Transformation of BPMN Diagrams to YAWL Nets

JianHong YE
School of Computer Science and Technology
Huaqiao University, Quanzhou Fujian 362021, China
Email: leafever@163.com
Wen SONG
School of Mathematics and Computer Engineering
Xihua University, Chengdu Sichuan 610039, China
Email: syl1505@163.com

Abstract— **Business Process Modeling Notation (BPMN) is the de facto standard for modeling business processes on a conceptual level. However, BPMN lacks a formal semantics and many of its features need to be further interpret, Consequently that hinders BPMN as a standard to statically check the semantic correctness of models. YAWL (Yet Another Workflow Language) allows the specification of executable workflow models. A transformation between these two languages enables the integration of different levels of abstraction in process modeling. This paper discusses how to transform BPMN diagrams to YAWL nets. The benefits of the transformation are threefold. Firstly, it clarifies the semantics of BPMN via a mapping to YAWL. Secondly, the deployment of BPMN business process models is simplified. Thirdly, BPMN models can be analyzed with YAWL verification tools.**

Index Terms— **BPMN, YAWL, Transformation, Algorithm**

I. INTRODUCTION

Process modeling is used at different levels of abstraction. First, models serve to communicate as-is business processes, pinpoint improvement options, conduct resource and cost analysis and capture to-be processes. The Business Process Modeling Notation (BPMN [1]) is the de facto standard for process modeling at this level. On the other hand languages targeting at technically realizing business processes is used as input for process execution engines. The YAWL [2] is a standard for implementing process-oriented composition of web services, with a strictly defined execution semantics, a first-class concept of "task", and sophisticated support for data mappings and task-to-resource allocation.

BPMN is a graph-oriented language in which controlling nodes can be connected in arbitrary way. It primarily targets at domain analysts and is supported by many modeling tools. The notation inherits and combines elements from a number of previously proposed notations for business process modeling, including the XML Process Definition Language (XPDL) [3] and the Activity Diagrams component of the Unified Modeling Notation (UML) [4].

BPMN provides a number of advantages to modeling business processes, it offers a process flow modeling technique that is more conducive to the way of business analysts model and its solid mathematical foundation is expressly designed to map to business execution languages. BPMN is already supported by more than 54 tools (see www.bpmn.org), but in its current form, BPMN lacks the semantic precision which is required to capture fully executable business processes. Consistent with the level of abstraction targeted by BPMN, none of these tools support the execution of BPMN models directly. Close inspection of existing translation from BPMN to BPEL standard, the one sketched in [1], [5]–[7], shows these translations fail to fulfill the key requirements, such as completeness, automation, readability, etc. The translation patterns and algorithms in these papers address issues that arise generally when translating from graph-oriented process languages to block-structured ones. However, mapping between graph-oriented and block-structured process definition languages is notoriously challenging, it is likely to require refinement as well as testing and debugging, which defeat the purpose of BPEL as a domain-specific language. Another attempt at defining a formal semantics for a subset of BPMN did so using Petri nets [8]–[11]. The proposed mapping serves not only the purpose of disambiguating the core constructs of BPMN, it also provides a foundation to statically check the semantic correctness of BPMN models. However, their semantics does not properly model multiple instances, exception handling, message flows and OR-join.

YAWL is a workflow language specially designed to support the 20 workflow patterns [12] that proposed by Van der Aalst, ter Hofstede, Kiepuszewski and Barros in an intuitive manner. YAWL can be used as a *lingua franca* to express the behavior of Web services (for example, described using BPEL or OWL-S [13]). YAWL has a well defined formal semantics. Furthermore, the basis on Petri nets provides a firm tool for the formal analysis of real-world services. In order to benefit from the expressive power of YAWL, a large amount of business process

Manuscript received Jun 11, 2009; revised Oct 13, 2009; accepted Oct 19, 2009

This work was supported in part by the Science Foundation of Huaqiao University grants 09BS514 and the Special Foundation for Young Scientists of Fujian Province.

doi:10.4304/jsw.5.4.396-404

models are mapping to YAWL, such as the Event-driven Process Chain (EPC) to YAWL [14] and BPEL to YAWL [15].

The transformation from BPMN to YAWL can be used as an instrument to implement process-oriented applications. It also opens the possibility of reusing static analysis techniques available for YAWL. Like Petri nets, YAWL has a formally defined semantics that enables the analysis of YAWL nets to detect semantic errors such as deadlocks. Close inspection of existing translations from BPMN to YAWL, e.g. sketched in [16], however, serveral properties and assignments are missing in the mapping, the translations fail to fulfill the following key requirements: (i) message flows are lost; (ii) transactions and compensation handlers are not covered; (iii) complex gateways are neglected. At first glance, this mapping may seem straightforward. Indeed, the conceptual mismatch between BPMN and YAWL is not as significant as the one between BPMN and BPEL, especially with regards to control-flow structures. However, mapping BPMN to YAWL turns out to be tricky in the details, revealing subtle differences between the two languages.

Our goal is to provide a methodology for transforming a model from BPMN to YAWL. The mapping is in the following five ways:

- Although BPMN and YAWL share most of their concepts, there is a fundamental difference in the way of joins and splits are treated in each language. While BPMN inherits the connector types from EPCs which define them as first class objects independent of functions, YAWL includes joins and splits in task objects. Accordingly, there is no direct equivalent in YAWL elements for BPMN connector chains, i.e. multiple consecutive connectors.
- YAWL requires processes to have only one start and one end condition. In BPMN, multiple start and end events are allowed.
- BPMN task or subprocess has a lot of attributes, which attributes can be applied in YAWL and which ones should be extended.
- A message flow is used to show the transmission of messages between two participants via communication actions such as send task, receive task, or message event in different pools in BPMN, although the information of pool or lane will be lost in conversion. How to map these messages to flow messages and not affect the YAWL initial marking is a challenge.
- BPMN exception handling is captured by exception flow. The conversion should be clear regarding the semantics of an exception handler attached on a task or subprocess.

The rest of the paper is organized as follows. Section II provides the mathematical notations. Our contribution starts in Section III, which illustrates the solutions of the transformation from BPMN to YAWL. Section IV presents the structure of the tool implementation and its application to static analysis of BPMN models. Finally, Section V concludes and outlines future work.

II. Basic Definitions

A BPMN process, which uses the core subset of BPMN elements as shown in Figure 1, is referred to as a core BPMN process. We define the syntax of the core BPD.

Definition 1 (Core BPMN Process): [17] A core BPMN process is a tuple: $M = (O, A, \varepsilon, g, P, L, T, S, T^R, \varepsilon^S, \varepsilon^I, \varepsilon^E, \varepsilon^{I_\phi}, \varepsilon^{I_M}, \varepsilon^{I_T}, \varepsilon^{I_E}, \varepsilon^{I_C}, g^{As}, g^{Aj}, g^{Xs}, g^{Xj}, g^{eXs}, g^{Os}, g^{Oj}, F, Cond, Exc, Mes)$

- O is a set of objects which can be partitioned into disjoint sets of activities A, events ε, gateways g.
- A can be partitioned into disjoint sets of tasks T and subprocesses invocation activities S.
- P/L is a set of pools/lanes. For any node $o \in O$, $P(o)/L(o)$ is a function showing which pool/lane the object o belongs to.
- $T^R \subseteq T$ is a set of receiving tasks.
- ε can be partitioned into disjoint sets of start events ε^S, intermediate events ε^I and events ε^E.
- ε^I can be further partitioned into disjoint sets of intermediate events without any trigger ε^{I_ϕ}, intermediate message events ε^{I_M}, intermediate timer events ε^{I_T}, intermediate exception events ε^{I_E}, and intermediate cancel events ε^{I_C}.
- g can be partitioned into disjoint sets of parallel fork gateways g^{As}, parallel join gateways g^{Aj}, data-based XOR decision gateways g^{Xs}, XOR merge gateways g^{Xj}, event-based XOR decision gateway g^{eXs}, inclusive decision gateways g^{Os} and inclusive merge gateway g^{Oj}.
- $F \subseteq O \times O$ is the control flow relation, i.e. a set of sequence flows connecting objects.
- $Cond : F \cap (g^{Xs} \times O) \rightarrow C$ is a function which maps sequence flows emanating from data-based XOR gateways to conditions.[1]
- $Exc : \varepsilon^I \rightarrow A$ is a function assigning an intermediate event to an activity such that the occurrence of the event signals an exception and thus interrupts the performance of the activity.
- $Mes \subseteq O_i \times O_j, P(o') \neq P(o''), o' \in O_i \wedge o'' \in O_j$, i.e. objects belong to different pools.

Definition 1 allows for graphs which are unconnected, not have start or end events, contain objects without any input and output, etc. Therefore we need to restrict the definition to well-formed core BPMN. Before this, we first define the predecessor and successor nodes.

Definition 2 (Predecessor and Successor Nodes): Let O be a set of objects and $Edge$ be a binary relation over O, $Edge = F \cup Cond \cup Exc$. For each node o, we define the set of predecessor nodes ${}^\bullet o = \{x \in O \mid (x, o) \in Edge \wedge o \in O \setminus dom(Exc)\}$, and the set of successor nodes $o^\bullet = \{x \in O \mid (o, x) \in Edge \wedge o \in O\}$.

[1] A condition is a Boolean function operating over a set of propositional variables that can be abstracted out of the control flow definition. The condition may evaluate to true or false, which determines whether or not the associated sequence flow is taken during the process execution.

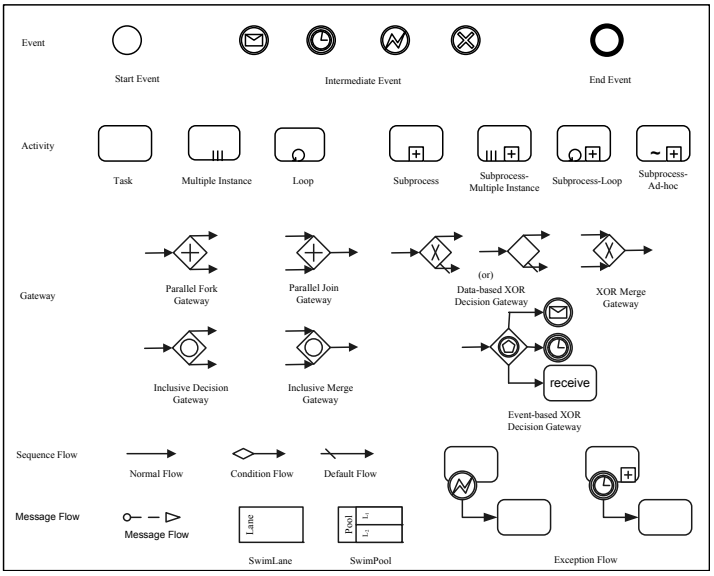

Figure 1. A core subset of BPMN elements

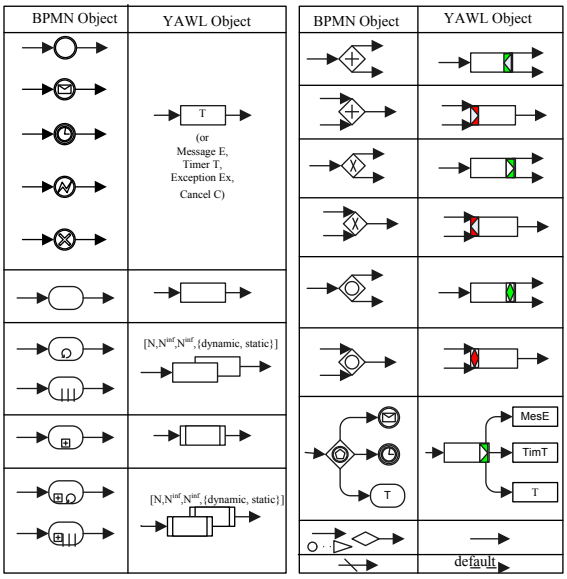

Figure 2. Activities, Events, Gateways and Sequence flow mapped to YAWL

Definition 3 (Well-formed core BPMN): [17] A core BPMN process P in Definition 1 is well formed if and only if relation F satisfies the following requirements:

- $\forall s \in \varepsilon^S \cup dom(Exc), {}^\bullet s = \phi \wedge |s^\bullet| = 1$, i.e. start events and exception events have an in-degree of zero and an out-degree of one,
- $\forall e \in \varepsilon^E, e^\bullet = \phi \wedge |{}^\bullet e| = 1$, i.e. end events have an out-degree of zero and an in-degree of one,
- $\forall x \in A \cup (\varepsilon^I \backslash dom(Exc)), |{}^\bullet x| = 1$ and $|x^\bullet| = 1$, i.e. activities and non-exception intermediate events have an in-degree of one and an out-degree of one,
- $\forall g \in g^{As} \cup g^{Xs} \cup g^{eXs} \cup g^{Os} : |{}^\bullet g| = 1 \wedge |g^\bullet| > 1$, i.e. fork or decision gateways have an in-degree of one and an out-degree of more than one,
- $\forall g \in g^{Aj} \cup g^{Xj} \cup g^{Oj} : |{}^\bullet g| > 1 \wedge |g^\bullet| = 1$, i.e. join or merge gateways have an out-degree of one and an in-degree of more than one,
- $\forall g \in g^{eXs}, g^\bullet \subseteq \varepsilon^{I_M} \cup \varepsilon^{I_T} \cup T^R$, i.e. event-based XOR decision gateways must be followed by intermediate message or timer events or receive tasks.
- $\forall g \in g^{Xs}, \exists x \in g^\bullet, Cond((g,x)) = \neg \wedge_{y \in g^\bullet \backslash \{x\}} = Cond((g,y))$, i.e. (g,x) is the default flow among all the outgoing flows from g,
- $\forall x \in O, \exists s \in \varepsilon^S \cup dom(Exc), \exists e \in \varepsilon^E, sF^*x \wedge xF^*e$, i.e. every object is on the path from a start event or an exception event to an end event.

Definition 4 (YAWL-Net): [2] A YAWL-net N is defined as a tuple $(C, i, o, T, Flow, split, join, rem, nofi)$ such that:

- C is a set of conditions.
- $i \in C$ is the unique input condition.
- $o \in C$ is the unique output condition.
- T is a set of tasks.
- $Flow \subseteq (C \setminus \{o\} \times T) \cup (T \times C \setminus \{i\}) \cup T \times T$ is the flow relation.
- split : $T \rightarrow \{AND, XOR, OR\}$ specifies the split behavior of each task.
- join : $T \rightarrow \{AND, XOR, OR\}$ specifies the join behavior of each task.
- rem : $T \rightarrow \rho(T \cup C \setminus \{i, o\})$ specifies the token to be removed from the tasks and conditions given in the mapping.
- nofi: $T \rightarrow N \times N^{inf} \times N^{inf} \times \{dynamic, static\}$ specifies the multiplicity of each task (minimum, maximum, threshold for continuation, and dynamic/static creation of instances).

III. From BPMN to YAWL

We only consider map well-formed core BPMN diagrams to YAWL nets in this paper, using simplified notation $M = (O, P, L, F, Exc, Mes)$ for their representation. The study is focused on control-flow constructs.

A. Activities, Events, Gateways, Sequence flow and Message Flow

Figure 2 gives the transformation from a set of BPMN activities, events, gateways and sequence flows. An intermediate event or task is mapped to an atomic task with one input and output. The other activities except for Ad-hoc subprocess are mapped onto corresponding composite task or multiple instance tasks with one predecessor and one successor. BPMN start and end events are easy to transform if there is only one. In this case, the BPMN start event maps to YAWL input condition and the end event to a output condition. Multiple start and end events will be discussed in the Section III-B.

As gateways are independent elements in BPMN, which is allowed to build so-called gateway chains, i.e. paths of two or more consecutive gateways. Splits and joins in YAWL are only allowed as part of tasks. As a consequence, there may be need to introduce empty

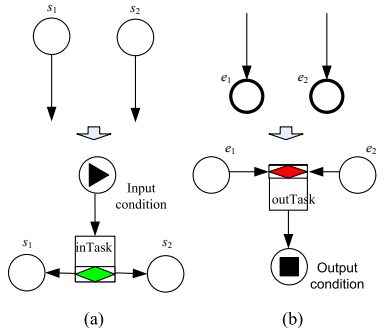

Figure 3. Transformation of multiple start and end events

tasks only to map a gateway. Figure 2 illustrates how a gateway is transformed, an additional empty task is required to include the join rule or split rule except event-based decision gateways. Especially, an event-based gateway is captured in a way that the gateway mapped to an empty task with split rule and all the elements of the gateway output corresponding atomic task compete for the determines which path is taken. Generally, sequence flow or message flow is transformed to a YAWL flow edge in straight-forward manner. More details are depicted in Figure 2.

B. Transformating Start and End Events

If there are multiple start events, they have to be bundled: the one YAWL input condition is followed by an empty task with an OR-split rule. Each BPMN start event is then mapped to a YAWL condition that is linked as a successor. Analogously, each of multiple BPMN end events is mapped to YAWL conditions which are all connected with an OR-join of an empty task that leads to the one YAWL output condition. This transformation rule makes these models difficult to analyze, because $2^{|n|}$ states have to be considered with n being the amount of BPNN start or end events. In this case, graph reduction rules could be applied in order to get compacter models. Yet, this issue is beyond the scope of this paper. The example is shown in Figure 3.

C. Activities Macro and Ad-hoc construct

An activity in a BPD can have attributes specifying its additional behavior, these attributes can be set to determine the values of *nofi* in the corresponding task in YAWL. Table I demonstrated how to map the BPMN attributes to the values of *nofi*.

There is another special attribute for Ad-hoc subprocess activity. This attribute defines if the activities within the process can be performed in parallel or must be performed sequentially. Without lose the generality, the multiple instance attributes for activities in Ad-hoc subprocess can referred from Table I. The example in Figure 4 considered each activity will be performed one at a time.

Algorithm 1: B_Mes2Y_Flow(Mes m)

Input: ε^S;ε^E;$m \in Mes$,$m = (x, y)|P(x) \neq P(y)$. Suppose there exists an input task t_{inTask} and output task $t_{outTask}$,$split(t_{inTask}) = OR$, $join(t_{outTask}) = OR$;
Output: yawl flow;

if $\forall s \in \varepsilon^S, \exists x \in \varepsilon^E \cup T \cup S, s.t.(x, s) \in Mes$ **then**
  add empty task t' with $join(t') = XOR$ to yawl;
  add $f \in Flow \wedge f = (t_{inTask}, t')$ to yawl;
  add $f \in Flow \wedge f = (t', s^\bullet)$ to yawl;
  if $x \in S \cup T \wedge |x^\bullet| = 1$ **then**
    add $f \in Flow \wedge f = (x, t') \wedge split(x) = AND$ to yawl;
  end
  else if $x \in S \cup T \wedge |x^\bullet| = 0$ **then**
    add $f \in Flow \wedge f = (x, t')$ to yawl;
  end
  else if $x \in \varepsilon^E$ **then**
    add $f \in Flow \wedge f = ({}^\bullet x, t')$ to yawl, delete x;
  end
  delete s;
end
if $\forall e \in \varepsilon^S, \exists y \in \varepsilon^S \cup T \cup S, s.t.(e, y) \in Mes$ **then**
  add empty task t'' with $split(t'') = AND$ to yawl;
  add $f \in Flow \wedge f = (t'', t_{outTask})$ to yawl;
  add $f \in Flow \wedge f = ({}^\bullet e, t'')$ to yawl;
  if $x \in S \cup T \wedge |{}^\bullet y| = 1$ **then**
    add $f \in Flow \wedge f = (t'', y) \wedge join(y) = AND$ to yawl;
  end
  else if $x \in S \cup T \wedge |y^\bullet| = 0$ **then**
    add $f \in Flow \wedge f = (t'', y)$ to yawl;
  end
  else if $y \in \varepsilon^S$ **then**
    add $f \in Flow \wedge f = (t'', y^\bullet)$ to yawl, delete y;
  end
  delete e;
end
while $|Mes| \neq \phi$ **do**
  if $x \in (T \cup S) \wedge y \in (T \cup S)$ **then**
    if $|x^\bullet| = 0 \wedge |{}^\bullet y| = 0$ **then**
      add $f \in Flow \wedge f = (x, y)$ to yawl;
    end
    else if $|x^\bullet| = 1 \wedge |{}^\bullet y| = 0$ **then**
      add $f \in Flow \wedge f = (x, y) \wedge split(x) = AND$ to yawl;
    end
    else if $|x^\bullet| = 0 \wedge |{}^\bullet y| = 1$ **then**
      add $f \in Flow \wedge f = (x, y) \wedge join(y) = AND$ to yawl;
    end
    else if $|x^\bullet| = 1 \wedge |{}^\bullet y| = 1$ **then**
      add $f \in Flow \wedge f = (x, y) \wedge split(x) = AND \wedge join(y) = AND$ to yawl;
    end
  end
  else if $x \in (T \cup S) \wedge y \in \varepsilon^S$ **then**
    if $|x^\bullet| = 0$ **then**
      add $f \in Flow \wedge f = (x, y^\bullet)$ to yawl;
    end
    else if $|x^\bullet| = 1$ **then**
      add $f \in Flow \wedge f = (x, y^\bullet) \wedge split(x) = AND$ to yawl;
    end
    delete ε^S;
  end
  else if $x \in \varepsilon^E \wedge y \in (T \cup S)$ **then**
    if $|{}^\bullet y| = 0$ **then**
      add $f \in Flow \wedge f = ({}^\bullet x, y)$ to yawl;
    end
    else if $|{}^\bullet y| = 1$ **then**
      add $f \in Flow \wedge f = ({}^\bullet x, y) \wedge join(y) = AND$ to yawl;
    end
    delete ε^E;
  end
  else if $x \in \varepsilon^E \wedge y \in\in \varepsilon^S$ **then**
    add $f \in Flow \wedge f = ({}^\bullet x, y^\bullet)$ to yawl, delete ε^E and ε^S;
  end
end

TABLE I.
HOW TO MAP THE BPMN ATTRIBUTES TO THE VALUES OF *nofi*

BPMN activity attributes				YAWL task *nofi*
LoopType	*LoopMax*	*LoopCondition*	*TestTime*	*nofi*
standard	m	The expression xq_3 should determine how many instances of activities are to be created	before	$[0, m, xq_3, static]$
standard	m	The expression xq_3 should determine how many instances of activities are to be created	after	$[1, m, xq_3, static]$

BPMN activity attributes					YAWL task *nofi*
LoopType	*LoopMax*	*MICondition*	*MIOrdering*	*MIFlow Condition*	*nofi*
multiInstance	m	All activity instances should generate a token when the instance is completed	parallel	none	$[n, m, m, static]$
multiInstance	m	The token should continue past the activity after all of the activity instances have completed	parallel	all	$[m, m, m, static]$
multiInstance	m	Only one token can passed from the activity	parallel	one	$[1, m, 1, static]$
multiInstance	dynamic	The Expression should determine when and how many tokens will continue past the activity	parallel	complex	$[xq_1, xq_2, xq_3, dynamic]$

Note that the n, m is integer and the expression $xq_i (i = 1, 2, 3)$ is evaluated at run time to help determine how many instances of activities are to be created. The LoopCondition and MICondition will be replaced in a mathematical expression to be either tested as True or False or to be evaluated to update the value of Properties in actual use

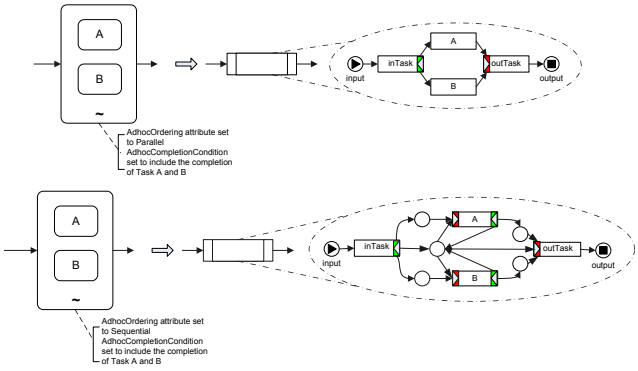

Figure 4. Transformation of Ad-hoc subprocess with different attribute of AdhocOrdering

D. Message Flow

The information of pools and lanes will be lost in the conversion. However, some related information such as message flow should be inherited. Message flows are used to model message passing between organizations or applications. It can be mapped to a flow message in YAWL and modify some objects' join or split attribute. Some special cases for the mapping involves the start event and end event. Mapping rules are established for distinguish these cases. Algorithm 1 gives the method for mapping the message flow. Figure 5 shows some examples with special rules. Note that in Definition 2, the predecessor or successor of nodes and the binary relation between them do not include message flow. On the other hand, some detail discussions involving activity attributes such as receive task, user task, service task etc. refer to [17].

E. Exception Handling

While the intermediate event is attached to the boundary of an activity, either a task or a subprocess, they create an exception flow. The event will respond to specific triggers to interrupt the activity and redirect the flow through the intermediate event. The source of the trigger may from external or caused by a "throw" intermediate event from any other active in the process. YAWL presents a direct and intuitive support of the remove tokens, sounds like "vacuum cleaner" removing tokens from selected parts of a net. At the same time, we have to impose one restriction that a subprocess associated with exception handing is not allowed to be interrupted by the occurrence of the exception event in itself, which will violate the seal principle in a subprocess. This is ambiguous in the BPMN specification states. We will describe two different source of the trigger circumstances in the following:

Firstly, assuming the *Timer, Message, Exception* and *Error* trigger will be invoked from the external of the process execution, Figure 6(1) shows the mapping of an exception associated with a task or subprocess via an exception task and cancel arc in YAWL.

Secondly, except the *Timer* trigger, an intermediate event attached on a task or subprocess will be invoked by an event occurs, and location in the process with the name exact consistency. Assuming there are two intermediate events with the same name, if ambiguity is possible, we use *throw* or *catch* as subscript convenient for marking their positions in the process. For example, i_{throw} is an intermediate exception event in the process, and i_{catch} refer to an intermediate event associated with a task invoking exception handing activities by i_{throw}. Figure 6(2) depicts the mapping, Algorithm 2 shows the exception flow conversion algorithm.

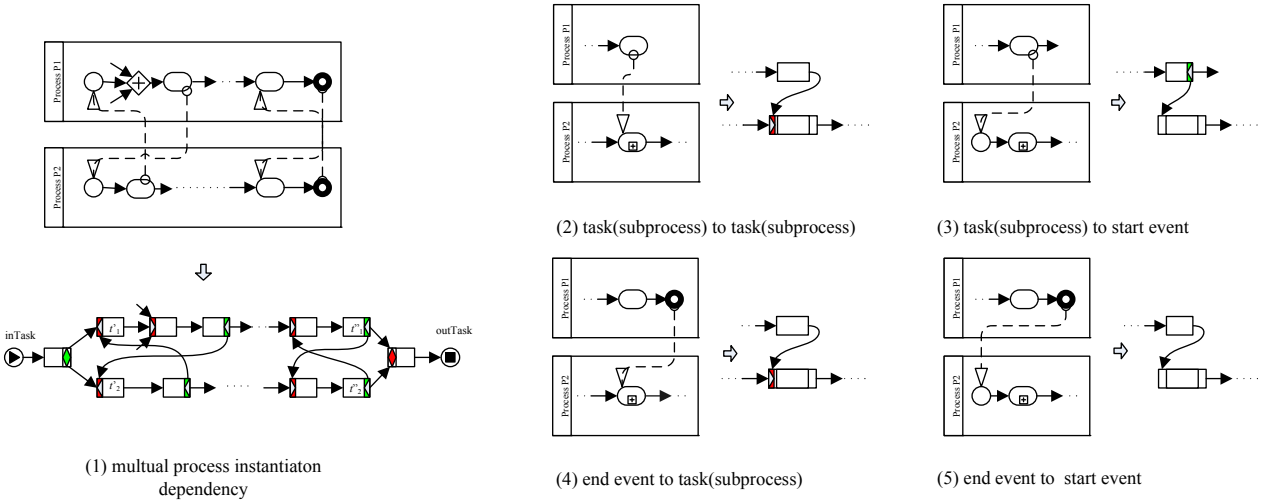

Figure 5. Examples of BPMN message flow to YAWL flow

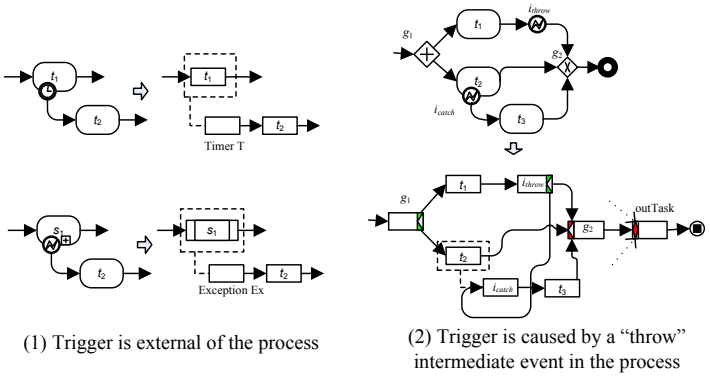

Figure 6. Mapping of exception flow

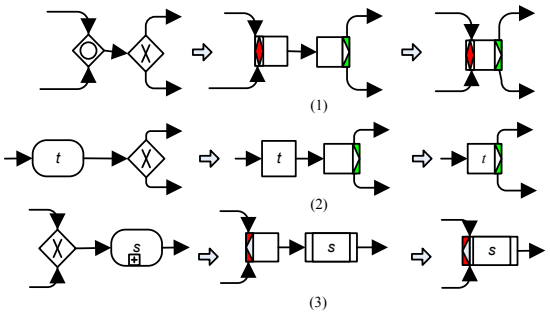

Figure 7. Reduction of connector chains events

Algorithm 2: B_Exc2Y_Flow(Exc e)

Input: $e \in Exc$,$e = (x, y)|x \in \varepsilon^I, y \in O, r$ is the Task(Subprocess) associated with x on the boundary;
Output: yawl flow;
if $Name(x)$ *is not exclusive, i.e.* $\exists z \in \varepsilon^I \wedge |^\bullet z| \neq \phi \wedge Name(x) \equiv Name(z)$ **then**
  map x to a task;
  add cancel $edge(r, x)$ to yawl;
  map z to a task and $Split(z) = AND$;
  add $f \in Flow \wedge f = (z, x)$ to yawl;
end
else
  map x to a task;
  add cancel edge (r, x) to yawl;
end

F. Transformating gateway chains

As joins and splits are first class elements of BPMN while in YAWL they are part of tasks. As a consequence, there may be need to introduce empty tasks only to map a connector. This is in particular the case with connector chains. Figure 7(1) illustrates how a connector chain is transformed. If a join connector is followed by a split, they can be combined into one empty task. Otherwise, splits and joins can be combined with the pre-event predecessor function or the post-event successor function, respectively.

In addition, if there have a split connector as the successor of one task (subprocess), which can be embedded into the task(subprocess). It is similar with a task (subprocess)'s predecessor. Figure 7(2) and (3) respectively show these possible.

G. Tranformation Algorithm and Example

We traverse the BPMN process graph and take advantage of the fact that YAWL does not enforce an alternation of tasks and conditions. Basically, we ignore events except start or end events. Therefore, most states of the generated YAWL process model are associated with implicit conditions.

Based on the mapping of each of the components aforementioned, we now define an algorithm to translate a well-formed core BPD into YAWL. The algorithm is arranged in two stages, the first stage is depending on the type of the current node, its predecessor nodes and successor nodes, respective elements of the YAWL target model are generated. The second stage is reducing gateway chains if necessary. These steps repeated until no elements can be folded.

Now, we can analyze the complexity of the algorithm 3. Given a well-formed BPD $M = (O, P, L, F, Exc, Mes)$,

Algorithm 3: WF_BPMN2YAWL (BPD M)

Input: a well-formed BPD $M = (O, P, L, F, Exc, Mes)$;
Output: a yawl model
$Y = (C, i, o, T, Flow, split, join, rem, nofi)$;

1 Add YAWL input and output condition:$i \in C$ and $o \in C$;
2 **if** $|\varepsilon^S| > 1$ **then**
3 add OR split task t_{inTask};
4 **end**
5 **if** $|\varepsilon^E| > 1$ **then**
6 add OR join task $t_{outTask}$;
7 **end**
8 **while** $|M| \neq \phi$ **do**
9 **for** $\forall node \in Mes$; *if node is not be mapped;* $M = M \setminus node$ **do**
10 B_Mes2Y_Flow($node$);
11 **end**
12 $M = M \setminus (P \cup L)$;
13 **for** $\forall node \in Exc$; *if node is not be mapped;* $M = M \setminus node$ **do**
14 B_Exc2Y_Flow($node$);
15 **end**
16 **for** $\forall node \in O$; ; $M = M \setminus node$ **do**
17 **if** $node \in \varepsilon^S$ *(or* $node \in \varepsilon^E$*)* **then**
18 add $c \in C$ and ${}^\bullet node = t_{inTask}$ (or $node^\bullet = t_{outTask}$);
19 **end**
20 **if** $node \in g$ **then**
21 add empty task with corresponding split or join type;
22 **end**
23 **if** $node \in T$ **then**
24 add task to YAWL with corresponding attributes;
25 **end**
26 **if** $node \in S$ **then**
27 add composite task to YAWL, all elements in this subprocess is represented by $node = (O', P', L', F', Exc', Mes')$, and *Do* WF_BPMN2YAWL($node$);
28 **end**
29 **if** $node \in F$ **then**
30 add an flow edge to YAWL;
31 **end**
32 **end**
33 **if** t_x, t_y *is close relations in YAWL* $\wedge$ *there are two empty task or one is task (composite task) and the other is empty task* $\wedge t_x^\bullet = t_y \wedge {}^\bullet t_y = t_x$ **then**
34 t_x and t_y can be fold to a new task(composite task)t, ${}^\bullet t = {}^\bullet t_x \wedge t^\bullet = t_y^\bullet$;
35 **end**
36 **end**

For each transformation of BPMN node, the conversion operation is performed in linear time and the algorithm will be completed. The completeness is granted because BPMN are coherent; every node will be ultimately processed when navigation begins from the start events. In the extreme case, the algorithm will terminate when each node is processed exactly once (no influence by the exception handling event). Assuming that converse a BPMN object onto a YAWL element is trivial, the complexity of the algorithm is $O(|O|+|P|+|L|+|F|+|Exc|+|Mes|)$.

We present the complaint handling process model shown in Figure 8. This example has been taken from [5]. We use this example to illustrate how a BPD can be translated into a YAWL net. First the complaint is registered (task *register*) then in parallel a questionnaire is sent to the complainant (task *send questionnaire*) and the complaint is evaluated (task *evaluate*). If the complainant returns the questionnaire within two weeks (event *returned-questionnaire*), task process questionnaire is executed. Otherwise (event *timeout*), the result of the questionnaire is discarded. After either the questionnaire is processed or a time-out has occurred, the result needs to be archived (task *archive*), and in parallel, if the complaint evaluation has been completed, the actual processing of the complaint (task *process complaint*) can start. Next, the processing of the complaint is checked via task check processing. If the check result is not ok, the complaint requires re-processing. Otherwise, if the check result is ok and also the questionnaire has been archived, a notice will be sent to inform the complainant about the completion of the complaint handling (task *send notice*). Figure 8 sketches the translation procedure following the translation algorithm as mentioned above.

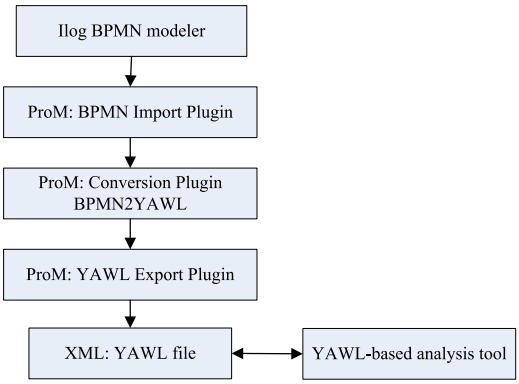

Figure 9. Structure of the BPMN to YAWL transformation

IV. Empirical Evaluation

As a proof of concept we have implemented the algorithm, an open-source plug-in called BPMN2YAWL is available in ProM 5.0. Each reader interested in this field can use the http://www.processmining.org web page for a more complete overview and download the latest version. Figure 9 shows the structure of the tool that we implemented.

We use the ILog BPMN Modeler[2]as a graphical editor to create BPMN models. To the author's knowledge, no existing tool for modeling BPMN can perform properties checks. We can analyst the BPMN model after them mapped to a YAWL net. The plugin in ProM subsequently transforms the BPMN models into a YAWL net and export the YAWL net as a XML file. This XML file can serve as input to a YAWL-based verification tool [11], [18]. A lot of properties such as the *deaklock_free*, *no dead_task*, *proper completion*, *no OR-join* and *soundness* [19]–[21] etc, can be checked via these tools.

We tested BPMN2YAWL on a set of models[3], some collected from the BPMN Web log[2], and the others are designed by the authors. Table II shows the size of each tested BPMN models in terms of number of tasks, events, gateways, subprocesses and message flows. It also shows the size of the resulting YAWL-nets in terms of number of conditions, atomic tasks and composite tasks. BPMN2YAWL was able to deal with all the models, although preprocessing was needed to transform some of them into well-formed BPMN models, how to generate a well BPD refer to [9], [17], [22], [23].

[2]http://www.ilog.com/products/jviews/diagrammer/bpmnmodeler/
[3]This set of test models are included in the distribution of the BPMN2YAWL tool, http://is.tm.tue.nl/trac/prom/wiki/TestBPMN

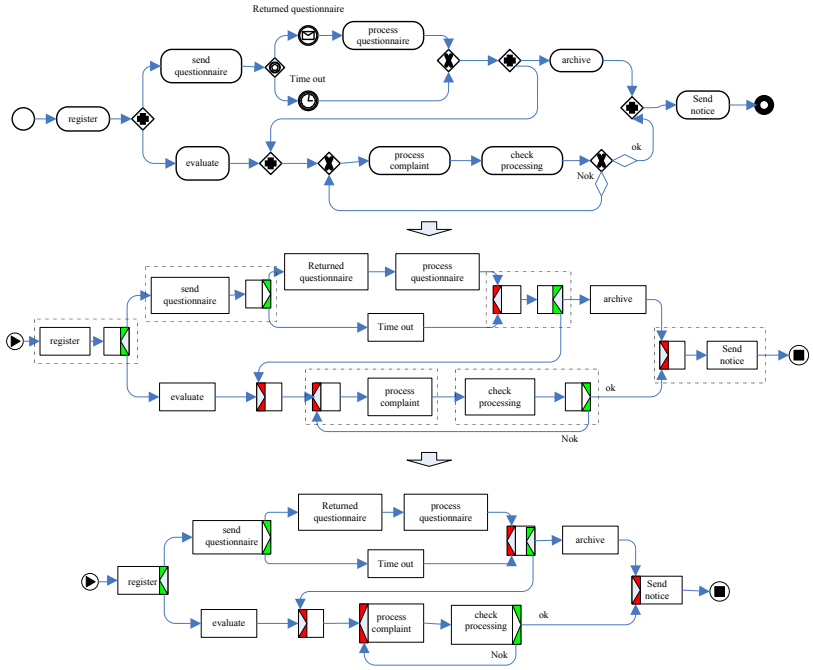

Figure 8. Translating the complaint handling process model into YAWL

TABLE II.
RESULTS OF IMPLEMENTING THE MAPPING TOOL TO EXISTING MODELS

Model No.	BPMN Model						YAWL net			
	tasks	events	gateways	subprocesses	sequence flows	message flows	composite tasks	atomic tasks	conditions	flows
0	5	4			9			7	6	15
1	6	5	2	3	8	2	3	16	12	30
2	20	10	5	6	30	9	6	47	20	81
3	20	4			28			22	6	34
4	2	2	2		6			6	4	10
5	2	4			3	1		6	4	10

Fig 10 shows a screenshot of ProM, generating the YAWL net from the No.1 BPMN model, which violate the *deaklock_free* and *soundness* properties via the analysis of YAWL editor[4]. Moreover, We detected model 2 contained incomplete process executions. Models 0, 3, 4 and 5 did not contain any errors.

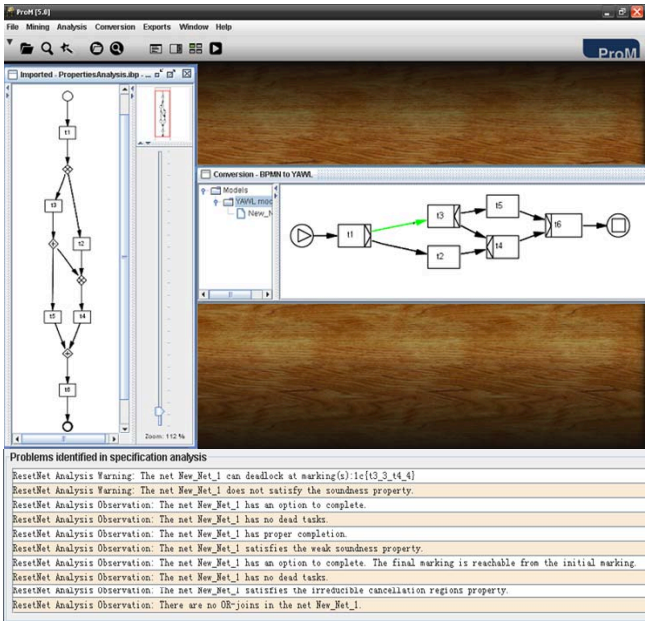

Figure 10. Screenshot of BPMN model to YAWL net in ProM and the analysis resulting by YAWL editor

V. CONCLUSION AND FUTURE WORK

Ongoing work aims at extending the BPMN2YAWL plugin in order to make the transformation reversible. After generating a model, the plugin will be able to propagate changes in the YAWL net into the BPMN diagram (and vice-versa) in order to maintain the models synchronized. We aim to analyze the whole BPMN process models with YAWL verification tools such as WofYAWL [11]. This will provide insight into the correctness of large enterprise models described in BPMN. We also plan to investigate process mining using BPMN processes and focusing on the OR-join semantic particularities of BPMN.

[4]http://www.yawl-system.com/

ACKNOWLEDGMENT

The authors wish to thank W.M.P. van der Aalst, Remco M.Dijkman and Xun PU for their careful reading of an earlier version of this paper.

REFERENCES

[1] O. M. Group, *Business Process Modeling Notation (BPMN) Version 1.0. OMG Final Adopted Specification*, Object Management Group, 2006.
[2] W. Aalst and A. Hofstede, "YAWL: Yet Another Workflow Language," *Information Systems*, vol. 30, pp. 245–275, 2005.
[3] WFMC, "Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface-XML Process Definition Language (XPDL) (WFMC-TC-1025)," Workflow Management Coalition, Tech. Rep., 2002.
[4] O. M. Group, *Unified Modeling Language: Superstructure. UML Superstructure Specification v2.0, formal*, Object Management Group, 2005.
[5] C. Ouyang, M. Dumas, A. Hofstede, and W. Aalst, "From BPMN process models to BPEL Web Services," in *On the 4th International Conference on Web Services (ICWS'06)*. IEEE, Chicago, Illinois, USA, 2006.
[6] C. Ouyang, M. Dumas, and A. Hofstede, "Pattern-based Translation of BPMN Process Models to BPEL Web Services," in *International Journal of Web Service Research*, vol. 5, 2007, pp. 42–62.
[7] C. Ouyang, M. Dumas, S. Breutel, and A. Hofstede, "Translating standard process models to BPEL," BPMcenter.org, Tech. Rep., 2005.
[8] M. Remco and C. Ouyang, "Formal Semantics and automated analysis of BPMN process models," Queensland University of Technology, Tech. Rep., 2007.
[9] Y. Peter and J. Gibbons, "A Process Semantics for BPMN," Oxford University Computing Laboratory, Tech. Rep., 2007.
[10] M. Dumas, A. Grosskopf, T. Hettel, and M. Wynn, "Semantics of Standard Process Models with OR-Joins," Queensland University of Technology, Tech. Rep., 2007.
[11] H. Verbeek and W. Aalst, "Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool," in *Application and Theory of Petri Nets 2000*, ser. Lecture Notes in Computer Science, M. Nielsen and D. Simpson, Eds., vol. 1825. Springer-Verlag, Berlin, 2000, pp. 475–484.
[12] W. Aalst, A. Hofstede, B. Kiepuszewski, and A. Barros, " Workflow Patterns," *Distributed and Parallel Databases*, vol. 14, pp. 5–51, 2003.
[13] O.-S. Coalition, *OWL-S: Semantic Markup for Web Services Version 1.1*, Web-Ontology Working Group, 2004.
[14] M. Jan, M. Michael, and N. Gustaf, "Transformation of yEPC business process models to YAWL," in *Proceedings of the 2006 ACM symposium on Applied computing (SAC '06)*, vol. 2. ACM, 2006, pp. 1262–1267.
[15] B. Anotnio and P. Razvan, "From BPEL Processes to YAWL Workflows," in *Proceedings of the 3 rd International Workshop on Web Services and Formal Methods (WS-FM 2006)*, ser. Lecture Notes in Computer Science, vol. 4184. Springer Berlin / Heidelberg, 2006, pp. 107–122.
[16] G. Decker, R. Dijkman, M. Dumas, and L. Garcia-Banuelos, "Transforming BPMN diagrams into YAWL nets," in *Business Process Management 6th International Conference, BPM 2008*, ser. Lecture Notes in Computer Science, M. R. M. Dumas and M.-C. Shan, Eds., vol. 5240. Springer-Verlag, Berlin, 2008, pp. 386–389.
[17] M. Remco, M. Dumas, and C. Ouyang, "Formal Semantics and Analysis of BPMN Process Models using Petri Nets," Queensland University of Technology, Tech. Rep., 2007.
[18] M. Wynn, D. Edmond, W. Aalst, and A. Hofstede, "Achieving a General, Formal and Deciable Approach to the OR-join in Workflow using Reset nets," in *Applications and Theory of Petri Nets 2005*, ser. Lecture Notes in Computer Science, vol. 3536. Springer-Verlag, 2005, pp. 423–443.
[19] W. Reisig, *Petri Nets: an Introduction*, 1st ed. Berlin: Springer-Verlag, 1985.
[20] W. Aalst and K. Hee, *Workflow Management: Models, Methods, and systems*, 1st ed. Cambridge, MA: MIT, 2002.
[21] T. Murata, "Petri nets: Properties, analysis, and applications," in *Proceedings of the IEEE*, vol. 77, no. 4. IEEE, 1989, pp. 541–580.
[22] J. Camara, C. Canal, J. Cubo, and A. Vallecillo, "Formalizing WSBPEL Business Process using Process Algebra," in *CONCUR'2005 Workshop on the Foundations of Coordination Languages and Software Architectures*, vol. ENTCS 154. Elsevier, 2005, pp. 159–173.
[23] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon, "Web Services Choreography Description Language 1.0," W3C Candidate Recommendation, Tech. Rep., 2005.

JianHong YE was born in Xiamen, China. He received his PhD degree in computer software and theory from University of Electronic Science and Technology of China in 2009, his MS degree in computer software and theory from the Xihua University in 2006, and his BS degree in computer science from the Fuzhou University in 2002.

He is currently a lecturer of computer science at Huaqiao University, China. He was a member of BETA laboratories from 2007 to 2008. His current research interests include application and theory of Petri nets, distributed and parallel computing.

JianHong YE is a member of China Computer Federation. He is a recipient of the Fujian Province National Science Foundation CAREER award.

Wen SONG was born in Chengdu, China. He received his MS degree in computer software and theory from the Guizhou University in 1991.

He is currently professor of computer science at Xihua University, China. His current research interests include theory of Petri nets and mathematical logic.

Prof. Wen SONG is a senior member of China Computer Federation and a member of Petri net Special Commission. He is a recipient of the China National Science Foundation CAREER award.