ABSTRACT
A number of business organizations are beginning to realize the importance of business process management (BPM) in achieving specific organizational goals through process automation. BPM developed as process-aware system that is based on a defined process model. This process model is translated into an executable language that can be executed by process engine. However, gaps often exist between the process model and the executable language. In other words, the process model cannot be fully represented in the executable language. In addition, a lack of formalized procedures during the model design phase will exacerbate ambiguities and inconsistencies, further complicating model verification and translation to the executable language. In this paper, we present a mechanism for transforming a BPMN-based process model into a formalized process model that can be better represented in WS-BPEL and verifying a model using a workflow-based verification mechanism, thus avoiding the ambiguities and inconsistencies inherent to BPMN-based process models.

KEY WORDS
BPMN, Business Process, Transformation to WS-BPEL, Formalized Model

1. Introduction
Information processing, networking, software, and the society that forms their basis constitute information technology. Today, information technology is the most effective tool for improving productivity and reducing cost. Thus, several organizations are beginning to realize the importance of Business Process Management (BPM) in achieving specific organizational goals through process automation. BPM developed as process-aware system that is based on a defined process model. Before using BPM, an organization must establish the process lifecycle, i.e., one complete cycle of a business process. This comprises six phases: modeling, simulation, deployment, execution, monitoring, and optimization [3]. During the modeling phase, an organization creates a business process model represented in Business Process Modeling Notation (BPMN), a standardized graphical notation for business process model [4]. This notation is then translated into Business Process Execution Language for Web Services (WS-BPEL) for simulation and execution. Therefore, the goal of the modeling phase is the development of a complete and consistent process model. However, existing verification tools for process models are mostly workflow-based. Thus, most current researches focused on transforming BPMN-based process models into workflow-based process models for verification [2]. This process inherently limited because some BPMN modeling elements cannot be mapped to a workflow-based model. Furthermore, many ambiguities and inconsistencies can arise in the BPMN-based process models owing to the lack of formalization, which can further complicate verification [8]. For this reason, recent research has focused only on a limited set of model elements or well-formed process models [2,5]. The fact remains that BPMN and WS-BPEL have different syntaxes and semantic structures: BPMN has a graph-oriented structure, and WS-BPEL a block-oriented structure. Thus a BPMN-based process model cannot be fully represented in WS-BPEL.

In this paper, we present a mechanism for transforming a BPMN-based process model into a formalized process model. Specifically, we present transformation mechanisms for

- reducing ambiguities and inconsistencies in BPMN-based process models that are used in actual business environments;
- complete mapping to WS-BPEL, i.e., without unconvertible model elements; and
- verifying a BPMN-based business process model using a workflow-based verification mechanism.

The remainder of the paper is organized as follows. Section 2 gives an overview of BPMN. In Section 3, broad surveys of related works are discussed. In Section 4,
we present mechanisms for the transformation to a formalized business process model. In Section 5, examples are provided. Finally, we present our conclusions in Section 6.

2. Background

BPMN is a standardized graphical notation for drawing business process models. It was developed by the Business Process Management Initiative and is currently maintained by the Object Management Group. The goal of BPMN is to provide a process modeling notation that is readily understood by all business analysts, technical developers, and business people who manage and monitor process management systems. BPMN is based on flowcharts and consists of four categories of modeling objects. Figure 1 shows the structure of BPMN and Figure 2 shows the four categories of modeling objects.

A flow object is an event, activity, or gateway. An event is represented by a circle and it affects the flow of the process. It usually has a cause (trigger) or impact (result). There are three types of events in BPMN: start, intermediate, and end events. An activity is represented by a rounded rectangle, and it represents the work performed within a business process. There are two types of activities: tasks and sub-process. A task is an atomic activity representing work to be performed. A sub-process is a non-atomic (compound) activity that includes other flow objects. A gateway is used to control the divergence and convergence of multiple sequence flows. In BPMN, there are five types of gateways: data-based XOR, event-based XOR, AND, OR, and complex gateways.

A connecting object is used to connect two graphical objects and show the flow progression of processes. There are three types of connecting objects: sequence flow, message flow, and association objects. Sequence flow is used to show the order in which activities are performed in a process. Message flow is used to show the flow of messages between two entities that are in separate pools. An association is used to associate information with flow objects.

A swimlane is a visual mechanism for organizing different activities into categories of the same functionality. There are two types of swimlanes: pools and lanes. A pool represents a participant in a process. To elaborate, it is a graphical container for portioning a set of activities. A lane is a sub-partition within a pool and is used to organize and categorize activities.

Artifacts help modelers and developers add information to a diagram, thus making the diagram more readable to the user. Artifacts are abstractions of data instances within the business process. BPMN provides three types of standard artifacts: data objects, groups, and text annotations. A data object is used to show which data are required or produced in an activity. A group is used to visually group different activities, but it does not affect the process flow. Text annotations allow the modeler to provide additional information to the reader of a BPMN diagram.

3. Related Works

3.1 Transformation Methods for Business Process Models

There has been considerable research on accomplishing transformation from a BPMN-based model to a workflow model when conducting workflow-based analysis and using verification tools. The reason for this is that workflow-based process models have various verification methods and analysis tools, but verification tools for BPMN-based process models have not been developed to a great extent. However, in most cases, the BPMN-based process model cannot be transformed completely to a workflow-based process model because some of the BPMN modeling elements are specified differently.

Major methods for business process transformation from a BPMN-based model to a workflow-based model are as follows.

- **BPMN to Petri Nets**: Petri nets have unique analysis and verification tools. However, existing workflow management systems do not use Petri nets as the modeling formalism, because Petri nets are not as easy to understand as verification methods based on activities and explicit control constructs [1]. In addition, Petri nets cannot support OR gateways, complex gateways, sub-processes, intermediate events, and other elements in BPMN [6].

- **BPMN to YAWL Nets**: YAWL nets are based on concepts of workflow nets, which are derived from Petri nets. YAWL has its own verification mechanisms, which are based on invariants and reset nets [7]. The method of using YAWL nets is the only method that considers OR gateways in a workflow-based process model. However, YAWL nets cannot support intermediate events, complex gateways,
activity (self) looping, or multiple instances in BPMN [10]. In this paper, we used the transformation of BPMN to YAWL nets for verification of process model especially in section IV.

3.2 Issues in Mapping BPMN to WS-BPEL

The BPMN specifications provide guidelines for mapping from BPMN to WS-BPEL, but they are only for limited sets of modeling elements or well-formed process models. Thus there are many issues have still not been fully clarified [4]. Table 1 lists modeling notations that cannot be converted to WS-BPEL.

### Table 1. BPMN Notations that cannot converted to WS-BPEL

<table>
<thead>
<tr>
<th>Modeling Notations that Cannot Be Converted to WS-BPEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Signal-triggered Start Event</td>
</tr>
<tr>
<td>- Cancellation-triggered Intermediate Event</td>
</tr>
<tr>
<td>- Signal-triggered and result-type Intermediate Event</td>
</tr>
<tr>
<td>- Cancellation-result-type End Event</td>
</tr>
<tr>
<td>- Signal-result-type End Event</td>
</tr>
<tr>
<td>- Link Intermediate Event</td>
</tr>
<tr>
<td>- Transaction Subprocess</td>
</tr>
<tr>
<td>- Ad hoc Subprocess</td>
</tr>
<tr>
<td>- Manual Task</td>
</tr>
<tr>
<td>- Complex Gateway</td>
</tr>
<tr>
<td>- Message Flows</td>
</tr>
<tr>
<td>- Swimlane (Pool, Lane, or Association)</td>
</tr>
<tr>
<td>- Artifact (Data Object, Group, or Annotation)</td>
</tr>
</tbody>
</table>

4. Transformation to a Formalized Business Process Model

BPMN-based process models allow, for example, the use of unconnected modeling objects, and modeling objects without any inputs or outputs. Furthermore, start and end events are not mandatory. Essentially, BPMN allows the use of modeling elements that cannot be represented in WS-BPEL. BPMN specifications state that multiple start events and unconnected modeling objects are to be avoided because the behavior of a process may be ambiguous and more difficult to understand [4]. However, the problem of representing several modeling elements in WS-BPEL is as yet not fully clarified. And as a result, several ambiguities and inconsistencies exist, which complicate model verification and conversion to WS-BPEL. Therefore, a mechanism is required for enabling transformation to a formalized process model.

Our transformation mechanism is presented in Figure 3. First the BPMN-based process model (source model) is transformed into a formalized process model (target model) on the basis of specific transformation rules. The formalized process model can subsequently be used for verification or can be represented in WS-BPEL for execution. Before the transformation, we should ensure the BPMN-based process model satisfies two assumptions.

**Assumption 1:** The BPMN-based process model should be a private (internal) process model, and not a collaboration process model. The BPMN-based process model can support a collaboration process. A collaboration process is the interaction between two or more business entities. However, workflow-based verification methods cannot support the collaboration process and the transformation is difficult. Thus, we only consider the private BPMN-based process model.

**Assumption 2:** Ad hoc processes in the BPMN-based process model should not be considered. An ad hoc process is a group of activities that have no predefined sequence relationship. An ad hoc process can be defined as a set of activities, but the sequence and number of performances for activities are not completely defined. Thus, ad hoc processes cannot be considered in the transformation.

We now define the BPMN-based process model for our transformation.

**Definition 1 (BPMN-based Process Model):** The BPMN-based business process model is the source model for the transformation processes.

- A BPMN-based business process model \(BP\) comprises event \(e\), activity \(a\), gateway \(g\), sequence flow \(f\), pool \(p\), and lane \(l\). That is, \(BP := \{e, a, g, f, p, l\}\).
- \(e\) can be divided into start event \(es\), intermediate event in a normal sequence flow \(ie\-NF\), intermediate event attached to the boundary of an activity \(ie\-AB\), and end event \(ee\). That is, \(e := \{es, ie\-NF, ie\-AB, ee\}\).
- \(es\) has three different types of triggers: message, timer, and conditional triggers. That is, \(es := \{\text{type(trigger)}\} = \{\text{message, timer, conditional}\}\).
- \(ie\-NF\) is used in a normal sequence flow. It has four different types of triggers: message, timer, conditional, and link. That is, \(ie\-NF := \{\text{type(trigger)}\} = \{\text{message, timer, conditional, link}\}\).
- \(ie\-AB\) is an event attached to the boundary of an activity. It has six different types of triggers: message, timer, error, and conditional triggers. That is, \(ie\-AB := \{\text{type(trigger)}\} = \{\text{message, timer, error, compensation, conditional, multiple}\}\).
- \(ee\) has three different types of results: message, error, and terminate results. That is, \(ee := \{\text{type(result)}\} = \{\text{message, error, terminate}\}\).
- Multiple \(es\) and multiple \(ee\) can be used in \(BP\).
- \(es\) and \(ee\) are not always required in \(BP\).
- \(es\) may have multiple outgoing sequence flows, but no incoming sequence flows.
- \(ee\) may have multiple incoming sequence flows, but no outgoing sequence flows.
- \(ie\-NF\) has only one incoming sequence flow and only one outgoing sequence flow.

![Figure 3. Proposed mechanism for transformation to a formalized process model](image-url)
- In BP, a cannot have an incoming sequence flow, and can have only one outgoing sequence flow.
- a in BP can be divided into task t and subprocess sp. That is, $a \to [t, sp]$.
- t comprises five different task types: service, receive, send, user, script types. That is, $t := \{\text{task} | \text{type} = \text{service, receive, send, user, script}\}$.
- a may have multiple incoming sequence flows and multiple outgoing sequence flows.
- g in BP can be divided into an AND gateway $g$-AND, OR gateway $g$-OR, data-based XOR gateway $g$-XOR(D), event-based XOR gateway $g$-XOR(E), and complex gateway $g$-COM. That is, $g := \{g$-AND, $g$-OR, $g$-XOR(D), $g$-XOR(E), $g$-COM\}.
- g may also have multiple incoming sequence flows and multiple outgoing sequence flows.
- Loop patterns in BP can be either activity (self) looping or have multiple instances; they are created by connecting a sequence flow to an “upstream” object.

We now define the formalized process model in our transformation mechanism.

**Definition 2 (Formalized Business Process Model):** The formalized business process model is the target model for the transformation. This model can be completely represented in WS-BPEL. This model is used for workflow-based verification.

- The formalized business process model $f$-BP comprises an event $e$, activity $a$, gateway $g$, and sequence flow $f$.
  That is, $f$-BP := \{e, a, g, f\}.
- e in $f$-BP can be divided into start event $es$ and end event $ee$. That is, $e := \{es, ee\}$.
- es has only one type of trigger: message trigger. That is, $es := \{\text{type(trigger)} = \text{message}\}$.
- ee has four different types of results: message, error, compensation, and terminate. That is, $ee := \{\text{type(result)} | \text{result} = \text{message, error, compensation, terminate}\}$.
- A single $es$ and single $ee$ must be used in $f$-BP.
- $es$ has just one outgoing sequence flow and must not have an incoming sequence flow.
- $ee$ has only one incoming sequence flow and must not have an outgoing sequence flow.
- a in $f$-BP can only be an atomic activity (task) $t$. That is, $a := \{t\}$.
- t is composed of three different task types: service, receive, send types. That is, $t := \{\text{task} | \text{type} = \text{service, receive, send}\}$.
- a has only one incoming sequence flow and only one outgoing sequence flow.
- g in $f$-BP can be divided into an AND gateway $g$-AND, OR gateway $g$-OR, and data-based XOR gateway $g$-XOR. That is, $g := \{g$-AND, $g$-OR, $g$-XOR\}$.
- g in $f$-BP is used for only one purpose (split $g(s)$ or join $g(j)$).
- When $g$ is used for splitting, it has only one incoming flow and multiple outgoing sequence flows.
- When $g$ is used for joining, it has multiple incoming flows and only one outgoing sequence flow.
- $f$-BP has only one loop pattern, which is created by connecting a sequence flow to an upstream object.

These are the 14 transformation rules required to create a formalized business process model. The rules are applied to the BPMN-based process model as follows:

### 4.1 Transformation of Ambiguities and Inconsistencies in Business Process Model

**Rule 1) Transformation of a Process Model That Does Not Have Start and End Events:** $es$ and $ee$ are optional objects in BP and so may not be present. By contrast, a valid WS-BPEL process must begin with a $<\text{receive}>$ element that has $\text{createInstance}$ set to “yes,” and must end with a $<\text{reply}>$ element for execution by a process engine [9]. According to the BPMN specification, these $<\text{receive}>$ and $<\text{reply}>$ elements represent $es$ and $ee$ in BP [4]. However, if $t := \{\text{task} | \text{type} = \text{receive}\}$ is present at the initial part, it can be converted into the necessary $<\text{receive}>$ element that has $\text{createInstance}$ set to “yes.” Further, if $t := \{\text{task} | \text{type} = \text{send}\}$ is present in the final part, it can be converted into the $<\text{reply}>$ element in WS-BPEL. Thus, the absence of start and end events can be overcome. However, the ambiguity created by the absence of start and end events leads to two problems: Firstly, it is difficult to identify the kind of input and output data used within the model. For example, if a process model uses $es$, the attributes of $es$ in BP will allow confirmation of the type of data. But if $t := \{\text{task} | \text{type} = \text{receive}\}$ is used instead, the input data is declared in the attributes of $t := \{\text{task} | \text{type} = \text{receive}\}$ and not presented in the process model. The use of wrong input data may cause run-time errors in the process engine. Secondly, these $es$ and $ee$ events are essential for workflow-based verification methods. Most workflow-based verification methods involve scanning the inside of the process model from the start event to the end event. It would be difficult to implement a workflow-based verification method without such events. Therefore, in the formalized process $f$-BP, a single start event $es$ and single end event $ee$ must be used. We thus transform a process that does not have $es$ and $ee$ into a process that does. Furthermore, we set the message trigger type in $es$ ($es := \{\text{type} | \text{trigger} = \text{message}\}$) and message result type in $ee$ ($ee := \{\text{type} | \text{result} = \text{message}\}$).

**Rule 2) Transformation of Start Event with Timer and Conditional Triggers:** A start event $es$ has three different types of triggers in BP. Among them, $es := \{\text{type} | \text{trigger} = \text{timer}\}$ can be set to a specific time or date when the process will be started. Furthermore, $es := \{\text{type} | \text{trigger} = \text{conditional}\}$ can be used to set a specific condition that must be satisfied for the process to start [4]. However, these trigger types are converted into $<\text{receive}>$ elements in WS-BPEL [9]. Thus, the trigger types are converted into the same element with $es := \{\text{type} | \text{trigger} = \text{message}\}$. This $<\text{receive}>$ element cannot contain any specific time/date or condition. Therefore, semantic inconsistencies can exist between the process model and the executable code. In the formalized process $f$-BP, $es := \{\text{type} | \text{trigger} = \text{timer}\}$ and $es := \{\text{type} | \text{trigger} = \text{conditional}\}$ are transformed as in Figures 4 and 5, respectively.
Rule 3) Transformation of Task with User and Script Types: A task \( t \) has five different types in \( f-BP \). Among the different types of \( t \), user type \( t := \text{task(type=user)} \), and script type \( t := \text{task(type=script)} \) can be used in the transformation. A \( t := \text{task(type=user)} \) is a task where a human performer performs the task with the assistance of a software application. A \( t := \text{task(type=script)} \) is a contained a script in a language that the process engine can interpret. Thus when the \( t := \text{task(type=script)} \) is ready to start, the process engine will execute the script [4]. However, these two different types of tasks are converted into same \(<\text{invoke}>\) elements in WS-BPEL [9].

(Generally, the \( t := \text{task(type=service)} \) is converted into \(<\text{invoke}>\) element in WS-BPEL.) Fortunately this \(<\text{invoke}>\) element has several attributes that can handle these \( t := \text{task(type=user)} \) and \( t := \text{task(type=script)} \). (i.e., Attributes of \(<\text{invoke}>\) element: inputVariable that contains Performers: String, partnerlink, portType, and etc.) In other words, these tasks has no semantically difference with \( t := \text{task(type=service)} \). Therefore, in \( f-BP \), transform \( t := \text{task(type=user)} \) and \( t := \text{task(type=script)} \) into a \( t := \text{task(type=service)} \) for reduce these ambiguities: Most of the tasks in process models are \( t := \text{task(type=service)} \), so that’s why we transform \( t := \text{task(type=service)} \) and \( t := \text{task(type=script)} \) into a \( t := \text{task(type=service)} \).

Rule 4) Transformation of Multiple Outgoing Sequence Flows from a Start Event and Multiple Incoming Sequence Flows to an End Event: A start event \( es \) may have multiple outgoing sequence flows and an end event \( ee \) may have multiple incoming sequence flow in \( BP \). WS-BPEL represents a sequence flow with the \(<\text{sequence}>\) or \(<\text{link}>\) elements [4,9]. The \(<\text{sequence}>\) element is used to define a set of flow objects to be performed sequentially. And the \(<\text{link}>\) element is used to define a set of parallel flows to flow objects. This \(<\text{link}>\) element must be used within a \(<\text{flow}>\) element in WS-BPEL. However, this \(<\text{link}>\) element, which is used within \(<\text{flow}>\) element can be also matched with \( g-\text{AND} \) gateways in \( BP \). According to Def 2, \( es \) has just one incoming sequence flow and no outgoing sequence flows in \( f-BP \). And \( ee \) has just one incoming sequence flow and no outgoing sequence flows in \( f-BP \). Moreover, current workflow verification methods can handle only one incoming sequence flow and one outgoing sequence flow from the end and start events. However, WS-BPEL provides the \(<\text{if}>\) and \(<\text{flow}>\) elements to control multiple incoming and outgoing sequence flows. These elements can be matched with \( g-\text{XOR} \) and \( g-\text{AND} \) gateways in \( BP \). Therefore, we transform the multiple incoming sequence flows to \( es \) into exactly one incoming sequence flow using the AND-split gateway \( g-\text{AND}(s) \). Further, we transform the multiple incoming sequence flows to \( ee \) into exactly one incoming sequence flow using the XOR-join gateway \( g-\text{XOR}(i) \).

Rule 5) Transformation of a Task with Multiple outgoing Sequence Flows and Multiple Incoming Sequence Flows: Task \( t \) may have multiple outgoing sequence flows and multiple incoming sequence flows in \( BP \), but in \( f-BP \) has only one incoming sequence flow and only one outgoing sequence flow. Due to the same reasons described in Rule 5, we transform multiple outgoing sequence flows from \( t \) into one outgoing sequence flow by using an AND-split gateway \( g-\text{AND}(s) \) after the object; this \( g-\text{AND}(s) \) send all the outgoing sequence flows of the object. And we transform multiple incoming sequence flows to a task into one incoming sequence flow by using an XOR-join gateway \( g-\text{XOR}(i) \) before the corresponding object; this \( g-\text{XOR}(i) \) receives all the incoming sequence flows of the object.
Rule 6) Transformation of a Gateway with Multiple Incoming and Multiple Outgoing Sequence Flows: A gateway in f-BP is used for only one purpose (split s(i) or join s(j)). Consequently, we decompose a gateway with multiple incoming and outgoing sequence flows into a split s(i) that has all the outgoing sequence flows and a join s(j) that has all the incoming sequence flows.

![Figure 6. Transformation of a gateway with multiple incoming and outgoing sequence flows](image)

4.2 Transformation of unsupportable model elements in Workflow Verification Methods

Rule 7) Transformation of an Intermediate Event in a Normal Sequence Flow: An intermediate event ie-NF that is placed within the normal sequence flow of a process can be used in BP. According to the BPMN specifications, all trigger-intermediate events ie-NF in BP can be represented by proper WS-BPEL elements except the link-trigger-type intermediate event ie-NF := type(trigger=link) [4,9]. However, current workflow verification methods cannot handle ie-NF. Therefore, we should transform ie-NF into a relevant graphical object in f-BP. In BP, there are several types of ie-NF. Thus, there are different methods of transformation for each. Firstly, ie-NF := type(trigger = message) states that a specific message data arrives from a previous object to trigger an event. It is represented by <receive> elements that have createInstance set to “no” or <invoke> elements in WS-BPEL. However, a receive-type task t := task (type = receive) and a service-type task t := task (type = service) are represented by <receive> elements that have createInstance set to “no” and <invoke> elements in WS-BPEL as well. Thus, ie-NF := type(trigger = message) and receive/service-type tasks in BP are semantically equivalent. Therefore, in f-BP, we transform ie-NF := type(trigger = message) into a receive/service-type task. Secondly, ie-NF := type(trigger = timer) represents a delay, which is either a command to wait of a specified duration or until a specific time. This ie-NF := type(trigger = timer) is represented by a <wait> element in WS-BPEL. If ie-t is used in BP, we transform ie into a time (wait) atomic task whose execution time can be changed using WS-BPEL time properties. Finally, ie-con represents a wait for some expression involving process data to become true. If ie-con is used in BP, we transform ie into an event task whose input/output rules can be changed using the task attribute in the BPMN specifications. Input/output rules are a collection of expressions, each of which specifies a required relationship between the input and output. Therefore, if the task is instantiated with a specified input, that task shall be completed with a specified output.

Rule 8) Transformation of an Intermediate Event Attached to the Boundary of an Activity: An intermediate event ie-AB represents an exception-handling event in BP. This rule is for the transformation of an exception-handling intermediate event into relevant graphical objects in f-BP. Exception-handling means that if a signal is received by ie-AB, we abort the task that ie-AB is attached to and proceed down the sequence flow (exception flow), coming out of the event. If no signal is received by ie-AB, the task is carried out normally and we proceed through the sequence flow (normal flow) directly out of the task. This exception-handling intermediate event ie-AB is converted into <faultHandler> in WS-BPEL [9]. There are different methods for transforming different exception-handling events. Among the several different types of ie, the timer-triggered ie-t, error-triggered ie-e, and cancellation-triggered ie-c can be used in the transformation. First, if ie-t is used, we transform ie into a wait task and split normal and exception flows using an XOR-split gateway g-XOR(). Second, ie-e represents a system fault in a task. This means the task cannot be completed successfully. If ie-e is used in BP, we transform ie into an error task and split normal and exception flows using an XOR-split gateway g-XOR(). Finally, ie-c is used for a transaction subprocess, but subprocess sp is not a graphical object in f-BP. Thus, we do not consider ie-c in exception handling.

Rule 9) Transformation of an Event-based XOR Gateway: An event-based XOR gateway g-XOR() is always followed by the triggering of an intermediate event ie-NF or a t := task(type=receive) in BP. According to Rule 7, however, ie-NF needs to be transformed into a task. This means that g-XOR() should be transformed into relevant graphical objects in f-BP. If g-XOR() is used in BP, we transform the event-based XOR gateway into a data-based XOR gateway g-XOR().

Rule 10) Transformation of an Activity (Self) Loop: An activity may have attributes that specify special behaviors such as repetition (i.e., the activity is executed multiple times sequentially) in BP. There are two variants of sequential activity repetition: one corresponds to a TestTime = before loop and the other to a TestTime = after loop. However, workflow verification methods cannot handle these activity loops. According to the BPMN specifications, a loop can be represented in WS-
BPEL by a `<while>` element, but an activity is usually represented by an `<invoke>` element. Thus, these activity loops must be transformed into other process model patterns. In f-BP, the transformation of an activity loop is determined by the `TestTime` attribute. If `TestTime = after`, the task is performed at least once. Thus, it is equivalent to a do-until loop. If `TestTime = before`, the task may not be performed. Thus, it is equivalent to a while-do loop. Therefore, we transform these activity loops into do-until and while-do loops.

![Transformation of a (self) looping activity](image)

**Rule 11) Transformation of Multiple Instances:** In the BPMN specifications, an activity may have multiple instantiations (i.e., the activity is executed multiple times concurrently). There are two variants of multiple instance activity: serial and parallel. Transformation of multiple instances is determined by the `MI_Ordering` attribute. If `MI_Ordering = serial`, we transform the activity into the serial type. If `MI_Ordering = parallel`, we transform the activity into the parallel type.

**Rule 12) Transformation of a Sub-process:** A sub-process `sp` is not a graphical object in f-BP. In our transformation mechanism, an internal process (low-level process) in `sp` is regarded as a stand-alone process and is subject to our transformation rules. After the transformation of the internal processes of `sp`, we transform the sub-process into a normal task as a high-level process.

### 4.3 Representation of Unconvertible Model Elements in WS-BPEL

**Rule 13) Transformation of a Link-type Intermediate Event:** A link event is a mechanism for connecting two sections of a process. A link event can be used to create a loop or to avoid long sequence flow lines in `BP`. However, link events cannot be represented in WS-BPEL and only start and end events are present in f-BP (i.e., `e` = `{es, ec}`). This means we should transform a link event into relevant graphical objects in f-BP. If a link event is used in `BP`, we remove the link events and directly link the selected tasks.

**Rule 14) Transformation of a Complex-split Gateway:** The behavior and usage of a complex gateway `g-COM` have not been established for the case of mapping to WS-BPEL. In addition, `g-COM` is not a graphical object in f-BP. Therefore, `g-COM` should be transformed into a relevant graphical object.

Table 2 shows a truth table for gateway control flow. It is a simple table for structural anomaly detection in a business process model. The table shows the same control-flow results for OR-split and complex-split gateways, which are joined by other gateways. Therefore, OR-split and complex-split gateways are not different from a control-flow perspective. If a complex-split gateway is used in `BP`, we transform it into an OR-split gateway.

**Table 2. Truth table for Gateway Control-Flow**

<table>
<thead>
<tr>
<th>Gateways</th>
<th>AND-Join</th>
<th>OR-Join</th>
<th>XOR-Join</th>
<th>Complex-Join</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>Split</td>
<td>(Lack of Sync.)</td>
<td>(Deadlock)</td>
<td>(Deadlock)</td>
<td>True for <code>N</code> = 1</td>
</tr>
<tr>
<td>OR</td>
<td>False</td>
<td>True</td>
<td>False</td>
<td>True for <code>N</code> = 1</td>
</tr>
<tr>
<td>Split</td>
<td>(Deadlock)</td>
<td>True</td>
<td>False</td>
<td>False for <code>N</code> &gt; 1</td>
</tr>
<tr>
<td>XOR</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True for <code>N</code> = 1</td>
</tr>
<tr>
<td>Split</td>
<td>(Deadlock)</td>
<td>False</td>
<td>False</td>
<td>False for <code>N</code> &gt; 1</td>
</tr>
<tr>
<td>Complex</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True for <code>N</code> = 1</td>
</tr>
<tr>
<td>Split</td>
<td>(Deadlock)</td>
<td>False</td>
<td>False</td>
<td>False for <code>N</code> &gt; 1</td>
</tr>
</tbody>
</table>

### 5. Transformation Example

We use a business process focusing on supply fulfillment to illustrate the transformation mechanism. Figure 9 shows the supply fulfillment process model using BPMN.

![Supply Fulfillment Business Process Model](image)

Following the transformation rules in Section III, we now translate the above BPMN-based process model to a formalized process model. Figure 10 shows the transformations are shown in Figure 10.

![Transformation to a Formalized Business Process Model](image)

A formalized business process model may also be transformed into YAWL nets. YAWL uses concepts from workflow nets, which are derived from Petri nets. YAWL
has its own verification mechanisms, which are based on invariants and reset nets [7]. Therefore, the presented BPMN-based process model can verify process anomalies by using workflow-based verification mechanisms. The mapping of BPMN elements to WS-BPEL can be accomplished without process anomalies.

![Figure 11. Transformation to a YAWL nets](image1)

![Figure 12. Example of Workflow-based process model verification result using YAWL nets and translating to WS-BPEL](image2)

As a result, the BPMN-based process model can subsequently be used for verification or can be represented in WS-BPEL for execution by formalized process model.

6. Conclusion

In this paper, we presented a mechanism for transforming a BPMN-based process model to a formalized process model that does not suffer the ambiguities and inconsistencies of the BPMN-based process model. This transformation mechanism will be useful for reducing ambiguities and inconsistencies in actual business environments. The mechanism also enables the use of workflow verification methods for detecting process anomalies. However, our transformation mechanism is only suitable for private (internal) processes.

In future research, we plan to extend the transformation mechanism to a collaboration process and develop transformation algorithms.

Acknowledgements

This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MOST) (No. R01-2007-000-20135-0)

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2007-313-D00757)

References


