# WorkflowNet2BPEL4WS: A Tool for Translating Unstructured Workflow Processes to Readable BPEL

Kristian Bisgaard Lassen[1] and Wil M.P. van der Aalst[2,1]

[1] Department of Computer Science, University of Aarhus, IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark. k.b.lassen@daimi.au.dk

[2] Department of Information Systems, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. w.m.p.v.d.aalst@tm.tue.nl

**Abstract.** This paper presents *WorkflowNet2BPEL4WS* a tool to automatically map a graphical workflow model expressed in terms of Workflow Nets (WF-nets) onto BPEL. The *Business Process Execution Language for Web Services* (BPEL) has emerged as the de-facto standard for implementing processes and is supported by an increasing number of systems (cf. the IBM WebSphere Choreographer and the Oracle BPEL Process Manager). While being a powerful language, BPEL is difficult to use. Its XML representation is very verbose and only readable for the trained eye. It offers many constructs and typically things can be implemented in many ways, e.g., using links and the flow construct or using sequences and switches. As a result only experienced users are able to select the right construct. Some vendors offer a graphical interface that generates BPEL code. However, the graphical representations are a direct reflection of the BPEL code and not easy to use by end-users. Therefore, we provide a mapping from WF-nets to BPEL. This mapping builds on the rich theory of Petri nets and can also be used to map other languages (e.g., UML, EPC, BPMN, etc.) onto BPEL. To evaluate WorkflowNet2BPEL4WS we used more than 100 processes modeled using Protos (the most widely used business process modeling tool in the Netherlands), automatically converted these into CPN Tools, and applied our mapping. The results of these evaluation are very encouraging and show the applicability of our approach.

**Keywords**: BPEL4WS, Petri nets, workflow management, business process management.

## 1 Introduction

The primary goal of this paper is to present *WorkflowNet2BPEL4WS*, a tool to automatically map WF-nets onto BPEL. The tool uses the approach described in [9] and assumes a WF-net modeled using CPN Tools [11, 27] and the resulting BPEL code can be used in systems such as the IBM WebSphere Choreographer [22] and the Oracle BPEL Process Manager [32]. Note that the approach is also applicable to other Petri-net-based tools (e.g., systems such as ProM, Yasper, WoPeD, and Protos that are able to export PNML). Moreover, the ideas are also applicable to other graph-based languages such as BPMN [44], UML activity diagrams [21], EPCs [24, 37], etc.

This introduction motivates the need for a tool like *WorkflowNet2BPEL4WS*. To do this we start by introducing BPEL followed by a brief discussion of WF-nets, an introduction to the tool, and some information about the evaluation using 100 Protos models.

### 1.1 BPEL

After more than a decade of attempts to standardize workflow languages (cf. [5, 31]), it seems that the *Business Process Execution Language for Web Services* (*BPEL4WS* or *BPEL* for short) [10] is emerging as the de-facto standard for executable process specification. Systems such as Oracle BPEL Process Manager, IBM WebSphere Application Server Enterprise, IBM WebSphere Studio Application Developer Integration Edition, and Microsoft BizTalk Server 2004 support BPEL, thus illustrating the practical relevance of this language.

Interestingly, BPEL was intended initially for cross-organizational processes in a web services context: "BPEL4WS provides a language for the formal specification of business processes and business interaction protocols. By doing so, it extends the Web Services interaction model and enables it to support business transactions." (see page 1 in [10]). However, it can also be used to support intra-organizational processes. The authors of BPEL [10] envision two possible uses of the language: "Business processes can be described in two ways. Executable business processes model actual behavior of a participant in a business interaction. Business protocols, in contrast, use process descriptions that specify the mutually visible message exchange behavior of each of the parties involved in the protocol, without revealing their internal behavior. The process descriptions for business protocols are called abstract processes. BPEL4WS is meant to be used to model the behavior of both executable and abstract processes." In this paper we only consider the use of BPEL as an execution language.

BPEL is an expressive language [45] (i.e., it can specify highly complex processes) and is supported by many systems. Unfortunately, BPEL is not a very intuitive language. Its XML representation is very verbose and there are many, rather advanced, constructs. Clearly, it is at another level than the graphical languages used by the traditional workflow management systems (e.g., Staffware, FileNet, COSA, Lotus Domino Workflow, SAP Workflow, etc.). This is the primary motivation of this for developing WorkflowNet2BPEL4WS, i.e., a tool to generate BPEL code from a graphical workflow language.

The modeling languages of traditional workflow management systems are executable but at the same time they appeal to managers and business analysts. Clearly, managers and business analysts will have problems understanding BPEL code. As a Turing complete[3] language BPEL can do, well, anything, but to do this it uses two styles of modeling: graph-based and structured. This can be explained by looking at its history: BPEL builds on IBM's WSFL (Web Services Flow Language) [28] and Microsoft's XLANG (Web Services for Business Process Design) [39] and combines accordingly the features of a block structured language inherited from XLANG with those for directed graphs originating from WSFL. As a result simple things can be implemented in two ways. For example a sequence can be realized using the `sequence` or `flow` elements, a choice based on certain data values can be realized using the `switch` or `flow` elements, etc. However, for certain constructs one is forced to use the block structured part of the language, e.g., a *deferred choice* [7] can only be modeled using the `pick` construct. For other constructs one is forced to use the links, i.e., the more graph-based oriented part of the language, e.g., two parallel processes with a one-way synchronization require a `link` inside a `flow`. In addition, there are very subtle restrictions on the use of links: "A link MUST NOT cross the boundary of a while

[3] Since BPEL offers typical constructs of programming languages, e.g., loops and if-the-else constructs, and XML data types it is easy to show that BPEL is Turing complete.

activity, a serializable scope, an event handler or a compensation handler... In addition, a link that crosses a fault-handler boundary MUST be outbound, that is, it MUST have its source activity within the fault handler and its target activity within a scope that encloses the scope associated with the fault handler. Finally, a link MUST NOT create a control cycle, that is, the source activity must not have the target activity as a logically preceding activity, where an activity A logically precedes an activity B if the initiation of B semantically requires the completion of A. Therefore, directed graphs created by links are always acyclic." (see page 64 in [10]). All of this makes the language complex for end-users. Therefore, there is a need for a "higher level" language for which one can generate *intuitive* and *maintainable* BPEL code.

Such a "higher level" language will not describe certain implementation details, e.g., particularities of a given legacy application. This needs to be added to the generated BPEL code. *Therefore, it is important that the generated BPEL code is intuitive and maintainable.* If the generated BPEL code is unnecessary complex or counter-intuitive, it cannot be extended or customized.

Note that tools such as Oracle BPEL Process Manager and IBM WebSphere Studio offer graphical modeling tools. *However, these tools reflect directly the BPEL code*, i.e., the designer needs to be aware of structure of the XML document and required BPEL constructs. For example, to model a *deferred choice* in the context of a parallel process [7] the user needs to add a level to the hierarchy (i.e., a `pick` defined at a lower level than the `flow`). Moreover, subtle requirements such as links not creating a cycle still need to be respected in the graphical representation. Therefore, it is interesting to look at a truly graph-based language with no technological-oriented syntactical restrictions and see whether it is possible to generate BPEL code.

### 1.2 WF-nets

In this paper we use a specific class of Petri nets, named *WorkFlow nets* (WF-nets) [1–3], as a *source language* to be mapped onto the *target language* BPEL. There are several reasons for selecting Petri nets as a source language. It is a simple graphical language with a strong theoretical foundation. Petri nets can express all the routing constructs present in existing workflow languages [4, 17, 42] and enforce no technological-oriented syntactical restrictions (e.g., no loops). Note that WF-nets are classical Petri nets without data, hierarchy, time and other extensions. Therefore, their applicability is limited. However, we do *not* propose WF-nets as the language to be used by end-users; we merely use it as the theoretical foundation. It can capture the control-flow structures present in other graphical languages, but it abstracts from other aspects such as data flow, work distribution, etc. Note that there are many Petri-net based modeling tools, e.g., general tools such as ExSpect, CPN Tools, etc. and more dedicated Petri-net-based workflow modeling and analysis tools such COSA, Protos, WoPeD, Yasper, and Protos. Clearly, these tools can be used to model WF-nets (possibly extended with time, data, hierarchy, etc.). Moreover, as demonstrated in the context of tools such as ProM [12] and Woflan [41], it is possible to map (abstractions of) languages like Staffware, MQSeries Workflow, EPCs, YAWL, etc. onto WF-nets. Hence, the mapping presented in this paper can be used as a basis for translations from other source languages such as UML activity diagrams [21], Event-driven Process Chains (EPCs) [24, 37], and the Business Process Modeling Notation (BPMN) [44]. Moreover, the basic ideas can also be used to map graph-based languages onto other (partly) block-structured languages.

### 1.3 WorkflowNet2BPEL4WS

In a technical report [9], we introduced an approach to automatically map a WF-net onto BPEL using an iterative approach. To support this approach, we implemented the tool *WorkflowNet2BPEL4WS*. This tool automatically translates Colored Petri Nets (CPNs, [27]) into BPEL code. These CPNs are specified using CPN Tools [11]. Note that a CPN may also contain detailed data transformations and stochastic information (e.g., delay distributions and probabilities). However, in the transformation, we abstract from data, time, and probabilities and mainly focus on the WF-net structure of the CPN. The code generated by WorkflowNet2BPEL4WS can be imported into any system that supports BPEL, e.g., IBM's WebSphere Studio [22].

### 1.4 Evaluation Using 100 Protos Models

To evaluate the applicability of our approach we used 100 processes modeled using Protos [36]. These models were automatically converted into CPN Tools using ProM [12] and then we used WorkflowNet2BPEL4WS to map them onto BPEL. Protos (Pallas Athena) uses a Petri-net-based modeling notation [36] and is a widely used business process modeling tool. It is used by more than 1500 organizations in more than 20 countries. The number of users that use Protos for designing processes is estimated to be 25000. Some of the organizations have modeled more than 1500 processes. The 100 process models used for the evaluation resulted from student projects where students had to model and redesign realistic business cases.

### 1.5 Outline

The remainder of this paper is organized as follows. First, we provide an overview of related work. Section 3 describes the approach used to map WF-nets onto BPEL using an iterative approach. Section 4 presents the implementation of WorkflowNet2BPEL4WS. Section 5 evaluates our approach using 100 Protos models. These models where then executed using IBM's WebSphere Studio. Section 6 concludes the paper.

## 2 Related Work

Since the early nineties workflow technology has matured [20] and several textbooks have been published, e.g., [6, 13, 23, 29]. During this period many languages for modeling workflows have been proposed, i.e., languages ranging from generic Petri-net-based languages to tailor-made domain-specific languages. The Workflow Management Coalition (WfMC) has tried to standardize workflow languages since 1994 but failed to do so [17]. XPDL, the language proposed by the WfMC, has semantic problems [4] and is rarely used. In a way BPEL [10] succeeded in doing what the WfMC was aiming at. However, BPEL is really at the implementation level rather than the workflow modeling level or the requirements level (thus providing the motivation for this paper).

Several attempts have been made to capture the behavior of BPEL [10] in some formal way. Some advocate the use of finite state machines [18, 19], others process algebras [16, 26], and yet others abstract state machines [14, 15] or Petri nets [33, 30, 38, 40]. For a detailed analysis of BPEL based on the workflow patterns [7] we refer to [45].

The work reported in this paper is also related to the various tools and mappings used to generate BPEL code being developed in industry. Tools such as the IBM WebSphere Choreographer and the Oracle BPEL Process Manager offer a graphical notation for BPEL. However, this notation directly reflects the code and there is no intelligent mapping as shown in this paper. This implies that users have to think in terms of BPEL constructs (e.g., blocks, syntactical restrictions on links, etc.). More related is the work of Steven White that discusses the mapping of BPMN onto BPEL [43] and the work by Jana Koehler and Rainer Hauser on removing loops in the context of BPEL [25]. Note that none of these publications provides a mapping of some (graphical) process modeling language onto BPEL: [43] merely presents the problem and discusses some issues using examples and [25] focusses on only one piece of the puzzle. Also related is the mapping presented in [34] where a subclass of BPMN is mapped onto BPEL using ECA rules as an intermediate format. Then these ECA rules are realized by BPEL event handlers (onEvent). Note that this mapping heavily relies on the implementation of events in BPEL. Moreover, the resulting code is not very readable for humans because this mapping does not try to identify patterns close to the BPEL constructs. A more recent mapping tries to overcome this problem [35] but has not been implemented yet.

The tool presented in this paper uses our translation which was described in detail in a technical report [9]. The work is also related to [8] where we describe a case study where for a new bank system requirements are mapped onto Colored Workflow Nets (a subclass of Colored Petri Nets) which are then implemented using BPEL in the IBM WebSphere environment.

## 3 Mapping WF-nets to BPEL

In this paper, we would like to focus on the WorkflowNet2BPEL4WS tool and the evaluation of it. Therefore, we do not show any details for the algorithms being used. Moreover, we do not give any proof of the correctness of our approach. For this we refer to the technical report [9] mentioned before. We also assume that the reader has basic knowledge of BPEL and Petri nets. This allows us to focus on the application of our translation from WF-nets to BPEL.

As indicated in the introduction, it is important that the generated BPEL code is intuitive and maintainable. If the generated BPEL code is unnecessary complex or counter-intuitive, it cannot be extended or customized. Therefore, we try to map parts of the WF-net onto BPEL constructs that fit best. For example, a sequence of transitions connected through places should be mapped onto a BPEL `sequence`. We aim at recognizing "sequences", "switches", "picks", "while's", and "flows" where the most specific construct has our preference, e.g., for a sequence we prefer to use the `sequence` element over the `flow` element even though both are possible. We aim at an iterative approach where the WF-net is reduced by packaging parts of the network into suitable BPEL constructs.

*We would like to stress that our goal is not to provide just any mapping of WF-nets onto BPEL.* Note that a large class of WF-nets can be mapped directly onto a BPEL `flow` construct. However, such a translation results in unreadable BPEL code. Instead we would like to map a graph-based language like WF-nets onto a hierarchical decomposition of specific BPEL constructs. For example, if the WF-net contains a sequence of transitions (i.e., activities) this should be mapped onto the more specific `sequence`

construct rather than the more general (and more verbose) `flow` construct. Hence, our goal is to generate readable and compact code.

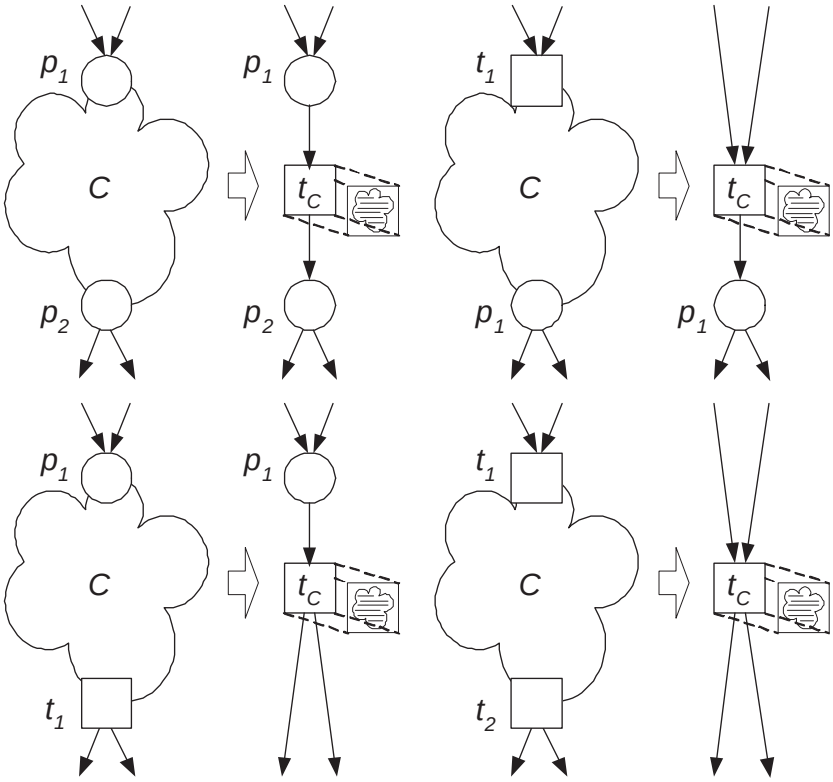


**Fig. 1.** Folding a component $C$ into a single transition $t_C$.

To map WF-nets onto (readable) BPEL code, we need to transform a graph structure to a block structure. For this purpose we use *components*. A component should be seen as a selected part of the WF-net that has a clear start and end. One can think of it as subnet satisfying properties similar to a WF-net. However, unlike a WF-net, a component may start and/or end with a transition, i.e., WF-nets are "place bordered" while components may be "place and/or transition bordered". The goal is to map components onto "BPEL blocks". For example, a component holding a purely sequential structure should be mapped onto a BPEL `sequence` while a component holding a parallel structure should be mapped onto a `flow`. Figure 1 shows the basic idea. We try to identify a place and/or transition bordered component $C$ and fold this into a single transition $t_C$. The annotation of $t_C$ hold the BPEL code corresponding to $C$. By repeating this process we hope to find a single transition annotated with the BPEL code of the entire process.

So we can summarize our approach as follows: *The idea is to start with an annotated WF-net where each transition is labeled with references to primitive activities such as* `invoke` *(invoking an operation on some web service),* `receive` *(waiting for a message from an external source),* `reply` *(replying to an external source),* `wait`

*(waiting for some time),* `assign` *(copying data from one place to another),* `throw` *(indicating errors in the execution), and* `empty` *(doing nothing). Taking this as starting point, a component in the annotated WF-net is mapped onto BPEL code. The component $C$ is replaced by transition $t_C$ whose inscription (cf. Figure 1) describes the BPEL code associated to the whole component. This process is repeated until there is just a single transition whose inscription corresponds to the BPEL specification of the entire process.* How this can be done is detailed using an example.

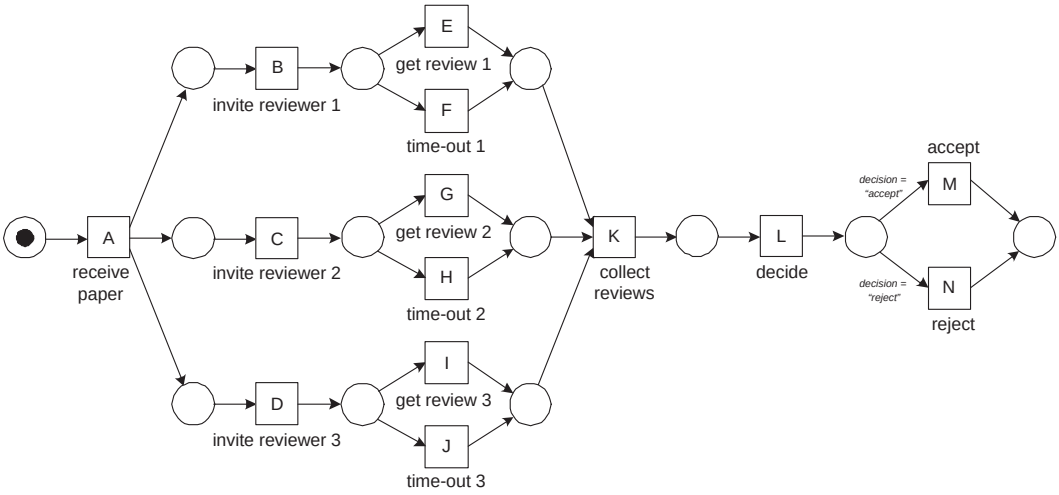


**Fig. 2.** A Petri net describing the process of reviewing papers.

Figure 2 shows a Petri net corresponding to a reviewing process. First we look for a *maximal sequence component*, i.e., a component representing a sequence that is chosen as large as possible. This component is mapped into a new transition with the corresponding BPEL annotation. In Figure 2 there is a sequence consisting of transition $K$ and $L$ and we replace this component by a transition $F1$. The resulting WF-net and the annotation of transition $F1$ is shown in Figure 3.

After folding $K$ and $L$ into $F1$ there is no sequence component remaining. Therefore, we replace $M$ and $N$ by a transition $F2$ tagged with a `switch` expression. Note that this component is not mapped onto a `pick` construct because of the inscriptions on the arcs suggesting some choice based on data rather than a time or message trigger. In our tool, we use a set of annotations to guide the generation of BPEL code. However, for the basic idea this is of less importance. Figure 4 shows the resulting WF-net.

Because of the introduction of $F2$ a new sequence is created. Clearly, this sequence is maximal and we can replace is by a transition $F3$ tagged with a `sequence` expression as shown in Figure 5

In Figure 5 there are three components representing a `pick` component. Note that we assume that these represent a `pick` because there are no conditions on the arcs or the places with multiple outgoing arcs. As indicated, the WorkflowNet2BPEL4WS tool can handle a variety of tags directing the mapping process. In case of a `pick` it is possible to describe more about the nature of the choice (e.g., events, timers, etc.). However, in this paper we focus on the control-flow. Each of the `pick` components can

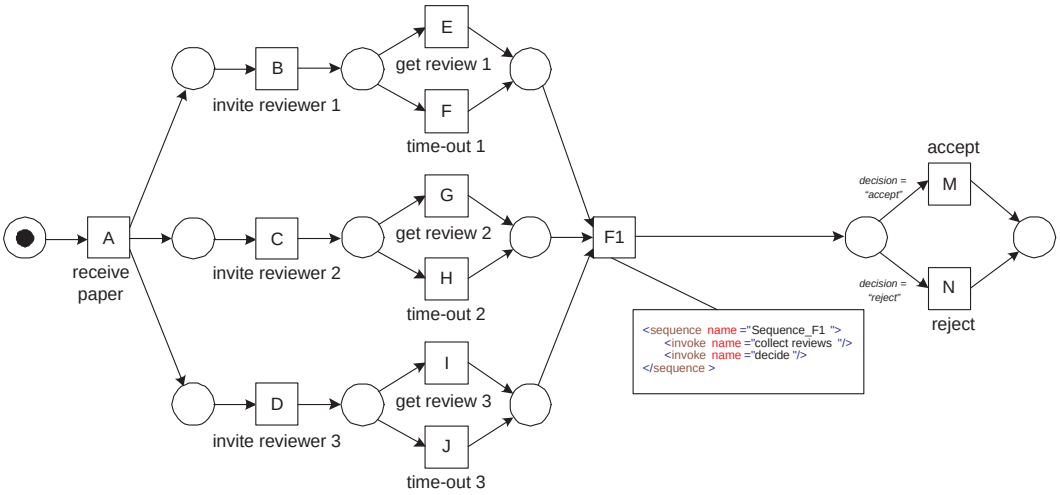


**Fig. 3.** The Petri net after replacing $K$ and $L$ by a transition $F1$ tagged with a `sequence` expression.

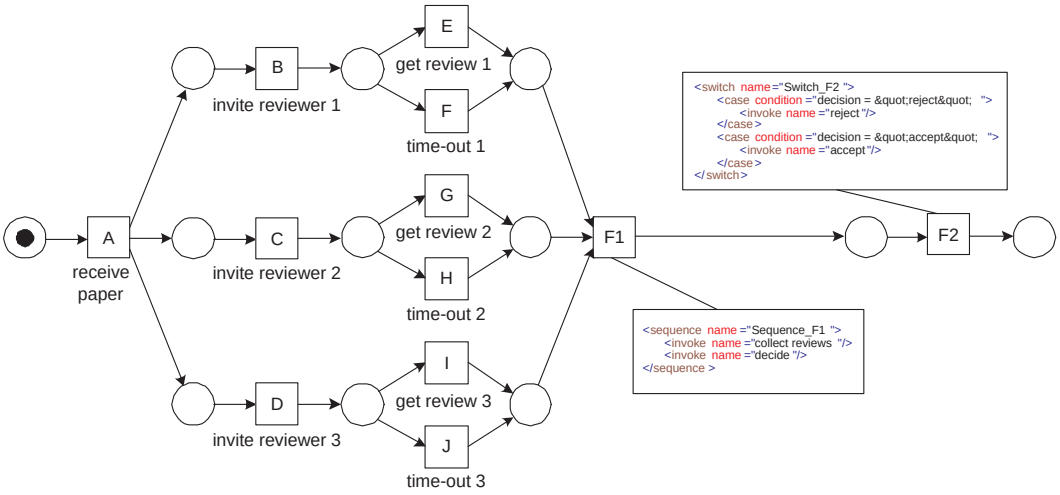


**Fig. 4.** The Petri net after replacing $M$ and $N$ by a transition $F2$ tagged with a `switch` expression.

be replaced by a single transition which is then merged with the preceding transition into a `sequence` transition.

Figure 6 illustrates the sequence of steps that are taken. Note that the intermediate result shown is actually not possible, i.e., first $D$ and $F4$ would be merged into $F5$ before moving to the other two parallel branches. However, $D$ and $F4$ are depicted separately to show the process in more intuitive manner. If $D$ and $F4$ are also merged (i.e., the sequence of $D$ and $F4$ is replaced by $F5$), there are 5 transitions remaining: $A$, $F3$, $F5$, $F7$, and $F9$. Together they form a `flow` component.

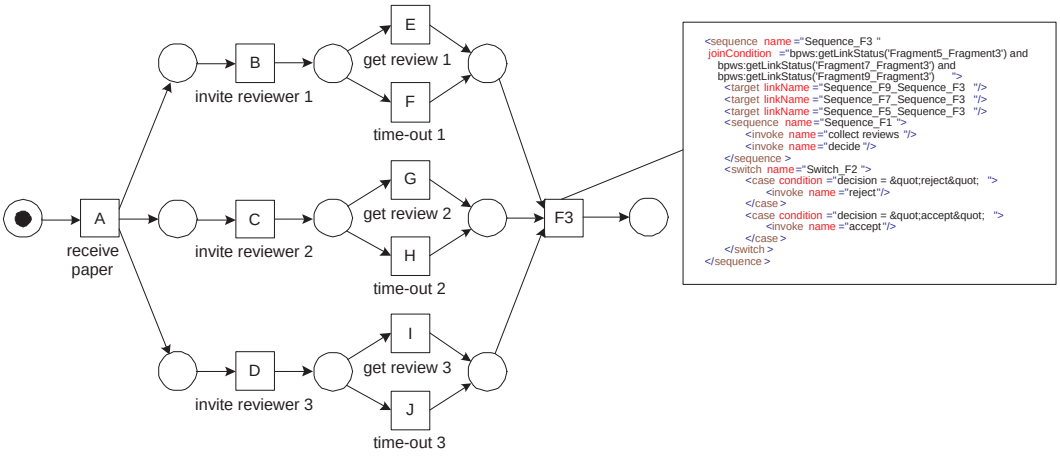


**Fig. 5.** The Petri net after replacing $F1$ and $F2$ by a transition $F3$ tagged with a `sequence` expression.

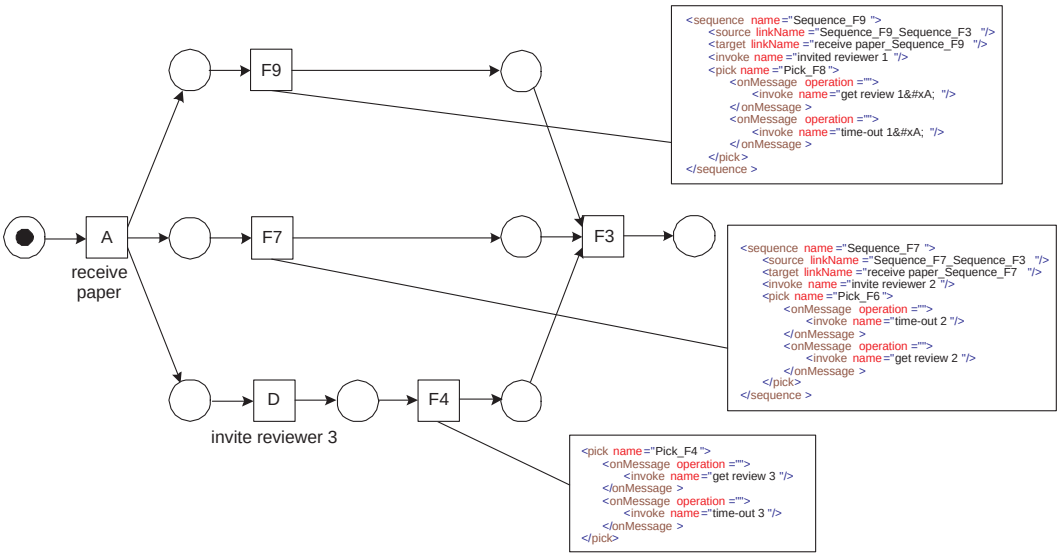


**Fig. 6.** The Petri net after replacing $I$ and $J$ by a transition $F4$ tagged with a `pick` expression. Then $D$ and $F4$ can be merged into a sequence $F5$. Similarly, $F7$ and $F9$ can be created.

Figure 7 shows the result after applying the last step. Note that this is the 10th folding and the result is a WF-net consisting of only one transition $F10$. The annotation of $F10$ is the BPEL code for the entire process. Hence, we provided an iterative approach to translate the Wf-net shown Figure 2 into BPEL template code.

## 4 Implementation

For a detailed description of the algorithm we refer to [9]. In this section, we only highlight the basic structure of the tool. The starting point is a WF-net. We assume that

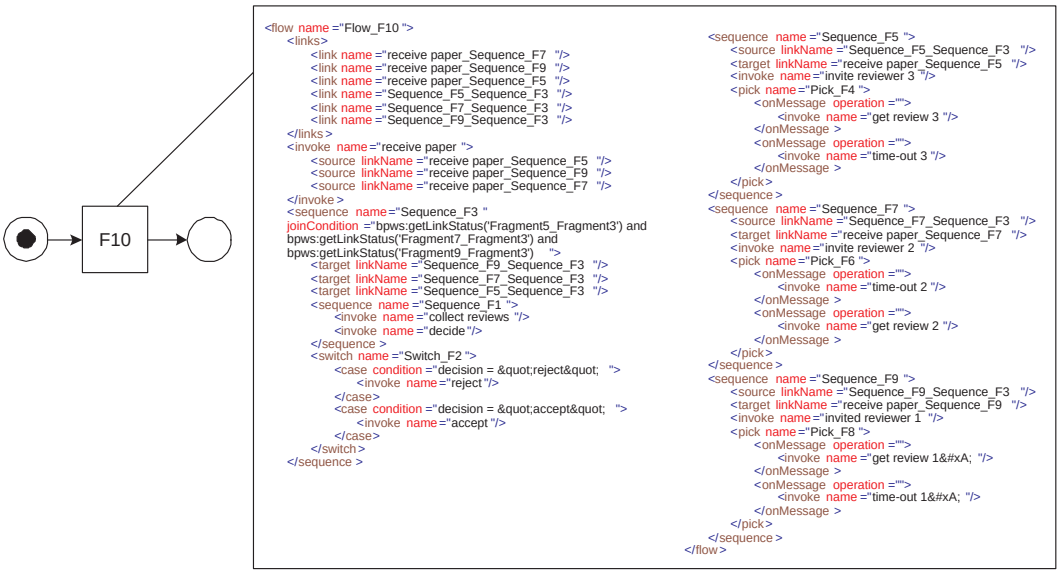


**Fig. 7.** The Petri net after replacing the remaining part by one transition $F10$ tagged with a `flow` expression.

this WF-net is modeled using CPN Tools [11, 27]. Note that through ProM [12] and Woflan [41], it is possible to map (abstractions of) languages like Protos, Staffware, MQSeries Workflow, EPCs, YAWL, etc. onto WF-nets and export them to CPN Tools. In CPN Tools we assume some annotation describing e.g. the nature of choice (`pick` or `switch`) and the content of the activity represented by an atomic transition. We allow for the annotation of activity types like `invoke` (invoking an operation on some web service), `receive` (waiting for a message from an external source), `reply` (replying to an external source), `wait` (waiting for some time), `assign` (copying data from one place to another), `throw` (indicating errors in the execution), and `empty` (doing nothing).

In principle, no annotations are needed for the translation of Workflow nets in CPN Tools to BPEL. If a choice construct (place with outgoing arcs) is not annotated, it is assumed to be part of a `switch`. If an atomic transition is not annotated, it is assumed to be an `invoke` activity.

An important assumption for the correctness of our approach is that the initial WF-net is safe and sound. This can be checked with tools such as ProM [12] and Woflan [41] and also the state-space tool of CPN Tools [11].

In [9] we described that the various components can be detected in a WF-net. Based on this, WorkflowNet2BPEL4WS uses the following algorithm to produce BPEL code.

**Definition 1 (Algorithm).** *Let $PN = (P, T, F, \tau_P, \tau_G, \tau_{MA}, \tau_T)$ be a safe and sound annotated WF-net.*

*(i)* $X := PN$
*(ii) while* $[X] \neq \emptyset$ *(i.e., $X$ contains a non-trivial component)*[4]
*(iii-a) If there is a maximal SEQUENCE component* $C \in [X]$*, select it and goto (vi).*

[4] Note that this is the case as long as $X$ is not reduced to a WF-net with just a single transition.

*(iii-b) If there is a SWITCH component $C \in [X]$, select it and goto (vi).*
*(iii-c) If there is a PICK component $C \in [X]$, select it and goto (vi).*
*(iii-d) If there is a WHILE component $C \in [X]$, select it and goto (vi).*
*(iii-e) If there is a maximal FLOW component $C \in [X]$, select it and goto (vi).*
*(iv) If there is a component $C \in [X]$ that appears in the component library, select it and goto (vi).*
*(v) Select a component $C \in [X]$ to be manually mapped onto BPEL and add it to the component library.*
*(vi) Attach the BPEL translation of $C$ to $t_C$ as illustrated in Figure 1.*
*(vii) $X := fold(PN, C)$ and return to (ii).*
*(viii) Output the BPEL code attached to the transition in $X$.*

The actual translation of components is done in step *(vi)* followed by the folding in step *(vii)*. The component to be translated/folded is selected in steps *(iii)*. If there is still a sequence remaining in the net, this is selected. A maximal sequence is selected to keep the translation as compact and simple as possible. Only if there are no sequences left in the WF-net, other components are considered. The next one in line is the SWITCH component followed by the PICK component and the WHILE component. Given the fact that SWITCH, PICK and WHILE components are disjoint, the order of steps *(iii-b)*, *(iii-c)*, and *(iii-d)* is irrelevant. Finally, maximal FLOW components are considered.

Not every net can be reduced into SEQUENCE, SWITCH, PICK, WHILE, and FLOW components. Therefore, steps *(iv)* and *(v)* have been added. The basic idea is to allow for ad-hoc translations. These translations are stored in a component library. If the WF-net cannot be reduced any further using the standard SEQUENCE, SWITCH, PICK, WHILE and FLOW components, then the algorithm searches the component library (note that it only has to consider the network structure and not the specific names and annotations). If the search is successful, the stored BPEL mapping can be applied (modulo renaming of nodes and arc and transition annotations). If there is not a matching component, the tool will save the irreducible net along with each of the components in the net. A manual translation can be then provided for one of the components and stored in the component library for future use.

The component library is composed of pairs of component and the translation of it into BPEL activity. When we match a component against a library component we take each pair of transitions that where matched successfully by that component and substitute the BPEL activity in the library BPEL specification code by that of the transition in the net that is being translated. So if a transition is annotated with a sequence and the corresponding transition in the library component is an invoke then the invoke in the library BPEL invoke is replaced by the sequence. If we did not do this, then each time a library component is used for reduction, all previous translations would get lost.

Figure 8 shows a screenshot of WorkflowNet2BPEL4WS in action. The WF-net Figure 2 is loaded into CPN Tools and WorkflowNet2BPEL4WS iteratively creates BPEL template code. Note that WorkflowNet2BPEL4WS saves the result of each step, i.e., for Figure 2 there are 9 intermediate and one final model generated. This allows the designer to check the translation process. If the net is irreducible, i.e. there is no predefined or library component that can match a component in the net, then WorkflowNet2BPEL4WS stores the irreducible net along with a copy of the components that exists in the irreducible net. This makes it easy to develop the library to cope with a particular Workflow net, by choosing a component in the list that WorkflowNet2BPEL provides and translating it, and adding the component and the translation of it to the library.

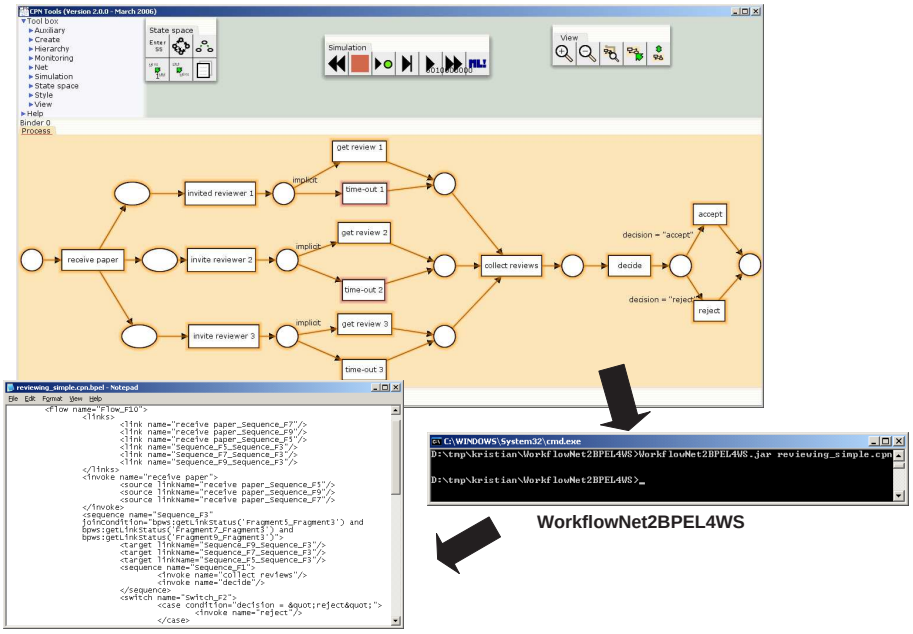


**Fig. 8.** A screenshot of CPN Tools and the resulting BPEL code after applying the WorkflowNet2BPEL4WS tool.

The default component matching order in the tool is the one described in [9], but it is possible to change the order in which components are matched. It is possible express that FLOW components are selected before SEQUENCE components, or that a user-defined component has priority over FLOW components. By doing this, the user of the tool can adjust the "style" of the generated BPEL and not just settle with some fixed order.

## 5 Evaluation

To evaluate our approach and to test the WorkflowNet2BPEL4WS tool, we used 100 process models developed by students in group projects. Each group modeled the processes related to a realistic business case using Protos. Each group was free to select their own business case. Using interviews and documentation, their assignment was to model the business processes, to analyze them, and to propose redesigns. It is important to stress that the processes were not selected or modeled by the authors of this paper, i.e., the groups were free to choose a business case. There were only requirements with respect to the size and complexity of the models. Moreover, the models had to be correct. Using Woflan the groups could verify the soundness of their models [41]. As a modeling tool they used Protos. The reason is that this is the most widely used business process modeling tool in the Netherlands (see Section 1.4). Moreover, we have developed interfaces to Petri-net-based analysis tools such as Woflan, ProM, ExSpect, and CPN Tools.

Although there were more than Protos 100 models, we made an random selection of 100 models. All these models where exported to our input format by, first importing

them into ProM and then exporting them to the CPN Tools format. On average the generated CPN models contained 23.66 places and 26.54 transitions.

The goal of the evaluation was to take the WorkflowNet2BPEL4WS tool and convert each of the 100 Protos models into BPEL. This was not trivial since only 11 of the 100 models could be reduced using the standard predefined components of the algorithm, i.e., just 11 models could be completely reduced into SEQUENCE, SWITCH, PICK, WHILE, and FLOW components. Therefore, we were forced to add components to our component library. We started with an empty component library and each time we encountered a WF-net that could not be reduced completely, we added a manual translation of the smallest irreducible component to our library. After adding 76 components to the library we succeeded in completely reducing all Protos models.

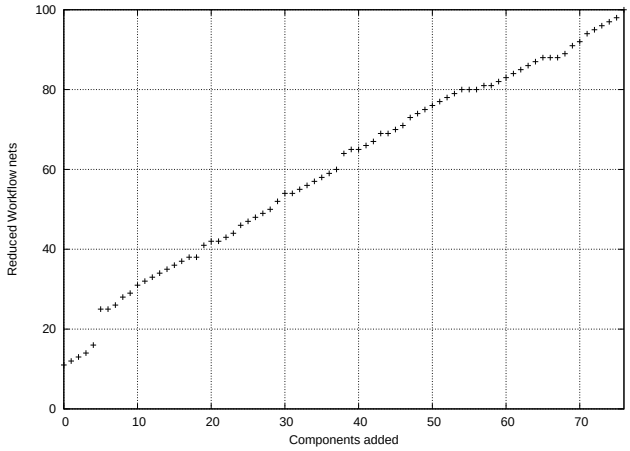


**Fig. 9.** A graph showing the relationship between added components and reduced nets.

In Figure 9 we see the relationship between components added to the library and the number of nets reduced. It starts out in (0,11) (11 nets could be reduced using the predefined components) and ends up in (100,76) (76 components had to be added to reduce all 100 nets).

For each library component we added, we translated all of the 100 nets to check the distribution of components types. Figure 10 shows how often each component type could be applied to reduce the WF-nets in each of the 76 steps. For example, the number of times a component could be reduced by mapping it onto a SEQUENCE increases from less than 700 until 743 (top line). This number increases because each time a component is added to the library a further reduction is possible and new SEQUENCE components may surface. The line "x" in Figure 10 shows the number of reductions possible because of the component library. Initially, this value is 0 because the component library is empty. After adding 76 ad-hoc components, 132 reductions result from the component library. After adding these 76 each WF-net can be reduced completely and as a result we obtain the BPEL code for all 100 models.

Figure 11 shows the final distribution of reductions, i.e., while generating the BPEL code 73 FLOW components, 132 LIBRARY components, 743 SEQUENCE compo-

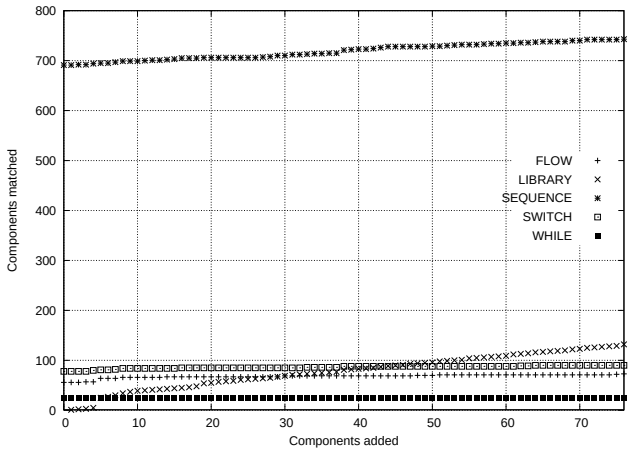


**Fig. 10.** A graph showing the distribution of components used.

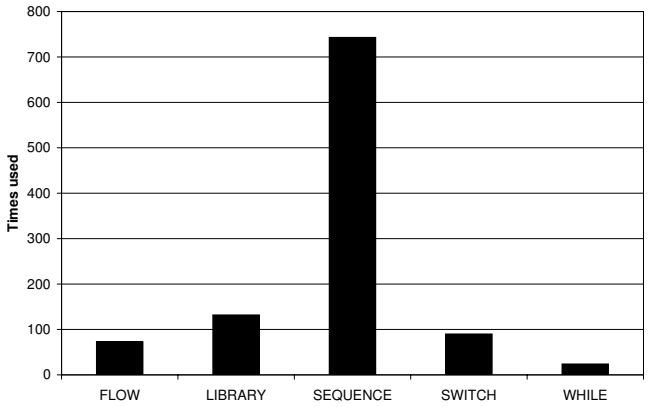


**Fig. 11.** The number of times each component type can be used for a reduction after adding 76 ad-hoc components to the library.

nents, 90 SWITCH components, and 24 WHILE components are encountered.[5] These numbers show that the standard reductions using the SEQUENCE, SWITCH, PICK, WHILE, and FLOW components are doing remarkably well. On the total of 1062 reductions required to generate the BPEL code, only 132 were due to LIBRARY components ($\simeq$12.4%).

After adding all 76 components to the library we checked to see how often a library component was used when translating all of the 100 WF-nets. This is shown in Fig-

[5] The reason that no PICK components where found was that the current export facility of ProM to CPN Tools do not annotate the WF-nets with this information. Since the only difference between a match for a PICK and a SWITCH component is the annotation of the place splitting the flow, the reduction process is not influenced. If we would have transferred this information from Protos, the sum of the number of SWITCH and PICK reductions would be 90 and the other numbers would not be affected. Note that this information is available in Protos but cannot (yet) be stored in the intermediate PNML format.

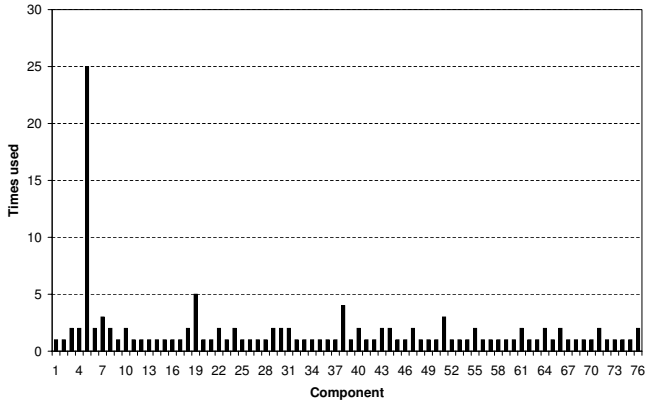


**Fig. 12.** A graph showing how many times each added component is used.

ure 12. The figure shows that most of the library components we used only once. Only one library component (number 5) was used frequently, i.e., of the 76 library components 50 components were used only once ($\simeq 66\%$), 21 were used twice ($\simeq 28\%$), 2 were used three times ($\simeq 3\%$), 1 was used four times ($\simeq 1\%$), 1 was used five times ($\simeq 1\%$), and 1 was used 25 times ($\simeq 1\%$).

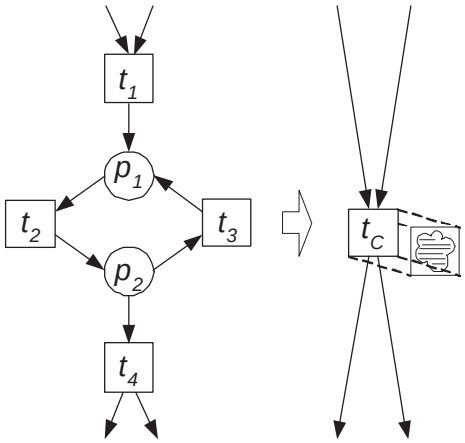


**Fig. 13.** The reduction using the library component that was used 25 times.

Figure 13 shows the library component that could be applied most frequently. Clearly, this is a good candidate to be added to the set of standard components. None of the other library components is used very frequently. This shows that some manual work will always be necessary. We would like to point out that students were encouraged to select complicated business processes, i.e., part of the grading was based on the use of as many workflow patterns as possible [7]. As a result we expect the selected set of Protos models to be more complicated than usual. For example, the students were encouraged to use the "Milestone Pattern" [7] which cannot be reduced using any of the standard

components and is difficult to capture this pattern in a single component to be added to the library. Therefore, the results should be interpreted as a worst-case scenario. We expect that in a most situations, more than 95% of the components can be reduced using the standard components and the component depicted in Figure 13.

The performance of the reductions and BPEL generation is not an issue. After adding the 76 library components, each of the 100 Workflow nets was translated within a few seconds.

## 6 Conclusion and Future Work

In this paper we presented an approach to generate BPEL code from WF-nets using reductions, i.e., components are folded into transitions labeled with BPEL code. The main goal is to generate *readable* BPEL code. Therefore, we did not aim at a complicated full translation (e.g., using events handler [34]) but at recognizing "natural BPEL constructs". The approach is supported by the translation tool *WorkflowNet2BPEL4WS*. The tool also supports an extensible component library, i.e., components and their preferred BPEL translations can be added thus allowing for different translation styles.

We have evaluated WorkflowNet2BPEL4WS and the underlying ideas using 100 complex Protos models. Based on this evaluation we estimate that more than 95% of real-life process models can be automatically translated into readable BPEL code. However, some manual interventions are needed to translate the remaining 5%. As demonstrated, larger component libraries could be used to achieve a fully automatic translation in most cases.

The current implementation can be used in conjunction with a wide variety of Petri net based tools, e.g., CPN Tools, ProM, Yasper, WoPeD, PNK, CPN-AMI, and Protos. Moreover, the ideas are also applicable to other graph-based languages such as BPMN, UML activity diagrams, EPCs , and proprietary workflow languages.

In the near future we plan to implement this algorithm directly into ProM [12] so it will be possible to translate a wide variety of process modeling languages to BPEL. Moreover, Pallas Athena is interested in integrating WorkflowNet2BPEL4WS into Protos. Give the widespread use of Protos, such an implementation would allow many organizations to generate BPEL code.

## References


1. W.M.P. van der Aalst. Verification of Workflow Nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer-Verlag, Berlin, 1997.
2. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
3. W.M.P. van der Aalst. Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques. In W.M.P. van der Aalst, J. Desel, and A. Oberweis, editors, *Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161–183. Springer-Verlag, Berlin, 2000.
4. W.M.P. van der Aalst. Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management. In J. Desel, W. Reisig, and G. Rozenberg, editors, *Lectures on Concurrency and Petri Nets*, volume 3098 of *Lecture Notes in Computer Science*, pages 1–65. Springer-Verlag, Berlin, 2004.

5. W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Web Service Composition Languages: Old Wine in New Bottles? In G. Chroust and C. Hofer, editors, *Proceeding of the 29th EUROMICRO Conference: New Waves in System Architecture*, pages 298–305. IEEE Computer Society, Los Alamitos, CA, 2003.
6. W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT press, Cambridge, MA, 2002.
7. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
8. W.M.P. van der Aalst, J.B. Jørgensen, and K.B. Lassen. Let's Go All the Way: From Requirements via Colored Workflow Nets to a BPEL Implementation of a New Bank System Paper. In R. Meersman and Z. Tari et al., editors, *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005*, volume 3760 of *Lecture Notes in Computer Science*, pages 22–39. Springer-Verlag, Berlin, 2005.
9. W.M.P. van der Aalst and K.B. Lassen. Translating Workflow Nets to BPEL4WS. BETA Working Paper Series, WP 145, Eindhoven University of Technology, Eindhoven, 2005.
10. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
11. CPN Group, University of Aarhus, Denmark. CPN Tools Home Page. http://wiki.daimi.au.dk/cpntools/.
12. B. van Dongen, A.K. Alves 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, *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
13. M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software through Process Technology*. Wiley & Sons, 2005.
14. D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In D. Beauquier and E. Börger and A. Slissenko, editor, *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, March 2005.
15. R. Farahbod, U. Glässer, and M. Vajihollahi. Specification and validation of the business process execution language for web services. In W. Zimmermann and B. Thalheim, editors, *Abstract State Machines 2004*, volume 3052 of *Lecture Notes in Computer Science*, pages 79–94, Lutherstadt Wittenberg, Germany, May 2004. Springer-Verlag, Berlin.
16. A. Ferrara. Web services: A process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.
17. L. Fischer, editor. *Workflow Handbook 2003, Workflow Management Coalition*. Future Strategies, Lighthouse Point, Florida, 2003.
18. J.A. Fisteus, L.S. Fernández, and C.D. Kloos. Formal verification of BPEL4WS business collaborations. In K. Bauknecht, M. Bichler, and B. Proll, editors, *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, August 2004. Springer-Verlag, Berlin.
19. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *International World Wide Web Conference: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA, 2004. ACM Press.
20. D. Georgakopoulos, M. Hornick, and A. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3:119–153, 1995.
21. Object Management Group. *OMG Unified Modeling Language 2.0*. OMG, http://www.omg.com/uml/, 2005.
22. IBM WebSphere. www.ibm-306.ibm.com/software/websphere.

23. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press, London, UK, 1996.
24. 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.
25. J. Koehler and R. Hauser. Untangling Unstructured Cyclic Flows A Solution Based on Continuations. In R. Meersman, Z. Tari, W.M.P. van der Aalst, C. Bussler, and A. Gal et al., editors, *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2004*, volume 3290 of *Lecture Notes in Computer Science*, pages 121–138, 2004.
26. M. Koshkina and F. van Breugel. Verification of Business Processes for Web Services. Technical report CS-2003-11, York University, October 2003. Available from: http://www.cs.yorku.ca/techreports/2003/.
27. L.M. Kristensen, S. Christensen, and K. Jensen. The Practitioner's Guide to Coloured Petri Nets. *International Journal on Software Tools for Technology Transfer*, 2(2):98–132, 1998.
28. F. Leymann. Web Services Flow Language, Version 1.0, 2001.
29. F. Leymann and D. Roller. *Production Workflow: Concepts and Techniques*. Prentice-Hall PTR, Upper Saddle River, New Jersey, USA, 1999.
30. A. Martens. Analyzing Web Service Based Business Processes. In M. Cerioli, editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
31. M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design and Application of workflow-driven Process Information Systems*. Logos, Berlin, 2004.
32. Oracle BPEL Process Manager. www.oracle.com/technology/products/ias/bpel.
33. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, , and H.M.W. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.
34. C. Ouyang, M. Dumas, S. Breutel, and A.H.M. ter Hofstede. Translating Standard Process Models to BPEL. BPM Center Report BPM-05-27, BPMcenter.org, 2005.
35. C. Ouyang, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Translating BPMN to BPEL. BPM Center Report BPM-06-02, BPMcenter.org, 2006.
36. Pallas Athena. *Protos User Manual*. Pallas Athena BV, Plasmolen, The Netherlands, 2004.
37. A.W. Scheer. *ARIS: Business Process Modelling*. Springer-Verlag, Berlin, 2000.
38. C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
39. S. Thatte. XLANG Web Services for Business Process Design, 2001.
40. H.M.W. Verbeek and W.M.P. van der Aalst. Analyzing BPEL Processes using Petri Nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 59–78. Florida International University, Miami, Florida, USA, 2005.
41. H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001.
42. WFMC. Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface – XML Process Definition Language (XPDL) (WFMC-TC-1025). Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002.
43. S. White. Using BPMN to Model a BPEL Process. *BPTrends*, 3(3):1–18, March 2005.
44. S.A. White et al. Business Process Modeling Notation (BPML), Version 1.0, 2004.
45. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In I.Y. Song, S.W. Liddle, T.W. Ling, and P. Scheuermann, editors, *22nd International Conference on Conceptual Modeling (ER 2003)*, volume 2813 of *Lecture Notes in Computer Science*, pages 200–215. Springer-Verlag, Berlin, 2003.