Translating labelled P/T nets into EPCs for sake of communication

Translating labelled P/T nets into EPCs for sake of communication

H.M.W. Verbeek, B.F. van Dongen

Department of Technology Management, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{h.m.w.verbeek,b.f.v.dongen}@tm.tue.nl

Abstract. Petri nets can be used to capture the behavior of a process in a formal and precise way. However, Petri nets are less suitable to communicate the process to its owner, as simple routing constructs in the process might require a large number of transitions. This paper introduces a translation from labelled P/T nets to EPCs in such a way that many transitions can be translated into one EPC connector. The algorithm even allows for translating a set of transitions into an OR connector, even though the concept of OR connectors (especially the OR join connector) has no real equal in Petri nets. Using this translation presented here, labelled P/T nets may be communicated to the process owner by means of the created EPC.

Introduction

In the scientific world, Petri nets are a well-known modelling language for processes, having a clear executable semantics. Petri nets consist of places, transitions, and arcs, where places capture the set of possible states the process can be in, transitions capture the set of possible activities in the process, and the arcs link the activities to the states (which activity can be performed in which state, and to which state would its execution lead?).

In the business world however, Petri nets are less known for describing business processes, since real life processes often contain complex routing constructs between activities. In Petri nets, these routing constructs need to be captured by transitions and places as well, making them very hard to express. If Petri nets are used for describing real life processes, the identity of a transition that corresponds to some routing construct is of no importance, whereas the identity of a transition that corresponds to an activity is of high importance. As a result, a distinction between these two classes of transitions has to be made. One way of doing this is by adding a label to every transition that corresponds to an activity, and to keep the other transitions unlabelled. The class of Petri nets including this distinction is called *labelled P/T nets*.

Typically, a process owner will be more aware of the activities in the process and less of the routing constructs connecting them and even if we distinguish two classes of transitions, if a model contains many unlabelled transitions, the

process owner might not be able to see the wood for the trees. Therefore, even a labelled P/T net is not really suitable to communicate a process to its owner. Instead, other modelling languages are more appropriate, as they are better suited to cope with more complex routing constructs. One example of such a modelling language is the language of Event-driven Process Chains [5], or EPCs for short. In our opinion, EPCs are better suited to communicate a process to its owner, as EPCs contain functions to model activities, and connectors to model routing. Furthermore, EPCs allow for OR connectors, which enables complex routing constructs to be expressed in a rather simple way.[1]

This paper investigates a sensible way to translate labelled P/T nets into EPCs. The EPCs that result from this translation should contain as few connectors as is possible, while retaining as much of the behavior as possible. Of course, this is a trade-of situation, in which some choices need to be made. This paper presents a set of choices which we believe to be appropriate, and which result in proper EPCs.

The remainder of this paper is organized as follows. First, we introduce a running example that we use to explain the issues at hand. Figure 1 shows the labelled P/T net for this running example. Second, we introduce the concepts of labelled P/T nets and EPCs. Third, we introduce the concepts of spheres and sphere types, which underly the choices made in this paper. Basically, a sphere of some transition is the largest part in the net that converges onto that transition, and we will translate every sphere into one EPC connector. Finally, we introduce the algorithm that actually translates a labelled P/T net into an EPC using these spheres and sphere types and we conclude the paper with some general conclusions.

1 Preliminaries

As we mentioned in the introduction, we will consider a special class of Petri nets, called labelled P/T nets, which we translate to EPCs using the concept of spheres. In this section, we introduce these concepts one by one, using a running example which is presented in Figure 1.

1.1 Petri nets

Petri nets are a formal language that can be used to specify processes. Since the language has a formal and executable semantics, processes modelled in terms of a Petri net can be executed by an information system. For an elaborate introduction to Petri nets, the reader is referred to [3, 6, 7]. The Petri nets we use in this paper correspond to a subclass of Petri nets, namely labelled Place/Transition nets. A labelled Place/Transition net (or labelled P/T net) consists of a number of places, a number of transitions, a flow relation between these places and transitions, and a number of transition labels.

[1] Note that this holds only on the conceptual level. When looking at the execution level, the semantics of the OR join are extremely complex.

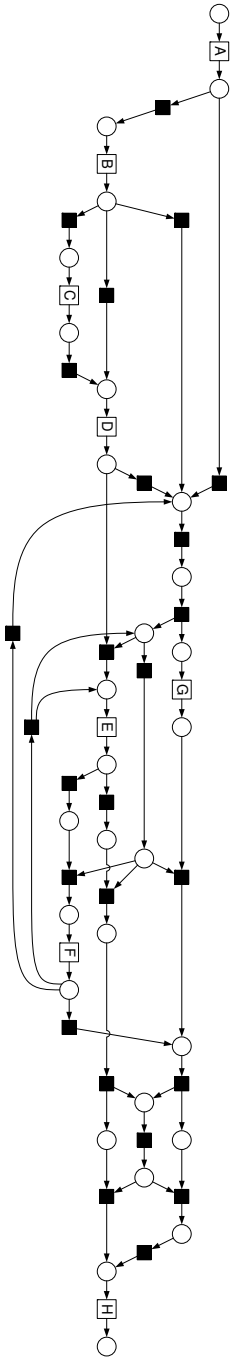

Fig. 1. The running example.

Definition 1 (Labelled P/T net). *Let P, T, L be finite, non-empty and mutually disjoint sets. Furthermore, let τ be a constant, such that $\tau \in L$. Let $F_i : T \to P$, $F_o : T \to P$, and $l : T \to L$. The tuple (P, T, F_i, F_o, L, l) is a labelled P/T net. Set P is the set of places, set T is the set of transitions, function F_i maps every transition to a set of input places, function F_o maps every transition to a set of output places, and function l maps every transition onto a label. We say transition t is labelled if and only if $l(t) \neq \tau$.*

Figure 1 shows a process described as a labelled P/T net, containing the labels "A" through "H". The transitions that are colored black correspond to the transitions that serve routing purposes only and these are the unlabelled transitions (i.e. $l(t) = \tau$). For places and transitions, the following preset/postset notation is standard, and is also used in this paper.

Definition 2 (Preset, postset). *Let $N = (P, T, F_i, F_o, L, l)$ be a labelled P/T net. For a transition $t \in T$, its preset is defined as $F_i(t)$, denoted $\bullet t$. For a place $p \in P$, its preset is defined as $\{t \in T | p \in F_o(t)\}$, denoted $\bullet p$. For a transition $t \in T$, its postset is defined as $F_o(t)$, denoted $t \bullet$. For a place $p \in P$, its postset is defined as $\{t \in T | p \in F_i(t)\}$, denoted $p \bullet$.*

Using the normal semantics of Petri nets, we know that transitions can fire if and only if their input places are marked. This is no different for labelled P/T nets. However, since the goal of these P/T nets is to describe a process, we are only interested in the relation between labelled transition, i.e. which labelled transition can be fired next? It is not important which unlabelled transitions have to be fired in between. We exploit this concept in the translation to EPCs.

1.2 Event-driven Process Chains

The idea behind Event-driven Process Chains (EPCs) is to provide an intuitive modelling language to model business processes. EPCs were introduced by Keller, Nüttgens and Scheer in 1992 [5]. It is important to realize that the language is not intended to be a *formal* specification of a business process. Instead, it serves mainly as a means of communication. In contrast of Petri nets, EPCs lack a simple executable semantics. However in this paper, we are interested in communicating our process to a process-owner, which is the reason EPCs were developed.

An EPC consists of three main elements:

Functions, which are the basic building blocks. A function corresponds to an activity (task, process step) which needs to be executed. A function is drawn as a box with rounded corners.

Events, which describe the situation before and/or after a function is executed. Functions are linked by events. An event may correspond to the postcondition of one function and act as a precondition of another function. Events are drawn as hexagons.

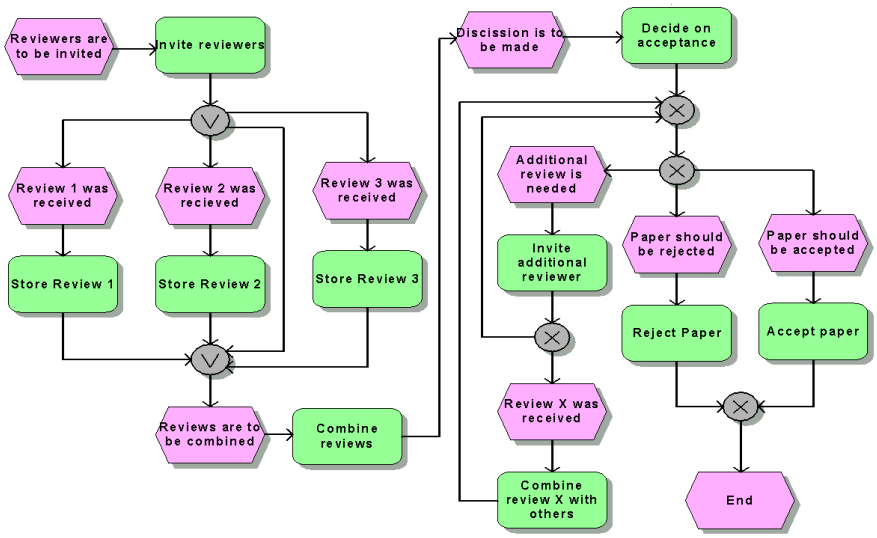

Fig. 2. An EPC describing a journal's reviewing process.

Connectors, which can be used to connect functions and events. This way, the flow of control is specified. There are three types of connectors: $\wedge$ (AND), $\times$ (XOR) and $\vee$ (OR). Connectors are drawn as circles, showing the type in the center of the circle.

By connecting these elements, the EPC defines the flow of a business process as a chain of events and functions reacting on these events. Functions, events and connectors can be connected with edges in such a way that

1. events have at most one incoming edge and at most one outgoing edge, but at least one incident edge (i.e. an incoming and/or an outgoing edge),
2. functions have precisely one incoming edge and precisely one outgoing edge,
3. connectors have either one incoming edge and multiple outgoing edges, or multiple incoming edges and one outgoing edge, and
4. on every path, functions and events alternate (in between two functions, there has to be an event, and vice versa).

An example of an EPC can be found in Figure 2, describing a review process of a journal, containing the activities of an editor.

2 The translation algorithm

The goal of our translation from labelled P/T nets to EPCs is to obtain an EPC that is easier to comprehend than the original labelled P/T net. Obviously, we could translate every labelled transition into a function, every silent transition into a pair of AND-connectors, and every place into a pair of XOR connectors.

However, the resulting EPC would typically not be comprehensive, as it will contain a lot of connectors. Instead, to obtain a comprehensible EPC, we need to restrict the number of connectors that are created during the translation. Thus, we need to restrict the number of connectors that are introduced. From the definitions of EPCs and labelled P/T nets, it is however clear that we need to translate labelled transitions into functions, since they both represent the activities in a process.

As we stated before, the unlabelled transitions in a P/T net are only there for routing purposes. To enable one labelled transition after firing another labelled transitions, some unlabelled transitions might have to fire. Therefore, for each labelled transitions, we can relate unlabelled transitions (and their connecting places) to it, since they describe which labelled transition could have fired before, and can fire next, thus enabling us to translate this behaviour to connectors of an EPC.

More precisely, based on the unlabelled transitions that can be related to a labelled transition, a connector of a certain type will be created before and/or after the corresponding function. Since we can put connectors *before* and *after* the corresponding function, we have to define two relations between unlabelled transitions on the one hand and labelled transitions on the other hand. The set of unlabelled transitions that are related to some labelled transition to decide the type of the join connector that will be created before the corresponding function, is called the *join sphere* of that labelled transition. Likewise, the *split sphere* of a labelled transition includes those silent transitions that are used to decide the type of the split connector that will be created after the corresponding function.

Later on, we see that we also want to determine both spheres of some silent transitions. Therefore, we define these spheres on silent transitions as well. Note however, that silent transitions are never translated into a function. Instead, such a silent transition would be translated into a join-split connector pair.

2.1 Join/split spheres

For every transition that we want to translate, we first determine its join sphere and its split sphere. Using these spheres, we later on determine the join and split type of the connectors that will be created before and after the corresponding function. Conceptually, the join sphere of a transition t contains all nodes for which all outgoing paths converge on transition t. Similarly, the split sphere of a transition t contains all nodes for which all incoming paths converge on transition t. Except for transition t itself (which may be labelled), the spheres of a transition t are only allowed to contain silent transitions.

Definition 3 (Join/split sphere). *Let $N = (P, T, F_i, F_o, L, l)$ be a labelled P/T net and let $t \in T$ be a transition of N. The join sphere of transition t, denoted $\triangleright t$, is the maximal set $S \subseteq P \cup T$ that satisfies the following conditions:*

- $t \in S$,
- $\forall_{p \in S \cap P} : |\bullet p| > 0$ *(no source places),*

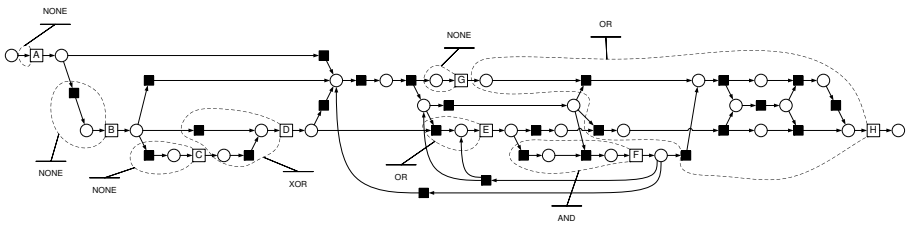

Fig. 3. The join spheres/types for the running example.

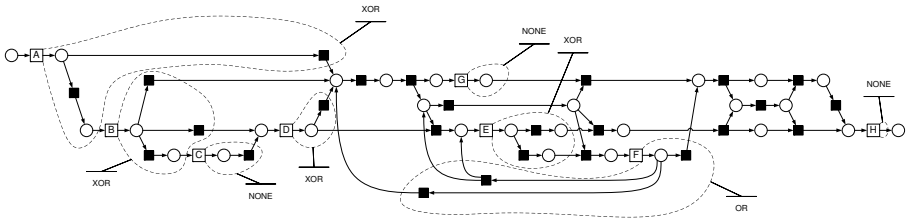

Fig. 4. The split spheres/types for the running example.

- $\forall_{u \in (S \cap T) \setminus \{t\}} : l(u) = \tau$ *(t is the only non-silent transition), and*
- $\forall_{n_1 \in S \setminus \{t\}} : \forall_{n_2 \in n_1 \bullet} : n_2 \in S$ *(all successors are part of the same sphere).*

The split sphere of transition t, denoted $t \triangleleft$*, is the maximal set* $S \subseteq P \cup T$ *that satisfies the following conditions:*

- $t \in S$,
- $\forall_{p \in S \cap P} : |p \bullet| > 0$ *(no sink places),*
- $\forall_{u \in (S \cap T) \setminus \{t\}} : l(u) = \tau$ *(t is the only non-silent transition), and*
- $\forall_{n_1 \in S \setminus \{t\}} : \forall_{n_2 \in \bullet n_1} : n_2 \in S$ *(all predecessors are part of the same sphere).*

Figures 3 and 4 show the join and split spheres for all labelled transitions in the running example.

2.2 Join/split types

For each of the join and split spheres of each transition, we can define the join/split behaviour of each transition. Later, we use that in our translation of EPCs. Based on the join sphere of transition t, we can determine its join type:

NONE If every node in the sphere (except for the transition t itself, which is allowed to have multiple outgoing arcs) has exactly one incoming arc and one outgoing arc (that is, if the sphere forms a simple sequence), then the join type is NONE. (A connector of type NONE will be reduced later on in the translation.)

AND If every place has exactly one incoming arc and one outgoing arc and if every transition has exactly one outgoing arc but possibly multiple incoming arcs (that is, if the sphere forms a tree where only transitions branch), then the join type is AND.

XOR If every transition has exactly one incoming arc and one outgoing arc and if every place has exactly one outgoing arc but possibly multiple incoming arcs (that is, of the sphere forms a tree where only places branch), then the join type is XOR.

OR Otherwise (that is, if the sphere is not a tree or if both transitions and places branch), the join type is OR.

In a similar way, the split type can be determined from the split sphere. Formally, the join/split types are defined as followes:

Definition 4 (Join/split type). *The join type of transition t, denoted $j(t)$ is:*

NONE iff $|\bullet t| = 1 \wedge \forall_{p \in (\triangleright t) \cap P} |\bullet p| = |p \bullet| = 1 \wedge \forall_{u \in ((\triangleright t) \cap T) \setminus \{t\}} |\bullet u| = |u \bullet| = 1$,
AND iff $|\bullet t| \geq 1 \wedge \forall_{p \in (\triangleright t) \cap P} |\bullet p| = |p \bullet| = 1 \wedge \forall_{u \in ((\triangleright t) \cap T) \setminus \{t\}} |\bullet u| \geq 1 \wedge |u \bullet| = 1$,
XOR iff $|\bullet t| = 1 \wedge \forall_{p \in (\triangleright t) \cap P} |\bullet p| \geq 1 \wedge |p \bullet| = 1 \wedge \forall_{u \in ((\triangleright t) \cap T) \setminus \{t\}} |\bullet u| = |u \bullet| = 1$, *and*
OR *otherwise.*

The split type of transition t, denoted $s(t)$ is:

NONE iff $|t \bullet| = 1 \wedge \forall_{p \in (t \triangleleft) \cap P} |p \bullet| = |\bullet p| = 1 \wedge \forall_{u \in ((t \triangleleft) \cap T) \setminus \{t\}} |u \bullet| = |\bullet u| = 1$,
AND iff $|t \bullet| \geq 1 \wedge \forall_{p \in (t \triangleleft) \cap P} |p \bullet| = |\bullet p| = 1 \wedge \forall_{u \in ((t \triangleleft) \cap T) \setminus \{t\}} |u \bullet| \geq 1 \wedge |\bullet u| = 1$,
XOR iff $|t \bullet| = 1 \wedge \forall_{p \in (t \triangleleft) \cap P} |p \bullet| \geq 1 \wedge |\bullet p| = 1 \wedge \forall_{u \in ((t \triangleleft) \cap T) \setminus \{t\}} |u \bullet| = |\bullet u| = 1$, *and*
OR *otherwise.*

Figures 3 and 4 also show the join and split types for all labelled transitions in the running example.

2.3 Edges

At this point, we know how to translate labelled transitions and we know how we can deal with unlabelled transitions using spheres. What remains is the flow relation, i.e. when are two elements of our EPC to be connected by an edge? Again, this can be decided using the spheres. If a split sphere of transition t overlaps the join sphere of transition u, then there will be a path from t to u. If we can find a node in the split sphere of transition t such that one of its output nodes belongs to the join type of transition u, then there will also be a path from t to u. If there is a path from transition t to transition u, then there will have to be an EPC edge from the corresponding split connector t to the corresponding join connector u. In a similar way, paths to and from non-functions can be determined.

2.4 Algorithm

Finally, we present our algorithm for translating a labelled P/T net into an EPC. The algorithm is presented in pseudo code in Figure 5 and has been implemented in the "Labelled WF-net to EPC" conversion plug-in of the ProM framework (See [4] and www.processmining.org for details about this open source tool).

One part of the algorithm we did not explain in detail yet is the creation of initial and final events. As shown in the algorithm, the source place needs to be translated into a combination of a start event, a start function, and a split connector. However, in some situations, this is not necessary. For this reason,

1. Let $J = \emptyset$ and $S = \emptyset$
2. For every labelled transition t
3. Add $\triangleright t$ to J and $t \triangleleft$ to S
4. Create a join connector j_t with type $j(t)$
5. Create an event e_t and a function f_t
6. Create a split connector s_t with type $s(t)$
7. Create edges from j_t to e_t, from e_t to f_t, and from f_t to s_t
8. As long as $T \setminus (J \cup S)$ is non-empty
9. Take a $t \in T \setminus (J \cup S)$ such that $|\triangleright t \cup t \triangleleft|$ is maximal
10. Add $\triangleright t$ to J and $t \triangleleft$ to S
11. Create a join connector j_t with type $j(t)$
12. Create a split connector s_t with type $s(t)$
13. Create an edge from j_t to s_t
14. As long as $P \setminus (J \cup S)$ is non-empty
15. Add p to J (or S)
16. Take a $p \in P \setminus (J \cup S)$
17. Create a join connector j_p with type XOR
18. Create a split connector s_p with type XOR
19. Create an edge from j_p to s_p
20. For every $t, u \in T$ such that $t \triangleleft \cap \triangleright u \neq \emptyset$
21. Create an edge from s_t to j_u
22. For every $t, u \in T$ such that $n_1 \in t \triangleleft$ and $n_2 \in \triangleright u$ exist such that $n_1 \in \bullet n_2$
23. Create an edge from s_t to j_u
24. For every $t \in T$ and $p \in P$ such that $t \triangleleft \cap \bullet p \neq \emptyset$
25. Create an edge from s_t to j_p
26. For every $t \in T$ and $p \in P$ such that $\triangleright t \cap p \bullet \neq \emptyset$
27. Create an edge from s_p to j_t
28. For every $p \in P$ such that $\bullet p = \emptyset$
29. If $|p \bullet| > 1$ or $\forall_{t \in p \bullet} l(t) = \tau$
30. Create a start event e_p and a start function f_p
31. Create an edges from e_p to f_p and from f_p to s_p
32. For every $p \in P$ such that $p \bullet = \emptyset$
33. Create an end event e_p
34. Create an edge from s_p to e_p
35. Reduce connectors that don't have multiple inputs or multiple outputs.

Fig. 5. The essence of the translation algorithm.

the source place is not translated into an event-function-connector combination if there is only one output transition, and if that output transition is labelled. Note that the event corresponding to that output transition will then act as start event.

2.5 Example

If we apply our algorithm on the example given in Figure 1, the algorithm will do the following:

– First, following lines 1 through 7, for each labelled transition, its join and split spheres are discovered and for each of these spheres a connector of the right type is created. Furthermore, for the transition an input event is created and a function is created. Recall that figures 3 and 4 show the join and split spheres of each labelled transition.

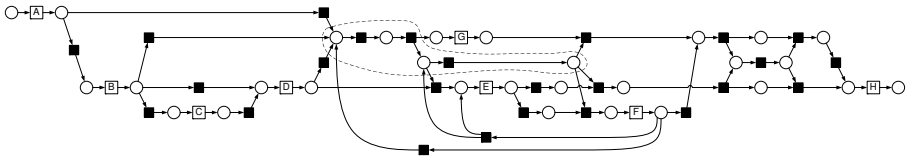

Fig. 6. The uncovered nodes.

– So far, we dealt with all labelled transitions and most of the unlabelled ones. However, some unlabelled transitions were not covered by any sphere. Figure 6 shows the nodes (places and transitions) that are not covered by a sphere so far. Lines 8 through 13 deal with the transitions in this set.

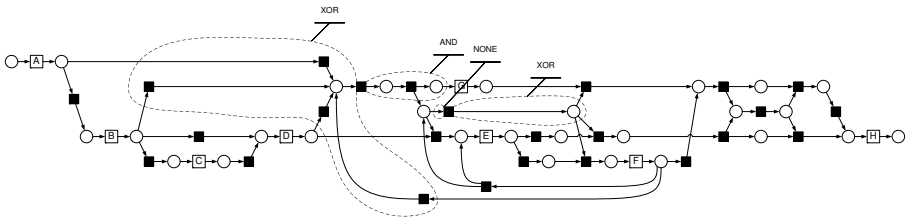

Fig. 7. The two spheres covering the uncovered transitions.

Every iteration, one of them is selected, in such a way that the size of its join and split sphere combined is maximal. Then, this transition is translated

into a AND-join, AND-split combination and the same procedure is follows as for a labelled transition. The result of this step is shown in Figure 7.
– Now, there can still be some places that were not translated yet, as shown in Figure 8. In lines 14 through 19, these places are dealt with using a similar procedure as for the uncovered transitions. The difference is that:
  • an source place is either translated into an event, a function, and an split connector, or it is translated into no EPC objects,
  • an sink place is translated into an event, and
  • any other place is translated into an XOR-join, XOR-split combination.

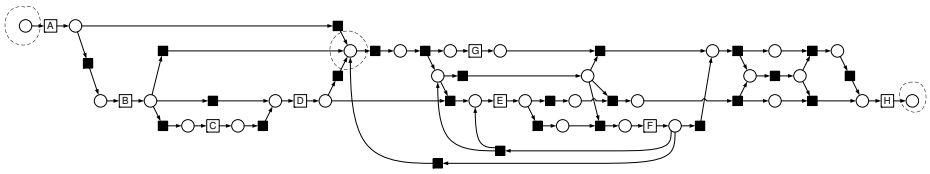

Fig. 8. The uncovered places.

– Finally, the edges are created as described before and the EPC is cleaned up by removing connectors with single input and output edges. The result of which is shown in Figure 9.

3 Conclusion

This paper introduces a translation from labelled P/T nets to EPCs. The goal of this translation is to improve the communication with the process owner. If we use a labelled P/T net to model a process, then the resulting model will be formal and executable, but it might contain few labelled transitions (which correspond to activities in the process) and many unlabelled transitions (which correspond to routing constructs). As a result, the process owner might have severe problems understanding this model of his process. By translating this labelled P/T net into an EPC in such a way that many unlabelled transitions are translated into one EPC connector, we will obtain an EPC that is more compact than the original labelled P/T net. As a result, the process owner will have less difficulties in understanding the model of his process.

Although the translation favors the AND and XOR connectors, it can result in EPCs that contains OR connectors. Although OR connectors in EPCs are problematic when it comes down to their formal semantics [1], humans seem to be able to grasp the meaning of an OR connector without that much difficulty. We expect that the process owner will be able to understand the EPC model

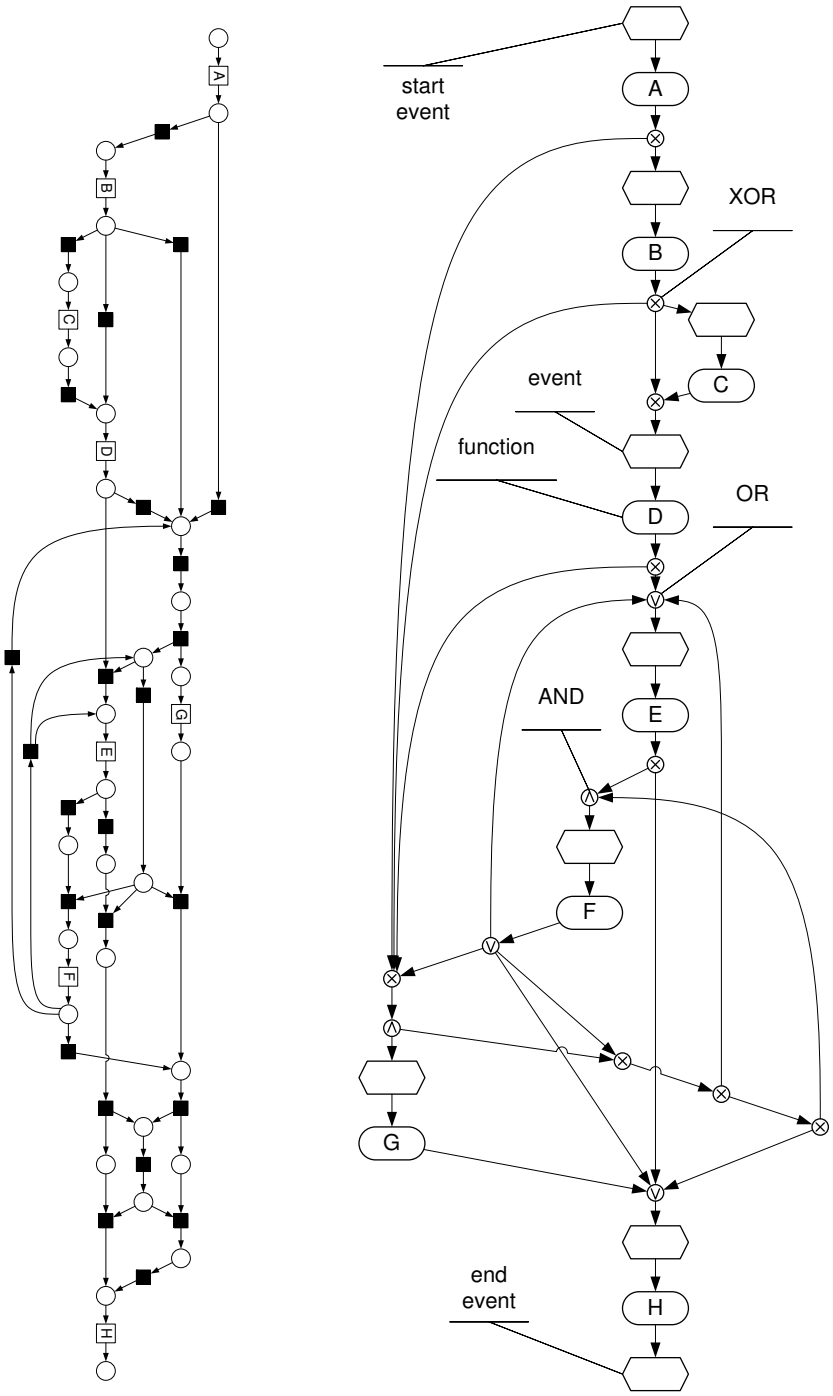

Fig. 9. The original P/T net and the resulting EPC.

of his process much quicker than a labelled P/T net model, even if that EPC contains several OR connectors.

A similar translation algorithm can be given for YAWL models [2] instead of EPCs. A labelled transition can be translated into a YAWL task and its input and output behavior can be determined using its sphere types. As long as they are not covered, unlabelled transitions can be translated into dummy YAWL tasks in a similar way. Places that are still uncovered can then be translated into YAWL conditions, where the input place is translated into an input condition and the output place into an output condition. This translation has also been implemented in a conversion plug-in of the ProM framework.

References

1. W.M.P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In M. Nüttgens and F.J. Rump, editors, *Proceedings of the EPK 2002: Business Process Management using EPCs*, pages 71–80. Gesellschaft für Informatik, Bonn, 2002.
2. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
3. J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, UK, 1995.
4. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The PRoM framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, Berlin, Germany, 2005.
5. G. Keller, M. Nüttgens, and A.W. Scheer. Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken, 1992.
6. T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
7. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science. Advances in Petri Nets*. Springer, Berlin, Germany, 1998.