Petri Net Transformations for Business Processes – A Survey

Niels Lohmann[1], Eric Verbeek[2], and Remco Dijkman[3]

[1] Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
`niels.lohmann@uni-rostock.de`
[2] Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`h.m.w.verbeek@tue.nl`
[3] Technische Universiteit Eindhoven
Department of Technology Management
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
`r.m.dijkman@tue.nl`

Abstract. In Process-Aware Information Systems, business processes are often modeled in an explicit way. Roughly speaking, the available business process modeling languages can be divided into two groups. Languages from the first group are preferred by academic people but shunned by business people, and include Petri nets and process algebras. These *academic* languages have a proper formal semantics, which allows the corresponding *academic* models to be verified in a formal way. Languages from the second group are preferred by business people but disliked by academic people, and include BPEL, BPMN, and EPCs. These *business* languages often lack any proper semantics, which often leads to debates on how to interpret certain *business* models. Nevertheless, business models are used in practice, whereas academic models are hardly used. To be able to use, for example, the abundance of Petri net verification techniques on business models, we need to be able to transform these models to Petri nets. In this paper, we investigate a number of Petri net transformations that already exist. For every transformation, we investigate the transformation itself, the constructs in the business models that are problematic for the transformation and the main applications for the transformation.

1 Introduction

Today, Business Process Management (BPM) is becoming more and more important to the business, which explains the increased popularity of business process modeling, and a plethora of similar but subtly different process modeling approaches has been proposed, including the Web Services Business Process Execution Language (BPEL) [1], the Event-driven Process Chains (EPCs) [2], the Yet Another Workflow Language (YAWL) [3], the Business Process Modeling Notation (BPMN) [4], process algebras [5] and Petri nets [6, 7]. The resulting babel has raised the issue of comparing the relative expressiveness between languages and translating models defined in one language into *equivalent* models defined in another language.

Academic people prefer languages like process algebras and Petri nets, as these languages have proper formal semantics and, hence, one can check relevant and interesting properties on corresponding models. Business people, however, prefer languages like BPEL, EPCs, and BPMN, which more than often lack these proper formal semantics. As a result, business processes found in practice are modeled in a way that leaves room for interpretation. An exception to this is YAWL, which has its roots in the academic world, but is actually used in practice, and has a semantics in terms of reset nets (which will be explained further on in this paper). YAWL supports the most frequent control-flow patterns found in the current workflow practice. As a result, most workflow languages can be mapped onto YAWL without loss of control-flow details, even languages allowing for advanced constructs such as cancelation regions and OR-joins. If we want to combine the best of both worlds, that is, to combine the ability to calculate certain properties on a formal semantics and the actual, existing in practice, *business* models, we need to transform these business models to languages with a formal semantics. In this paper, we will cover a number of transformations from business models in BPEL, EPCs, YAWL, and BPMN, focusing on transformations onto Petri nets.

The remainder of this paper is organized as follows. Section 2 introduces the concepts (like Petri nets and relevant properties). These concepts are presented in an informal way, for a more formal description, we refer to the literature on the transformations itself. This section also explains ‘workflow patterns’ (frequently used constructs in business process modeling) and how these patterns can be transformed into Petri nets. Finally, this section presents a running example, which we use throughout the paper to illustrate the transformations. Sections 3, 4, 5 and 6 introduce the business modeling languages, their transformations into Petri nets and the constructs that are difficult or impossible to transform. Each transformation is explained by reference to the workflow patterns that the language supports. For additional details on these transformations, we refer to the existing literature on these transformations. Section 7 concludes the paper.

2 Preliminaries

This section first presents the different types of Petri net used to formalize the business process modeling languages. Second, it explains some well-known patterns from the area of business process modeling, their representation using Petri nets and possible difficulties to represent them. Third, it presents an example business process that we will use throughout this paper to illustrate the modeling languages.

2.1 Petri Net Classes

We assume the standard definition of *Petri nets* [6, 7] to consist of two finite disjoint sets of places and transitions (graphically represented by circles and squares) together with a flow relation (represented as directed arcs).

A Petri net is called a *workflow net* [8] if it has a distinct source place, a distinct sink place, and if all nodes lie on some path from this source place to the sink place. Typically, a token in the source place signifies a new case, whereas a token in the sink place signifies a completed case. All transitions in the workflow net should contribute to forwarding some case from the new state to the completed state.

A workflow net net is called *sound* [8] if any case can always complete in a proper way (that is, without tokens being marooned) and if no transition is dead. Note that the workflow net requirement is a structural requirement, whereas the soundness requirement is a behavioral requirement. A workflow net is called *relaxed sound* [9] iff every transition can help in forwarding some case from the new state to the completed state. Note that this requirement is less strict than the soundness requirement, as the option to complete properly might not be guaranteed for every reachable marking.

A *reset net* [10] is a Petri net extended by reset arcs. A transition that is connected to a place with a reset arc removes *all* tokens on that place upon firing. Reset nets are more expressive than classical Petri nets: some forms of verification are undecidable in reset nets, while they are decidable in classical Petri nets.

An *open net* [11] is a Petri net extended with a set of interface places and a set of desirable final markings. The interface places are partitioned to input and output places. Open nets thereby extend classical workflow nets with an asynchronous interface to explicitly model message exchange. An important correctness criterion for open nets is *controllability* [12]. An open net is controllable if another open net exists such that their composition (where the communication places of both nets have been glued) always ends up in a desired final marking. Note that (relaxed) soundness does not imply controllability, or vice versa.

2.2 Workflow Patterns and Petri Nets

A collection of workflow patterns has been developed to analyze the expressive power of languages for workflow and business process modeling. Patterns with respect to the control-flow aspect, on which we focus in this paper, are described in [13, 14]. The expressive power of modeling languages can be explained in terms of which patterns they support. (For an overview of support from languages in this paper, see [14].) Therefore, we use the workflow patterns from [14] as a frame of reference here. We show how some of them can be mapped to Petri nets, or what the problems are when they cannot be mapped. This provides the reader with information about which languages support which patterns and the particularities of mapping these patterns to Petri nets. Hence, it gives a good overview of the state-of-the-art in Petri net transformation of business process modeling languages.

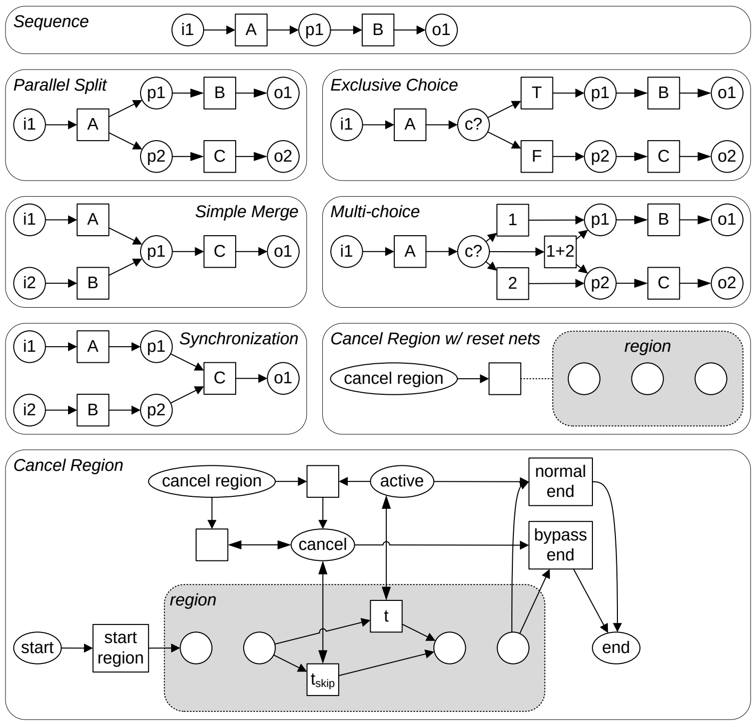

Fig. 1. Workflow patterns in Petri nets.

Figure 1 shows the mapping of some of the workflow patterns to Petri nets. Transitions that represent tasks are given the labels A, B or C and transitions that affect the flow of control, but that do not represent tasks are given more descriptive labels. The so-called *simple patterns* can easily be represented in Petri nets. These are:

– 'Sequence', a task is enabled after another task is completed;
– 'Parallel Split' (also called 'AND-split'), all outgoing branches are enabled at the same time;
– 'Synchronization' (also called 'AND-join'), the process must wait for all incoming branches to complete before it can continue;
– 'Exclusive Choice' (also called 'XOR-split'), the execution of one out of a number of branches is chosen;
– 'Simple Merge' (also called 'XOR-join'), the process continues when one incoming branch completes.

Another pattern that can easily be represented by Petri net is the 'Deferred Choice' pattern, which differs from the 'Exclusive Choice' pattern in the fact that the branch is chosen by the environment rather than by the system itself.

Patterns that are harder to represent in Petri nets include the 'Multi-choice', in which the execution of a number of branches is chosen, and the 'Cancel Re-

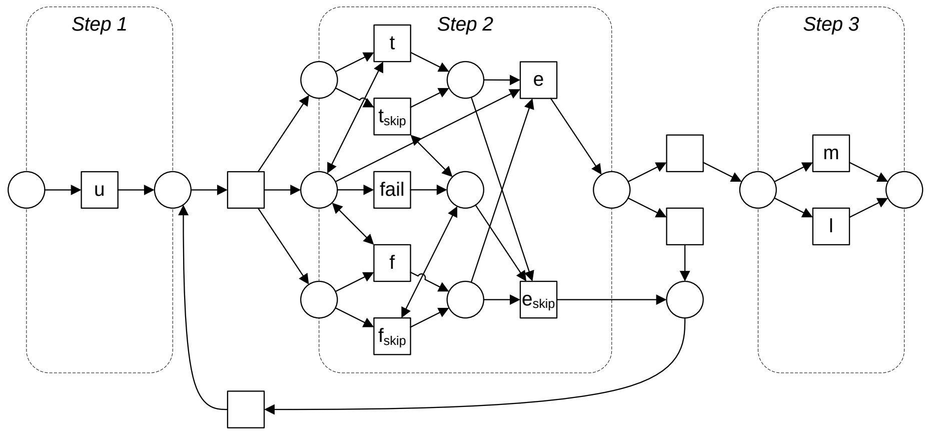

Fig. 2. Example process modeled as a Petri net.

gion', in which the execution of a set of tasks is disabled. These patterns lead to nets that become hard to read. For example, although the 'Multi-choice' in Fig. 1 is still readable, it becomes hard to read when there any number out of three or four outgoing branches can be chosen. To make these patterns more readable, another class of Petri net can be chosen. Figure 1 illustrates this for the 'Cancel Region', which is both represented as a classical Petri net and as a reset net.

Patterns that cannot be represented as a classical Petri net include the 'General Synchronizing Merge', which corresponds to a wait-and-see synchronizing construct. To represent such patterns, we would need very sophisticated classes of Petri nets, for which analysis might just be infeasible.

2.3 Example Process

We will illustrate each of the modeling techniques, using the same example process. As example process we take an image editing process. First, the customer uploads an image (u). Second, the following procedure is applied: The image is finished (f) and concurrently a thumbnail is created (t). Afterwards the results are evaluated (e). If a failure occurred or if the evaluation is negative, the procedure is repeated. Third, if the image is too big, it is stored temporarily and only a link is sent to the customer (l); otherwise the image is sent by e-mail (m). At any point during the second step, a failure occurs if the format in which the figure is stored cannot be imported by the tool that are used to process the image. The other steps are assumed to be infallible. Figure 2 shows the example modeled as a workflow net.

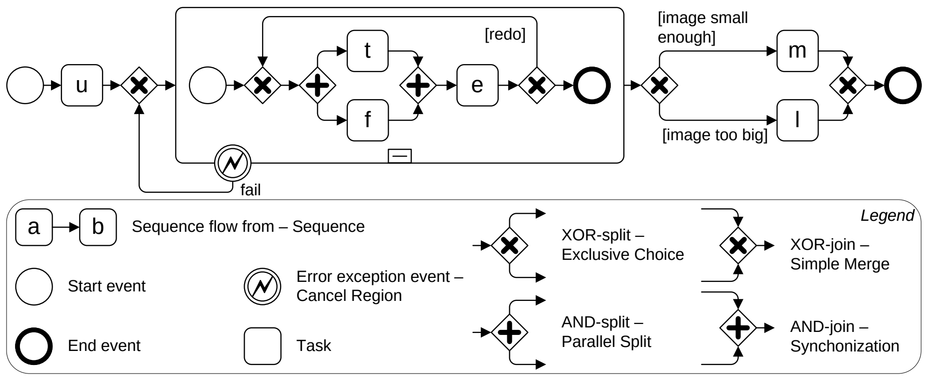
[redo]
t
f
u
e
[image small enough]
[image too big]
m
l
fail
Legend
a
b
Sequence flow from – Sequence
Start event
End event
Error exception event – Cancel Region
Task
XOR-split – Exclusive Choice
AND-split – Parallel Split
XOR-join – Simple Merge
AND-join – Synchonization

Fig. 3. The example process as a BPMN process.

3 BPMN

The Business Process Modeling Notation (BPMN) [4] is developed as a standard for business process modeling. This section briefly explains the language, the main challenges when transforming BPMN models to Petri nets and the transformation itself. The section focuses on BPMN version 1.0, because at the time that the transformation was developed that was the current version. Therefore, comments on BPMN apply to version 1.0 only.

3.1 Language

BPMN is a rich language that provides the modeler with a large collection of object types to represent various aspects of a business process, including the control-flow, data, resources and exceptions. BPMN is mainly meant for modeling business processes at a conceptual level, meaning that it is mainly intended for drawing process models that will be used for communication between stakeholders in the processes. As a consequence, formal rigor and conciseness were not primary concerns when developing the BPMN specification.

The three types of BPMN objects that can be used to represent the control-flow aspect of a process are *activities*, *events*, *gateways*. Many subtypes of these objects exist. Control-flow objects can be connected by sequence flows, which are directed arcs that represent the flow of control from one object to the next. Figure 3 illustrates some of these objects, by representing the example process in BPMN and by relating the object types to the workflow patterns explained in Sect. 2.2.

3.2 Transformation Challenges

Due to the large number of object types that constitute BPMN it is hard to define a mapping and show (or prove) that the mapping works for all possible combinations of these object types. Especially, because the mapping of a composition of object types is not the same as the composition of the mapping of those object types. This complicates, for example, defining mapping rules for interruptions of sub-process invocations.

BPMN frequently introduces shorthands and alternatives for representing certain constructs. For example, an activity with multiple incoming flows will start as soon as the control is passed to one of these flows. Hence, an activity with multiple incoming flows behaves similar to an activity that is preceded by an XOR-join. This further complicates the mapping.

Version 1.0 of the BPMN standard contains inconsistencies and ambiguities. We uncovered several while defining the mapping [15]. This illustrates that defining a mapping to a formal language can be useful to uncover flaws in an informal language.

3.3 Transformation and Application

In prior work we defined a mapping from a restricted version of BPMN to workflow nets [15]. The restrictions include that the BPMN models must have a single start and a single end event. Also, activities with multiple concurrent instances and some types of gateways cannot be used (in particular OR-gateways, which represent the 'Multi-choice' and 'General Synchronizing Merge' patterns). The mapping focuses on the control-flow aspect of processes.

The mapping is developed by defining mappings for each activity, events and gateway object type, similar to the way in which Petri net mappings are defined for each workflow pattern in Sect. 2.2. When transforming a model, first each object is mapped onto a partial Petri net and second the partial Petri nets are composed into a complete model. Although this approach works for many constructs, some constructs cannot simply be mapped and then composed. The mapping from BPMN to workflow nets allows the soundness of these nets to be analyzed (see Sect. 2).

To the best of our knowledge the only other mapping from BPMN to a formal language is from BPMN to CSP [16].

4 EPCs

Event-driven Process Chains (EPCs) [2] were developed to provide an intuitive modeling language to model business processes. This section briefly explains ECPs, the main challenges when transforming EPCs to Petri nets and the transformation itself.

4.1 Language

Like BPMN, EPCs are meant for modeling business processes at a conceptual level: EPCs are not intended to be a formal specification of a business process. Instead, it serves mainly as a means of communication. There are, however, some conceptual differences between BPMN and EPCs:

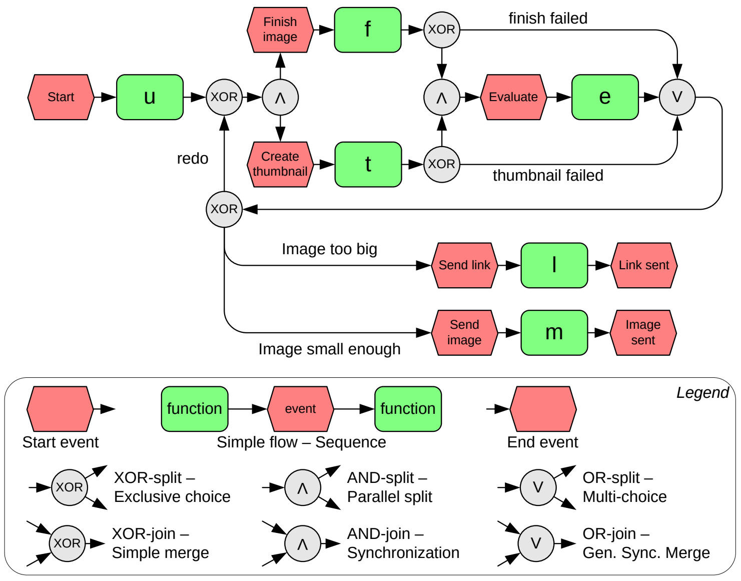

Fig. 4. The example process as an EPC.

- BPMN supports the 'Cancel Region', whereas EPCs do not, and
- EPCs supports the 'General Synchronizing Merge', whereas BPMN does not.

Three types of EPC objects can be used to model the control-flow aspect of a process: *functions*, *events*, and *connectors*. In a natural way, these types correspond to the BPMN *activities*, *events*, and *gateways*. However, EPCs do not allow for exceptions, and it supports only a limited set of connectors, which is shown by Fig. 4. Apart from the full set of connectors, this figure also shows an the example process as an EPC, and it relates the object types to the workflow patterns explained in Section 2.2.

4.2 Transformation Challenges

A main challenge in EPCs is the semantics of the constructs that support the 'Simple Merge' and 'General Synchronizing Merge' patterns, viz. the XOR-join connector and the OR-join connector. Everybody agrees that the XOR-join connector should be enabled if one of its inputs is enabled, but this agreement is lacking in case more than one inputs is enabled. Some say that the XOR-join should be executed for every single enabled input, while others say that the connector should block if multiple inputs are enabled. An even bigger problem is the OR-join connector, for which a definitive semantics has lead to extensive discussions in literature and to different solutions, all of which fail for some EPCs [17–19]. As a result, not everybody will agree on a given mapping, as not everyone will agree with the semantics used by it.

Furthermore, an EPC allows for multiple start events and multiple final events, but not all combinations of these events are possible. Although the process designer might know the possible combinations, an EPC does not contain this information.

4.3 Transformation and Application

The transformation to workflow nets as introduced in [20] explicitly targets the verification of EPCs, and assumes that an OR-join is enabled as soon as any of its inputs are enabled, and that an XOR-join with multiple inputs enabled will be executed multiple times. This transformation results in a safe[4] workflow net, as it introduces a so-called shadow place for every place, and uses a number of (optional) EPC reduction rules prior to transforming the EPC.

As mentioned above, an EPC allows for multiple start events and multiple final events. However, the EPC designer might be aware of the fact that certain combination of start events will not occur, and that the process will behave in such a way that certain combinations of final events are impossible as well. Clearly, this knowledge of the designer is vital for the verification, but unfortunately not included in the EPC. Therefore, the verification approach in [20] proposes to query the user for this information. First, the user has to select which combinations of start events are possible. Second, based on this information a state space is build that possibly contains multiple subsets of final events. Third, the user has to select the subsets of final events that indeed are possible. For the example process, three subsets of final events were detected (next to a number of deadlock states that also include non-final events, which are assumed to be ignored by default), from which one (the subset containing both Image sent and Link sent, which appears to be possible) needs to be ignored. Fourth, the soundness property is checked on the resulting state space. If the state space corresponds to a sound net, then the EPC is correct: A desired subset of final events will always be reachable. Otherwise, the relaxed soundness property is checked, where an OR-join (OR-split) transition is allowed to be non-relaxed sound if and only if its inputs (outputs) are covered by relaxed sound OR-join (OR-split) transitions. If the state space is relaxed sound, then the EPC can be correct, although it allows for undesirable behavior. Otherwise, the EPC is incorrect, as certain parts of the EPC cannot lead to any desired subset of final events, when executed.

The example EPC can be correct, but allows for undesired behavior. For example, if Finish image fails, then Create thumbnail should fail as well to be able to reach a desirable subset of final events (that is, {Link sent}, or {Image sent}).

5 BPEL

The Web Services Business Process Execution Language (BPEL) [1], is a language for describing the behavior of business processes based on Web services.

[4] A net is safe if and only if every place in every reachable marking contains at most one token. As a result, the set of reachable markings is finite.

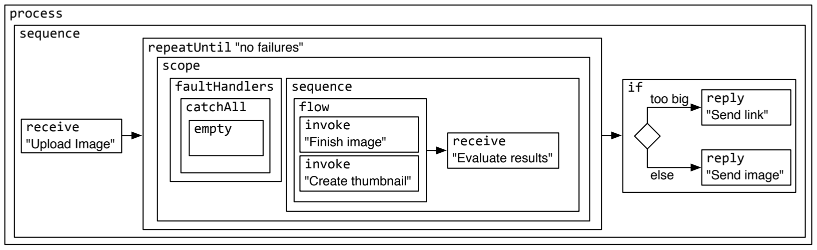

Fig. 5. The example process as a BPEL process.

That makes BPEL a language for the *programming in the large* paradigm. Its focus is — unlike modifying variable values in classical programming languages such as C or Java — the message exchange and interaction with other Web services. Advanced concepts such as instantiation, a complex exception handling, and long running transactions are further features that are needed to implement business processes.

5.1 Language

Activities organize the communication with partners, variable manipulation, etc. They can be ordered using structured activities which makes BPEL similar to a block-based language. To support the simple patterns depicted in Fig. 1, control links can be used to express splits, choices and merges. Due to restrictions, BPEL avoids the problems occurring with the OR-join.

Being an execution language, the exceptional behavior which also includes cancelation of parts of the process is described in great detail in the BPEL specification [1]. In addition, BPEL supports the concept of hierarchical scopes that model local units to which a local exception management (implemented by fault, termination, and compensation handlers) is bound.

Example Process BPEL is an XML-based execution language without standardized graphical representation. Figure 5 shows a possible implementation of the example process using BPEL, in a schematic way. The process "Image editing" contains a sequence, which in turn contains receive "Upload image", repeatUntil "no failures", and if "too big", etc.

5.2 Transformation challenges

The positive control flow of a BPEL process (i. e., the sheer business process) can be straightforwardly mapped to Petri nets by defining a translation of each of BPEL's activity type. The biggest challenge is the transition from the positive to the negative control flow. The BPEL specification defines the following steps to be performed in case a fault occurs. (1) All running activities in the scope of the faulty activity have to be stopped. (2) The fault handler of the scope is called. (3) If the fault could be handled, the execution continues with the scope's successor. If the fault could not be handled, it is escalated to the parent scope.

This procedure requires a global state (i. e., all running activities are stopped) to be reached before invoking a fault handler. Petri nets naturally model distributed systems with concurrently acting *local* components. The formalization of the enforcement of a *global* state with Petri nets is therefore cumbersome, because it requires all components to synchronize. The stopping of originally independently running activities can be achieved on two ways.

(a) The request to stop is propagated from the scope to each running activity.
(b) A global "variable" modeled by a place describes the "mode" of the scope; that is, whether the scope's internal activities should be executed or stopped.

Option (a) has the advantage that the BPEL process's nature of a distributed system is mirrored in the Petri net model: Activities embedded into a flow are executed concurrently. However, this concurrency introduces a lot of intermediate states that model the situation in which a fault is detected by a scope yet not fully propagated. With a global state like in (b), stopping can be modeled more easily, but implicitly *schedules* originally concurrent activities.

A similar problem arises when modeling the dead-path elimination [1] which requires to skip activities than cannot be executed due to their join condition. Again, this can be achieved through propagation (of the information whether a branch is executed or skipped) or global places (a global status place which controls whether an activity is executed or skipped).

5.3 Transformation and Application

Though there exist many works to formalize BPEL using Petri nets (see [21] for an overview), only two Petri net transformations are *feature-complete*; that is, covering all mentioned activities and aspects of a BPEL process. These formalizations are from the Humboldt-Universität zu Berlin together with the University of Rostock (abbreviated with "HR"), and from the Queensland University of Technology ("QUT"). A detailed comparison between these semantics can be found in [22]. Here, it suffices to mention that the HR transformations uses propagation (see Sect. 5.2) and selectively results in either a normal Petri net or an open net, whereas the QUT transformation uses global places and results in a workflow net.

HR Transformation to Petri nets The HR transformation implemented in the tool BPEL2oWFN [23] can be used to translate BPEL into a standard Petri net without interface places. This Petri net can be analyzed for deadlocks or other classical Petri net properties, soundness, as well as temporal logical formulas. A case study is presented in [24] shows that the internal behavior of large processes with nested scopes and complex exception handling can be analyzed using the model checking tool LoLA [25]. Furthermore, the semantics could be validated by proving deadlock-freedom of the patterns.

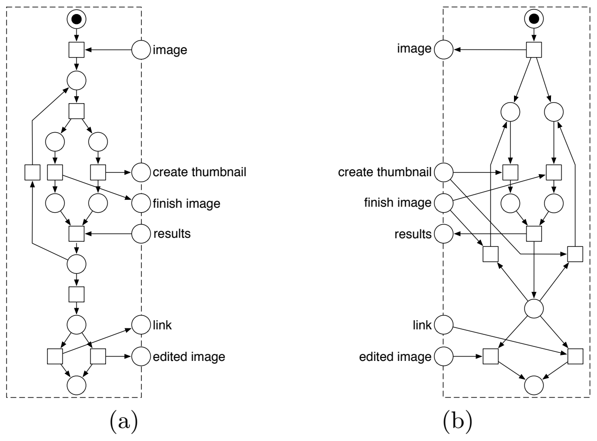

Fig. 6. The BPEL process transformed into an open net (a) and a synthesized partner open net (b). To increase legibility, fault handling is not depicted.

HR Transformation to Open Nets With the explicitly modeled interface of the open net, the communication behavior of the BPEL processes can be analyzed. The tool Fiona [23] can check controllability [12], synthesize a partner process which can be translated back to BPEL [26], or calculate the operating guideline [27] of the net. This operating guideline characterizes all partners that communicate deadlock-freely with the original net and can be used for service discovery. An extension to formalize choreographies [28] further allows to apply the mentioned analysis techniques to a choreography of many BPEL processes instead of just a single process.

Figure 6(a) shows the result of transforming the BPEL process to an open net. The net is controllable, and Fig. 6(b) shows a synthesized partner open net. The composition of the open nets is free of deadlocks, and a desired final marking of the composition always reachable.

QUT Transformation to Workflow Nets The QUT transformation was developed to decide soundness on the resulting workflow net, using the WofBPEL tool [29], which is a spawn-of of the workflow verification tool Woflan [30, 31]. However, the transformation does not allow for improper completion and BPEL processes by definition have the option to complete. Thus, the soundness check boils down to a check on dead transitions. Next to the soundness check, the WofBPEL tool can also check whether an incoming message can be handled by multiple elements (which is considered an anomaly in BPEL) and can augment the BPEL model with information on when to garbage collect queued messages. Based on this information, the BPEL garbage collector can decide to remove for a certain running instance certain incoming messages from the message queue as it is certain that these messages cannot be handled anymore by the instance.

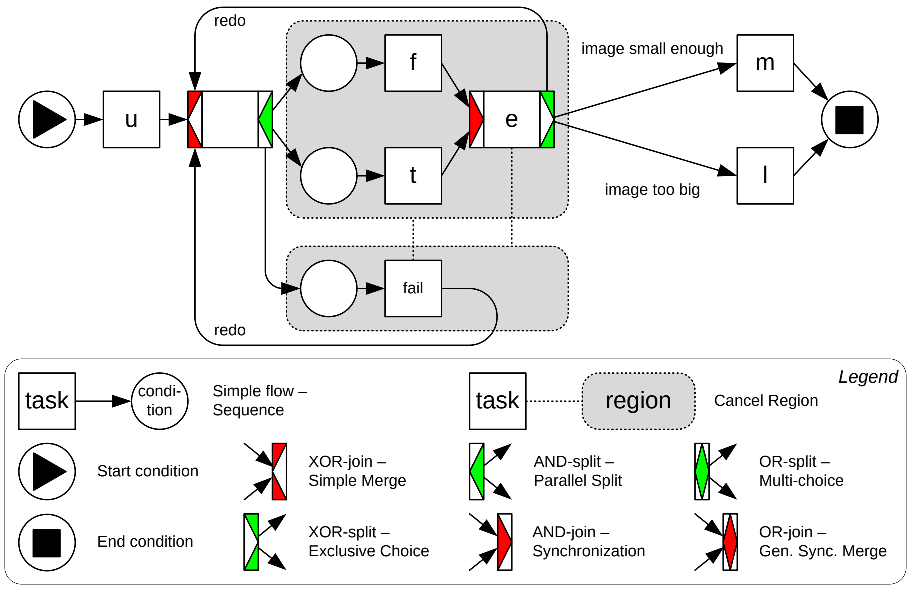

Fig. 7. The example process as a YAWL model.

6 YAWL

The Yet Another Workflow Language (YAWL) [3] was originally conceived as a workflow language that would support 19 of the 20 most frequent used patterns found in existing workflow languages. As such, YAWL supports the 'Multiple Instance' pattern, the 'General Synchronizing Merge' pattern, and the 'Cancel Region' pattern. The only pattern not supported by YAWL is the 'Implicit Termination' pattern (A process implicitly terminates when there is no more work to do and the process is not in a deadlock.), and the authors of YAWL deliberately chose not to support this pattern. Lately, the patterns have been revised and extended [14], and YAWL is being extended to support the new patterns

6.1 Language

In YAWL, two objects are used to model the control-flow aspect of a process: *tasks* and *conditions*. Loosely speaking the former correspond to *activities* (BPMN) and *functions* (EPC), and the latter to *events* (both BPMN and EPC). The BPMN *gateways* and EPC *connectors* are modeled by specifying the *join* and *split* behavior of a task. Like EPCs, YAWL supports AND, XOR, and OR splits and joins. Unlike EPCs, the semantics of the OR-join is well-defined, and an engine exists that supports the execution of any YAWL model. As such, a YAWL model can both act as a conceptual model and an IT model. Figure 7 shows a possible implementation of the example process using YAWL.

6.2 Transformation Challenges

The formalization of YAWL is straightforward, as it has a proper formal semantics. Challenges in YAWL include the OR-join and the cancelation regions. The YAWL OR-join comes with a semantics that includes backwards reasoning and coverability in reset nets, which is impossible to capture in a classical Petri nets. The cancelations regions are hard to capture (though possible) in classical Petri nets, but are straightforward to capture when using reset nets.

6.3 Transformation and Application

YAWL comes with a transformation to reset nets, which is straightforward except for the OR-join [32]. Furthermore, there is also a transformation to workflow nets that *covers* the behavior of the YAWL model [33]: any behavior exhibited by the YAWL model will also be present in the Petri net, but not vice versa. Finally, there is a transformation (see [34]) that is used to obtain a Petri-net-based simulation model (using CPN tools [35]) for an operational YAWL model. To keep things simple for the time being, this transformation assumes that there are no cancelation regions, and that an OR-join is enabled a soon as any of its inputs are enabled.

The transformation to reset nets that comes with YAWL is used by the YAWL engine to check which tasks are enabled [32]. For an AND-join task and an XOR-join task this check is quite simple (a task is enabled if and only if any of the corresponding transitions in the reset net is enabled), but for an OR-join task this check is quite complex and involves a coverability check on any corresponding input place in the reset net that is not marked. As coverability is decidable for reset nets, this procedure is decidable as well.

This transformation is also used to verify YAWL models [36]. In the absence of OR-joins, a YAWL model can be transformed to a reset net, which can (possibly) be verified for soundness. If the reset net is to complicated to be checked successfully, a set of reduction rules is given to simplify the reset net prior to checking soundness [37].

The transformation from [33] is also used to verify YAWL models, but is restricted to relaxed soundness. If the state space is too complex to be constructed, transitions invariants can be used to estimate relaxed soundness. This approach is correct (errors reported are really errors), but not necessarily complete (not every error might get reported).

The other transformation to workflow nets is used to transform an existing YAWL model into a colored Petri net that can be simulated by CPN Tools. If an event log from the given YAWL model is provided during the transformation, then relevant information such as organizational details and performance characteristics are included in the resulting simulation model.

7 Conclusion

Many transformations to Petri nets currently exist, and several of these transformations struggle with concepts that are hard to handle in Petri nets, like

OR-joins and exceptions, but other transformations simply can abstract from these concepts, either because the source language does not support the concept as well, or because the application of the transformation allows for the abstraction. For example, YAWL and EPCs do not support exceptions, and some of the YAWL and EPC transformations can abstract from the OR-join because the relaxed soundness property allows this.

Our experience indicates that transforming an informal and complex language like BPEL to a low-level Petri net language is quite difficult without first having formalized the language in a proper way. Many BPEL constructs require a dedicated and possibly complex solution in Petri net terms, and to keep these solutions nicely orthogonal (we do not want one solution to obstruct a second) is not an easy task. Therefore, it seems sensible to:

– first, formalize the language using, for example, a high-level Petri net language, and
– second, transform the high-level formalization to the target low-level Petri net language.

Almost all transformations have been implemented in tools, and most of these tools are included in the ProM framework [38]. For example, the transformations from EPCs has been implemented in regular ProM conversion plug-ins, and the transformations from BPEL have been implemented in tools for which ProM conversion wrapper plug-ins have been implemented. The transformations from YAWL models to reset nets and the transformations from BPMN to workflow nets have been implemented in separate tools and it is expected that these transformations will be included in the ProM framework in the near future.

The applications of the different transformations differ. Several transformations are used to verify the business process at hand, others are also used for the actual execution of the business process (the transformation from YAWL to reset nets is a good example for this). Furthermore, some transformations exist that aim to simplify the source language, examples include the EPC reductions and the well-known Petri-net reduction rules by Murata [7].

Informal languages often describe alternatives and shorthands to represent process parts that have the same (formal) semantics. For the four languages described here this holds only for BPMN. Other OMG standards also make use of alternatives and shorthands. In our experience alternatives and shorthands are most efficiently dealt with by creating a 'normal form' version of the language and defining the mapping for this normal form. Alternatives and shorthands should first be translated to the normal form. They will then be mapped to the formalism of choice automatically.

From the transformation of a very detailed language such as BPEL into a simple formalism like Petri nets, we learned that the applications of techniques well-known in the field of compiler theory greatly systematize and simplify the transformation. In particular, using high-level Petri nets as *intermediate formalism* to explicitly model data aspects yields a better understanding of BPEL. Only when a low-level pattern is actually needed (e.g., for verification), we abstract from data aspects. In addition *static analysis* [39] allows for an improved translation by collecting information on the context of each activity. This information can be used to chose the best fitting pattern (e.g., depending on the presence of handlers or the chosen verification goal) from a pattern repository. This *flexible model generation* [40] has been shown to yield very compact transformation results, and can be similarly applied to all presented source languages.

References

1. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS (2007)
2. Keller, G., Nüttgens, M., Scheer, A.: Semantische Processmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89 (in German), University of Saarland, Saarbrücken (1992)
3. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language. Information Systems **30**(4) (2005) 245–275
4. OMG: Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification, Object Management Group (2006)
5. Baeten, J.C.M., Weijland, W.P.: Process Algebra. Volume 18 of Cambridge tracts in theoretical computer science. Cambridge University Press, Cambridge (1990)
6. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer (1985)
7. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4) (1989) 541–580
8. van der Aalst, W.M.P.: The application of Petri nets to workflow management. The Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
9. Dehnert, J.: A Methodology for Workflow Modelling: from Business Process Modelling towards Sound Workflow Specification. PhD thesis, Technische Universität Berlin, Berlin, Germany (2003)
10. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset nets between decidability and undecidability. In: ICALP 1998. Volume 1443 of LNCS., Springer (1998) 103–115
11. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1**(3) (2005) 35–43
12. Wolf, K.: Does my service have partners? Transactions on Petri Nets and Other Models of Concurrency (2008) (Accepted for publication).
13. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distributed and Parallel Databases **14**(3) (2003) 5–51
14. Russell, N., ter Hofstede, A., van der Aalst, W., Mulyar, N.: Workflow control-flow patterns: A revised view. Report BPM-06-22, BPM Center (2006)
15. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information & Software Technology (2008) accepted for publication.
16. Wong, P.Y., Gibbons, J.: A Process Semantics for BPMN. In: Proceedings of 10th International Conference on Formal Engineering Methods. LNCS (2008) To appear. Extended version available at `http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmnsem.pdf`.
17. Rittgen, P.: Modified EPCs and their Formal Semantics. Technical report 99/19, University of Koblenz-Landau, Koblenz, Germany (1999)

18. van der Aalst, W.M.P., Desel, J., Kindler, E.: On the Semantics of EPCs: A Vicious Circle. In: EPK 2002, Trier, Germany, GI (2002) 71–80
19. Kindler, E.: On the semantics of EPCs: A framework for resolving the vicious circle. Data and Knowledge Engineering **56**(1) (2006) 23–40
20. van Dongen, B.F., Jansen-Vullers, M.H., Verbeek, H.M.W., van der Aalst, W.M.P.: Verification of the SAP reference models using EPC reduction, state space analysis, and invariants. Computers in Industry **58**(6) (2007) 578–601
21. Breugel, F.v., Koshkina, M.: Models and verification of BPEL. `http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf` (2006)
22. Lohmann, N., Verbeek, H.M.W., Ouyang, C., Stahl, C., van der Aalst, W.M.P.: Comparing and evaluating Petri net semantics for BPEL. Computer Science Report 07/23, Eindhoven University of Technology, Eindhoven, The Netherlands (2007)
23. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: BPM 2006. Volume 4102 of LNCS., Springer (2006) 17–32
24. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In: BPM 2005. Volume 3649 of LNCS., Springer (2005) 220–235
25. Schmidt, K.: LoLA: A low level analyser. In: ICATPN 2000. Volume 1825 of LNCS., Springer (2000) 465–474
26. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. Volume P-127 of LNI., GI (2008) 57–72
27. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. Volume 4546 of LNCS., Springer (2007) 321–341
28. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. Volume 4937 of LNCS., Springer (2008) 46–60
29. Ouyang, C., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M., Verbeek, H.M.W.: WofBPEL: A tool for automated analysis of BPEL processes. In: ICSOC 2005. Volume 3826 of LCNS., Springer (2005) 484–489
30. Verbeek, H.M.W., van der Aalst, W.M.P.: Woflan 2.0: A Petri-net-based workflow diagnosis tool. In: Application and Theory of Petri Nets 2000. Volume 1825 of LCNS., Springer (2000) 475–484
31. Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P.: Diagnozing workflow processes using woflan. The Computer Journal **44**(4) (2001) 246–279
32. Wynn, M.T., Edmond, D., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. In: ICATPN 2005. Volume 3536 of LNCS., Springer (2005) 423–443
33. Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Verifying workflows with cancellation regions and OR-joins: An approach based on relaxed soundness and invariants. The Computer Journal **50**(3) (2007) 294–314
34. Rozinat, A., Wynn, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Fidge, C.: Workflow simulation for operational decision support using yawl and prom. BPM Center Report BPM-08-04, BPMcenter.org (2008)
35. CPN Group, University of Aarhus, Denmark: CPN Tools Home Page. (http://wiki.daimi.au.dk/cpntools/)
36. Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Verifying workflows with cancellation regions and OR-joins: An approach based on reset nets and reachability analysis. In: BPM 2006. Volume 4102., Springer (2006) 389–394
37. Wynn, M.T., Verbeek, H.M.W., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Reduction rules for reset workflow nets. BPM Center Report BPM-06-25, BPMcenter.org (2006)

38. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The ProM framework: A new era in process mining tool support. In: ICATPN 2005. Volume 3536 of LNCS., Springer (2005) 444–454
39. Nielson, F., Nielson, H.R., Hankin, C.: Principles of Program Analysis. Springer (1999)
40. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation. Data Knowl. Eng. **64**(1) (2008) 38–54