Towards a Comprehensive Formal Model for Business Processes

Khalil Mecheraoui, Nabil Belala(✉), and Djamel Eddine Saïdouni

MISC Laboratory, University of Constantine 2 – Abdelhamid Mehri, Constantine, Algeria
{mecheraoui,belala,saidouni}@misc-umc.org

Abstract. In order to enable enterprises to operate more effectively and efficiently, providing a high reliability of the specification of business processes is an active research subject. However, the proposed approaches use limited models. WS-BPEL (or BPEL for short) is the most well known and used orchestration language describing the Web services composition in form of business process. However, BPEL lacks a formal semantics. This paper introduces High-Level Time Open Workflow Nets with Action Duration model (HL-DToN) which is an extension of Time Petri Nets with Action Duration (DTPN) for tackling different aspects of business processes as far as possible. We use this model either to specify WS-BPEL processes or to directly specify business processes.

Keywords: Business processes · Open workflow model · Data modeling · Time constraints · Action durations · True-concurrency semantics · Formal specification

1 Introduction

To achieve its goals, an enterprise executes a collection of activities, sequentially or in a concurrently, in a specific chronological and logical order, and possibly in different locations. This procedure, called *business process*, is designed to show how and when the work is done with a beginning, an end, and identified inputs and outputs. To improve business processes, Business Process Management (BPM) is the key to align enterprises activities with the needs of their clients [1]. Therefore, there are some aspects to be considered in BPM such as time constraints, data manipulation, and structural properties of business processes. Furthermore, with the emergence of new information technologies, the enterprises want their customers and partners to be able to access quickly and directly to their functionality. To achieve this, Web services are software systems designed to support interoperability and carry out the business tasks over the web. However, as the capability of completely separate Web services is limited, one have to create new functionality by composing existing Web services in such way to interact together. To achieve that, many languages that support the descriptions of orchestration and choreography are proposed (e.g. [2–4] and [5]).

© Springer International Publishing Switzerland 2016
G. Dregvaite and R. Damasevicius (Eds.): ICIST 2016, CCIS 639, pp. 174–186, 2016.
DOI: 10.1007/978-3-319-46254-7_14

Web Services Business Process Execution Language (WS-BPEL) [5] is the most well known and used composition language which defines business processes by the orchestration of various partner interactions. However, WS-BPEL is defined informally in natural language. Thus, it lacks a formal semantics which allows a direct formal verification of well-functioning of web services. To solve this problem, one can specify business processes written in a WS-BPEL process through a formal model. Therefore, providing a formal semantics for BPEL (BPEL4WS 1.1 [6] or its revision WS-BPEL 2.0 [5]) is a research subject in several work using many approaches (e.g. [7–11]) based on various formal models. Nevertheless, many proposed approaches in the literature use limited models, either in terms of time characteristics, data modeling and manipulation, the composition and the interactions among processes; or in terms of the use of the interleaving semantics in low-level specification.

In this paper, we introduce and formalize *High-Level Time Open Workflow Nets with Action Duration* (HL-DToN). It is an extension of Time Petri Nets with Action Duration (DTPN) [12] which is proposed to deal with different aspects of business processes by adding an interface for inbound and outbound message exchange, and enabling representing and manipulating data. Furthermore, we consider both time constraints and action durations under a true-concurrency semantics to naturally express concurrent and parallel behaviors of processes. In fact, HL-DToN model can be considered as a convergence of the three models: DTPN, HLPN [13] and oWFN [14] in which we can specify WS-BPEL or directly specify business processes considering different aspects.

The remainder of this paper is organized as follows. First, we introduce in Sect. 2 some extensions of Petri nets. In Sect. 3, HL-DToN are introducted. Section 4 is devoted to show strengths of HL-DToN by providing two patterns for two activities of WS-BPEL 2.0. Finally, we give some conclusions and perspectives of our work.

2 Preliminaries

2.1 Petri Nets

Petri nets [15] are both formal and graphical modeling formalism, allowing the specification of distributed and concurrent systems by modeling states, events, conditions, synchronization, etc. Formally, $Q = (P, T, F, B)$ is a Petri net where P is a set of places, Tr is a set of transitions such that $P \cap Tr = \emptyset$. $B : P \times Tr \to \mathbb{N}$ is a backward incidence function such that $B(p_i, t_j)$ represents the arc weight from t_j to p_i and $F : P \times Tr \to \mathbb{N}$ is a forward incidence function such that $F(p_i, t_j)$ represents arc weight from p_i to t_j.

2.2 High Level Petri Nets

High-Level Petri Nets (HLPN) are defined in the international standard `ISO/IEC 15909-1` [13] developed in 2004. This standard is mainly providing a glossary of

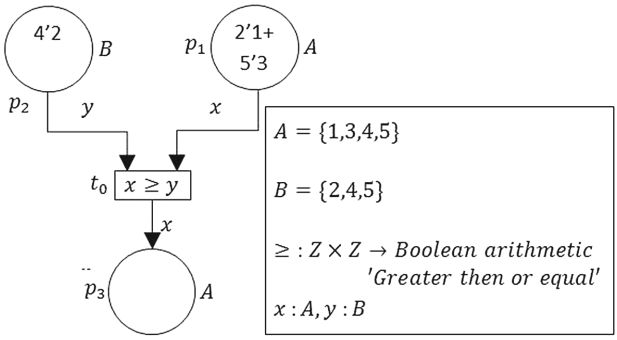

Fig. 1. An example of HLPNG

terms and abbreviations, the mathematical conventions needed for the definition of HLPN, the semantic model for HLPN, the formal definition for High-Level Petri Net Graph (HLPNG) which is the graphical form of HLPN, and the formal concepts of marking, enabling and transition rules.

A High-Level Petri Net Graph (HLPNG) includes a graph and a declaration part in which all the types, functions, constants and variables used for the graph are defined. Note that using HLPNG, we can find transition conditions (boolean expressions) associated with transitions, each place may contain a multiset of tokens, and over arcs we can find annotations (constants, variables and function images). An example of HLPNG is depicted in Fig. 1.

2.3 Workflow Nets

The workflow (workflow process) is the automation of a business process, in whole or part [16]. To model workflow processes, van der Aalst [17] defines the classical workflow net (WF-net) which is a special class of Petri nets satisfying two requirements. First, it has two special places: i and o. i is a source place, i.e., $^\circ i = \emptyset$; and o is a final place, i.e., $o^\circ = \emptyset$. A token in i corresponds to an instance that must be treated while a token in o corresponds to an instance that has been treated. Furthermore, for every node there exists a path from i to o which covers the node.

It is important to note that WF-nets ignore important aspects such as temporal characteristics of processes. Also, this model is unable to handle the interaction between workflow processes. Therefore, there are many extensions of WF-nets in the literature (see e.g. [14,18] and [19]).

Based on van der Aalst WF-nets and Timed Petri Nets of Jenner [20], Sbaï proposes a workflow net model handling the durations of tasks by associating with each transition an amount of time. This temporal extension is called *Time WF-nets* (TWF-nets) [18]. In this model, the tokens in an input place of a transition which can be fired t are removed by its firing, and the output places cannot be marked before the expiry of the duration associated to t. Thus, the

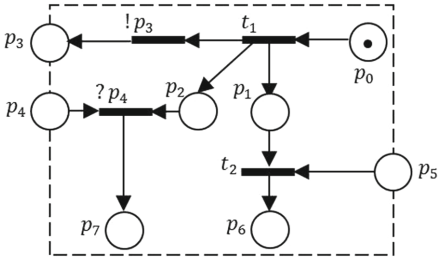

Fig. 2. Open workflow net

semantics of TWF-nets model is similar to the semantics of DTPN. We note that this model has weaknesses. For instance, it is not suitable for the composition of several processes.

In general, workflow processes are supposed to communicate with other processes. Therefore, in [14] the authors propose an open version of classical workflow nets named *open workflow nets* (oWFN). This open version is used to model workflow processes which communicate with other processes via communication channels (communication places). Graphically, a oWFN is surrounded by a dashed box, with the interface places. Also, a transition t connected to an input (resp. output) interface place p is labeled with $?p$ (resp. $!p$). Note that this open model ignores temporal characteristics of processes too. As an example, Fig. 2 shows an oWFN with three interface places (the input places p_4 and p_5 and the output place p_3), an initial place p_0, and two final places p_6 and p_7.

We know that in some real scenarios we cannot precisely determine the execution duration of an activity but it can be done in a time interval. In [19], the authors propose to adopt the Time Petri Nets of Merlin [21] to provide an extension of workflow nets called Time open Workflow Nets (ToWF-nets). In fact, this model associates two dates *min* and *max* with each transition of an open workflow net to define its interval. However, note that this model does not support data modeling and manipulation.

As we have seen, the different workflow nets presented above have weaknesses. Also, it is important to note that all these extensions are not based on a true-concurrency semantics.

2.4 Time Petri Nets with Action Duration

In [12], *Time Petri Nets with Action Duration* (DTPN), which can be considered as a generalization of Merlin's TPNs [21], T-TdPNs [22] and P-TdPN [23], are proposed. This model supports both time constraints and action durations under a true-concurrency semantics. A time constraint (temporal interval) associated to a transition t defines the firing interval in which t can be fired while an action duration indicates the duration of its corresponding action. A true-concurrency semantics was given for DTPN in terms of *Durational Action Timed Automata* (DATA) [24].

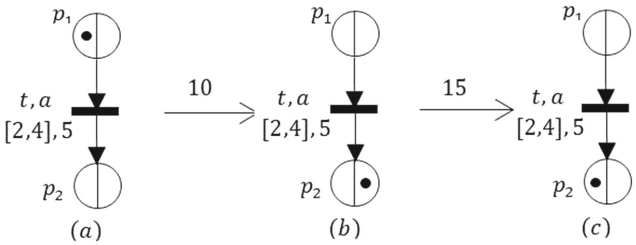

Fig. 3. Marking in DTPN

In DTPN, there are two sets of tokens, namely unavailable tokens (or *bound tokens*) and available tokens (or *free tokens*). Tokens are put on the right side of an output place of a transition t as long as the actions associated to this transition are running. We call these tokens unavailable tokens. An unavailable token becomes available at the end of the execution of the associated action. Hence, it is put at the left side. As an example, Fig. 3 describes a marked DTPN in which the token in the input place p_1 became available at the time 7.

Definition 1 (DTPN). *Let $\mathbb{T}$ be a non-negative temporal domain (like $\mathbb{Q}^+$ or $\mathbb{R}^+$) and Act be a finite set of actions, i.e. an alphabet*[1]*. A* Time Petri Net with Action Duration *(DTPN) on $\mathbb{T}$ and of support Act is a tuple $N = (P, Tr, B, F, \lambda, SI, \Gamma)$ such that*

- *$Q = (P, Tr, F, B)$ is a finite Petri net, i.e. $|P \cup Tr| \in \mathbb{N}$.;*
- *$\lambda : Tr \to Act \cup \{\tau\}$ is a labeling function of a DTPN. If $\lambda(t) \in Act$ then t is called observable or external;*
- *$SI : Tr \to \mathbb{T} \times \mathbb{T} \cup \infty$ is a function that associates to each transition a static firing interval;*
- *$\Gamma : Act \to \mathbb{T}$ is a function that associates to each action its static duration.*

In [12], a true-concurrency semantics is given to DTPN in terms of Durational Action Timed Automata.

3 High-Level Time Open Workflow Nets with Action Duration

3.1 Time Open Workflow Nets with Action Duration (DToN)

Given that workflow services are supposed to communicate with other workflow services, and that time in workflow processes is very important and crucial to determine and control activities life cycle, we employ Time Petri Nets with Action Duration model in the context of workflow processes. We provide the

[1] Assuming that $\tau \notin Act$ (τ indicates *invisible* action, also known as silent or internal action).

formal model *Time Open Workflow Nets with Action Duration* (DToN) which is extended from DTPN as an open variant adapted to workflow. In fact, it is enriched with an interface for joining each DToN pattern with other DToN patterns to allow inbound and outbound message exchange. Specifically, it is enriched with communication places for asynchronous communication in which each communication place of a DToN pattern represents a channel to send (receive) messages to (from) other DToN patterns.

Definition 2 (DToN). *A* Time Open Workflow Net with Action Duration *is a Time Petri Net with Action Duration* $PN = (P, Tr, F, B, \lambda, SI, \Gamma)$ *with*

- *Two sets:* $P_{in}, P_{out} \subseteq P$ *such that* $\forall t \in Tr : B(p,t) = 0, \forall p \in P_{in}$ *and* $\forall t \in Tr : F(p,t) = 0, \forall p \in P_{out}$*;*
- *An initial marking* M_0 *and a set of final markings* Ω *such that* $\forall M \in \Omega : \nexists t \in Tr, M[t\rangle$ *and* $\forall M \in \Omega \cup \{M_0\} : M(p) = 0, \forall p \in P_{in} \cup P_{out}$*.*

According to Definition 2, the set of places P contains inner places and also two sets of special places that are P_{in} and P_{out}. These two sets represent respectively interface input places which are input places, not output, and interface output places which are output places, not input. Also, in DToN, two conditions must be met: no transition can be fired at any final marking, and in the initial and the final markings, the interface places should not be marked. Finally, note that DToN model considers both time constraints and action durations under a true-concurrency semantics.

Graphically, a DToN is surrounded by a dashed frame, with the interface places in which all interface places have a second label depicted outside the place. This latter is used for the composition or to show the purpose of the place. We note that initial and final places are always labeled respectively by *Initial* and *Final*. An example is shown in Fig. 4 which represents a DToN pattern named X and contains: two inner places p_1 and p_2, and four interface places on the dashed frame. Interface places are the initial place *Initial*, the final place *Final*, the interface output place α, and the interface input place β.

In order to compose open workflow nets, some approaches are provided. In [14] Massuthe, Reisig, and Schmidt propose the composition of two oWFN by

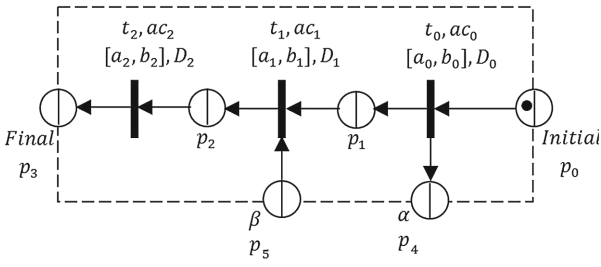

Fig. 4. DToN pattern X

sharing input and output elements in common, in which the composition of two oWFN A and B is an oWFN. Each input interface place of A must be joined with an output interface place of B and turns into an inner place if these two places have the same identifier, and vice-versa. However, it is important to note that this approach only supports the composition of two oWFN, no more. In [25], the authors propose the composition of Timed Open Workflow Nets (ToN) by adding a mediation net to deal with message mismatches in which the composition of two ToN A and B via a mediation net MN is called Mediation-Aided Composition of ToN (MToN).

In this paper, we propose a simple approach to compose DToN patterns. We model each process in a composition by a DToN pattern. All interface places have a second label (label *2*) depicted outside the place in which the interface places of these several patterns, only which they have the same label –label *2*–, are joined without turning them into inner places. Except initial and final places because an initial place can generally be joined either with another initial place or with a final place (e.g. to sequentially compose patterns).

3.2 High-Level Time Open Workflow Nets with Action Duration

To model data flow and data manipulation based on the international standard `ISO/IEC 15909-1` [13], we extend Time Open Workflow Nets with Action Duration to *High-Level Time Open Workflow Nets with Action Duration* (HL-DToN). This high-level model allows us to completely integrate data while such data can also be ignored by moving to DToN.

Definition 3. *A* High-Level Time Open Workflow Nets with Action Duration *is a tuple* $HN = (P, Tr, F, B, \lambda, SI, \Gamma, D, Type, M_0)$ *where*

- $N = (P, Tr, F, B, \lambda, SI, \Gamma)$ *is a DToN with* $F, B : TR \rightarrow \Phi PL$[2] *are the Pre and Post mappings with* $TR = \{(t, m) | t \in Tr, m \in Type(t)\}$ *and* $PL = \{(p, g) | p \in P, g \in Type(p)\}$*;*
- D *is a non-empty finite set of domains. Each element of* D *is called* type*;*
- $Type : P \cup Tr \rightarrow D$ *is a function used to assign types to places and to determine transition mode;*
- $M_0 \in \Phi PL$ *is a multiset called the* initial marking *of the net.*

Transition mode is defined such as in [13]. It is an assignment of values to the transition variables (variables that occur in the transition condition and the annotations of the arc involving the transition) that satisfies the transition condition. F function determines token demands (multisets of free tokens) on places for each transition mode, and B function determines output tokens (multisets of bound tokens) for places for each transition mode.

Note that Definition 3 covers important aspects of data modeling and manipulation. Graphically, places may contain a multiset with two kinds of tokens (available or unavailable tokens). Over arcs we can find constants, variables and

[2] ΦPL is the set of multisets over PL.

function images. Transitions can be associated with boolean expressions (transition conditions). Thus, a transition is annotated in this model by its name, a name of its associated action, a time interval (firing interval), a duration of execution, and a boolean expression (the transition condition).

Marking of HL-DToN. A marking M is a multiset of available and unavailable tokens of correct type for all places. Formally, $M \in \Phi PL$.

Enabling a Transition. A transition $t \in Tr$ is *enabled* in a transition mode $\widehat{t}$ and a marking M iff

$$\forall p \in P, F_{\widehat{t}}(p,t) \leq M(p)$$

Enabling free tokens are the free tokens resulting from evaluating input arc's expression in and with respect to a specific transition mode.

Firing Rule. Assuming that the firing interval of a transition t is $[min, max]$ and t is enabled in a transition mode at the time ϑ, t will be fired in the time interval $[\vartheta + min, \vartheta + max]$. When t is fired, its input arc enabling free tokens with respect to that mode are dropped from the left side of the input place (the available input place's marking) and the multiset of tokens of the evaluated output arc expression is added to the left side of the output place (the unavailable input place's marking).

Formally, The firing of $t \in Tr$ in transition mode $\widehat{t}$ and marking M, results a new marking M' defined by:

$$\forall p \in P, M'(p) = M(p) - F_{\widehat{t}}(p,t) + B_{\widehat{t}}(p,t)$$

Now, we can make a comparison among Sect. 2 models and HL-DToN. Table 1 shows that HL-DToN overcomes the others. In each column, we use '+' for *yes* and '-' for *no*. The meaning of the column headers are as follows: A: supports the control of data flow using arc annotations and transition conditions. B: able to use complex structured data as tokens. C: adapted to workflow. D: suitable for the composition of several processes to handle the interaction between them. E: able to handle the duration of tasks. F: supports both time constraints and action durations under a true-concurrency semantics.

Table 1. Comparison of different formal models

	A	B	C	D	E	F		A	B	C	D	E	F
PN	−	−	−	−	−	−	oWFN	−	−	+	+	−	−
HLPN	+	+	−	−	−	−	ToWFN-net	−	−	+	+	+	−
WF-net	−	−	+	−	−	−	DTPN	−	−	−	−	+	+
TWF-net	−	−	+	−	+	−	HL-DToN	+	+	+	+	+	+

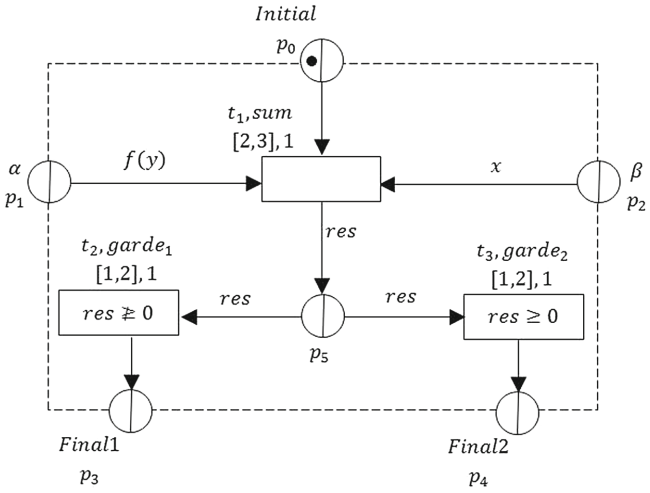

Fig. 5. Pattern for checking the result of $x + f(y)$

As example, we provide the HL-DToN pattern depicted in Fig. 5. This pattern is used to check if the result of $x+f(y)$ is negative or not. The control flow starts with an available token in the initial place p_0 and it ends either with an available token in the final place p_3 or in the final place p_4.

First, transition t_1 allows us to calculate the sum of x plus $f(y)$. The result of this sum is saved in the variable res. In the second step, this information is analyzed using both t_2 and t_3. Either the result is greater than or equal zero ($res \geq 0$) and therefore t_3 will be fired after a period between 1 and 2 units of time, or the result is negative and therefore t_2 will be fired after a period between 1 and 2 units of time. In the first case, p_4 will be marked by an available token after 1 unit of time. In the second case, p_3 will be marked by an available token after 1 unit of time.

4 An HL-DToN Semantics for WS-BPEL

WS-BPEL is the most used language to specify the behavior of business processes based on Web services. In our approach, which is similar to the one in [9], given that the construction of the WS-BPEL process is performed by the composition of its constructs, it is clear that each construct should be modeled, at least, by an HL-DToN pattern. This latter has an interface used to join this pattern with other patterns. The composition of all patterns forms an HL-DToN representing a formal semantics for the WS-BPEL process.

Thus, we must provide at least one pattern of each basic and structured activity, links, and the four handlers. In this paper, we provide two HL-DToN patterns. One for ⟨`wait`⟩ activity to specify a delay for a certain period of time,

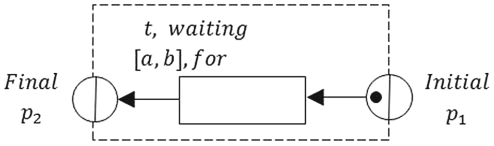

Fig. 6. A pattern for a ⟨`wait`⟩ activity

and the other for the activity ⟨`forEach`⟩ in the case of *parallel* = *yes*. Other patterns of the other activities can be found in [26].

⟨`wait`⟩ **Activity.** A ⟨`wait`⟩ activity is used to specify a delay for a certain period of time or until a certain deadline is reached [5].

The pattern of a ⟨`wait`⟩ activity to specify a delay of a certain period of time is given in Fig. 6. In this pattern, we represent the delay for a certain period by the duration of the action associated to transition t. It is clear that this transformation is very easy due to the existence of action durations associated to transitions in HL-DToN.

⟨`forEach`⟩ **Activity.** The ⟨`forEach`⟩ activity iterates its child scope activity exactly $N + 1$ times where N equals `finalCounterValue` minus `startCounterValue`. If *parallel=yes* then this is a parallel ⟨`forEach`⟩ where the $N + 1$ instances of the enclosed ⟨`scope`⟩ activity should occur in parallel [5].

The pattern of the activity ⟨`forEach`⟩ in the case of *parallel=yes* is depicted in Fig. 7. To simplify the pattern and such as they are defined in [9], we add the object variable and we use the variable X (in capital letter) representing a set of variables. Also, we use read arcs witch are unfolded to loops. Furthermore, it is important to note that this pattern represents the case in which the optional element ⟨`completionCondition`⟩[3] does not exist.

As depicted in Fig. 7, if the initial place is marked by an available token, the stored data in the object *Obj* will be read using two read arcs. Its values are saved in X and analyzed using t_1 and t_4. If an error[4] occurs ($EB =$ `true`), the error information will be saved in *error* and the interface place $failed$ will be marked. This latter must be joined with the associated interface place on a pattern of a fault handler (labeled by $failed$). If there is not an error ($EB =$ `false`) and then the two places p_4 et p_5 become marked by available tokens, the two transitions t_2 and t_3 will be used respectively to evaluate the expressions in `finalCounterValue` and `startCounterValue`. Their two results will be respectively saved in fcv and scv. Thereafter, the transition conditions $fcv \geq scv$ (t_5) and $fcv \ngeq scv$ (t_6) must be evaluated[5]. If $fcv \geq scv$ (t_5), the two variables fcv and scv are used to calculate the arc weight from t_5 to the initial place (p_8) of

[3] It is used to force early termination of some of the children (in the parallel case) [5].

[4] For instance, because of a selection failure.

[5] In WS-BPEL, "if `startCounterValue` is greater than `finalCounterValue`, then the child ⟨`scope`⟩ activity must not be performed and the ⟨`forEach`⟩ activity is complete." [5].

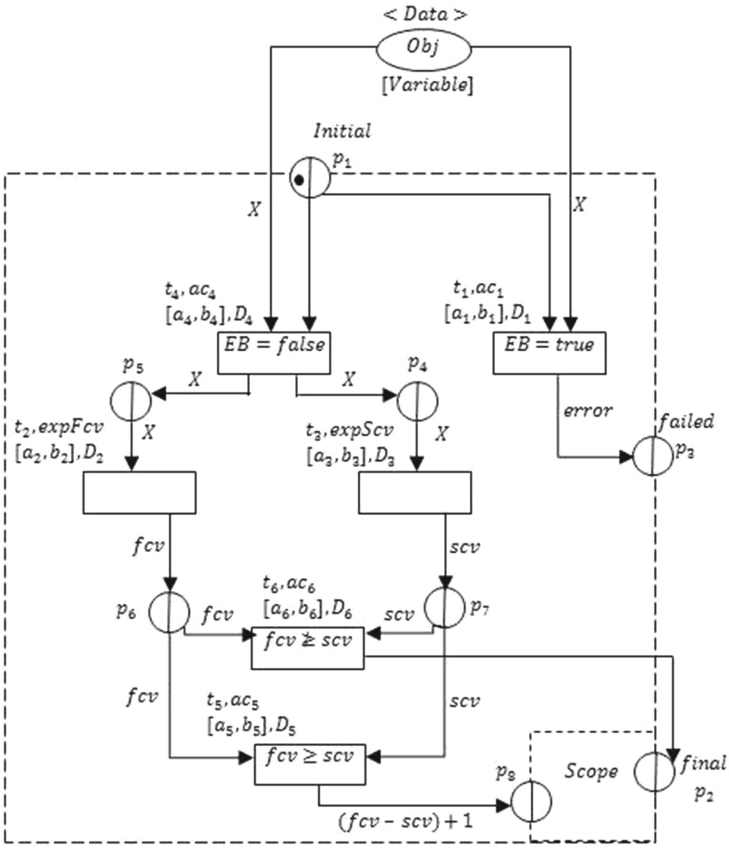

Fig. 7. Pattern for ⟨`forEach`⟩ activity in case of *parallel=yes*

the enclosed ⟨`scope`⟩. This allows us to iterate the child scope $N + 1$ times in parallel where N equals $(fcv - scv) + 1$ (this is thanks to DTPN semantics).

Finally, it is important to note that this approach differentiates from the already existing approaches by covering different characteristics of business processes. For instance, it covers the aspect of action duration and time constraints, unlike the other approaches.

5 Conclusion

In this paper, we introduced the formal model of High-Level Time Open Workflow Nets with Action Duration (HL-DToN) by which we do not only specify WS-BPEL processes but directly specify business processes as well. This model is able to enable inbound and outbound message exchange, to consider time constraints and action durations under a true-concurrency semantics, and also to cover important aspects of data modeling and manipulation.

Furthermore, a transformation of two WS-BPEL activities, featuring the advantages of the model, is presented. Actually, we provided a pattern of a ⟨`wait`⟩ activity that specifies a delay for a certain period of time, and a pattern for the activity ⟨`forEach`⟩ in the case of *parallel=yes*.

In the near future, we plan to express action durations using random variables as those present in some stochastic extensions of Petri nets. Also, we project to continue the transformation of WS-BPEL constructs using HL-DToN. Thereafter, we aim to extend the use of HL-DToN to verify qualitative and quantitative business processes properties.

References

1. Rogge-Solti, A., Kasneci, G.: Temporal anomaly detection in business processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 234–249. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10172-9_15
2. Thatte, S.: XLANG: Web services for business process design. Microsoft Corporation (2001)
3. Leymann, F., et al.: Web services flow language (WSFL 1.0) (2001)
4. Kavantzas, N., et al.: Web services choreography description language version 1.0. W3C Candidate Recommendation **9**, 290–313 (2005)
5. OASIS Standard. WSBPEL Ver. 2.0 (2007). http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
6. Andrews, T., et al.: Business Process Execution Language for Web services (2003)
7. Thivolle, D.: Langages modernes pour la modélisation et la vérification des systèmes asynchrones. Ph.D. thesis, Université de Grenoble and Université Polytechnique de Bucarest (2011)
8. Cavalli, A., et al.: Definition of the mapping from BPEL to WS-TEFSM. In: Livrable WEBMOV-FC-D2.3/T2.4 (2008)
9. Stahl, C.: A Petri net semantics for BPEL (2005)
10. Lohmann, N.: A feature-complete petri net semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008). doi:10.1007/978-3-540-79230-7_6
11. Abouzaid, F., Mullins, J.: A calculus for generation, verification and refinement of BPEL specifications. Electron. Notes Theor. Comput. Sci. **200**(3), 43–65 (2007)
12. Belala, N., et al.: Time petri nets with action duration: a true concurrency real-time model. Int. J. Embed. Real-Time Commun. Syst. (IJERTCS) **4**(2), 62–83 (2013)
13. ISO/JTC1/SC7/ WG19: International Standard ISO/IEC 15909: Software and Systems Engineering - High-level Petri Nets, Part 1: Concepts, Definitions and Graphical Notation (2004)
14. Reisig, W., Massuthe, P., Schmidt, K.: An Operating Guideline Approach to the SOA (2005)
15. Petri, C.: Kommunikation mit Automaten. Ph.D. thesis. Schriften des Instituts für instrumentelle Mathematik, University of Bonn, Germany (1962)
16. WFMC: Workflow management coalition terminology and glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels (1999)
17. van der Aalst, W.M.P.: Verification of workflow nets. In: Azéma, P., Balbo, G. (eds.) Application and Theory of Petri Nets 1997. LNCS, vol. 1248, pp. 407–426. Springer, Heidelberg (1997). doi:10.1007/3-540-63139-9_48

18. Sbaï, Z.: Contribution á la Modélisation et á la Vérification de Processus Workflow. Ph.D. thesis, Ecole Doctorale Informatique, Télécommunications et Electronique de paris. France (2010)
19. Sbaï, Z., Barkaoui, K., Boucheneb, H.: Compatibility analysis of time open workflow nets. In: PNSE 2014 - Petri Nets and Software Engineering (2014)
20. Jenner, L.: Further studies on timed testing of concurrent systems. Technical Report 4, Institute für Mathematik, Universität Augsburg (1998)
21. Merlin, P.M.: A study of the recoverability of computing systems. Ph.D. thesis. University of California, Irvine, USA (1974)
22. Ramchandani, C.: Analysis of asynchronous concurrent systems by timed Petri nets. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA (1974)
23. Sifakis, J.: Use of petri nets for performance evaluation. Model. Perform. Eval. Comput. Syst. **4**, 75–93 (1977)
24. Saïdouni, D.E., Belala, N.: Actions duration in timed models. In: The International Arab Conference on Information Technology (2006)
25. Du, Y., et al.: Timed compatibility analysis of Web service composition: A modular approach based on Petri nets. Autom. Sci. Eng. **11**(2), 594–606 (2014)
26. Mecheraoui, K.: Spécification formelle des processus métiers par l'utilisation des réseaux de Petri temporellement temporisés. MSc Thesis, Ecole Nationale Supérieure d'Informatique (2015)