Journal of King Saud University – Computer and Information Sciences xxx (xxxx) xxx

Contents lists available at ScienceDirect

Journal of King Saud University – Computer and Information Sciences

journal homepage: www.sciencedirect.com

Towards modelling and analyzing timed workflow systems with complex synchronizations

Abdia Hamdani [a], Abdelkrim Abdelli [b,*]

[a] University of Tiaret, Algeria
[b] LSI laboratory, USTHB University, Algiers, Algeria

ARTICLE INFO

Article history:
Received 15 February 2019
Revised 28 May 2019
Accepted 14 August 2019
Available online xxxx

Keywords:
Workflow
Pattern
Synchronization
Rendezvous
Petri Nets
Real time systems

ABSTRACT

Nowadays, workflow systems are becoming very complex, involving time constraints, priorities on tasks as well as elaborated synchronization schemes. As a result, the specification and the verification of such systems are demanding much theory to prove their correctness and determine their qualitative and quantitative properties. In this paper, we mainly deal with complex synchronization on timed workflow systems. To address this issue, we redefine the concept of rendezvous as a mechanism to synchronize time constrained concurrent tasks of different privileges (Master, Slave). We discuss the different strategies that a rendezvous can follow, and hence we determine all the synchronizations patterns that could be considered in specifying its behaviour. From there, we deduce 36 generic synchronization rules that can be associated with the concept of rendezvous. Based on this theory, we extend the Time Petri Nets formalism to the concept of rendezvous to introduce the RTPN model (Time Petri Nets with rendezvous). Then, a subclass of RTPNs called Time Workflow-nets with Rendezvous (RTWF-nets), are defined for the modelling and the analysis of timed workflow systems. Finally, a case study is addressed to show how our framework can deal efficiently and elegantly with complex synchronization requirements.

1. Introduction

Software engineering is a an important research topic in computer science that dates back to the early 70's and which is continuing to grow fast. Over time, the software has become more and more complex while having to respond continually to user's requirements to improve its cost, its reliability its efficiency, and its performances. The need in terms of defining formal methodologies to verify the correctness of the software is becoming a challenging issue, especially with the advent of real time systems. Indeed, it needed to be recognised that to better manage the dysfunctions and avoid unexpected process errors, we need as designers to anticipate them by simulating and model-checking the behaviour of the software before its implementation.

The communication, the synchronization and the delays are important aspects to deal with when several occurrences of software processes are planned to execute in parallel while sharing objects and resources. We need to guarantee a reliable and continuous coordination to maintain the information exchange in the due time. To that end, different models have been introduced to specify and study aspects related to process execution, as for instance *workflow*. The latter allows to describe clearly how and when elementary tasks of a process must be accomplished by specifying the *control flow* (Khoshafian et al., 1998).

The research on workflow systems has started in the late of the 70's, and many approaches have been defined so far. The description of the *control flow* may include different mechanisms (e.g., sequence, choice, parallelism and synchronization), usually called workflow *patterns* (Van der Aalst and ter Hofstede, 2005). For instance, synchronization in a workflow can be seen as a meeting point during the process execution where a set of tasks has to wait for others according to a given scheme (e.g, the AND-join synchronizer pattern). Nowadays, one of the challenge in workflow systems is to cope with situations where a variable number of tasks should be executed under different synchronization and time constraints patterns. Indeed, adapting, replanning, and synchronizing workflows in response to an unexpected behaviour, delays, or

* Corresponding author at: Bp 32 El Alia Babezzouar, Algiers, Algeria.
E-mail address: abdelli@lsi-usthb.dz (A. Abdelli).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.jksuci.2019.08.007
1319-1578/

technical conditions are necessary to guarantee the reliability of the systems.

Furthermore, such requirements are becoming critical aspects in many domains as for example healthcare workflows (Combi and Posenato, 2009). For example, in the transplantation surgery activity, we require the concurrent presence of the organ to be implanted, the blood for the patient and the patient, that must all arrive within the same hour at the hospital to avoid any functional degradation of the implant. Moreover, in warehouse management the workflow requires some sort of synchronization to ensure that all the products are delivered to the customer in a due time to avoid lack of on-site storage. For example, two tasks A and B that correspond to the shipments to be made by two suppliers. The final delivery must begin after all products are available and none of the products must wait more than 5 time units at the warehouse.

As an additional constraint, tasks may not have the same priorities towards synchronization, as for instance, in the lip-synchronization problem. The audio (master with priority), and the video (slave) are connected through a master synchronization. On the other hand, the audio may synchronize also with a telepointer (slave) through another synchronization (Owezarski and Boyer, 2010). The following delays, for example, must be met: the video may be ahead of audio but no more than 80 ms, while it should not be advancing earlier than 80 ms. The telepointer must not be advancing to audio earlier than 740 ms.

In our previous works, we introduced a new Time Petri Net extension called RTPN (Time Petri Net with Rendezvous) (Hamdani and Abdelli, 2017). This model enriches the TPN semantics with the paradigm of rendezvous which provides a flexible mechanism to design a synchronization process between different time constrained concurrent tasks of different privileges. Hence, the main synchronization rules that can be induced from the concept of rendezvous have been investigated and discussed.

In Hamdani and Abdelli (2018), we put forward a methodology to specify and analyze workflow systems using the RTPN formalism. For this effect, we explored the main synchronization strategies that can be associated with the concept of a rendezvous. The Time Workflow-nets with Rendezvous (RTWF-nets), were introduced as a subclass of RTPNs, to model complex workflow systems.

In this paper, we mainly address the problem of specifying timed workflow systems with complex synchronization requirements. To this end, the paradigm of rendezvous is extended providing to the designer more expressiveness and more flexibility. Hence, we are able to generate more synchronization strategies and induce thereof any type of synchronization rule.

According to the extended semantics of a rendezvous, we define the global strategies that govern the behaviour of the latter, then we derive an exhaustive list of all the possible synchronization patterns and sub-patterns that can be obtained thereof. Hence, we define altogether 27 master and 9 non master generic synchronization rules that determine all the possible pattern combinations that could be designed from the concept of rendezvous. Each generic rule can be in turn generates sub rules to specify more precise and peculiar synchronizations cases. The formal syntax and semantics of the RTPN already introduced in Hamdani and Abdelli (2017) are revisited to cope with the additional features introduced in the definition of the concept rendezvous. Finally, a case study is considered to advocate the purpose of our model in dealing with complex synchronizations in timed workflow systems.

The remainder of this paper is organized as follows: In Section 2, we discuss the related works. In Section 3, we present formally the concept of rendezvous and infer the different synchronization patterns that could be derived thereof. An exhaustive list of generic synchronization rules is then presented and discussed. Section 4 lays down the formal syntax and semantics of the RTPN and RTWFN models before presenting the modelling approach. In Section 5, we give a case-study example. Finally, conclusions and comments on future work are given.

2. Related works

In this section, we review the most important works of the literature that address the formal specification and the verification of workflow systems.

a) *Petri Nets based Models:*

Thanks to its well defined syntax and formal semantics, its local state-based system description, and the abundant analysis techniques, the use of Petri net (PN) model as a mathematical foundation for the formal analysis of workflow systems has been considered in many works. Basic PNs have been first extended in van der Aalst (1998)to define the *WorkFlow net* (WF-net) to specify the processing of a workflow. In Van der Aalst and ter Hofstede (2005) WF-net is considered for the modelling, the validation and the verification of business procedures.

The WFnet has to be extended afterwards to deal with new aspects: (data, time, etc. . .). A various extensions were, thus, put forward in the literature: *(i)* the extension with time (Li et al., 2003); *(ii)* the extension with color to model data (Russell and ter Hofstede, 2009); *(iii)* the extension with hierarchy to structure large models (Bæk et al., 2008); *(iv)* and finally a general model that combines some of the previous features Barreto and Julia, 2017.

Many other PN extensions have been proposed in the literature:

For example, in Wang and Zeng (2008), the authors introduce the R/NT-WF Net to model workflows constrained by resources and a non-determined time. A procedure is given to compute the earliest and the latest times to start each activity. In Barkaoui et al. (2008), a new formalism called the Time Recursive ECATNets (T-RECATNets) is proposed for modelling and analysing time constrained reconfigurable workflows. In Bertolini et al. (2012) component-based timed-arc Petri Nets (CTAPN) are defined to model collaborative healthcare workflows. The authors in Cicirelli et al. (2013) consider the TSPN model for modelling and enactment of complex workflows. The authors in Thabet Kotb and Badreddin (2005), introduce the WFCS-nets (workflow with critical sections nets), to deal with synchronization and time constraints among activities while considering critical sections. In van der Aalst and Weske (2013) and Boukhedouma et al. (2017), patterns are adapted to cope with inter-organisational workflows (IOWF). The authors in del Foyo and Silva (2008), apply Time PNs to model and analyse the temporal behaviour of workflow systems using TINA as a tool to support the verifications of the activities deadlines. The time information located in the transitions of the automaton derived from the (TPN), allows checking for temporal quantitative properties using TCTL logics. In Atluri and Huang (2000), Color Timed Petri Net (CTPN) is considered to represent the WAM: (workflow authorization model) to design the separation of duties constraints on multiple workflow instances. In Yuyue and Jiang (2003), the Logical Time Workflow nets (LTWN) and Inter-organizational LTWNs (ILTWN) based on logical time Petri net (LTPN) are presented for specifying and verifying real-time cooperative systems. Moreover, the logical correctness as well as the bounded response time properties were investigated. In Boucheneb and Barkaoui (2014), an approach

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

describing how the qualitative and quantitative analysis of the framework can be performed by using TCTL model checking is presented. In Hamdani and Abdelli (2018), the authors introduce the concept of rendezvous to specify complex synchronizations in timed workflow systems. They give orientations on the strategies to adopt to obtain a variety of synchronization patterns.

b) *Graph based Models:*
Graphs have been also considered as a formal tool to specify workflow systems. In Marjanovic and Orlowska (1999), tasks are the vertices of the graph stamped with their durations. The edges represent the tasks execution. In Bettini et al. (2002), the vertices are stamped with the starting and the ending instants of the related task whereas the edge is given the maximum allowed duration of the task. In Combi and Posenato (2009), the authors introduce the directed graph wherein tasks are represented as in Bettini et al. (2002). The controllability of a workflow path is verified for any possible combination of durations and delays. In Eder et al. (2013), the Timed Workflow Graph (*TWG*) is defined providing temporal constraints between activities as upper and lower bounds constraints. No delay between activities is defined.

c) *Others Models:*
Other formalisms have been also considered to specify workflow systems, as the use of *computational tree logic* (*CTL*) in Rusinkiewicz et al. (1991), the *event algebra* in Gunter et al. (2009), *multi-agent theory* in Alfonso-Cendn et al. (2016), *UML activity diagrams* in Karhof et al. (2016), and *State Charts* Zhang et al. (2012).

d) *Discussion:*
The majority of the previous works do not address synchronizations with time constraints in the modelling of control flow. The synchronization requirements handled so far in the literature are very basic and do not cover all the possible patterns. There is a need to consider more schemes and to provide a more expressive formal model to handle these requirements. For this purpose, we propose in this paper to extend the work defined in Hamdani and Abdelli (2018). We revisit the concept of rendezvous by extending its semantics to cover a large range of synchronizations. We provide an extensive list of synchronizations patterns from which we deduce 36 synchronizations rules. The concept of rendezvous is then associated with TPN model to provide a powerful tool to design and analyse timed workflow systems with complex synchronization requirements.

3. Definition of the proposed synchronizations model

Synchronization and time constraints are becoming important features to deal with in Workflow systems. Indeed, modelling and analyzing such requirements is a very complex task as it takes to introduce new tools to tackle the additional challenges. We present in this section the essence of our approach by describing the different paradigms used in our modelling framework. We recall the concept of rendezvous already introduced in Hamdani and Abdelli (2017) and Hamdani and Abdelli (2018). The semantics of the latter is however extended to permit more expressiveness and more flexibility.

3.1. Tasks, rendezvous, locality and delay parameters

The concept of a rendezvous relays on the concept of tasks. In a workflow model, we assume the existence of a set of elementary work units called tasks, that collectively achieve a given process. An elementary task is an atomic activity that cannot be divided into sub-processes. We assume in our model that tasks can have different privileges (*Master* or *Slave*) and are associated with a time interval defining their earliest starting and the latest ending times.

The way tasks interact and sychronize is described by the concept of rendezvous. Informally, a *Rendezvous* is seen as a time point during the workflow process where a set of tasks has to interact with each other to synchronize their executions according to given requirements. More formally, a rendezvous, noted R, is the tuple $(T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$, such that:

– T_m denotes the non empty set of *master tasks* involved in the synchronization pattern; T_s is the set of *slave tasks*, such that $T_s \cap T_m = \varnothing$. The set T_s can be empty, thus denoting that all the tasks involved in the pattern have the same privileges. In the sequel, we note $T_r = T_s \cup T_m$. We assume that each task t is characterized by its own time constraints, given in the form of an interval $[x(t), y(t)]$. $x(t)$ (resp. $y(t)$) denotes the earliest starting (resp. the latest ending) time of the task t. $y(t) - x(t)$ will thus denote the maximal duration of the task t. The difference between a master task and a slave task is that either the start or the end of the rendezvous must be decided exclusively by the execution of the master tasks. However, slave tasks do not necessarily impact the synchronization unless they are given this privilege, as we will see in the sequel.

– (Loc^-, Loc^+) are the time localities that determine the time landmarks of the synchronization. Loc^- is called the *start time locality* and Loc^+ is the *end time locality*. Loc^- introduces the rule to compute the preferable time to start the rendezvous, whereas Loc^+ defines the rule to compute the preferable time to achieve it. The value of the locality Loc^- resp Loc^+ is encoded by the couple (Set^-, Op^-) resp. (Set^+, Op^+). Hence we introduce the following expressions to compute the date of both localities: $Loc^- = \underset{\forall t \in Set^-}{Op^-} \{x(t)\}$, $Loc^+ = \underset{\forall t \in Set^+}{Op^+} \{y(t)\}$ where Set^+, $Set^- \in \{T_r, T_m\}$ and $Op^-, Op^+ \in \{MIN, MAX\}$.

Comparatively to the semantics defined in Hamdani and Abdelli (2017) and Hamdani and Abdelli (2018), we extend here the latter to permit more localities values in the expression of a rendzvous. Concretely, each locality can take 4 different values rather than two. Therefore, we introduce more flexibility in the design of complex synchronization and extend the range of synchronization patterns that can be generated by our model. According to the type of synchronization pattern that we want to specify, the start time locality Loc^- can refer to one of the following values:

- (T_r, MAX): That means that the start of the synchronization is determined when the last task starts its execution. This locality value requires that all the tasks should start conjointly in order to hold the synchronization.
- (T_m, MAX): That means that the start of the synchronization is determined by the last master task that started its execution. These patterns require that all the master tasks should synchronize their start (see Fig. 1).
- (T_m, MIN): This means that the start of the synchronization is given by the first master task to execute. Such a locality value requires that at least one master task must be available to execute in order to start the synchronization.
- (T_r, MIN): This means that the start of the synchronization is decided by the first task to execute. This synchronization requires that at least one task must be available to execute in order to start the synchronization.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

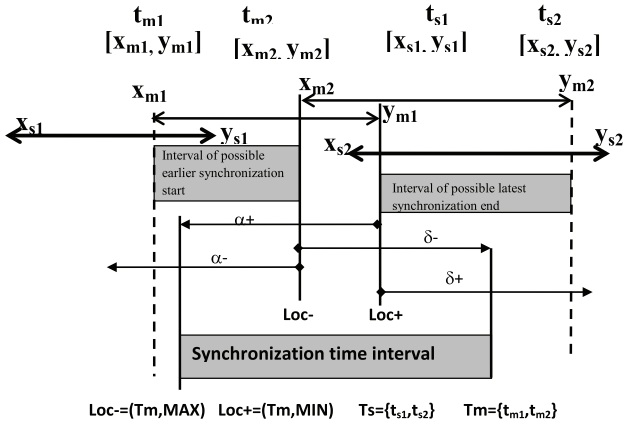

Fig. 1. Localities, advance and delay in a rendezvous.

Likewise, Loc^+ is associated with four locality values:

- (T_r, MIN): This means that the end of the synchronization is determined by the first task to achieve its execution. This requires that all the tasks have already started their execution in order to achieve conjointly the synchronization.
- (T_m, MIN): This means that the end of the synchronization occurs when the first master task ends its execution. This scheme demands that all the master tasks have already started their execution in order to achieve conjointly the rendezvous (see Fig. 1).
- (T_m, MAX): This denotes that the end of the synchronization is given by the last master task to end its execution. Such a locality value requires that at least one master task must be running in order to end the synchronization.
- (T_r, MAX): This denotes that the synchronization ends when the last task achieves its execution. Such a synchronization requires that at least one task must be running in order to end the rendezvous.

To be coherent with the master tasks semantics, we require in our model that at least one of the localities refers to the set of master tasks, namely either $Set^- = T_m$or $Set^+ = T_m$. This enforces that either the start or the end of the synchronization is exclusively decided by the execution of master tasks.

– The parameters $(\alpha^-, \alpha^+, \delta^-, \delta^+)$ are positive rational numbers from $\mathbb{Q}^+ \cup \{+\infty\}$ that specify advance or delay for the localities. Concretely, in order to relax the time constraints determined by the locality values defined previously, we use these additional parameters to express a time drift relatively to the synchronizations points defined by Loc^- and Loc^+. α^- (respectively α^+) specifies the acceptable advance relatively to the locality Loc^- (respectively, Loc^+) to start the synchronization. Concretely, α^- denotes that the earliest start of the synchronization can be advanced α^- time units before Loc^- time value, providing that at least one task of set^- has already started. On the other side, α^+ denotes that the earliest start of the rendezvous can be advanced α^+ time units before the occurrence of Loc^+, but not after Loc^-, providing that at least one task of set^- has already started its execution (see Fig. 1).

Moreover, δ^- (respectively δ^+) specifies the acceptable delay relatively to the locality Loc^- (respectively, Loc^+) to end the synchronization. δ^- denotes that the synchronization must end no later than δ^- time units after Loc^-, and not before Loc^+, providing that at least one task of set^+ is still running. δ^+ denotes that the end of the synchronization can be delayed to no longer than δ^+ time units after Loc^+ providing that at least one task of set^+ has not yet ended its execution (see Fig. 1).

3.2. Timed rendezvous patterns

By assuming the semantics of a rendezvous given previously, we explore in this section the different global synchronization strategies that can be generated according the values taken by the localities. Then, we enumerate all the patterns and sub-patterns that can be induced by each strategy. We discuss each case and highlight its semantics through a case study example. Finally, by combining these patterns, we introduce and discuss all the possible synchronization rules that can be associated with the concept of rendezvous. Notice that this study is more exhaustive than that given in Hamdani and Abdelli (2018) which relies on the basic version of the rendezvous and where only subsets of strategies, patterns and synchronization rules have been briefly introduced according to the old semantics of the concept of rendezvous.

Let $R = (T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$ be a rendezvous. According to the value of the delay parameters and the two localities, we define and discuss in the sequel a panel of useful synchronization patterns that can be derived from the concept of rendezvous. At first level, by combining the values of the localities Loc^- and Loc^+ we obtain twelve general rules that we regroup in four general strategies:

- $(Op^- = MAX)$: In such a strategy, the start of the synchronization is decided when all the tasks of Set^- are ready to start. This strategy is called the *Co-Begin* strategy.
- $(Op^- = MIN)$: In such a strategy, the earliest time to meet the synchronization is decided by the first task among Set^- that started its execution. This strategy is called the *One-Begin* strategy.
- $(Op^+ = MIN)$: In such a strategy, the end of the rendezvous is determined when all the tasks of Set^+ are ready to end their execution. This strategy is called the *Co-End* strategy.
- $(Op^+ = MAX)$: In such a strategy, the latest time to end the synchronization is decided when all the tasks of Set^+ have achieved their execution in asynchronous manner. This strategy is called the *One-End* strategy.

The *One-Begin* and the *CoBegin* strategies set up the way the tasks should synchronize their start whereas the *One-End* and the *CoEnd* specify how the tasks should synchronize their end. Each strategy defines a class of patterns. By restricting the conditions, we can define sub-patterns in each class as we will discuss hereafter. Then, by considering specific values for the delay parameters, each class of patterns can be relaxed to cover partially or totally the patterns of another class.

Example: To better understand the previous concepts, let us consider the following workflow example. A travel agency needs to organize national and international trips. The set of the following tasks are required: After registering the client request, the operator has to process the requests to find out propositions that will be delivered to the client. If the client agrees to one of the proposition, the booking is made. The booking consists on different subtasks that are processed in parallel: booking the flight (*Book-flight* task), the hotel (*Book-hotel* task), and the car (*Book-car* task) as well as buying a travel insurance for cancellation and luggage loss (*Insur-cancel* and *Insu-loss* tasks). According to the client desire, the booking can include either all the tasks or only a subset, as for instance when it comes to buy a travel insurance. The time

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

constraints associated with each task determine the earlier date to start the booking and the latest date to achieve it. Each task has it own time constraints depending on the availability, client and administrative requirements (e.g visa). The processing of the tasks may require a specific synchronization by the operator to match the results according to different requirements. Moreover, a specific task can be advanced or delayed relatively to others to cope with flight or room unavailabilityies or with visa requirements.

3.2.1. One-Begin pattern

Formally, the *One-begin* pattern is obtained when the condition $Op^- = MIN$ is satisfied. This means that the beginning of the rendezvous is decided by the first task among Set^- to start its execution. Following the case T_s is empty or not, two subclasses of *One-Begin* patterns can be derived:

- If $T_s = \varnothing$: We have necessarily $Set^- = T_m$. All the tasks considered in the *One-Begin* pattern have the same privileges. We call this subclass, the *Close One-begin*(COB).
- –If $T_s \neq \varnothing$: In this subclass, the presence of at least one slave task is required in the *One-Begin* pattern. We call this subclass of patterns, the *Open One-Begin*. Moreover, following the value of Set^- two types of *Open One-Begin* are reported:
  - –If $Set^- = T_m$: The start of the synchronization is decided by the first master task to execute. We call this type of patterns, the *Master Open One-Begin* (MOOB).
  - –If $Set^- = T_r$: The beginning of the synchronization is decided by the first task to execute. This type of pattern is called the *Extended Open One-begin* (EOOB).

The difference between the COB and the MOOB is that the end of the latter can be decided by the end of a slave task, whereas the end of the former is necessarily decided by a master task. However, the start of both is decided by the first master task to execute.

All the patterns defined as *One-Begin* cannot be restricted or relaxed when specifying in addition the parameters (α^-, α^+). This is because the start of the synchronization cannot be advanced to earlier than the first executed task among Set^-.

The Fig. 2 illustrates the class of *One-Begin* patterns. We have $T_m = \{t_{m1}, t_{m2}\}$ and $T_s = \{t_{s1}, t_{s2}\}$. The dark grey boxes denote the nominal duration of the synchronization, the light grey boxes denote the extension in the duration of the synchronization.

In the COB pattern, we exclude the slave tasks $\{t_{s1}, t_{s2}\}$. The synchronization holds when the first master task is executed (t_{m1}); such that x_{tm1} denotes the date of Loc^-. The end of the execution of the COB depends on the value of Loc^+. This covers the range $[y_{tm1}, y_{tm2}]$ from the first master task that ends its execution (t_{m1} if $Loc^+ = (T_m, MIN)$) to the last one (t_{m2} if $Loc^+ = (T_m, MAX)$).

In case the slave tasks are involved in the pattern, the MOOB pattern is offered when the first master task is executed (t_{m1}), whereas the EOOB starts when the first task begins (t_{s1}).

Depending on the value of Loc^+, the execution of the MOOB pattern can end in the best case when the first task ends its execution (t_{s1} if $Loc^+ = (T_r, MIN)$). The duration of the pattern can be extended to the last task that achieves its execution (t_{s2} if $Loc^+ = (T_r, MAX)$). For the EOOB, the achievement of the synchronization must be decided by the end of a master task. Therefore, the nominal duration lasts till the first master task to end (t_{m1} if $Loc^+ = (T_m, MIN)$) and can be extended to the last master task to end (t_{m2} if $Loc^+ = (T_m, MAX)$).

Example: Let us consider the travel agency example, wherein we assume the following subtasks for a domestic trip booking process $T_m = \{Book - hotel, Book - Flight, Book - Car\}$and $T_s = \{Insu - trip, Insu - Loss\}$. The booking tasks are executed in parallel and it may be needed to synchronize their executions.

If all the items are booked separately then we apply a *One-Begin* synchronization. The booking date is given by the date of the first task to be processed. In the case the customer does not need to buy the travel insurance proposed by the agency, we have $T_s = \varnothing$; the COB pattern is considered here. However, if he is not sure whether he will buy the insurance and asks for additional time to decide, then we apply a MOOB synchronization. In the case he wants to book for the insurances first, to prevent any cancellation or mishap, then we apply the EOOB pattern.

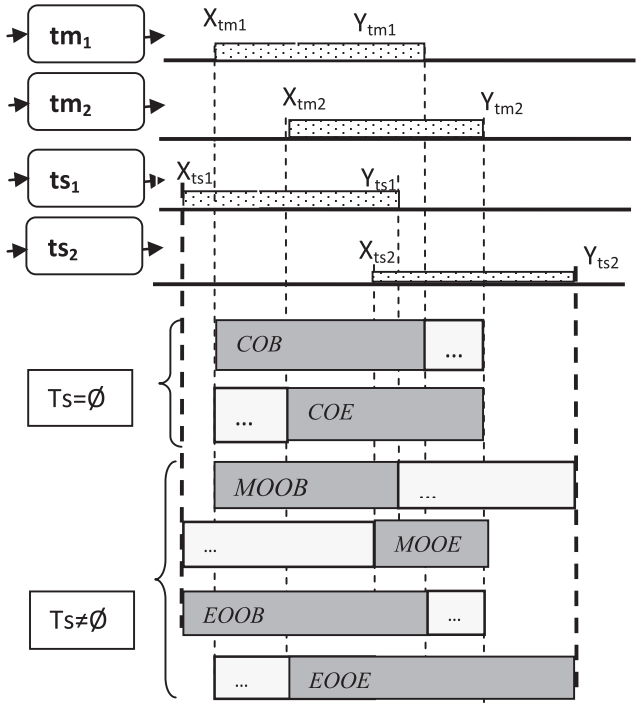

Fig. 2. One-End and One-Begin Patterns.

3.2.2. One-End pattern

Formally, the *One-End* patterns is derived when $Op^+ = MAX$. This means that the synchronization ends when the last master task of Set^+ achieves its execution. Following the case T_s is empty or not, two subclasses of *One-End* patterns can be derived:

- If $T_s = \varnothing$: We have necessarily $Set^+ = T_m$. All the tasks considered in the *One-End* pattern have the same privileges. We call this subclass of patterns *Close One-End patterns* (COE).
- If $T_s \neq \varnothing$: In this subclass, the presence of at least one slave task is needed. We call this subclass the *Open One-End*. According to the value of Set^+, two types of *Open One-End* are reported:
  - If $Set^+ = T_m$: The end of the synchronization is decided by the first master task to end its execution. We call this type of patterns the *Master Open One-End* (MOOE).
  - If $Set^+ = T_r$: The synchronization achieves when the last task ends its execution. This type of pattern is called the *Extended Open One-End* (EOOE).

Although the synchronization in the COE and the MOOE ends in the same way, the start may be different. Indeed, the start of the MOOE can be decided by the start of a slave task, whereas the start of the COE is necessarily decided by the start of a master task.

As for the *One-Begin* class, all the patterns defined in this *One-End* class cannot be relaxed when varying in addition the delays

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

parameters (δ^-, δ^+). This is because the end of the synchronization cannot be delayed to later than the last executed task among Set^+.

In Fig. 2, we depict the semantics of the *One-End* patterns. In case of a COE, the tasks $\{t_{s1}, t_{s2}\}$ are ignored. The synchronization ends when the last master task is achieved (t_{m2}); y_{tm2} denotes the date of Loc_+. The start of the pattern is decided by a master task and depends on the value of Loc^-. This covers the range $[x_{tm1}, x_{tm2}]$ from the first master task that begins its execution (t_{m1} if $Loc^- = (T_m, MIN)$) to the last one (t_{m2} if $Loc^- = (T_m, MAX)$).

In case of a MOOE pattern, the latter ends when the last master task ends (t_{m2}) whereas the EOOE ends when the last task is achieved (t_{s2}). Depending on the value of Loc^-, the beginning of the MOOE covers the range from the date when the first task starts its execution (t_{s1} if $Loc^- = (T_r, MIN)$) to the date when the last task begins its execution (t_{s2} if $Loc^- = (T_r, MAX)$).

Regarding the EOOE pattern, its start must be decided by the beginning of a master task. Hence, its nominal duration is bounded by the last master task to begin (t_{m2} if $Loc^- = (T_m, MAX)$) and can be extended to the first master task to start (t_{m1} if $Loc^- = (T_m, MIN)$).

Example: Let us go back to our booking example. If we consider the following subtasks for a domestic trip booking process T_m = {*Book-hotel, Book-Flight, Book-Car*} and T_s = {*Insu-trip, Insu-Loss*}. In case the client does not require insurance ($T_s = \varnothing$), the booking ends when all the master tasks are processed; we apply here the COE pattern. However, in the case he needs an insurance, two cases can be seen. If he wants to book later the insurance through the travel agency, we apply the EOOE pattern. Otherwise, if he has already booked first for the insurance, and needs only to complete the booking of the required items, we apply the MOOE pattern.

3.2.3. CoBegin patterns

The class of *CoBegin* patterns regroups all the patterns such that the start of the synchronization occurs when all the tasks of Set^- are ready to start conjointly their execution. Formally, we have $Op^- = MAX$. At the first level, two subclasses of the *CoBegin* can be considered following the case T_s is empty or not:

- If $T_s = \varnothing$: In this subclass, all the tasks considered in the *CoBegin* have the same privileges. We call this subclass of patterns, the *close CoBegin*.
- If $T_s \neq \varnothing$: In this subclass, the presence of at least one slave task is needed. We call this subclass of patterns the *Open CoBegin*. Within this subclass, we report different types of patterns following the value of Set^-:
  - –: $Set^- = T_m$: In this particular case, the beginning of the synchronization is decided when all the master tasks start conjointly their execution. We call this type of pattern a *Master Open CoBegin*.
  - –: $Set^- = T_r$: In this case, the synchronization begins when all the tasks are ready to start their execution. We call this type of pattern, the *Extended Open CoBegin*.

When considering in addition the associated delay parameters (α^-, α^+), the subclasses of the *CoBegin* can be relaxed to cover other classes of patterns.

- If $(\alpha^- = 0)$: This means that the start of the synchronization cannot be advanced relatively to the date Loc^- of the related *CoBegin* pattern. This type of pattern is called a *Synchronous CoBegin*. Hence, following the subclass of pattern we are dealing with, we obtain the *Synchronous Close CoBegin*(SCCB), the *Synchronous Master Open CoBegin* (SMOCB) or the *Synchronous Extended Open CoBegin* (SEOCB).
- If $(\alpha^- \neq 0)$: In this case, different types even classes of patterns are covered:
  - –if $(\alpha^+ \in [0, +\infty[)$ or $(\alpha^- \in]0, +\infty[)$: In this case, the *CoBegin* is relaxed to permit that the early start of the synchronization be advanced in time relatively to the associated *synchronous CoBegin*, but not earlier than the related *One-Begin*. Following the subclass of pattern, we obtain the *Advanced Close CoBegin* (ACCB), the *Advanced Master Open CoBegin* (AMOCB) or the *Advanced Extended Open CoBegin* (AEOCB).
  - –if $(\alpha^- = +\infty)$ and $(\alpha^+ = +\infty)$: Assuming these conditions, the *CoBegin* is relaxed to the maximum by advancing its execution to behave like the related *One-Begin*.

The different variants of the *CoBegin* patterns are depicted in Fig. 3a. The SCCB starts when all the master tasks are ready to start (t_{m2} is the last master task to start). Its nominal duration is reached when all the master tasks are ready to end (when t_{m1} ends), and can be extended to the last master task to achieve its execution (t_{m2} if $Loc^+ = (T_m, MAX)$). The SCCB can be advanced but no earlier than the start of the COB (start of t_{m1}). The SMOCB patterns are similar to the SCCB ones in the way they start. However, the nominal durations of SMOCB are set up when the first task ends (t_{s1} if $Loc^+ = (T_r, MIN)$) and can be extended to the last task to end (t_{s2} if $Loc^+ = (T_r, MAX)$).

Regarding the SEOCB, this pattern starts when all its tasks are ready to begin (when t_{s2} starts) and can be advanced to no earlier than the first task to begin (t_{s1}) (Start of the EOOB). Its nominal duration is set up when the first master task ends (t_{m1} if $Loc^+ = (T_m, MIN)$) and can be extended to the last one to end (t_{m2} if $Loc^+ = (T_m, MAX)$). It is noteworthy to recall that if $Set^- = T_r$, then we have necessarily $Set^+ = T_m$. Conversely, if $Set^- = T_m$, then Set^+ can refer either to T_r or T_m.

Example: Let us go back to our booking example. If we consider the following subtasks for a domestic trip booking process T_m = {*Book-hotel, Book-Flight, Book-Car*} and T_s = {*Insu-trip, Insu-Loss*}. In case the client does not require insurance ($T_s = \varnothing$) and has preference for a hotel and type of car, the operator can decide to search for availability of the car, the room and the flight within the same request in order to match the availability dates according to user requirements. In this case, we apply the SCCB pattern. If there is a high demand for instance for the hotel the operator can decide to advance the start of the booking by first booking for any available room in the required hotel. Then he matches the latter with other results once the booking of the flight and the car are processed. In this case we apply the ACCB pattern. However, in the case the client needs to add an insurance to the process, two cases can be seen. If he wants to book later the insurance through the travel agency, we apply the SMOCB pattern. However, we apply the AMOCB in case the booking of the room should be performed before the other required tasks. On the other hand, if the client is sure to buy the insurances, the operator can include this task when starting to process all the requests together; we apply in this case the SEOCB pattern. The AEOCB is considered in case the booking for the room is processed beforehand.

3.2.4. CoEnd patterns

This class includes all the patterns such that the latest time to end the synchronization is decided when all the tasks among Set^+ are ready to achieve conjointly their execution. Formally, we have $Op^+ = MIN$. At the first level, two subclasses of the *CoEnd* can be considered following the case T_s is empty or not.

- If $T_s = \varnothing$: We have necessarily $Set^+ = T_m$. All the tasks considered in the *CoEnd* pattern have the same privileges. We call this subclass of patterns, the *Close CoEnd*.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

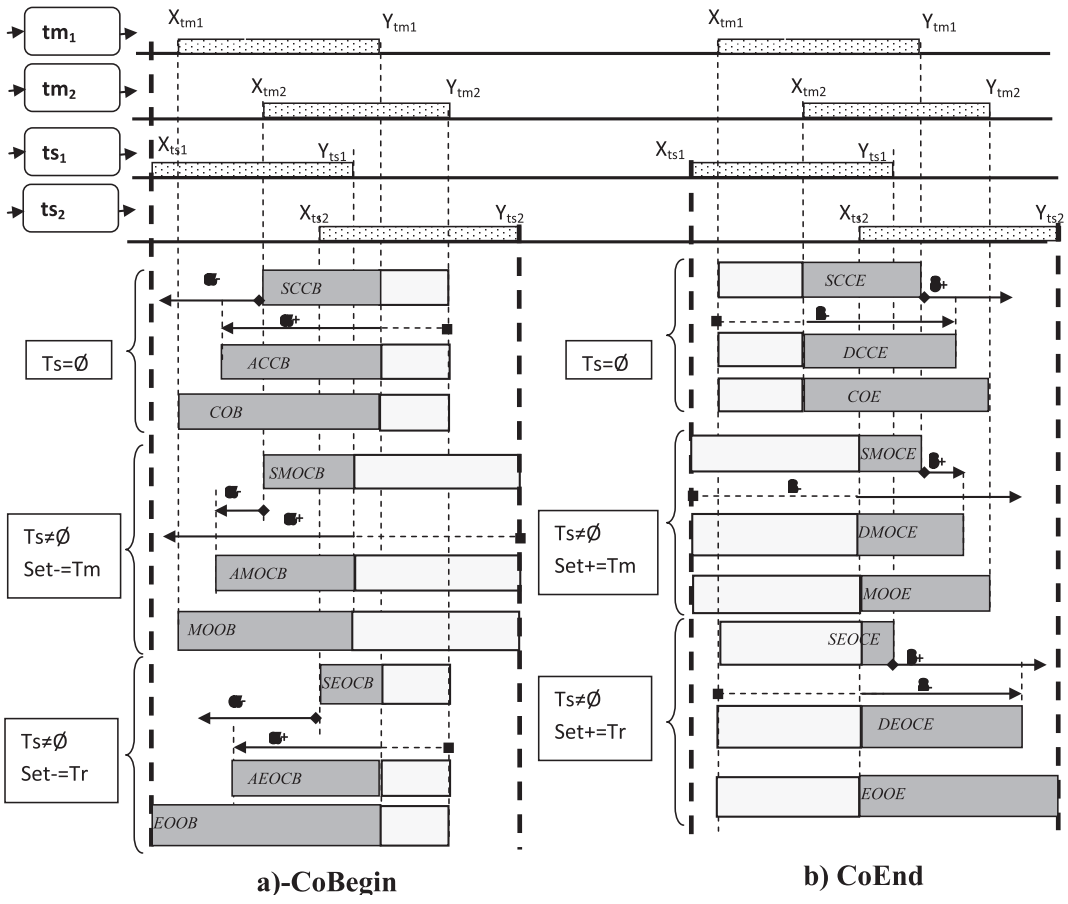

Fig. 3. CoEnd and CoBegin Patterns.

- If $T_s \neq \varnothing$: In this subclass, the presence of at least one slave task is required in the *CoEnd*. We call this subclass, the *Open CoEnd*. Following the value of Set^+, we report two sub-patterns:
  - : $Set^+ = T_m$: In this case, the synchronization ends when the first master task achieves its execution, providing that all the master tasks have already started. We call this type of patterns a *Master Open CoEnd*.
  - : $Set^+ = T_r$: The end of the rendezvous is decided when the first task is accomplished, providing that all the tasks have already started. We call this sub-patterns the *Extended Open CoEnd*.

When considering in addition the associated delay parameters $((\delta^-, \delta^+)$, the different subclasses of the *CoEnd* can be relaxed to cover new types of pattern even other previous classes.

- If $(\delta^+ = 0)$: This means that the end of the synchronization is decided by the end of the last task to execute without considering any time delay. We call this pattern the *synchronous CoEnd*. Following the subclass of pattern we consider, we have the *Synchronous Close CoEnd* (SCCE), the *Synchronous Master Open CoEnd* (SMOCE) and the *Synchronous Extended Open CoEnd* (SEOCE).
- If $(\delta^+ \neq 0)$: In this case, different subtypes even classes of patterns are covered:
  - –if $(\delta^- \in [0, +\infty[)$ or $(\delta^+ \in]0, +\infty[)$: In this case, the *CoEnd* is relaxed to permit that the *Synchronous CoEnd* pattern be delayed in time, but no later than the related *One-End* pattern. We call this pattern the *delayed CoEnd*. According to the subclass to which the pattern belongs, we obtain: the *Delayed Close CoEnd* (DCCE), the *Delayed Master Open CoEnd* (DMOCE) and the *Delayed Extended Open CoEnd* (DEOCE).
  - –if $(\delta^- = +\infty)$ and $(\delta^+ = +\infty)$: In this case, the *Synchronous CoEnd* is relaxed to the maximum by delaying the end of its execution to behave like the related *One-End* pattern.

The different classes and types of the *CoEnd* are illustrated in Fig. 3b. The SCCE ends when all its master tasks are ready to end conjointly (t_{m1} is the first master task to end forcing t_{m2} to end). Its nominal duration is delimited by the last master task to start (t_{m2}), and can be extended to the first one (t_{m1}). The SCCE can be delayed but no later than the COE, to specify the DCCE pattern. The *master Open CoEnd* patterns are similar to their related *Close CoEnd* in the way they end. However, their nominal duration is set up when the last task starts (t_{s2} if $Loc^- = (T_r, MAX)$) and can be extended to the first task to start (t_{s1} if $Loc^- = (T_r, MIN)$).

The SEOCE achieves when all its tasks can terminate conjointly (when t_{s1} ends), and its end can be delayed to no later than the last task to achieve its execution (t_{s2}), to specify the DEOCE pattern. Its nominal duration is delimited by the last master task to start (t_{m2} if $Loc^- = (T_m, MAX)$) and can be extended to the start of t_{m1}, if $Loc^- = (T_m, MIN)$.

Example: Let us go back to our working example. As stated previously, we consider the following subtasks for a domestic trip booking process T_m = {*Book-hotel, Book-Flight, Book-Car*} and T_s = {*Insu-trip, Insu-Loss*}. In case the client does not want to book for insurance ($T_s = \varnothing$) but prefers to stay in a specific hotel and books for a type of car during the period of his trip. The operator can decide to perform the booking of all the tasks in one same request in order to match the availabilities. Hence, the booking is ended

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

when all the tasks are achieved conjointly. In this case we apply the SCCE pattern. In case the booking date of the room does not match exactly the trip period, the operator can then delay the end of the booking waiting for a room to free. This process can be designed by the DCCE pattern.

However, in the case the client needs to add an insurance to his trip, two cases can be seen: If he is not sure whether he will buy it through the travel agency, he needs to confirm his decision before the end of the booking of the required tasks. The operator will perform the latter in one request without including the booking of the insurances which will be performed separately in case the client confirms his decision before the due time. This process can be specified by applying the SMOCE pattern. If in addition, the dates of the room can be subject to change due to a mismatch with the trip period, we enforce the DMOCE pattern. On the other hand, if the client wants to buy the insurances through the travel agency, the operator can decide to include the latter in the same request as for the required tasks. The booking is then achieved when all the tasks are processed conjointly according to the client requirements. The SEOCE pattern is considered in this case. If the process needs to be delayed waiting for a room to free, then we consider the DEOCE.

3.2.5. Rendezvous synchronization rules

A rendezvous is driven by a strategy to start the synchronization and another one to end it. By combining the previous patterns, we can define generic rules that settle the global behaviour of the rendezvous. The chart depicted in Table 1 enumerates the set of pattern combinations with the associated rule-names. Having said that, it needs to be understood that some strategies cannot be combined, as for instance the *extended One-Begin* with *extended One-End*. This is because we need that at least one of the two strategies be driven by master tasks. Concretely, if all the tasks are of the same privilege ($T_s = \varnothing$), then we can derive 9 rules. These are obtained by considering a Loc^- strategy from {SCCB, ACCB, COB} with a Loc^+ strategy from {SCCE, DCCB, COE}. We call these rules the *non master rules*. On the other hand, if tasks are not of the same privileges ($T_s \neq \varnothing$), we derive, what we identify as the *master rules*. The Loc^- strategies from {SMOCB, AMOCB, MOOB} can be combined with any of the 6 Loc^+ strategies, thus yielding to 18 possible master rules. However, the Loc^- strategies from {SEOCB, AEOCB, EOOB} can be combined only with {SMOCE, DMOCE, MOOE}, inducing 9 additional master rules. In overall, we generate 27 master rules among the 36 rules. However, certain rules can be refined further by considering different combinations of the parameters $\alpha^-, \alpha^+, \delta^-, \delta^+$).

The *AND* related rules denote a synchronization where either the *Cobegin* or the *CoEnd* strategies are considered. The *Or* related rules are those which adopt the *One-begin* strategy, whereas the *Weak* related rules apply the *One-End* strategy. The*Strong* rules are those which are involving slave tasks in one of the two combined strategies. The rules identified as *Advanced*, are those with ($\alpha^- > 0$). This denotes that the start of the synchronization is advanced relatively to the start of the original rule (e.g. The *advanced AND*starts before the *AND*). On the other hand, The rules noted as *Delayed* are those with ($\delta^+ > 0$). This specifies that the end

Table 1
Synchronization rules.

Rules	Loc^-	Loc^+	(Loc^-, Loc^+)	$(\alpha^-, \alpha^+, \delta^-, \delta^+)$	Cons
Non master rules	$T_s = \varnothing$				
AND	SCCB	SCCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, 0)$	No
Delayed AND	SCCB	DCCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, > 0)$	Yes
Weak-AND	SCCB	COE	$((T_m, MAX), (T_m, MAX))$	$(0, ?, ?, ?)$	Yes
Open-AND	ACCB	DCCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, > 0)$	Yes
Advanced And	ACCB	SCCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, 0)$	Yes
Advanced WeakAnd	ACCB	COE	$((T_m, MAX), (T_m, MAX))$	$(> 0, ?, ?, ?)$	Yes
Or	COB	COE	$((T_m, MIN), (T_m, MAX))$	$(?, ?, ?, ?)$	Yes
Or-AND	COB	SCCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, 0)$	Yes
Delayed Or-AND	COB	DCCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, > 0)$	Yes
Master rules	$T_s \neq \varnothing$				
Master-AND	SMOCB	SMOCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, 0)$	No
Del Master-AND	SMOCB	DMOCE	$((T_m, MAX), (T_m, MIN))$	$(0, ?, ?, > 0)$	Yes
Strong Weak Master-AND	SMOCB	EOOE	$((T_m, MAX), (T_r, MAX))$	$(0, ?, ?, ?)$	Yes
Del Strong Master-AND	SMOCB	DEOCE	$((T_m, MAX), (T_r, MIN))$	$(0, ?, ?, > 0)$	Yes
Weak Master-AND	SMOCB	MOOE	$((T_m, MAX), (T_m, MAX))$	$(0, ?, ?, ?)$	Yes
Strong Master-AND	SMOCB	SEOCE	$((T_m, MAX), (T_r, MIN))$	$(0, ?, ?, 0)$	No
Master-Or-AND	MOOB	SMOCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, 0)$	Yes
Del Master-Or-AND	MOOB	DMOCE	$((T_m, MIN), (T_m, MIN))$	$(?, ?, ?, > 0)$	Yes
Strong Master-Or-AND	MOOB	SEOCE	$((T_m, MIN), (T_r, MIN))$	$(?, ?, ?, 0)$	No
Master-Or	MOOB	MOOE	$((T_m, MIN), (T_m, MAX))$	$(?, ?, ?, ?)$	Yes
Strong Master-Or	MOOB	EOOE	$((T_m, MIN), (T_r, MAX))$	$(?, ?, ?, ?)$	Yes
Del Strong Master-Or-AND	MOOB	DEOCE	$((T_m, MIN), (T_r, MIN))$	$(?, ?, ?, > 0)$	Yes
Del Strong AND-Master	SEOCB	DMOCE	$((T_r, MAX), (T_m, MIN))$	$(0, ?, ?, > 0)$	Yes
Strong AND-Master	SEOCB	SMOCE	$((T_r, MAX), (T_m, MIN))$	$(0, ?, ?, 0)$	No
Strong Weak AND-Master	SEOCB	MOOE	$((T_r, MAX), (T_m, MAX))$	$(0, ?, ?, ?)$	No
Strong Open AND-Master	AEOCB	DMOCE	$((T_r, MAX), (T_m, MIN))$	$(> 0, ?, ?, > 0)$	Yes
Adv Strong AND-Master	AEOCB	SMOCE	$((T_r, MAX), (T_m, MIN))$	$(> 0, ?, ?, 0)$	Yes
Adv Strong Weak AND-Master	AEOCB	MOOE	$((T_r, MAX), (T_m, MAX))$	$(> 0, ?, ?, ?)$	Yes
Del Strong OR-AND Master	EOOB	DMOCE	$((T_r, MIN), (T_m, MIN))$	$(?, ?, ?, > 0)$	Yes
Strong OR-AND Master	EOOB	SMOCE	$((T_r, MIN), (T_m, MIN))$	$(?, ?, ?, 0)$	Yes
Strong Or-Master	EOOB	MOOE	$((T_r, MIN), (T_m, MAX))$	$(?, ?, ?, ?)$	Yes
Adv Master-AND	AMOCB	SMOCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, 0)$	Yes
Master Open-AND	AMOCB	DMOCE	$((T_m, MAX), (T_m, MIN))$	$(> 0, ?, ?, > 0)$	Yes
Adv Strong Master-AND	AMOCB	SEOCE	$((T_m, MAX), (T_r, MIN))$	$(> 0, ?, ?, 0)$	Yes
Adv Weak Master-AND	AMOCB	MOOE	$((T_m, MAX), (T_m, MAX))$	$(> 0, ?, ?, ?)$	Yes
Adv Strong Weak Master-AND	AMOCB	EOOE	$((T_m, MAX), (T_r, MAX))$	$(> 0, ?, ?, ?)$	Yes
Strong Master Open-AND	AMOCB	DEOCE	$((T_m, MAX), (T_r, MIN))$	$(> 0, ?, ?, > 0)$	Yes

"?": denotes any value in the range, whereas "> 0" denotes any strict positive value.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

of the synchronization is delayed relatively to the end of the original rule (e.g The *Delayed AND*ends after the *AND*). Finally, the OPEN related rules as those with ($\alpha^- > 0$ and $\delta^+ > 0$), namely the start of the synchronization is advanced whereas its end is delayed relatively to the original rule (e.g The *Open AND* starts before and ends after the *AND*).

The execution of a rule is said to be *Critical* if the earliest start and the latest end of the synchronization refer to the same date. The duration of the synchronization in this case is atomic and cannot be extended. Moreover, the execution of a rule is said to be inconsistent if the earliest start of the synchronization cannot occur before the latest end of the synchronization. In this case, the synchronization cannot hold. This happens for instance in the AND rule when the time intervals associated with master tasks are not overlapping. The column *Const* in Table 1 identifies the rules which are always consistent whatever the time constraints of the tasks we consider.

3.3. Discussion

A large range of synchronization schemes defined in the literature are covered by our model, in addition to new rules that are introduced as the *Advanced* and *Delayed* related rules. For example, the ten rules of the *TSPN* model (Cicirelli et al., 2013), as well as the seven synchronization schemes of the *Allen* theory (James, 1983), are all covered by the concept of rendezvous. Our model covers also the *Wait* constraints of the parallel pattern (Combi and Posenato, 2009). However, our model is more expressive as it considers time constraints in form of intervals (not instants) while assuming different tasks privileges.

4. Time Workflow Net with Rendezvous: RTWFN

In this section, we first present the syntax and the semantics of the RTPN model. The latter extends the first versions introduced in Hamdani and Abdelli (2017) and Hamdani and Abdelli (2018) to new localities values while adapting the semantics. Then, we introduce a subclass of RTPN, called RTWFN (Workflow nets with timed rendezvous). This model is dedicated to specify workflow systems. Finally, we present a RTWFN based modelling approach for workflow systems.

4.1. Time Petri Net with Rendezvous: RTPN

In the theory of Petri Nets (PN), A classical Petri Net is a directed bipartite graph with two types of nodes, called places and transitions. Oriented Arcs are used to connect places to transitions or transitions to places. Graphically, places are drawn as circles and transitions as rectangles. In the sequel, we assume that the reader is familiar with Petri nets theory and we recall only the theory needed to introduce the *RTPN* model. The *RTPN* model, is in extension of Time Petri Nets (TPN). In a TPN a time interval is associated with each transition to specify in addition time constraints. The *RTPN* model adds to the latter the possibility to express synchronizations between different transitions given by the concept of rendezvous introduced previously. We revisit hereafter the syntax and the semantics of the *RTPN* model given in Hamdani and Abdelli (2017) by considering the new semantics of the rendezvous as introduced in the previous section. Formally, the new syntax of the *RTPN* model is defined as follows:

Definition 1. An *RTPN* is given by the tuple $(P, T, B, F, M_0, I_s, RDV_s)$ where: P and T are respectively two non empty disjoint sets of places and transitions; B and F are respectively the backward and the forward incidence functions $B : P \times T \longrightarrow N = \{0, 1, 2, \ldots\}$; $F : P \times T \longrightarrow N$; M_0 is the initial *marking* function that associates with each place a number of tokens $M_0 : P \longrightarrow N$; I_s is the delay interval mapping function; $I_s : T \longrightarrow Q^+ \times Q^+ \cup \{+\infty\}$, where Q^+ is a set of null or positive rational values. We write $I_s(t) = [x_0(t), y_0(t)]$. This gives the static time interval within which the transition t can fire, such that $0 \leqslant x_0(t) \leqslant y_0(t)$;

RDV_s denotes a finite set of *Rendezvous*. A *Rendezvous* R of RDV_s is a synchronization scheme that has the form $(T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$, where T_m and T_s are subsets of transitions. $T_m \cap T_s = \varnothing$ and we note $T_r = T_m \cup T_s$. T_m is a non empty and finite set of master transitions, and T_s the finite set of slave transitions. Loc^-, Loc^+ are the localities considered in the rendezvous. We have $Loc^- = (Set^-, Op^-)$ and $Loc^+ = (Set^+, Op^+)$, such that: $Set^-, Set^+ \in \{T_m, T_r\}$ and $Op^-, Op^+ \in \{MIN, MAX\}$. We assume that at least one of the parameters Set^- and Set^+ must be equal to T_m.

Finally, $\alpha^-, \alpha^+, \delta^+$ and δ^- are the delay parameters that take their values in $Q^+ \cup \{+\infty\}$.

Graphically, as depicted in Fig.4 an *RTPN* is similar to a PN. However the meta-data annotating the graph is different. This includes inter alia rendezvous and time information to specify synchronization and time constraints. A transition models a single task which is subject to time constraints given by the associated interval. The execution (Firing) of a transition requires to consume a predefined number of tokens (defining the marking) that must be located in its entry places. A place represents generally a resource or an event whereas the tokens denote the number of instances present in the given place at a given instant (defining the Marking). If a transition is governed by a rendezvous, this means that its firing depends additionally on the firing of other transitions according to the synchronization rule induced by the rendezvous.

Moreover, in the *RTPN* model, it is admitted to call a same transition in different rendezvous declarations. This allows to design the case where a single task can be governed by different alternative synchronizations. The selection of the rendezvous to enforce when more than once are offered can be decided in non deterministic manner. However, for more flexibility, we can extend the syntax of the *RTPN* with a priority function on the set of rendezvous to resolve the non determinism. In other respects, a transition that is not involved in any rendezvous of the *RTPN* is to progress in an asynchronous manner as in TPN since it is not compelled by any synchronization scheme.

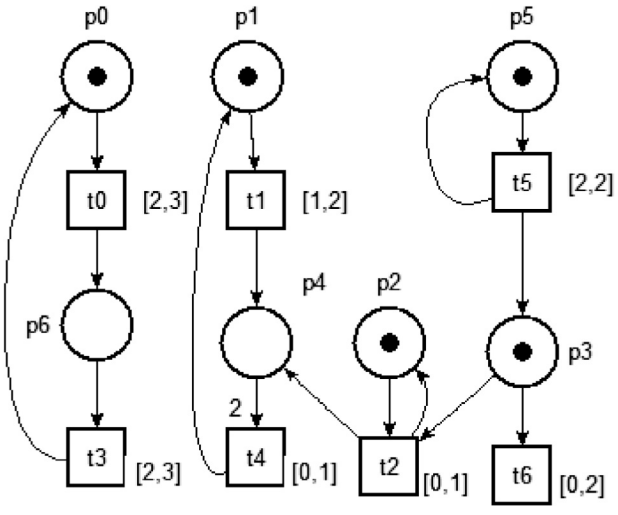

Fig. 4. An RTPN example.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

Let us consider the *TPN* example shown in Fig.4, extended to the following set of rendezvous $RDV_s = \{R_1, R_2, R_3, R_4\}$ such that:

$R_1 = (\{t_0, t_1\}, \varnothing, ((\{(t_0, t_1\}, MAX), (\{t_0, t_1\}, MIN)), (0, 0, 0, 0))$

$R_2 = (\{t_2\}, \{t_1\}, ((\{t_1, t_2\}, MIN), (\{t_2\}), MIN), (+\infty, +\infty, +\infty, 0))$

$R_3 = (\{t_6\}, \{t_2\}, ((\{t_6\}, MAX), (\{t_2, t_6\}, MAX)), (0, +\infty, +\infty, +\infty))$

$R_4 = (\{t_3, t_4\}, \varnothing, (\{t_3, t_4\}, MIN), (\{t_3, t_4\}, MIN)), (+\infty, +\infty, +\infty, 0))$

$R_5 = (\{t_3\}, \varnothing, (\{t_3\}, MAX), (\{t_3\}, MIN)), (0, 0, 0, 0))$

$R_6 = (\{t_4\}, \varnothing, (\{t_4\}, MAX), (\{t_4\}, MIN)), (0, 0, 0, 0))$

According to Table 1, the rendezvous R_1, R_5, R_6 follows the *And* rule whereas R_2 and R_3 consider respectively, the *Strong Or-AND-Master* and the *Strong Weak Master-AND* rules. Finally, R_4 is driven by the *Or-AND* rule.

Each transition defines one task characterized with its own time constraints given in the form of an interval. The arc connecting each input place to a transition models the resources needed for the execution of the task. The availability of resource instances is specified by the current marking of the places. We notice that the transitions t_1 and t_2 are involved in two different rendezvous. t_1 can progress following the synchronization scheme defined in R_1 or R_2, whereas t_2 can progress according to R_2 or R_3. Likewise the transitions t_3 and t_4 can synchronize their execution through the rendezvous R_4 or progress in asynchronous manner through the rendezvous R_5 and R_6 when R_4 cannot hold. Finally, as t_5 is not involved in any defined rendezvous, it should therefore progress in asynchronous manner.

4.2. RTPN formal semantics

Before laying down the formal semantics of the model. We introduce, first, some notations:

Let $RT := (P, T, B, F, M_0, I_s, RDV_s)$ be a *RTPN* and.

$R := (T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$ be a rendezvous from RDV_s.

- We note $Tr(R) = T_r$ the set of transitions defined in R.
- We introduce the set of asynchronous *Rendezvous*, noted RDV_a. $R_a \in RDV_a$ has the form $(\{t\}, \varnothing, ((\{t\}, MAX), (\{t\}, MIN)), (0, +\infty, +\infty, 0))$ such that t is a transition of T which is not involved in any predefined *rendezvous*, $\forall R \in RDV_s, t \notin Tr(R)$. The set RDV defines all the rendezvous that might fire in the model, we write: $RDV = RDV_s \cup RDV_a$.
- We call a marking the function, noted M that associates with each place a number of tokens, $M : P \longrightarrow \mathbb{N}$;
- A transition t is said to be *enabled* for the marking M, if $\forall p \in P, B(p, t) \leqslant M(p)$; the number of tokens in each input place of t is greater or equal to the valuation of the arc connecting this place to the transition t. Thereafter, we denote by $E(M)$ the set of transitions *enabled* for the marking M.
- A *rendezvous* R is said to be enabled for the marking M if all its transitions are enabled for M and we denote by $ER(M)$ the set of all *rendezvous* of RDV that are enabled for M. $ER(M) = \{R \in RDV | Tr(R) \subseteq E(M)\}$.
- Let M be a marking; two transitions t_i and t_j enabled for M are said to be *conflicting* for M, if $\exists p \in P, \quad B(p, t_i) + B(p, t_j) > M(p)$. Hence, we note $Conf(M)$ the relation built on $E(M)^2$ such that $(t_i, t_j) \in Conf(M)$, iff t_i and t_j are in conflict for the marking M. Hence, a rendezvous R is said to be conflicting with itself for M, if there exist two transitions $t_i, t_j \in Tr(R)$ that are in conflict for M.
- Let M be a marking; two *rendezvous* R_i and R_j enabled for M are said to be *conflicting* for M, if there exists $t_i \in Tr(R_i)$ and $t_j \in Tr(R_j)$ such that $(t_i, t_j) \in Conf(M)$.

It is noteworthy that if at least two transitions of a same rendezvous R are in conflict, then R can not hold. This denotes the case where two tasks are in conflict for the same resources which further need to synchronize their execution.

To illustrate the previous definitions, let us go back to our *RTPN*example. The predefined set of rendezvous RDV_s is extended with the asynchronous rendezvous $R_7 := (\{t_5\}, \varnothing, ((\{t_5\}, MAX), (\{t_5\}, MIN)), (0, +\infty, +\infty, 0))$.

The initial marking is given by $M_0 : p_0, p_1, p_2, p_3, p_5 \mapsto 1$; $p_4, p_6 \mapsto 0$. Hence, the set of enabled transitions is $E(M_0) = \{t_0, t_1, t_2, t_5, t_6\}$. The set of enabled rendezvous is $ER(M_0) = \{R_1, R_2, R_3, R_5\}$. As we can see the transitions t_2 and t_6 are in conflict for M_0, hence the rendezvous R_3 is in conflict with itself and cannot fire from M_0. Moreover, the rendezvous R_2, R_3 are in conflict for M_0, the firing of one of them will disable the other.

The *RTPN* model evolves by firing a *rendezvous* at each step. This implies the firing of all its transitions providing that some conditions are satisfied. Firing a rendezvous relies on the marking and on the corresponding synchronization rule which entails to meet the dynamic time constraints of the model.

Different semantics can be considered with the *RTPN*: a mono-server semantics, namely for any marking only one instance of a transition and by extension a rendezvous can be enabled for a marking. Or the multi-server one which considers that a transition and hence a rendezvous can be enabled more than once for a given marking (refer to the papers (Abdelli, 2015; Boucheneb et al., 2013). In the sequel, we consider a mono-server semantics rather than the multi-server one, by enforcing the threshold granularity policy and the intermediate memory policy. In other terms, if the current marking can enable different instances of a same transition, only one instance is considered. However, the semantics may be adapted to consider other firing policies as well as the multi-server case. Assuming that, we introduce hereafter the formal expression of a reachable state.

Definition 2. A reachable state, noted e, of a *RTPN* is a tuple $e = (M, V)$, where M is the reachable marking and V is the function that associates with each enabled transition its dynamic firing interval $V(t) := [x(t), y(t)]$. We note $e_0 = [M_0, V_0]$ the initial state, where: $V_0(t) := [x_0(t), y_0(t)]$.

The initial state of our *RTPN* example is given by (M_0, V_0) such that $V_0 : t_0 \mapsto [2, 3]; t_1 \mapsto [1, 2]; t_2 \mapsto [0, 1]; t_5 \mapsto [2, 2]; t_6 \mapsto [0, 2]$.

Given a reachable state e, we see next how to compute the firing interval of an enabled rendezvous.

Definition 3. Let $e = (M, V)$ be a reachable state of the *RTPN*. For each enabled rendezvous R, such that $R = (T_m, T_s, (Loc^-, Loc^+), (\alpha^-, \alpha^+, \delta^-, \delta^+))$ where $Loc^- = (Set^-, Op^-)$ and $Loc^+ = (Set^+, Op^+)$. We define the firing interval of R, as follows: $[Low(R), Up(R)]$:=

$$\left[MAX \left\{ \begin{array}{l} \underset{\forall t \in Set^-}{Op^-} \{x(t)\} - \alpha^- \\ MIN \left(\underset{\forall t \in Set^+}{Op^+} \{y(t)\} - \alpha^+, \ \underset{\forall t \in Set^-}{Op^-} \{x(t)\} \right) \\ \underset{\forall t \in Set^+}{MIN} \{x(t)\} \end{array} \right., \ MIN \left\{ \begin{array}{l} \underset{\forall t \in Set^+}{Op^+} \{y(t)\} + \delta^+ \\ MAX \left(\underset{\forall t \in Set^-}{Op^-} \{x(t)\} + \delta^-, \ \underset{\forall t \in Set^+}{Op^+} \{y(t)\} \right) \\ \underset{\forall t \in Set^+}{MAX} \{y(t)\} \end{array} \right. \right]$$

$[Low(R), Up(R)]$ determines the interval of relative times within which the *rendezvous* R has to fire in respects of the considered synchronization rule. This interval is computed from the dynamic firing intervals of transitions of $Tr(R)$ according to locality values and the associated delay parameters. Notice that as the number of localities values has been extended to four, the formulas provided in the previous definition allow therefore to induce more rules and synchronization schemes than in the old semantics of the *RTPN* model Hamdani and Abdelli (2017).

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

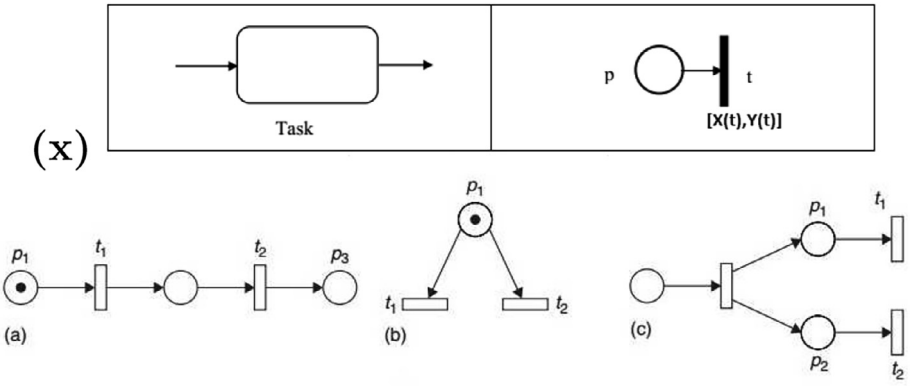

Fig. 5. Modelling approach with RTWFN.

For example, if we apply this general rule on the *RTPN* example, we obtain: R_1 can fire within $\left[\underset{\forall t\in\{t_0,t_1\}}{MAX}\{x(t)\}, \underset{\forall t\in\{t_0,t_1\}}{MIN}\{y(t)\}\right]$ = [2,2]. This requires that the transitions t_0, t_1 be fired in full synchronous manner. Concerning R_2, it can fire within $\left[\underset{\forall t\in\{t_1,t_2\}}{MIN}\{x(t)\}, y(t_2)\right]$ = [0,1]. Moreover, R_3 may fire within $\left[x(t_6), \underset{\forall t\in\{t_2,t_6\}}{MAX}\{y(t)\}\right]$ = [0,2]. Finally, the asynchronous rendezvous R_7 can fire within $[x(t_5), y(t_5)] = [2,2]$.

We give hereafter the sufficient and the necessary conditions to fire a rendezvous.

Definition 4. An enabled rendezvous R_f can fire from a reachable state, $e = (M, V)$, at a relative date d, iff:

(i) R_f is not conflicting with itself
(ii) The lower bound of R_f is reached: $0 \leqslant Low(R_f) \leqslant d$;
(iii) No upper bound of any enabled rendezvous is overtaken: $d \leqslant \underset{R\in ER(M)}{MIN}\{Up(R)\}$;

In our *RTPN* example, only R_2 can fire from the initial state e_0. Indeed, R_3 is in conflict with itself and hence is not firable. Moreover, the time cannot progress to reach the lower bound of R_1 and R_7.

We introduce now the *RTPN* formal semantics as a labelled transition system.

Definition 5. The semantics of an *RTPN* is defined as a labelled transition system $ST = (\Gamma, e_0, \rightarrow)$such that:

- Γ is the set of all reachable states;
- $\rightarrow: \Gamma \times (T \times \mathbb{Q}^+) \times \Gamma$ is the transition relation between states such that: $((M,V),(R_f,d),(M^\uparrow,V^\uparrow)) \in \rightarrow$, iff the firing conditions of Definition 4 are satisfied, such that:
- $(\forall p \in P, \forall t \in Tr(R))$, $M^\uparrow(p) := M(p) - B(p,t) + F(p,t)$
- The function $V^\uparrow$ is computed by adopting the intermediate memory policy, as follows:
  1. Remove the transitions that are disabled for $M^\uparrow$
  2. For each transition*t* that was enabled for M and remains enabled for $M^\uparrow$, shift $V(t)$ by the value*d* towards the origin of time and truncate to non negative value its lower bound if necessary. $V^\uparrow = [MAX(0, x(t) - d), y(t) - d]$.
  3. Add the transitions newly enabled for $M^\uparrow$ constrained by their static firing intervals:$V^\uparrow(t) = [x_0(t), y_0(t)]$.

Firing a rendezvous R_f implies the firing of all its involved transitions (i.e., $t \in Tr(R_f)$), which yields a new accessible state $e^\uparrow$ where the corresponding marking $M^\uparrow$ is obtained from M after firing sequentially the transitions of $Tr(R_f)$. Notice that the upper bound of the dynamic interval of an enabled transition can be overtaken by the progression of time $(y(t) < 0)$ unlike in *TPN*. This denotes that the related task has already ended its execution while the synchronization has not yet taken place. Therefore, in the *RTPN* the time constraints of enabled transitions follow a *weak semantics*, while those of enabled rendezvous obey to a *strong semantics*.

Moreover, due mainly to the density of time, the *RTPN* reachability graph defined in Definition 5, in an infinite representation of all timed behaviours of the modelled system. The analysis and verification of the properties of the model require the construction of a finite abstraction of the infinite state space that preserves a subset of the properties of interest. This could be achieved by adapting the state class graph method for instance as in Time Petri Net that mainly conserves the linear properties of the underlying model.

Let us consider the firing of the rendezvous R_2 at instant $d = 1$ from e_0 to reach the new state $e_1(M_1, V_1)$, such that: The new marking $M_1 : p_0, p_2, p_4, p_5, \mapsto 1; p_1, p_3, p_6 \mapsto 0$ Hence, the set of enabled transitions is $E(M_1) = \{t_0, t_5\}$. $V_1 : t_5 \mapsto [2-1, 2-1] = [1,1]; t_0 \mapsto [2-1, 3-1] = [1,2]$. Only R_5 is enabled an firable at instant 2 from the state e_1.

4.3. Time Workflow Net with Rendezvous: RTWFN

We introduce, in the following, the RTWFN model which is a particular case of an RTPN.

Definition 6. A RTWFN noted RT_w is a tuple (RT, p_b, p_e), such that:

- $RT = (P, T, B, F, M_0, I_s, RDV_s)$ is a Time Petri Net with rendezvous.
- p_b is a special place of P called the beginning place of the workflow net, and we have: $\bullet p_b = \varnothing$ and $M_0(p_b) \neq 0$;
- p_e is a special place of P called the ending place of the workflow, and we have: $p_e\bullet = \varnothing$ and $M_0(p_e) = 0$; where: $\bullet x$ denotes the set of input transitions connected to the place x while $x\bullet$: gives the set of output transitions connected to x.

The place p_b denotes the source of the net while the place p_e the sink of the net. The RTWFN should verify that there exists a run from the initial marking including the place p_b to a final marking including the place p_e ; we say that the net is *strongly connected*.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

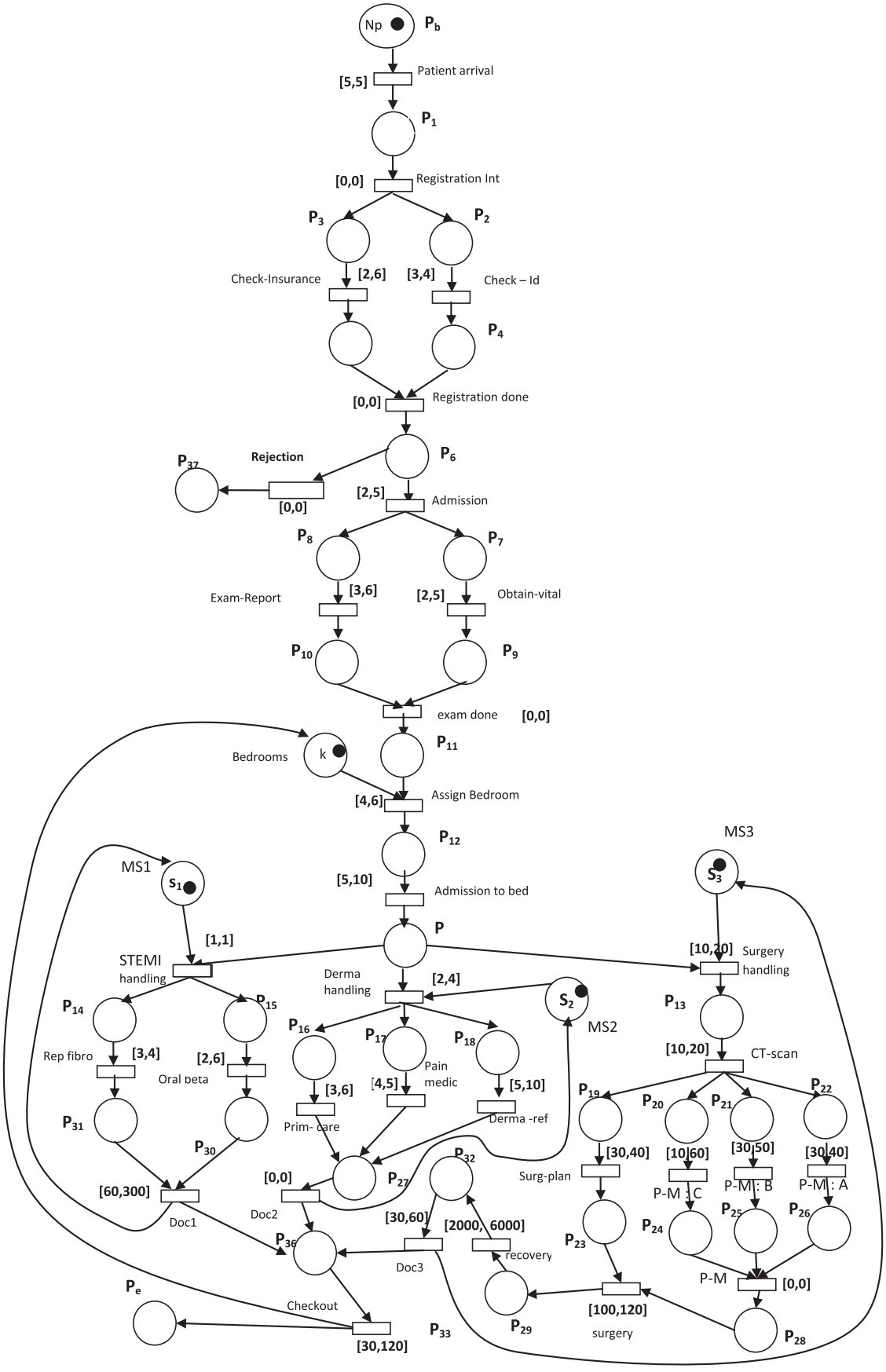

Fig. 6. Patient workflow modelled by an RTWFN.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

4.4. Modeling workflow with synchronization and time delay using RTWF-Net

The RTWFN model of the whole workflow constrained by synchronization and time delays can be obtained by the following approach:

1) First we create the places p_e and p_b.
2) *A single task:* Each elementary unit: (*task*) is mapped into a transition $t \in T$ and an input place $p \in P$. For time constraints a time interval is associated with each transition's task $I(t) = [x(t), y(t)]$, thus, defining the earliest and the latest time delay of the task. If no time constraint are imposed, we have $I(t) = [0, +\infty]$. Otherwise with $I(t) = [0, 0]$ the task t cannot be delayed and must occur as soon as the input place is marked (See Fig. 5(x)). If the task is the first in the process then its input place is p_b. If it is the last in the workflow then its output place is p_e.
3) *Sequence*: In the example of Fig. 5a, tasks t_1 and t_2 are executed sequentially, representing precedence constraints of task execution in the workflow;
4) *Choice*: In Fig. 5b, t_1 and t_2 are in conflict and can never occur both;
5) *Concurrency*: In Fig. 5c, tasks are in concurrency; they occur in parallel and are not in conflict. Their execution can be governed by synchronization rules that are expressed in the form of rendezvous.

4.5. Cases study – the patient worflow

In this section, we present a case-study to highlight the expression power of the RTWFN model in modelling workflow systems with complex synchronization schemes.

We consider here the example of the *Patient workflow*. The associated RTWFN is depicted in Fig. 6. This example is a combination and an adaptation of many health-care workflows already addressed in the literature (Lanz et al., 2010; Salimifard et al., 2013; Combi and Posenato, 2009). The problem is as follows: Every 5 min (transition *patient-arrival*), one patient comes to the private hospital. We assume that in average *Np* patients arrive everyday which is denoted by the initial marking of the place P_b. At the reception desk, the patient is asked to give both his id and insurance cards, then called to take a seat in the waiting room. Once his case is handled, its two cards are inserted in the system for authentication and verification (see transitions *Check-Insurance, Check-Id*). The execution of both tasks should be coordinated as data need to be exchanged between different systems. However, in case one of the two tasks is late to respond the coordination can be delayed for no later than to 2 time units after the maximal due time. The latencies of both systems to respond to the query are respectively [2,6] and [3,4]. To specify these requirements, we consider the following rendezvous:$R_0 := (T_m = \{Check\text{–}insurance, Check - id\}, \varnothing, ((T_m, MIN), (T_m, MIN)), (\infty, \infty, \infty, 2))$.

The associated synchronization rule to R_0 refers to the non master rule*Delayed Or-And*. If the insurance and the id are valid, the patient is admitted and called to move to a second room for a first diagnosis (transition *Admission*). Otherwise, his admission is rejected (transition *Reject*). We assume that the move lasts between 2 and 5 min.

Once the patient is examined for a first diagnosis, a nurse checks for his vital data (transition *Obtain-vital*) while the emergency doctor asks questions and reports the results on a dedicated document (transition *Exam-Report*). The doctor should states in its report a first diagnosis and assigns the next service to handle the patient. Both tasks should start together and the exam ends when the last task ends. To specify the previous constraints, we consider the following rendezvous associated with a *weak-And* rule: $R_1 := (T_m = \{Exam - Report, Obtain - vital\}, \varnothing, ((T_m, MAX), (T_m, MAX)), (0, \infty, \infty, 0))$.

Once the exam is achieved, the patient is admitted to the service provider to be assigned a free bedroom (transition *assign-bedroom*). We assume that the hospital possesses k bedrooms which is denoted by the initial marking of the place *bedrooms*. If no bedroom is available, the patient should wait until one is released. The patient is then admitted to its bedroom and waits to be handled by the staff of the service he was directed to. We consider in our case three services which are: the SETMI, the dermatology and the surgery services The places Ms_1, Ms_2 and Ms_3 denote the number of patients that can be handled in parallel in the three services. We assume that the capacity of the three services are respectively s_1, s_2 and s_3 and depend on the available medical teams in each service. The patient can be handled only if a team is free (transitions *SETMI-handling, Derma-Handling, Surgery-handling*).

The STEMI (segment Elevation Myocardial Infraction) care requires two therapeutic actions: In presence of a myocardial infection, a reperfusion is obtained through a fibrinolytic therapy (master task denoted by the transition *Rep-fibr*). The complementary action consists of the administration of beta blocker drugs (denoted by the slave transition *Oral-beta*). We want to express the fact that the oral therapy neither can start more than 1 min before the start of reperfusion nor can finish more than 2 time unit after reperfusion ends. To model this synchronization, we consider the following rendezvous associated with the master rule *Strong-Open-AND-Master*.$R_2 := (T_m = \{rep - fibr\}, \ T_s = \{oral - beta\}, ((T_m, MAX), (T_m, MIN)), (1, \infty, \infty, 2))$.

As concerns the dermatology care, it is composed of three parallel independent therapeutic actions: a primary care referral, a pain medication and dermatology referral actions. Following the case, one, two or the three actions are required. To model these requirement we consider the following rendezvous with the rule $OR.R_3 := (T_m = \{Prim - care, derma - ref, pain - medic\}, \varnothing, ((T_m, MIN), (T_m, MAX)), (\infty, \infty, \infty, \infty))$.

The last service deals with surgery care following a femur fracture diagnosis. After a CT-scan is done, the next action consists of two main parallel tasks: surgical planning and pain medication. The latter task requires the administration of two mandatory drugs $p - m - A$ and $p - m - B$ and an optional one $p - m - C$. A and B must be taken together, however C has to be given 5 time units before. Example: Administration of an anti-biotic with an anti-inflammatory and gastric bandages. For these requirements, we consider the following rendezvous having the master rule *Adv Weak-Master-AND* rule. $R_4 := (T_m = \{p - m - A, p - m - B\}, (T_s = \{p - m - C\}, ((T_m, MAX), (T_m, MAX)), (5, \infty, \infty, \infty))$.

Meanwhile to the medication action, the surgery is being planned (transition *Surg-plan*). After the surgery action, a period of recovery is observed.

Whatever the service we consider, the patient is kept in observation in its room a certain time once the medical care is achieved. In parallel, his health report is produced (transitions *Doc1,Doc2, Doc3*). Then he is asked to checkout and the bedroom is freed up for the admission of a new patient.

5. Conclusion

In this paper, we have mainly addressed the modelling of timed workflow systems with complex synchronizations. We first, introduced the theory behind the concept of rendezvous and hence inferred all the possible synchronizations patterns that govern its behaviour. By combining the patterns, we deduce 36 generic

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007

synchronization rules that can be associated with the execution of a rendezvous. Afterwards, we enrich the Time Petri Net model with the rendezvous mechanism to define a formal framework, called *Time Petri Nets with rendezvous* (RTPN). A sub class of RTPN is then introduced called *time workflow nets with rendezvous* RTWFN. The latter provides a large panel of concise synchronization patterns that can deal with any complex synchronization scheme in timed workflow systems. Further work will lead us to investigate a methodology for the analysis of the quantitative and the qualitative properties of the RTWFN model, based on state class graph construction and model-checking.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Abdelli, A., 2015. Towards a general model to handle multi-enabledness in time petri nets. Formalisms Reuse Syst. Integr., 103–131

Alfonso-Cendn, J. et al., 2016. Neurocomputing 176, 91–97.

Atluri, Vijayalakshmi, Huang, Wei-kuang, 2000. A Petri net based safety analysis of workflow authorization models. J. Comput. Secur. 8 (2/3), 209–240.

Bæk, J.J., Bisgaard, L.K., Van der Aalst, W.L.M.P., 2008. From task descriptions via colored Petri nets towards an implementation of a new electronic patient record workflow system. STTT 10 (1), 15–28.

Barkaoui, K., Boucheneb, H., Hicheur, A., 2008. Modelling and analysis of time-constrained flexible workflows with time recursive ECATNets. WS-FM. pp. 19–36.

Barreto, F.M., Julia, S., 2017. Modeling of video games using workflow nets and state graphs. SBES, 261–266.

Bertolini, C., Liu, Z., Srba, J., 2012. Verification of timed healthcare workflows using component timed-arc petri nets. FHIES. pp. 19–36.

Bettini, C., Wang, X.S., Jajodia, S., 2002. Temporal reasoning in workflow systems. Distrib. Parallel Databases 11 (3), 269–306.

Boucheneb, H., Barkaoui, K., 2014. Partial order reduction for checking soundness of time workflow nets. Inf. Sci. 282, 261–276.

Boucheneb, H., Lime, D., Roux, O.H., 2013. On multi-enabledness in time petri nets. Petri Nets, 130–149.

Boukhedouma, S., Alimazighi, Z., Oussalah, M., 2017. Adaptation and evolution frameworks for service based inter-organizational workflows. IJEBR 13 (2), 28–57.

Cicirelli, F., Furfaro, A., Nigro, L., 2013. Using time stream Petri nets for workflow modelling analysis and enactment. Simulation 89 (1), 68–86.

Combi, C., Posenato, R., 2009. Controllability in temporal conceptual workflow schemata. BPM, 64–79.

del Foyo, P.M.G., Silva, J.R., 2008. Using time Petri nets for modelling and verification of timed constrained workflow systems. ABCM Symp. Ser. Mechatron. 3, 471–478.

Eder, J., Panagos, E., Rabinovich, M., 2013. Time constraints in workflow systems. Semin. Contrib. Inf. Syst. Eng., 191–205

Gunter, E.L., Yasmeen, A., Gunter, C.A., Nguyen, A., 2009. Specifying and analyzing workflows for automated identification and data capture. HICSS. pp. 1–11.

Hamdani, Abdia, Abdelli, Abdelkrim, 2017. Time Petri Net with Rendezvous. IEEE Codit, Barcelona, Spain.

Hamdani, Abdia, Abdelli, Abdelkrim, 2018. Using the RTPN model for the modelling of complex Workflow systems. In: ICAASE Conference, CEUR Proceedings.

James, F. Allen, 1983. Maintaining knowledge about temporal intervals. Commun. ACM 26 (11), 832–843.

Karhof, A. et al., 2016. On the de-facto standard of event-driven process chains: reviewing epc implementations in process modelling tools. Modellierung, 77–92.

Khoshafian, S., Buckiewicz, M., 1998. Groupware et workflow. Dnod ed. p. 297 (ISBN 9782225829260).

Lanz, A., Weber, B., Reichert, M., 2010. Workflow time patterns for process-aware information systems. BMMDS/EMMSAD, 94–107.

Li, J., Fan, Y., Zhou, M., 2003. Timing constraint workflow nets for workflow analysis. IEEE Trans. Syst. Man Cybern. Part A 33 (2), 179–193.

Marjanovic, O., Orlowska, M.E., 1999. On modeling and verification of temporal constraints in production workflows. Knowl. Inf. Syst. 1 (2), 157–192.

Owezarski, P., Boyer, M., 2010. Modeling of multimedia architectures: the case of videoconferencing with guaranteed quality of service. ISTE, pp. 501–525.

Rusinkiewicz, M., Sheth, A.P., Karabatis, G., 1991. Specifying interdatabase dependencies in a multidatabase environment. IEEE Comput. 24 (12), 46–53.

Russell, N., ter Hofstede, A.H.M., 2009. newYAWL: towards workflow 2.0. Trans. Petri Nets Other Models Concurrency 2, 79–97.

Salimifard, K., Hosseini, S.Y., Moradi, M.S., 2013. Improving emergency department processes using coloured petri nets. PNSE+ModPE, 335–349.

Thabet Kotb, Y., Badreddin, E., 2005. Synchronization among activities in a workflow using extended workflow petri nets. CEC. pp. 548–551.

van der Aalst, W.M.P., 1998. The application of petri nets to workflow management. J. Circuits Syst. Comput. 8 (1), 21–66.

Van der Aalst, W.M.P., ter Hofstede, A.H.M., 2005. YAWL: yet another workflow language. Inf. Syst. 30 (4), 245–275.

van der Aalst, W.M.P., Weske, M., 2013. Reflections on a decade of interorganizational workflow research. Semin. Contrib. Inf. Syst. Eng., 307–313

Wang, H., Zeng, Q., 2008. Modeling and analysis for workflow constrained by resources and nondetermined time: an approach based on petri nets. IEEE Trans. Syst. Man Cybern. Part A 38 (4), 802–817.

Yuyue, Du., Jiang, Changjun, 2003. Towards a workflow model of real-time cooperative systems. ICFEM, 452–470.

Zhang, W.W., Beaubouef, T., Ye, H., 2012. State chart: a visual language for workflow specification. Int. J. Comput. Theor. Eng. 4 (6), 921–925.

Please cite this article as: A. Hamdani and A. Abdelli, Towards modelling and analyzing timed workflow systems with complex synchronizations, Journal of King Saud University – Computer and Information Sciences, https://doi.org/10.1016/j.jksuci.2019.08.007