A Methodology for Generating BPEL Models from a Business Process Textual Description

Wiem Khlif[1], Nadia Aloui[1] and Nourchène Elleuch Ben Ayed[2]
[1]*Mir@cl Laboratory, University of Sfax, Sfax, Tunisia*
[2]*Higher Colleges of Technology, ADW, U.A.E.*

Keywords: BPEL Model, Alignment Textual Description.

Abstract: Generating BPEL model from a Textual Description (TD) is essential to its reliable analysis. Nonetheless, creating or preserving TD-BPEL alignment is an issue when an organization develops or changes a BEPL model. Hence, it is possible to detect misalignment between BPEL model and text if changes are not applied to both representations. This paper proposes a new methodology that assists business analyst to derive BPEL models, which are aligned with their corresponding textual description. It uses the business concept's template that is augmented by a set of transformation rules. Compared to existing methods, our methodology offers a complete alignment, which covers all BPEL elements. It is evaluated experimentally using the recall and precision rates.

1 INTRODUCTION

Business process modelling constitutes an important asset for expressing software requirements and handling organizational change. What becomes challenging with business process modelling is the fact that the development of Business Process Models (BPM) is a time-consuming task due to the availability of different notations. The business analyst uses Business Process Modelling Notation (BPMN) for designing and improving the business process, whereas Business Process Execution Language (BPEL) is used by the technical analyst and programmer when implementing it. These BPMN models are as blueprints to define the BPEL model.

Several approaches were proposed in the literature (Aysolmaz et al., 2018) (Doux et al., 2013) (Van der Aa et al., 2019) to increase the agility of model. These approaches are classified into two categories: the first category addresses the transformation of BPMN model into a textual description and vice versa (Aysolmaz et al., 2018) (Van der Aa et al., 2019), while the second category deals with the transformation from the BPMN model into BPEL and vice versa (Doux et al., 2013).

Regarding the first category, the transformation from BPMN model to textual description can be automatic or semi-automatic (Aysolmaz et al., 2018).

For the transformation from textual description to BPMN model, (Van der Aa et al., 2019) present an automatic approach that combines existing tools from natural language processing in an innovative way and augments them with a suitable anaphora resolution mechanism.

For the second category, (Doux et al., 2013) propose a set of BPMN patterns and their corresponding BPEL structured constructs as well as an algorithm automating this mapping. The latter turns out to be rather complex because of inherent differences between these two languages: BPMN process models are graph-oriented (with only minor topological restrictions), while BPEL process definitions are block-structured.

To overcome these limitations, this paper addresses the challenge of deriving a BPEL model from a textual description. We propose to enhance MONET (a syste**M**atic derivati**O**n of a bpm**N** mod**E**l from business process Textual description) methodology (Khlif et al., 2020) which mainly focus on the derivation of BPMN models from given textual descriptions. This generation is based on the business concepts used as a mean to split the business process textual description into goal-specific descriptions. Each business concept (BC) is described by a well-formed template respecting a set of linguistic patterns. In addition, MONET proposes business transformation rules that transform each linguistic

Khlif, W., Aloui, N. and Ben Ayed, N.
A Methodology for Generating BPEL Models from a Business Process Textual Description.
DOI: 10.5220/0010457403230330
In *Proceedings of the 16th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2021)*, pages 323-330
ISBN: 978-989-758-508-1

pattern to its corresponding BPMN elements. The enhancement of MONET methodology goes through differents steps. First, defining new transformation rules that aim to derive BPEL from a textual decription (TD). Second, evaluating the quality of the produced model in terms of their precision and domain coverage.

The remainder of the paper is structured as follows: Section 2 presents an overview of MONET methodology (a syste**M**atic derivati**O**n of a bpm**N** mod**E**ls from business process Textual description) and the related works. Section 3 determines the transformation rules that allow the derivation of BPEL model from the business concept template. Section 4 illustrates our tool MONEET that implements the transformation rules and the ontology to produce BPEL model and evaluate it through the recall and precision. Finally, Section 5 summarizes the results and draws the future works.

2 BACKGROUND

2.1 Overview of MONET

MONET (a syste**M**atic derivati**O**n of a bpm**N** mod**E**ls from business process **T**extual description) is a methodology that generates BPMN model from its corresponding documentation (Khlif et al., 2020). It is composed of two phases: BPMN model derivation phase and evaluation phase. The derivation phase is organized around a set of three steps that are a pre-processing, a definition of the transformation rules, and their implementations. A pre-processing step during which the business analyst cleans first the business process description, written with a natural language. Then, the business analyst identifies the business goals to divide the business process description into business concepts. For each business concept, the business analyst prepares its TD according to a specific template (Khlif et al., 2020) which is composed of three blocks. The first block summarizes the business concept. The second block describes the main, alternative, and error scenarios. The third block illustrates business objects as result of the execution of the BC. For more details, reader can refer to (Khlif et al., 2020).

A transformation-definition step during which the business designer defines an ontology to analyze the semantic of the business concepts' template. It is used to define the business transformation rules.

A Transformation-implementation step during which the business engineer formalizes/implements the transformation rules, which provide for the automated generation of the BPMN model.

2.2 Related Work

Many researchers proposed a number of methods for generating BPMN model from its textual description and vice versa (Aysolmaz et al., 2018) (Van der Aa et al., 2019), and from model to another at different abstraction levels ie. from BPMN to BPEL and vice versa (Doux et al., 2013).

On the one hand, model-to-text transformation (Aysolmaz et al., 2018) proposed a semi-automated approach technique that transforms process models into intuitive natural language texts.

On the other hand, text-to-model transformation techniques cover a diversity of models. (Van der Aa et al., 2019) offered an approach that derives automatically BPMN models from natural language text based on a tailored Natural Language Processing technique that identifies activities and their inter-relations. The authors of (Doux et al., 2013) address the issues related to model-to-model transformation from BPMN to BPEL and vice versa. They proposed pattern-based transformation from BPMN to BPEL using ATL.

What become problematic with these works (Doux et al., 2013) is the patterns identification and the different types of process models: BPEL (block-based) and BPMN (graph based). To overcome these limits, (Yongchareon et al., 2020) present a unified framework, namely UniFlexView, for supporting automatic and consistent process view construction. Based on this framework, process modellers can use the proposed View Definition Language to specify their view construction requirements disregarding the types of process models.

In summary, many researchers studied the transformation between BPMN model and its textual description or between BPMN and BPEL models. However, there is no works that focus on the generation of BPEL from the documentation of the business process. Our objective is to facilitate the task of the business designer and developer to obtain BPEL model at a high level of granularity.

3 FROM TEXTUAL DESCRIPTION TO BPEL MODEL

We propose to extend MONET methodology to generate BPEL model from its textual description.
We called the new methodology MONEET (a syste**M**atic derivati**O**n of a bpm**N** and bp**E**l mod**E**ls from business process Textual description).

MONEET is composed of two phases: BPEL model derivation phase and evaluation phase. As MONET, the derivation phase is organized around a set of three steps that are a pre-processing, a definition of the transformation rules, and their implementations. The pre-processing step is quite similar to MONET. However, we defined new transformation rules to generate the BPEL model. We describe in the following sections the transformation rules and we evaluate the obtained model.

We defined eighteen transformation rules. Each transformation rule operates on the different components of the template.

R1. Each trigger is transformed into an event that will be linked to the first element of the current business concept. Based on the trigger type, we add the corresponding event.

R1.1. If the trigger type describes the time, so add the following code:

- Case of start event, which is only applied on an Event Sub-Process :

```
<eventHandlers> <onAlarm>[timer-spec]
 <scope> [current business concept] </scope>
 </onAlarm>
</eventHandlers>
```

- Case of intermediate event:

```
<wait name = "[trigger-name]" for="[trigger-TimeCycle]"/>
```

R1.2. If the trigger type describes any action that refers to a specific addressee and represents or contains information for the addressee, so add the following code:

- Case of start event:

```
<receive name= [trigger-name] partnerLink = [Participant] createInstance="yes|no"/>
```

- Case of intermediate event

```
<receive name= [trigger-name] partnerLink = [Participant] createInstance="no"/>
```

We note that conditional and signal event cannot be mapped to BPEL.

R2. Each participant is transformed into partner link depending on its type.

R2.1. If a participant invokes the BPEL process, so add the following code:

```
<process name=[process.name]/>
<partenerLink name="[participant name]" myRole = "[processNameProvider]"/>
```

R2.2. if a participant is invoked by the BPEL process, so add the following code:

```
<partnerLink name="[participant name]" myRole = "[processNameProvider]"
partnerRole= "[ParticipantNameRequester]"/>
```

R3. Each relationship between a business concept and its successors respects the linguistic pattern: **[<Pre-condition>] <Current Business Concept ID> is related <sequentially | exclusively | parallel | inclusively>to<Business Concept ID>**.

R3.1. If the relationship is <sequentially>, then add the following code:

```
<sequence>  <scope> [current BC] </scope>
  <scope> [successor of the current BC] </scope>
</sequence>
```

R3.2. If the relationship is <parallel>, then add the following code:

```
<flow> [current BC] [successor of the current BC]
</flow>
```

R3.3. If the relationship is <exclusively> and there is a precondition, then add following code:

```
<if >[precondition] [current BC]
 <else> [BC successor] </else> </if>
```

R3.4. If the relationship is <inclusively> and there is a precondition, add the following code:

```
<flow> <links>
   <link name= "link1" >
   <link name= "link2" >
   <link name= "link3" >
   <link name= "link4" > </links>
  <source linkName="link1"> <transitioncondition>
   precondition1 <transitioncondition>
</source>
<source linkName= " link2" >
   <transitioncondition>precondition2
<transitioncondition> </source>
<flow>
 <target linkName= "link 1" > </target>
 <source linkName= "link 3" > </source>
 [current BC] </flow>
<flow> <target linkName= " link 2" > </target>
 <source linkName= " link 4" > </source>
 [business concept successor] </flow>
<target linkName= " link3" ></target>
<target linkName= " link4" ></target>
</flow>
```

R4. For each step of a BC's scenario respecting the linguistic pattern: **[<Pre-condition>] <Task#> < Task Description > <Task Type >**, then add the following:

R4.1. If the task description is « Action verb + BusinessObject », then add an invoke activity presented by the following BPEL code and call R4.4, R4.5, and/or R4.6.

```
< invoke name="[Action verb + BusinessObject]"
  portType="[Task-operation-interface]"
operation="[Task-operation]" > </invoke>
```

R4.2. If the task description is « Action verb + NominalGroup », then add an invoke activity that has the same name of the pattern and call R4.4, R4.5 and/or R4.6. If the pre/post-modifier is a noun that merely represents a pure value, so there is no variable (data object) to add. Otherwise, if the pre/post-modifier is a complex noun (an entity) then add a variable corresponding to the data object.

```
<invoke name="[Action verb + NominalGroup]"
   partnerLink="[participant]"
   portType="[Task-operation-interface]"
operation="[Task-operation]"> </invoke>
```

R4.3. If the task description is « CommunicationVerb + BusinessObject | NominalGroup+ [[to ReceiverName(s)] | [from SenderName]] », then add the following code corresponding to an invoke or receive activity that has the same name of the pattern and a variable for each BusinessObject or NominalGroup.

```
<invoke
name="[CommunicationVerb+BusinessObject|Nomina
lGroup]" partenerLink= "[ReceiverName]">
     <toPart part ="[variable.name]"
fromVariable="[variable.name]"/>
</invoke>
```

Or

```
<receive
name="[CommunicationVerb+BusinessObject|Nomina
lGroup]"
partnerLink=[SenderName] portType="[Task-
operation-interface]" operation="[Task-operation]" >
     <fromPart part ="[variable.name]" fromVariable =
"[variable.name]"/>
</receive>
```

R4.4. If the task type is ActivePER, then add a variable presented by the following code:

```
<fromPart part = "[variable.name]" fromVariable =
"[variable.name]"/>
```

R4.5. If the task type is ActiveRET, then add a variable expressed as follows:

```
<toPart part = "[variable.name]" fromVariable =
"[variable.name]"/>
```

R4.6. If the task type is ActiveREP, then add a reply activity represented as follows:

```
<reply> </reply>
```

R5. Each relationship between the task and its successors respects the linguistic pattern: **[<Pre-condition>] <Current Task ID> is related <sequentially | exclusively | parallel | inclusively>to<Task ID>**

R5.1. If the relationship is <sequentially> and if the current activity and its direct successor are in the same main process, then add the following code:

```
</sequence> [current task] [successor task]
</sequence>
```

Otherwise add

- Receive activity

```
<receive name="[direct successor task]"
partenerLink="[participant]"></receive>
```

- Invoke activity

```
<invoke name= "[direct successor task]"
partenerLink="[participant]" ></invoke>
```

R5.2. If the relationship is <parallel>, then add

```
<flow> [current task] [successor task] </flow>
```

R5.3. If the relationship is <exclusively> and there is a precondition, add the following code:

```
<if >[precondition][current task]
 <else> [ task successor] </else> </if>
```

R5.4. If the relationship is <inclusively> and there is a precondition, then add the following code:

```
<flow> <links>
<flow> <links>
<link name= " link1" >
<link name= " link2" >
 <link name= " link3" > </links>
<source linkName=”link1”>
<transitioncondition>precondition1
<transitioncondition> </source>
<Source linkName= " link2" >
<transitioncondition>precondition2
<transitioncondition> </source>
<flow>
<target linkName= " link 1" > </target>
<source linkName= " link 3" > </source>
[current task] </flow>
<flow> <target linkName= " link 2" > </target>
<target linkName= " link3" ></target> </flow>
```

R5.5. If the relationship is <sequentially>, and there is a <complete> construct related to a task, then add an end event based on the following code:

- None end event

```
<empty name="[e-name]"> </empty>
```

- Message End Events

```
<invoke name="[e-name]"
 partnerLink="[Q, e-operation-interface]"
portType="[e-operation-interface]"  operation="[e-operation]">  </invoke>
```

- Error end event:

```
<throw faultName="[e -name]"> </throw>
```

- *Compensation end event*

```
<compensate/>  or  <compensateScope target="[referencedActivity]"/>
```

- Terminate End Events

```
<exit> </exit>
```

4 MONEET TOOL

To facilitate the application of our methodology, we enrich MONET tool (Khlif et al., 2020) by a module that derives BPEL model from a given textual description. We called MONEET Tool, which is implemented as an EclipseTM plug-in (Eclipse, 2011) and is composed of three main modules: Parser, generator, and evaluator.

The pre-processing engine uses as input the TD of a BPEL model written in a natural language. It cleans the file using. The cleaned file is used by the business analyst to manually determine the business goals. The Business Goal (BG4) definition and its description is presented in (Khlif et al., 2020). Figure 1 shows the template corresponding to BC4.

Next, the analyst selects one or more BCs. If he selects one BC, the corresponding fragment is generated. Else, the business analyst can select all business concepts to transform. The generator engine uses the ontology and applies the transformation rules to derive the BPEL model. Figure 2 illustrates the generated BPEL model: “Supply Management Process”. First, by applying **R2.1** we add a process name "*Supply Management Process*" and a partnerLink "*Inventory Management*". Second by applying **R1,** the process is activated by the trigger "*Item and Invoice are received*". The transformation of the main scenario calls **R4.2** and **R4.5** that generate an invoke activity labelled "*Check item and invoice*".

Then, we add two variables labelled "*item*" and "*invoice*" to this activity. **R4.2** produces an invoke activity labelled "*Establish a payment*" (respectively, "*Put items in stock*"). By applying **R5.1,** we generate an orchestration logic between "*Control result*" and "*Check item and invoice*". Then, by applying **R5.2**, we add a flow activity between "*establish payment*" and "*put item in stock*". The transformation of the alternative scenario calls **R4.2, R4.5** and **R4.6** that produces an invoke activity labelled "Reconciliation order/invoice". Then, we add two variables labelled "order" and "*invoice*" to this activity and we add a reply activity "send expired product". **R4.1** produces

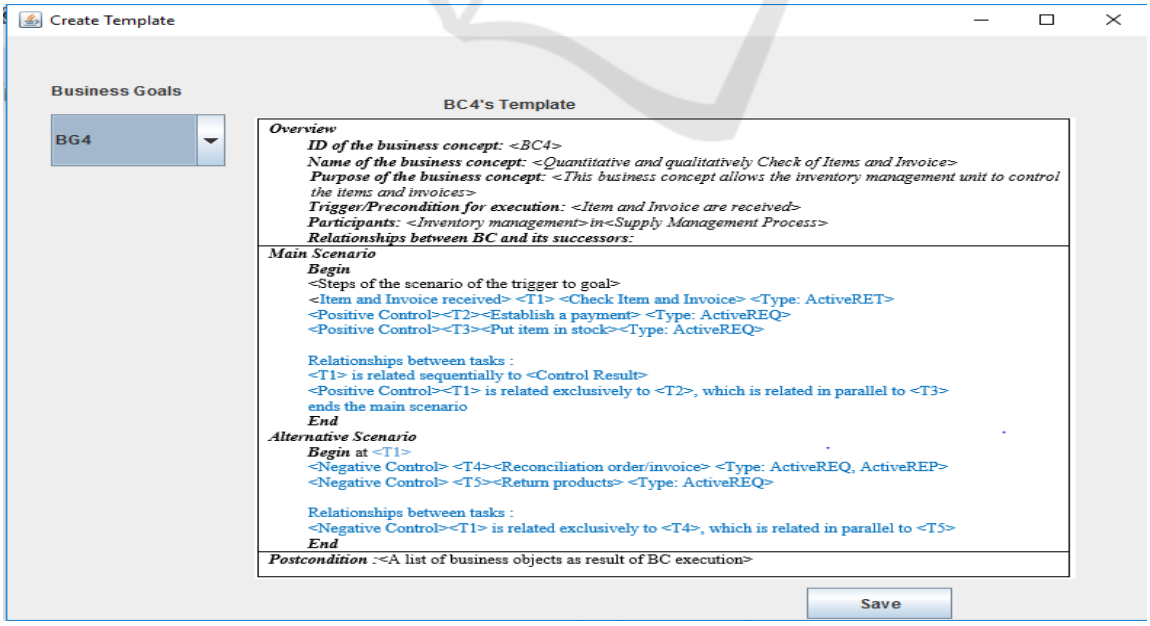

Figure 1: BC4’s enhanced template (Khlif et al., 2020).

an invoke activity labelled "return products".
Then, by applying **R5.2**, we add a flow activity between "reconciliation order/invoice" and "return products".

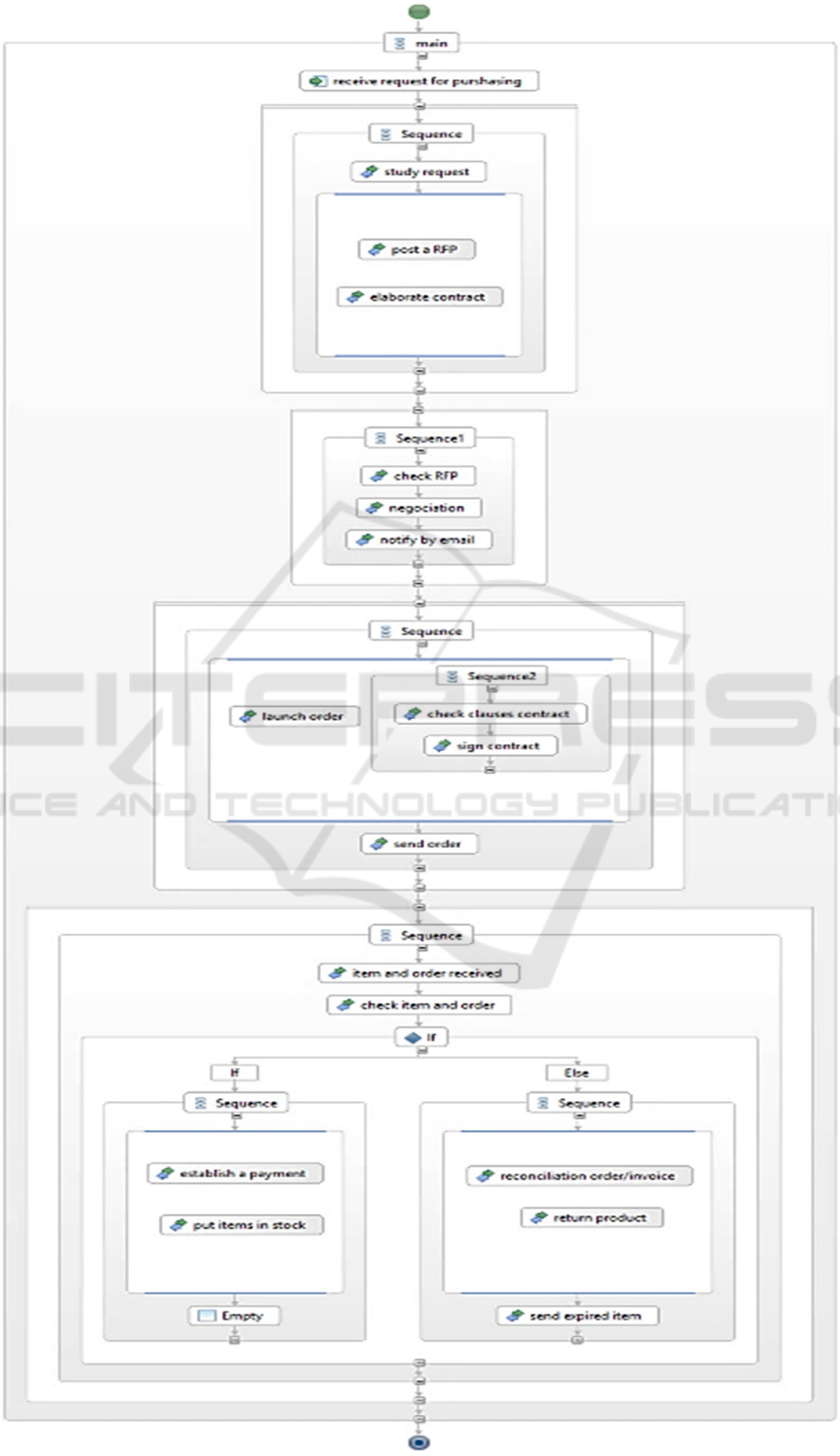

Figure 2: The generated BPEL model:"SupplyManagementProcess ".

Table 1: BPEL code corresponding to BC4.

BC4	Textual description	Rules	Code BPEL
Overview	Participants: <Inventory management> In<SupplyManagement Process>	R2.1	<process name= "SupplyManagementProces" </process> <partnerLink name= "InventoryManagemen" myRole="SupplyManagementProcesProvider" />
	<Item and Invoice are received>	R1	<receive name= item and invoice received partnerLink= "supplier" createInstance="*yes*"/>
Main scenario	<T1> <Check Item and Invoice> <Type: ActiveRET>	R4.2 R4.5	<invoke name="check item and invoice " <toPart part = "item" fromVariable = "item"/> <toPart part = "invoice" fromVariable = "invoice"/>
	<Positive Control> <T2><Establish a payment> <Type: ActiveREQ>	R4.2	< invoke name="establish a payment" portType="[T2-operation-interface]" operation="[T2-operation]"> </invoke>
	<T3><Put item in stock><Type: ActiveREQ>	R4.2	< invoke name="put item in stock" portType="[T3-operation-interface]" operation="[T3-operation]"> </invoke>
	<T1> is related sequentially to <Control Result>	R5.1	<sequence><T1><control result></sequence>
	which is related in parallel to <T3>	R5.2	<flow> [T2] [T3] </flow>
Alternative scenario	<T4><Reconciliation order/invoice> <Type:ActiveRET, ActiveREP>	R4.2 R4.5 R4.6	<invoke name="reconciliation order/invoice " portType="[T4-operation-interface]" operation="[T4-operation]"> </invoke> <toPart part = "order" fromVariable = "order"/> <toPart part = "invoice" fromVariable = "invoice"/> <reply name="send expired product"> </reply>
	<T5><Return products> <Type: ActiveREQ>	R4.1	< invoke name="return products" portType="[T5-operation-interface]" operation="[T5-operation]"> </invoke>
	<T4> is related in parallel to<T5>	R5.2	<flow> [T4] [T5] </flow>

Table 1 illustrates the BPEL code corresponding to BC4. The evaluator evaluates the BPEL model through the calculation of recall and precision rates. The high scores for both ratios (Recall=0,86 and Precision=0,95) mean that the generated BPEL model covers the whole domain precisely in accordance with the experts'(See Figure 3).

5 CONCLUSIONS

Deriving BPEL model from BPMN model can be a challenging task in the business process modelling project. The difference between the notations may degrade the quality of generated BPEL model. For that reason, this paper proposed a transformation-based methodology to generate a BPEL model from its textual description instead of BPMN models. To our best knowledge, there are no works that have investigated this research problem. Furthermore, compared to existing works dealing with BPMN-to-BPEL transformation, our methodology provided an enriched template defined in terms of structured linguistic patterns, as the starting point. Then, it defined transformation rules that derive each linguistic patterns to its corresponding BPEL elements. These transformation rules are automated and MONEET tool is implemented to derive the BPEL model an evaluate its quality using the recall and precision ratios.

REFERENCES

Aysolmaz, B., Leopold, H., Reijers, H.A., Demirörs, O., 2018. A semi-automated approach for generating natural language requirements documents based on business process models. In *Journal of Information & Software Technology*, V. 93, pp. 14-29.

Doux, G., Jouault, F., Bézivin, J., 2013. Transforming BPMN process models to BPEL process definitions with ATL. In *Proceedings of the 5th International Workshop on Graph-Based Tools*.

Eclipse Specification, 2011. Available from: http://www.eclipse.org/.

Khlif, W., Elleuch Ben Ayed, N., Chihi, F., 2020. Towards a systematic derivation of BPMN model from business process textual description. In *International Conference on Software Engineering and Knowledge Engineering* venue 2020. july 9 – 19.

Van der Aa, H., Ciccio, C.D., Leopold, H., Reijers, H.A., 2019. Extracting Declarative Process Models from Natural Language. In *31st International Proceedings of Conference on Advanced Information Systems*

Engineering. Rome, Italy, June 3-7, pp. 365-38.

Yongchareon, S., *Liu, C., Zhao,X., 2020.* UniFlexView: A unified framework for consistent construction of BPMN and BPEL process views. *In journal of concurrency and computation: Practice and experience, V 32.*

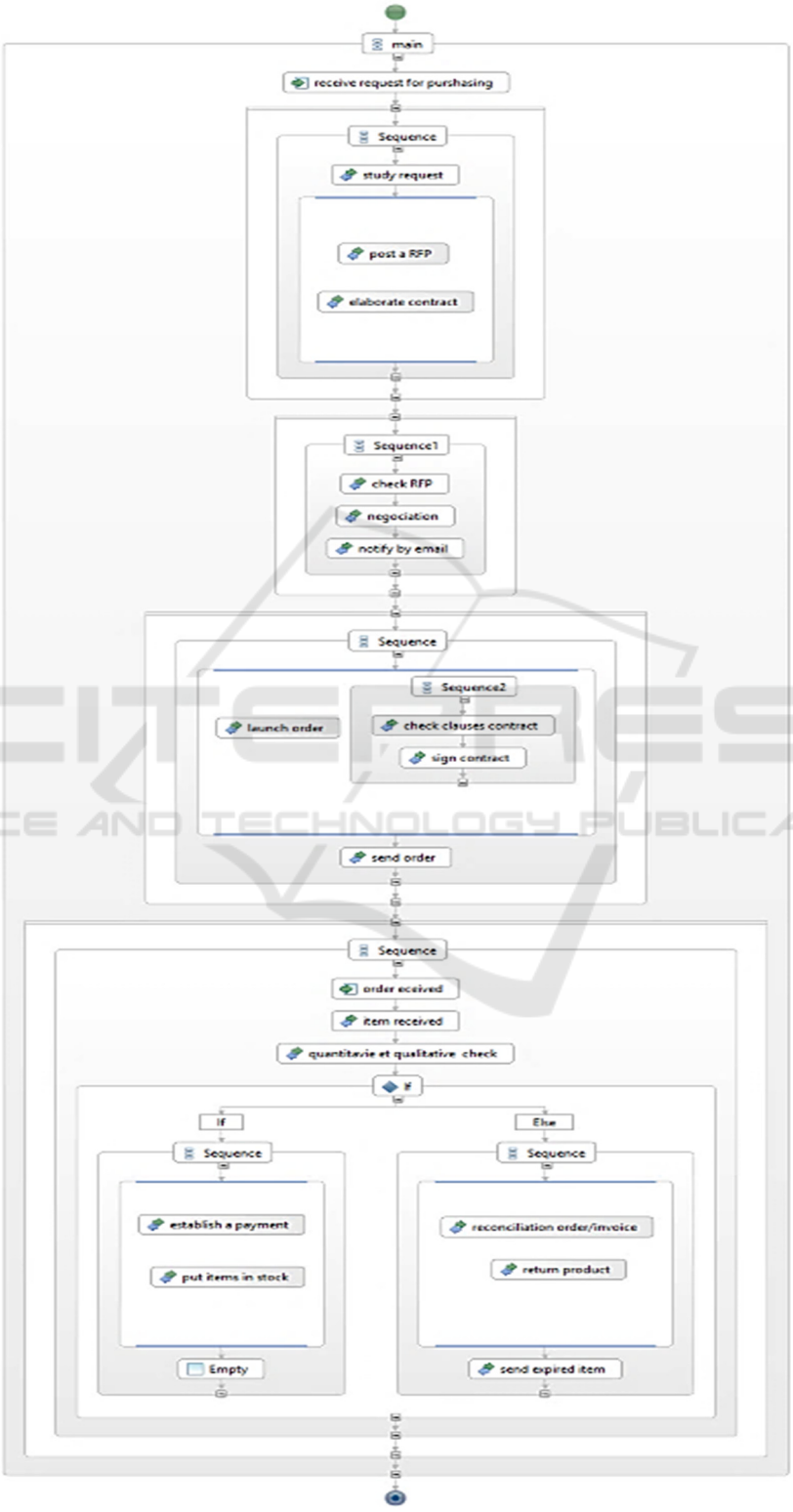

Figure 3: The elaborated BPEL model by the expert.