On Modelling and Analyzing Composite Resources' Consumption Cycles using Time Petri-Nets

Amel Benna[1], Fatma Masmoudi[2], Mohamed Sellami[3], Zakaria Maamar[4] and Rachid Hadjidj[5]
[1]*CERIST, Algiers, Algeria*
[2]*Prince Sattam Bin Abdulaziz University, Alkharj, K.S.A.*
[3]*Samovar, Télécom SudParis, Institut Polytechnique de Paris, France*
[4]*Zayed University, Dubai, U.A.E.*
[5]*Department of Computer Science, ABMMC, Doha, Qatar*

Keywords: Composition, Consumption, Petri Net, Resource.

Abstract: ICT community cornerstones (IoT in particular) gain competitive advantage from using physical resources. This paper adopts Time Petri-Nets (TPNs) to model and analyze the consumption cycles of composite resources. These resources consist of primitive, and even other composite, resources that are associated with consumption properties and could be subject to disruptions. These properties are specialized into unlimited, shareable, limited, limited-but-renewable, and non-shareable, and could impact the availability of resources. This impact becomes a concern when disruptions suspend ongoing consumption cycles to make room for the unplanned consumptions. Resuming the suspended consumption cycles depends on the resources' consumption properties. To ensure correct modeling and analysis of consumption cycles, whether disrupted or not, TPNs are adopted to verify that composite resources are reachable, bound, fair, and live.

1 INTRODUCTION

There is a consensus in the Information and Communication Technology (ICT) community that the Internet of Things (IoT) is helping achieve Mark Weiser's vision about ubiquitous computing that is "*...The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*" (Weiser, 1991). Whether visible or invisible, today's things like ambient sensors and smart watches are used anytime, anywhere producing massive amount of data about people and other "things" like vegetable freshness in a transit facility, and patients' vitals in an ICU. It is predicted that the total economic impact of IoT will reach between $3.9 trillion and $11.1 trillion per year by the year 2025 (DZone, 2021).

To sustain the IoT economic impact mentioned above, particular measures should be taken to address things' processing, storage, and communication limitations. Indeed, not all things (e.g., ambient sensors) are embedded with powerful processing capabilities and not all things (e.g., smart watches) can store large amount of data nor communicate data quickly. In the literature, some measures consist of coupling IoT applications to cloud computing so, that, appropriate processing, storage, and communication resources are made available based on these applications' functional and non-functional requirements (Li et al., 2020; Ren et al., 2017). In a previous work (Maamar et al., 2021), we associated cloud resources with consumption properties specialized into unlimited, limited, limited-but-renewable, shareable, and non-shareable, allowing a better control over these resources in terms of availability, consistency, and accountability. Each property is modeled, with Finite State Machine (FSM), as cycle that tracks the progress of consuming a resource by an application.

While IoT application/cloud coupling seems the way to move forward, many critical concerns are barely touched upon like first, guaranteeing the continuous availability of cloud resources following the occurrence of disruptive events (e.g., urgent upgrades to counter cyber-attacks and urgent demands to execute last-minute requests) and second, composing re-

Benna, A., Masmoudi, F., Sellami, M., Maamar, Z. and Hadjidj, R.
On Modelling and Analyzing Composite Resources' Consumption Cycles using Time Petri-Nets.
DOI: 10.5220/0010971900003176
In *Proceedings of the 17th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2022)*, pages 243-250
ISBN: 978-989-758-568-5; ISSN: 2184-4895

sources together to satisfy IoT applications' requirements. We focus in this paper on resource composition by specializing resources into primitive (pr) and composite (cr) and then, modeling and verifying composite resources' consumption cycles using Time Petri-Nets (TPNs) (Merlin and Farber, 1976). Compared to what we did in (Maamar et al., 2021) when modeling consumption cycles of separate prs using FSM, this modeling technique falls short of capturing the composition of these resources as well as their concurrent consumption. TPNs are widely used in the ICT community for representing the structure and timing of processes and distributed systems in a more realistic way.

For illustration purposes, let us assume a composite resource that calls for some primitive resources. For a successful composite resource modeling and verification, we raise many questions for instance, how to ensure the consistency of this composite resource's consumption cycle that is made of separate but collaborating primitive resources' consumption cycles, how to verify the correctness of the composite resource's consumption cycle, and how to track the overall consumption of the composite resource despite the disruptions that could impact its primitive resources. In this paper, we present a TPN-based approach for modeling and verifying composite resources' consumption cycles. The rest of this paper is organized as follows. Section 2 presents a motivating example. Section 3 briefly defines TPNs and primitive resources. How these resources are modeled in TPNs is presented in Section 4. Contrarily, Section 5 is dedicated to modeling and analysing composite resources in TPNs. Prior to concluding in Section 7, we discuss the simulation of the modeled TPNs and the verification of some associated properties in Section 6.

2 MOTIVATING EXAMPLE

Our motivating example, yet, simple illustrates how resources could be composed when completing jobs. A surveillance organization is in charge of monitoring different parts of the city using cameras broadcasting real-time images to a private cloud for storage (Figure. 1). To comply with local authorities' regulations, the cameras' recordings must be backed up for at least a year in a highly durable manner. For this purpose, the organization deploys the data storage on Amazon Web Services (AWS) public cloud. This consists of compressing the recordings using an in house tool (app_1) that consumes CPU thanks to a private cloud-based virtual machine (pr_1). During compression, the recordings are tagged with details like timestamp and location before saving them on the AWS cloud offering the necessary storage resources (pr_2 and pr_3). During each backup request, a serverless solution is configured by calling a Lambda function[1] (f_1) that saves the recordings in an archive storage (pr_2) and the associated details in a NoSQL database (pr_3) on the public cloud.

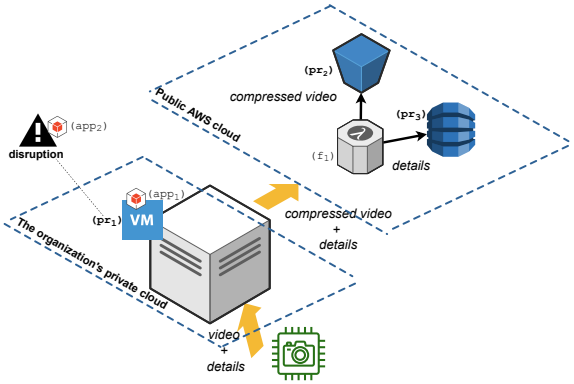

Figure 1: Video surveillance backup in action.

During the consumption of the composite resource cr={pr_1, pr_2, pr_3}, many disruptive events could happen. For instance, the full capacity of pr_1 was needed to compress app_1's recordings but, then, an urgent request to scan app_2's recordings is received suspending the compression. Handling the disruption that pr_1 faces along with ensuring the consumption continuity of other resources part of cr (i.e., pr_2 and pr_3) might vary depending on these resources' characteristics like availability and shareability. With respect to pr_1's disruption, many questions are raised as per the following cases: (*i*) since pr_3 is consumed for storing the compressed recordings' details and other data transmitted by app_2, will pr_3 remain available for consumption by app_2 despite the disruption impacting pr_1? (*ii*) since pr_2 is only consumed during the backup of recordings by f_1, will the disruption of pr_1 impact pr_2?, and (*iii*) assuming that the consumption of pr_1 happens sequentially with pr_2 and pr_3, what will be impact of pr_1's disruption , will pr_2 and pr_3 remain available for consumption, and is this impact the same in case the consumption of pr_1 happens concurrently with pr_2, and pr_3? In this paper, not only we address these questions but also other cases by modeling and analysing resources' consumption cycles using TPNs ensuring the completeness of these cycles, for example.

[1] AWS Lambda is an event-driven, serverless computing platform.

3 BACKGROUND

This section briefly presents TPNs, and, then primitive resources' consumption properties and cycles.

3.1 TPN in Brief

Petri Nets (PN)s are widely used for the study and analysis of concurrent systems (Peterson, 1977). PN is a directed bipartite graph that has 2 types of vertices : place (which can contain tokens) and transition. Distribution of tokens over the places represent a configuration of the net called a marking. Arcs run from a place to a transition or vice versa, never between places or between transitions. A transition is enabled if each place connected to it as input contains at least a number of tokens greater or equal to the weight of corresponding input arc. Time Petri Nets (TPNs) (Merlin and Farber, 1976) extend PNs to enforce the duration of a system's activities. Different variations of TPNs exist with focus in this paper on those with bounded data variables, called Interpreted Time Petri Nets (ITPN) (Hadjidj and Boucheneb, 2013), to model resources' consumption cycles (Section 3.2). In an ITPN, each transition has, on top of a time interval, a guard and a set of update operations on user defined data variables and the number of token. An enabled transition can be fired if its associated guard is true in the current state of the ITPN model. Once a transition is fired, its updates are applied.

A number of TPN software tools exist[2] allowing for instance, to evaluate some performance characteristics and formally verify a TPN against a set of properties (Berthomieu and Diaz, 1991). In our work, we use Real Time Studio (RT-Studio[3]). It allows both guards and update actions on transitions and the use of variables, that other TPN tools do not support (e.g., TAPAAL).

3.2 Primitive Resources in Brief

As stated earlier, jobs consume resources with respect to one of the 5 consumption properties (Maamar et al., 2021): unlimited (ul), shareable (s), limited (l), limited but-renewable (lr), and non-shareable (ns). The first two are not explained due to their simplicity.

- Limited means that the consumption of a resource is restricted to a particular capacity and/or period of time.
- Limited-but-renewable means that the consumption of a resource continues to happen since the (initial) agreed-upon capacity has been increased and/or the (initial) agreed-upon period of time has been extended.
- Non-shareable means that the concurrent consumption of a resource must be coordinated (e.g., one at a time).

Each consumption property is associated with a consumption cycle (*cc*) depicting the current state of a primitive resource. For instance, the consumption cycle of a limited resource pr is represented as follows cc_1: prepared $\xrightarrow{consumptionApproval}$ consumed $\xrightarrow{consumptionUpdate}$ done $\xrightarrow{consumptionCompletion}$ withdrawn, where prepared and consumed are examples of states and consumptionApproval and consumptionUpdate are examples of transitions linking these states together. In preparation for modeling and verifying composite resources' consumption cycles in ITPNs (TPN for short), we briefly discuss next how primitive resources are modeled in TPNs. We also show how to handle suspensions due to disruptions.

4 MODELING PRIMITIVE RESOURCES USING TPNS

We identify 6 places and 9 transitions, connecting these places together, that would constitute a primitive resource's non-disrupted consumption cycle in a TPN. The places are prepared (the resource is created), consumed (the resource is bound by a consumer due to the ongoing consumption), done (the resource is unbound by a consumer after successful consumption), locked (the resource is booked for a consumer in preparation for its consumption), unlocked (the resource is released by a consumer after its consumption), and withdrawn (the resource ceases to exit after unbinding all consumers from the resource).

For illustration purposes, we consider the TPN-based consumption cycles without/with disruption for l, lr, and s resources. In the remaining illustrative figures of a TPN-based consumption cycles, blue places and transitions correspond to the non-disrupted part. Contrarily, gray places and transitions correspond to this cycle's disrupted part.

4.1 Limited Resource-model

A l resource's consumption cycles can be impacted by its capacity/and or availability. Figure. 2 illus-

[2] www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html

[3] sites.google.com/view/rachid-hadjidj/rt-studio.

Table 1: Some conditions and actions on transitions for a l resource.

Transition	Condition	Action
consApproval	job's time-interval/capacity falls into resource's time-interval/capacity	—
consUpdate	no disruption or all disruptive jobs completed their consumption (consumed = 1) and enough capacity/time	update resource's capacity/time interval
consUpdateS	consumed > 1 (at least one disruptive job did not complete its consumption)	update resource's capacity/time interval
consRejection	job's time interval/capacity do not falls into resource's time interval/capacity	—
suspension	disrupting job's time interval/capacity falls into resource's time interval/capacity	update resource's capacity/time interval

trates these cycles. A transition consRejection from prepared to withdrawn is enabled if the required capacity to consume the job exceeds the capacity of the resource or the job's time interval does not fall into the resource's time interval.

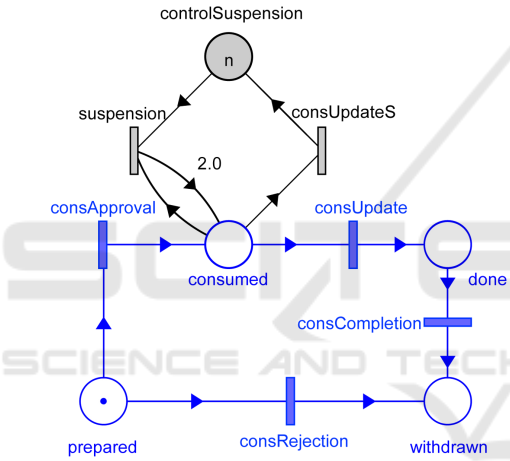

Figure 2: Consumption cycle for a l primitive resource.

During the consumption cycle of this resource, disruptions could occur initiating a disrupted consumption cycle (*d-cc*). For instance, $d\text{-}cc_{\mathtt{l}}$: consumed $\xrightarrow{suspension}$ consumed $\xrightarrow{consUpdateS}$ controlSuspension . The suspension transition initiates the disruptive consumption cycle by putting on-hold the consumption of the active job making room for the disrupting job[4]. Since the disrupting job can in turn be suspended and so on, we assume that firing consUpdateS transition concludes the consumption of the last job that triggered a disruption. The suspension transition is disabled when the number of allowed suspensions is reached while the disruptive jobs have not completed their consumptions. To control this number of allowed suspensions, we define a place called controlSuspension[5]. For instance, in Figure. 2, if n is set to three, a new job that suspended the initial job can in turn be suspended to initiate a new job that also can be suspended.

To resume the initial/first suspended job, the transition consUpdate from consumed to done needs to satisfy the condition that a resource still has some capacity and/or time left for the suspended job and all the disruptive jobs completed their consumptions. Otherwise, this job's consumption cycle remains suspended and no transition is enabled. In Table. 1, we associate transitions in Figure. 2 with the necessary firing conditions (guards) and actions to take in response to this firing.

4.2 Limited-but-Renewable Resource-model

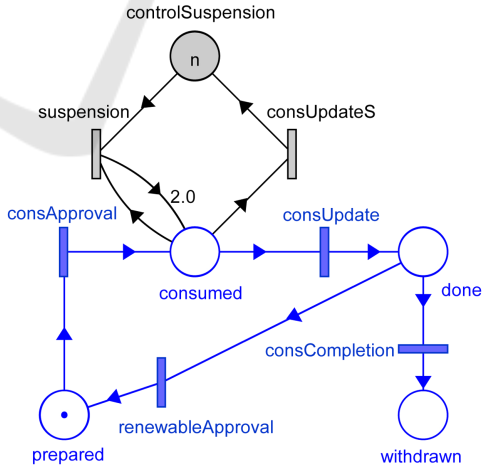

Figure 3: Consumption cycle for a lr primitive resource.

For a lr resource in term of either capacity or time, its non-disrupted consumption cycle (*cc*) is represented as follows: $cc_{\mathtt{lr}}$: prepared $\xrightarrow{consApproval}$

[4]Firing the suspension transition leads to accumulating a new number of tokens at the output place, i.e 2 tokens are added to the consumed place waiting for consumption.

[5]where n refers to the number of allowed disruptions

consumed $\xrightarrow{consUpdate}$ done $\xrightarrow{renewableApproval}$ prepared. And, its TPN-based consumption cycle is illustrated in blue in Figure. 3. In this cycle, the transition from done to prepared allows a resource to be regenerated for another round of consumption, if deemed necessary. Otherwise, the consumption cycle takes on withdrawn state. For a lr resource, in addition to conditions and actions on consApproval, consUpdate, and consUpdateS transitions described in Table. 1, firing the renewableApproval transition extends the resource's capacity and/or time interval. The disruptive cycle for a lr resource follows the same disruptive cycle of a l resource. However, as lr resource is extensible, a resumption's failure after disruption is less recurrent.

4.3 Shareable Resource-model

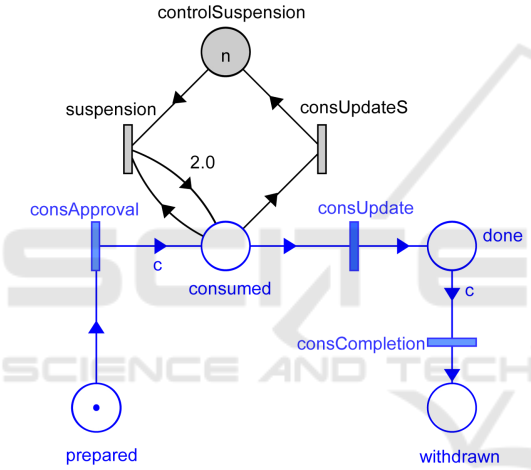

Figure 4: Consumption cycle for a s primitive resource.

Modeling a primitive resource's consumption cycle using a TPN depends on its consumption property and potential occurrence of disruptions.

A s resource allows its concurrent consumption by many jobs. In this case, at least 2 consumers that require the same resource are simultaneously activated. Figure. 4 illustrates the s resource's TPN-based consumption cycles. The transition from prepared to consumed is satisfied if 2 or several consumers that require the same resource are simultaneously activated. When no pending jobs consuming this resource exist, the transition from done to withdrawn is enabled. If a disruption occurs, we assume that all jobs are suspended making room for the disrupting job. Table. 2 associates transitions in Figure. 4 with the necessary firing conditions (guards) and actions to take in response to this firing.

5 MODELING AND ANALYZING COMPOSITE RESOURCES USING TPNs

Setting-up the consumption of a composite resource refers to a group of sequential and/or concurrent primitive resources that could be disrupted impacting this consumption. Using Backus-Naur Form notation (Knuth, 1964), we specify a composite resource with Listing 1:

Listing 1: BNF representation of composite resources.

⟨cr⟩ ::= ⟨resource⟩ ⟨chronology⟩ ⟨resource⟩
⟨resource⟩ ::= ⟨pr⟩ | (⟨cr⟩)
⟨pr⟩ ::= (⟨name⟩ , ⟨property⟩)
⟨property⟩ ::= ul | l | s | ns | lr
⟨chronology⟩ ::= *sequential* | *concurrent*
⟨name⟩ ::= STRING | INT | ⟨name⟩STRING | ⟨name⟩INT

Applying the notation above to the motivating example results into (pr_1,l) *sequential* ((pr_2,lr) *concurrent* (pr_3,s)) schema. Here, pr_1 has a limited CPU capacity that is consumed during the compression of recordings. This happens sequentially with pr_2 and pr_3 both consumed concurrently. pr_2 has a limited-but-renewable storage capacity/time and pr_3 has a shareable but limited storage capacity. To model a composition of resources we develop first, a TPN for the controller that will oversee the composition progress and second, a TPN for each primitive resource in this composition according to its consumption property. Figure. 5 is the controller's TPN that includes 1 place, controller, with m[6] tokens and 2 transitions, launch! and success?, that initiate and receive the end of the consumption of primitive resources, respectively.

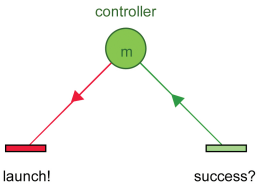

Figure 5: Controller overseeing the consumption of primitive resources.

Regarding launch and success transitions, they synchronize the different TPNs of the composed primitive resource. For each primitive resource's TPN, launch? transition, used to receive messages from the controller to initiate the resource's consumption,

[6]Varies according to the number of resources that are consumed concurrently.

Table 2: Some conditions and actions on transitions for a s resource.

Transition	Condition	Action
consApproval	c$^{*}>$1 and consumers job's time-interval/capacity falls into resource's time-interval/capacity	–
consUpdate	controlSuspension$= n^{**}$, no disruption or all disruptive jobs completed their consumption and enough capacity/time	update resource's capacity/time-interval
consUpdateS	controlSuspension $< n$ (at least one disruptive job did not complete its consumption)	update resource's capacity/time-interval
consCompletion	no pending consumer consuming this resource exists	–

* c refers to the number of concurrent consumers'
** n refers to the number of allowed disruptions.

is connected to the prepared place in this TPN, and success! transition, used to send messages to the controller, is connected to both done and withdrawn places in this TPN as well. These messages received by the controller can inform either about the successful consumption completion and unbinding of resource by the its consumer or about unbinding of all its consumers.

Depending on the consumption properties and consumption chronology, the occurrence of a disruption targeting a primitive resource (could be many) could impact the consumption of the remaining primitive resources. Table. 3 summarizes this impact on the composition of pr as per the occurrence or not of the consumption resumption of the disrupted resource.

Figure. 6 is a simulated TPN, using RT-Studio, of the composite resource described in the motivating example. The following discusses the disruption impact on each primitive resource per type of consumption, sequential *versus* concurrent.

- pr_1 disruption during sequential consumption: the primitive resource consumed after pr_1 might be impacted due to pr_1's disruption. Indeed, an unsuccessful resumption could make the consumption of both pr_2 and pr_3 fail and thus, the composite resource consumption too. The consumption cycle of pr_1 after firing launch? is described as follows: $pr_1.cc_{lr}$: pr_1.prepared $\xrightarrow{pr_1.consApproval}$ pr_1.consumed $\xrightarrow{pr_1.suspension}$ pr_1.suspended $\xrightarrow{pr_1.consumed}$ pr_1.consumed $\xrightarrow{pr_1.consUpdateS}$ pr_1.controlSuspension. The transition pr_1.consUpdate remains disabled and only a new disruptive job that falls into the time interval and remaining capacity of the resource can be consumed. Otherwise all transitions are disabled. It is worth noting that a disrupted unlimited primitive resource has no impact on the consumption of other primitive resources.
- pr_2/pr_3 disruption during concurrent consumption: the respective consumption of pr_2 and pr_3 happens independently from each other and the disruption targets either pr_2 or pr_3. Assuming that the consumption's resumption of pr_2 is unsuccessful leading to its failure, pr_3 would complete its consumption and *vice-versa*. After firing launch? the consumption cycles for each primitive resource are as follows:
  - $pr_1.cc_l$: pr_1.prepared $\xrightarrow{pr_1.consApproval}$ pr_1.consumed $\xrightarrow{pr_1.consUpdate}$ pr_1.done, where pr_1 is consumed by *app1*;
  - $pr_2.cc_{lr}$: pr_2.prepared $\xrightarrow{pr_2.consApproval}$ pr_2.consumed $\xrightarrow{pr_2.consUpdate}$ pr_2.done $\xrightarrow{pr_2.renewableApproval}$ pr_2.prepared $\xrightarrow{pr_2.consApproval}$ pr_2.consumed $\xrightarrow{pr_2.suspension}$ pr_2.consumed $\xrightarrow{pr_2.consUpdateS}$ pr_2.controlSuspension, where pr_2 is consumed by *app1* with an extended capacity/time;
  - $pr_3.cc_s$: pr_3.prepared $\xrightarrow{pr_3.consApproval}$ pr_3.consumed $\xrightarrow{pr_3.consUpdate}$ pr_3.consumed $\xrightarrow{pr_3.consUpdate}$ pr_3.done, where pr_3 is consumed by *app1* and *app2*.

In summary, a failed resumption of a l/lr/s $pr_i \subset$ cr leads to the failure of the consumption of all the primitive resources, pr_j, that are expected to be consumed sequentially after a peer, for instance pr_i. Otherwise, when the primitive resources are consumed concurrently and regardless of their consumption properties, a disruption does not have any impact on other resources. The consumption continuity of the composite resource is maintained but not the composition model; ((pr_1,l) *sequential* (pr_3,s)) instead of ((pr_1,l) *sequential* ((pr_2,lr) concurrent (pr_3,s))) when pr_2 is disturbed, for example. Moreover, even if an ul $pr_j \subset$ cr will always resume after a disruption, it could have an impact on the remaining consumption cycle of l/lr/s/ns $pr_i \subset$ cr if the consumption time interval of pr_i is exceeded after the consumption of pr_j. On another side, a ns $pr_k \subset$ cr that remains locked has the same impact as a non resumption of a l/lr/s resource after its disruption.

Table 3: Impact of disruption on composition of resources as per chronology consumption and pr consumption properties.

Consumption chronology	**Consumption properties of the disrupted resources**		
	ul	**l/s/lr**	
		Successful resumption of all disrupted pr	**Unsuccessful resumption of one of the disrupted pr.**
Sequential	successful consumption	successful consumption	consumption failure of remaining resources.
Concurrent			consumption of remaining resources but without following/respecting the `cr` chronology.
Sequential Concurrent			consumption of remaining concurrent resources but without following/ respecting the `cr` chronology.

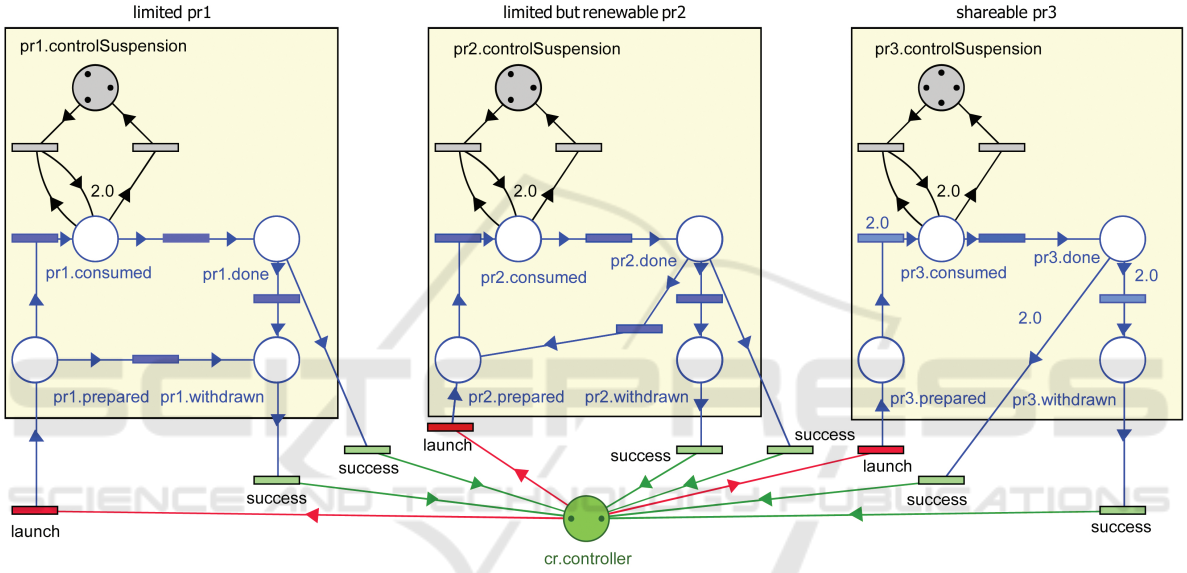

Figure 6: Illustration of a simulated composition of 3 primitives resources.

6 SIMULATING SOME TPNS

We discuss the simulation of some `TPNs` that were used for modeling primitive and composite resources. Along this simulation some TPNs' properties were verified. An online demo is available at https://youtu.be/aHPJl9Q7Xko. Verifying some important `PN` properties like `reachability`, `boundedness`, `fairness`, and `liveness` allowed to assess some good behaviors of our models.

- `Reachability` assesses if a marking M_n is reachable from the initial marking M_0 via a sequence of transitions that transforms M_0 into M_n.
- `Boundedness` states that the number of tokens in each place of the `TPN` does not exceed a finite number k for any marking reachable from the initial marking M_0. For instance, we can verify that `lr` resource `TPN` is bounded in a way that each place in the consumption cycle holds a limited number of tokens at any time.
- `Fairness` states that every transition is expected to be fired frequently/infinitely in any firing sequence. This is satisfied in our case since transitions are enabled over and over again.
- `Liveness` states that for every reachable marking it is possible to fire all transitions at least once by some firing sequence, denoting the absence of deadlocks. In our case, the `lr` pr_2 cycle is live since there is no deadlock during the execution.

For the analyse and verification of our TPN model using model checking for instance, we first need to represent the behavior of a system as a state graph, then specify properties of interest in a temporal logic , and finally explore the state graph to determine whether these properties hold or not. `Reachability` analysis of our `TPNs` relies on the construction of so-

called Strong State Class Graph (SSCG) (Berthomieu and Vernadat, 2003). RT-Studio allows to construct several abstractions[7] of state spaces for TPNs suitable to verify Reachability, but also linear and branching properties.

We used RT-Studio tool to compute the state space graph to preserve Linear Temporal Logic (LTL) and branching properties (CTL*) of the simulated TPN model shown in Figure. 6. This TPN model components in line with our motivation example: cameras broadcasting real-time images to a private cloud for storage purpose consists of one instance of : the controller model (cr), the limited resource model (pr_1), the limited but renewable resource model (pr_2), and the shareable resource model (pr_3). These components are synchronized on transitions to simulate (pr_1,l) *sequential* ((pr_2,lr) *concurrent* (pr_3,s)) schema. An initial capacity and time interval are associated with each resource. The controlSuspension admits 3 suspensions for both pr_1 and pr_2 while it allows 4 suspensions for pr_3. The latter is shared between *app1* (englobing the Lambda function) and *app2* which requires that pr_3's number of concurrent customers restricted to only 2. The generated SSCG can be used as starting point in a refinement process to generate Atomic State Class Graph (ASCG) allowing to preserve branching properties (CTL*) of the TPN model.

7 CONCLUSION

This paper examines the modeling and analysis of composite resources consumption consisting of several primitive (and even composite) resources arranged sequentially and/or concurrently. During resources consumption, disruptions could arise leading to potential concerns like the suspension of ongoing business operations, which is not a "pleasant" situation for organizations. To mitigate such a situation, we resorted to using TPN to capture composite resources' consumption cycles according to their primitive resources' consumption properties specialised into unlimited, shareable, limited, limited-but-renewable, and non-shareable. Along this TPNs modeling, we used RT-Studio to verify composite resources' reachability, boundedness, fairness, and liveness properties. In term of future work, we would like to ensure the correctness of composite resources' consumption cycles through model checking.

[7] SSCG and ASCG for interval characterization and Concrete State Zone Graph (CSZG) for clock characterization.

REFERENCES

Berthomieu, B. and Diaz, M. (1991). Modeling and Verification of Time Dependent Systems using Time Petri Nets. *IEEE Transactions on Software Engineering*, 17(3):259–273.

Berthomieu, B. and Vernadat, F. (2003). State class constructions for branching analysis of time petri nets. In Garavel, H. and Hatcliff, J., editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 442–457, Berlin, Heidelberg. Springer.

DZone (https://dzone.com/guides/iot-applications-protocols-and-best-practices, 2017 (visited in March 2021)). IoT: Application, Protocols, and Best Practices. Technical report, DZone.

Hadjidj, R. and Boucheneb, H. (2013). RT-Studio: A Tool for Modular Design and Analysis of Realtime Systems Using Interpreted Time Petri Nets. In *Joint Proceedings of PNSE'2013 and ModBE'2013*, volume 989 of *CEUR Workshop Proceedings*, pages 247–254, Milan, Italy. CEUR-WS.org.

Knuth, D. E. (1964). Backus Normal Form vs. Backus Naur Form. *Communications of the ACM*, 7(12):735–736.

Li, Z., Wen, L., Liu, J., Jia, Q., Che, C., Shi, C., and Cai, H. (2020). Fog and Cloud Computing Assisted IoT Model Based Personal Emergency Monitoring and Diseases Prediction Services. *Computing and Informatics*, 39(1):5–27.

Maamar, Z., Sellami, M., and Masmoudi, F. (2021). A Transactional Approach to Enforce Resource Availabilities: Application to the Cloud. In *Proceedings of RCIS'2021*, volume 415 of *Lecture Notes in Business Information Processing*, pages 249–264, Cyprus (online). Springer.

Merlin, P. and Farber, D. (1976). Recoverability of Communication Protocols - Implications of a Theoretical Study. *IEEE Transactions on Communications*, 24(9):1036–1043.

Peterson, J. (1977). Petri Nets. *ACM Computing Surveys*, 9(3):223–252.

Ren, J., Guo, H., Xu, C., and Zhang, Y. (2017). Serving at the Edge: A Scalable IoT Architecture Based on Transparent Computing. *IEEE Network*, 31(5):96–105.

Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 265(3):66–75.