $S_1 = (P_1, T_1, F_1, D_1, D_{I_1}, D_{O_1}, L_1, A_1)$ *and* $S_2 = (P_2, T_2, F_2, D_2, D_{I_2}, D_{O_2}, L_2, A_2)$ *be two annotated SISO-nets. Let C, with source node* i_C *and sink node* o_C*, be a non-trivial component in* S_1*. Let* $\texttt{in}(S_2)$ *be a place if and only if* i_C *is a place and let* $\texttt{out}(S_2)$ *be a place if and only if* o_C *is a place. Operation* `replace` *substitutes* $S_1||_C$ *in* S_1 *with* S_2 *resulting in* $\texttt{replace}(S_1, C, S_2) = (P_3, T_3, F_3, D_3, D_{I_3}, D_{O_3}, L_3, A_3)$ *with:*

- $P_3 = (P_1 \setminus C) \cup P_2$ *is the set of places*[4]*,*
- $T_3 = (T_1 \setminus C) \cup T_2$ *is the set of transitions,*
- $F_3 = (F_1 \cap ((P_3 \times T_3) \cup (T_3 \times P_3))) \cup$
  $F_2 \cup$
  $\{(n, \texttt{in}(S_2)) \mid (n, i_C) \in F_1\} \cup$
  $\{(\texttt{out}(S_2), n) \mid (o_C, n) \in F_1\}$ *is the flow relation,*
- $D_3 = \left(\bigcup_{t \in T_1 \setminus C} D_{I_1}(t) \cup D_{O_1}(t) \right) \cup D_2$ *is the set of dependencies,*
- $D_{I_3} \in T_3 \to \mathcal{P}(D_3)$ *is the set of input dependencies such that:*
  - $\forall_{t \in T_1 \setminus C} : D_{I_3}(t) = D_{I_1}(t)$*, and*
  - $\forall_{t \in T_2} : D_{I_3}(t) = D_{I_2}(t)$*,*
- $D_{O_3} \in T_3 \to \mathcal{P}(D_3)$ *is the set of output dependencies such that:*
  - $\forall_{t \in T_1 \setminus C} : D_{O_3}(t) = D_{O_1}(t)$*, and*
  - $\forall_{t \in T_2} : D_{O_3}(t) = D_{O_2}(t)$*,*
- $L_3 = \left(\bigcup_{t \in T_1 \setminus C} A_1(t) \right) \cup L_2$ *is the set of labels, and*
- $A_3 \in T_3 \not\to L_3$ *is the label assignment such that:*
  - $dom(A_3) = (dom(A_1) \setminus C) \cup dom(A_2)$*,*
  - $\forall_{t \in dom(A_1) \setminus C} : A_3(t) = A_1(t)$*, and*
  - $\forall_{t \in dom(A_2)} : A_3(t) = A_2(t)$*.*

In the next theorem we show that the result of the replacement is again a safe and sound annotated SISO-net.

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

Theorem 2. *Let S_1 and S_2 be two annotated SISO-nets. Let C be a non-trivial component in S_1. Let $S_3 = \texttt{replace}(S_1, C, S_2)$.*
Then:

- *S_3 is an annotated SISO-net.*
- *If S_1 is safe and sound and S_2 is safe and sound, then S_3 is safe and sound.*

Proof. See [11].

The net resulting from the replacement may contain routing transitions (AND-splits and -joins), that have become superfluous. An AND-split (AND-join) is superfluous when it is preceded (followed) by exactly one transition, i.e., the two transitions are in a sequence. The routing transition and the other transition may be combined (thus removing the routing transition) with one of the soundness preserving transformation rules, namely *aggregation*, described by van der Aalst [1].

4 Other Transformations

When generating alternative process models the selection and replacement operations are the same regardless of the change that is made. The type of change is determined by the transformation that is used. In the previous section, the parallel transformation was used to generate a more parallel process redesign. In this section we describe three other process transformation informally: *sequence*, *unfold* and *merge*.

In [11], we introduce the notion of a *layered annotated SISO-net* to allow an annotated SISO-net to contain *aggregated transitions*. An aggregated transition has an underlying sub process that divides the aggregated transition into several smaller pieces of work. Such a sub process is used for more complex transitions to give more insight in what should exactly be done. 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 (sub) SISO-net for each of the aggregated transitions at the upper layer. The *unfold* and *merge* transformation are performed on a layered annotated SISO-net and the *parallel* and *sequence* transformation are performed on the upper layer of a layered annotated SISO-net. The definitions and details of the transformations can be found in [11].

Sequence transformation: This transformation is the counterpart of the parallel transformation. A sequential process may be perceived as a simpler process by employees and clients, because the order of the transitions is fixed. Further, transitions are likely to be executed in the most logical way which may reduce errors. Furthermore, the synchronization that is required after the execution of transitions in parallel is not necessary in sequential processes [14].

With the sequence transformation transitions are placed in a fixed order, i.e., a sequence. As a preparatory step, the selected component is translated into a graph. The *sequence* operation is performed on this graph. Afterwards, the graph is further transformed into an annotated SISO-net in a similar way as the graph resulting from the *parallel* operation (Definition 7).

Unfold transformation: With this transformation aggregated transitions are split up into several smaller transitions. Unfolding may result in a higher run-time flexibility,

because the scheduling and execution of some smaller transitions allows for more possibilities. Furthermore, a transition that is too large may be unworkable and dividing it into smaller transitions could enhance the quality realized by the involved resource(s). A drawback of smaller transitions are the longer set-up times, i.e., the time a resource spends to become familiar with a case [14].

The unfold transformation is performed by replacing the aggregated transitions (which are at the upper layer of the process) by the underlying lower layer SISO-nets. A lower layer SISO-net is defined such that it 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 structure of a SISO-net.

Merge transformation: This transformation is the counterpart of the unfold transformation. With the merging of transitions, multiple transitions are combined into one aggregated transition. Combining transitions results in the reduction of setup times, i.e., only one resource has to get acquainted with the case before the execution of the aggregated transition. An additional reason to merge transitions is the expected positive effect on the quality of the delivered work due to fewer hand-overs of the work between resources [14].

With the merge transformation several similar transitions, i.e., transitions with the same label, are combined into one aggregated transition. The transformation is started with the translation of an unfolded component into an annotated graph. Edges between the nodes in the graph are created in such a way that nodes with the same label are connected while preserving all dependencies. Within this graph the largest possible groups of sub graphs with the same label are determined. The *merge* operation creates a layered annotated graph with the identified sub graphs as its lower layer graphs. Then, this layered annotated graph is translated into a layered annotated SISO-net.

5 Conclusion and Outlook

In this paper, we present a concrete method for the generation of alternative process models. It is part of our evolutionary approach to process redesign based on BPR best practices and 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 actual change in the process is made with process transformations. With the process transformations we aim for *extreme* changes, e.g., we place as many transitions as possible in parallel. Of course, depending on the specific process the redesign effort is made for, more conservative changes may be preferable. In a supporting tool it will be possible to adapt the transformations and generate such redesigns.

The transformations are performed with a generic set of process attributes that may be specified depending on the specific attributes of the process under consideration. Therefore, the presented 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 further development of the theory on evolutionary process redesign, we work on the development of a tool that provides concrete support to practitioners. The ProM tool [2] 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 interact with the redesigner. Our approach is a solution that should primarily help redesign novices in finding process alternatives based on 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. van der Aalst, W.M.P.: Verification of Workflow Nets. In: Azéma, P., Balbo, G. (eds.) ICATPN 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997)
2. van der Aalst, W.M.P., van Dongen, B.F., Gunther, C.W., Mans, R.S., Alves de Medeiros, A.K., Rozinat, A., Rubin, V., Song, M., Verbeek, H.M.W., Weijters, A.J.M.M.: ProM 4.0: Comprehensive support for *real* process analysis. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 484–494. Springer, Heidelberg (2007)
3. van der Aalst, W.M.P., Bisgaard Lassen, K.: Translating Unstructured Workflow Processes to Readable BPEL: Theory and Implementation. Information and Software Technology 50(3), 131–159 (2008)
4. van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., Verbeek, H.M.W.: Business Process Mining: An Industrial Application. Information Systems 32(1), 713–732 (2007)
5. Al-Mashari, M., Zairi, M.: BPR implementation process: An analysis of key success and failure factors. Business Process Management Journal 5(1), 87–112 (1999)
6. Desel, J., Esparza, J.: Free Choice Petri Nets. Cambridge Tracts in Theoretical Computer Science, vol. 40. Cambridge University Press, Cambridge (1995)
7. Grover, V., Jeong, S., Kettinger, W., Teng, J.: The implementation of Business Process Reengineering. Journal of Management Information Systems 12(1), 109–144 (1995)
8. Hammer, M., Champy, J.: Reengineering the corporation: a manifesto for business revolution. Harper Business Editions, New York (1993)
9. Kettinger, W., Teng, J., Guha, J.: Business process change: A study of methodologies, techniques, and tools. MIS Quarterly 21(1), 55–80 (1997)
10. Netjes, M., Limam Mansar, S., Reijers, H.A., van der Aalst, W.M.P.: An Evolutionary Approach for Business Process Redesign: Towards an Intelligent System. In: Cardoso, J., Cordeiro, J., Filipe, J. (eds.) Proceedings of the 9th International Conference on Enterprise Information Systems (ICEIS 2007), pp. 484–494. INSTICC, Setubal (2007)
11. Netjes, M., Reijers, H.A., van der Aalst, W.M.P.: The Creation of Process Redesigns by Selecting, Transforming and Replacing Process Parts. BETA Working Paper Series, WP 240, Eindhoven University of Technology, Eindhoven (2008)

12. Netjes, M., Vanderfeesten, I., Reijers, H.A.: "Intelligent" Tools for Workflow Process Redesign: A Research Agenda. In: Bussler, C.J., Haller, A. (eds.) BPM 2005. LNCS, vol. 3812, pp. 444–453. Springer, Heidelberg (2006)
13. Nissen, M.: Redesigning Reengineering through Measurement-Driven Inference. MIS Quarterly 22(4), 509–534 (1998)
14. Reijers, H.A.: Design and Control of Workflow Processes: Business Process Management for the Service Industry. LNCS, vol. 2617. Springer, Heidelberg (2003)
15. Reijers, H.A., Limam Mansar, S.: 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)
16. Sun, S.X., Leon Zhao, J., Nunamaker, J.F., Liu Sheng, O.R.: Formulating the Data-Flow Perspective for Business Process Management. Information Systems Research 17(4), 374–391 (2006)
17. Weber, B., Rinderle-Ma, S.B., Reichert, M.U.: Change Support in Process-Aware Information Systems - A Pattern-Based Analysis. Technical Report TR-CTIT-07-76, ISSN 1381-3625 (2007), http://eprints.eemcs.utwente.nl/11331