Formal Modelling and Verification of Cloud Resource Allocation in Business Processes

Ikram Garfatta[1(✉)], Kais Klai[2], Mohamed Graiet[3], and Walid Gaaloul[4]

[1] Faculty of Sciences, University of Monastir, Monastir, Tunisia
ikram.garfatta@gmail.com
[2] LIPN, CNRS UMR 7030, University of Paris 13, Villetaneuse, France
kais.klai@lipn.univ-paris13.fr
[3] ISIMM, Universiy of Monastir, Monastir, Tunisia
mohamed.graiet@imag.fr
[4] Telecom SudParis, UMR 5157 Samovar, University of Paris-Saclay, Paris, France
walid.gaaloul@telecom-sudparis.eu

Abstract. Cloud environments have been increasingly used by companies for deploying and executing business processes to enhance their performance while lowering the operating cost. Nevertheless, the combination of business processes and Cloud environments is a field that needs to be further studied since it lacks an explicit and formal description of the resource perspective in the existing business processes and especially of Cloud-related properties, namely vertical/horizontal elasticity. Therefore, this field cannot yet fully benefit of what Cloud environments can offer. Besides the lack in formalization, there is also a need for a verification method to check the correctness of allocations. In fact, without formal verification, the designer can easily model erroneous allocations which lead to runtime errors if left untreated at design-time. In this work, we address the above shortcomings by proposing a formal model for the Cloud resource perspective in business processes using the Coloured Petri net formalism, which can be used to check the correctness of Cloud resource allocation at design-time.

Keywords: Business Process Models · Formal verification
Cloud resources · Elasticity · Coloured Petri net

1 Introduction

Finding the right compromise between the best performance and the lowest cost has always been sought by enterprises that deal with Business Process Management Systems. Combining BPM with Cloud environments has proved to be of great benefit for such enterprises in their quest, especially considering that properties such as elasticity and shareability are at the very essence of the definition of Cloud Computing. Researchers have been investigating the possibilities

H. Panetto et al. (Eds.): OTM 2018 Conferences, LNCS 11229, pp. 552–567, 2018.
https://doi.org/10.1007/978-3-030-02610-3_31

of hosting entire Business Processes in the Cloud, going to the extent of proposing the Business Process as a Service (BPaaS) as a new service model that takes part in the Cloud computing paradigm [23].

On the other hand, the resource perspective in BPM has caught the interest of researchers since many activities may require certain resources for their proper execution. Many studies have dealt with the human resources management [6,7], but non-human resources, especially Cloud resources, were seldom considered.

Taking advantage of Cloud resources, however, is not only achieved by implementing the whole Business Process on the Cloud. A Business Process (BP) can use resources managed by a Cloud provider without having to be part of that Cloud environment. The communication between the BP and the Cloud provider has to be well defined for the interaction to be flawless. In an effort to outline this relationship between the two domains, a first attempt was made in [10]. The idea was to extend the Business Process Model to include configurable Cloud resources allocation. The extension comprises three operators essentially, which are designed to define the way the Cloud provider assigns the resources to the process activities taking into account both the elasticity and shareability concepts, and emphasizing on the type and capacity of the required resources. These proposed operators, though designed to deal with configurable allocations, can also be used for non-configurable ones if we consider them after configuration. Despite the visual representation it offers, this proposition remains unverified and lacking formalism. In fact, the interaction between the Business Process and the Cloud resources provider, however intuitive it may seem, is a very intricate task. Thus, using the operators in [10] can easily result into faulty allocations. For instance, two activities may request the same resource to consume at the same time with a total required capacity that exceeds the capacity the resource can provide. This allocation, though representable using the operators in [10], is a situation that ought to be avoided.

To be able to avoid unsound allocations, a verification method needs to be applied at design time to insure a correct behaviour at runtime. To achieve this goal, we propose a formal model to represent the Cloud resource allocation mechanism in BP models, using Coloured Petri Nets [12]. Having formalized the allocation aspect, we are then able to verify essential structural and behavioural properties by formally analysing and verifying the model.

The remainder of this paper is organized as follows: the related work is presented in Sect. 2. Section 3 introduces basic concepts used in the other sections and our proposed model is detailed in Sect. 4. Results for a case study are presented in Sect. 5. Finally, we conclude and provide insights for future work in Sect. 6.

2 Related Work

Business processes usually include activities that need resources to carry out their expected tasks. The required resources can be classified into two categories: human resources which are basically the workforce needed for the execution of

the activities, and non-human resources. In literature, more attention has been drawn to the first category, whereas only a few researches have dealt with the second one. Our work focuses on this disregarded category, and more precisely on Cloud resources.

The representation of the resource perspective in the context of business processes has been the subject of some works in literature. For example, in [18], the authors have extended the BPMN 2.0 metamodel in order to model and visualize the resource perspective requirements. This extension was however proved to be against the workflow resource patterns [17]. This work was then followed by the proposition of an approach [8,19] that enables the implementation of the requirements of the resource perspective in extended BPMN models and resource structure models into BPEL definitions in an attempt to provide a support to the lifecycle (i.e., definition, implementation, verification and validation) of the resource perspective requirements in the development of PAISs based on WfMSs.

Petri net has been prominently present in research dealing with the verification of the structural aspect in business processes. This formalism was used in [1] to define a number of workflow patterns that formalize the control flow in business processes. Other works used Petri net to verify workflow specifications [20] with focus on checking their soundness [21]. In [17], series of workflow resource patterns were proposed to capture the various ways in which human resources are represented and used in business processes. Using ordinary Petri net to formalize Cloud resource allocation would result in massive impractical models. We use Coloured Petri net to get a concise and more elegant representation without losing the benefit of formality that Petri net offers. Another asset of our CPN model is its support for multi-tenancy. Not only does it allow the allocation of multiple resources to one activity, it also covers the sharing of the same Cloud resources by multiple process instances that may pertain to different processes.

Despite the research effort put into the modelling of human resource, works that deal with non-human resources, especially Cloud resources, are still scarce and no formal patterns have been proposed for their modelling. For instance, an extension to BPMN was proposed in [10]. This work aimed at representing the Cloud perspective but lacked expressivity and formality. In our work, we present a formal model that offers a more rigorous expressiveness. In a previous work [5], an Event-B formal specification of the Cloud resource perspective was proposed as a step towards its validation. This work, however, does not treat the resource management as minutely as we do in our present work by detailing the interaction between the Cloud provider and business activities. This is probably due to the fact that Event-B is not the most suitable language to formally describe behavioural properties.

Cloud properties have recently been addressed in some researches in an attempt at their modelling and analysis. A temporal logic called CLTLt(D) (Timed Constraint LTL) was used by [4] to formalize elasticity in Cloud-based systems. In [2], authors define a formal framework for the description and evaluation of service-based business processes elasticity. Elasticity mechanisms and strategies for service-based business processes were described in [14] using Petri

net. An analytical model based on Markov chain was used in [24] to evaluate elasticity strategies and help Cloud providers decide on which strategy to implement. In [13] authors propose an elasticity model description language for StratModel on which they based their framework for the evaluation of elasticity strategies.

Although the elasticity property has been relatively recurrent in literature when business processes are discussed in a Cloud computing context, these works do not approach elasticity from the same point that we consider in this present work. In fact, they all regard elasticity as a behaviour of the whole system where this property manifests as adding/removing copies of a service or resizing its capacity depending on the considered type of elasticity. In this paper however, we are interested in elasticity at a lower lever, that is to say, we focus on elasticity as the behaviour of each service that requires some Cloud resources to finish its execution. This property concerns the made requests as they may vary from one execution to another of the same business process, as well as it concerns the required resources which can exhibit an elastic behaviour to satisfy the requests.

3 Preliminaries

In this section, we present essential concepts for the comprehension of this paper.

3.1 Cloud Resource Allocation in Business Processes

The allocation mechanism was treated in [11] where a resource perspective extension to BPMN was proposed. This extension allows a description of the resources as well as their allocation management. This work was then followed by [10] where the focus was directed on the configurable allocation of Cloud resources. The authors considered three types of Cloud resources, namely network, storage and compute resources and focused on two key properties, namely elasticity and shareability of resources. They adopted a pattern-based modelling approach and hence their proposition consisted of three operators mainly: the assignment, elasticity and sharing/batching operators. The application of the said operators allows for a customizable selection of the required resources by a business process while taking into account their properties. Using this approach results in models that are better in terms of expressivity than the models in which the resource perspective had to be hard-coded, while being lower in terms of complexity. These operators, however designed to express a configurable resource allocation, can be used in their configured form to express a basic allocation. Since we are not interested in our work in the configuration aspect of the allocation, we present these operators stripped of the configuration-related notions.

The resource assignment operator can be considered as the main operator since it is the one that expresses which resources are actually allocated to which activity. Two parameters are used in this operator, namely a *type* and a *range*. The former basically expresses whether all of the connected resources are assigned to the activity (type AND), only one of them is assigned to the activity

(type XOR) or an unfixed number of the connected resources are assigned to the activity (type OR). On the other hand, the *range* parameter is but a configuration guideline specified by the Cloud provider to additionally constraint the minimal and maximal number of resources of each type that should be assigned to each activity. The second operator is used to express the elasticity aspect of a set of resources. It is used to specify the way the resources scale up and down (vertical, horizontal or hybrid elasticity). The third and last operator deals with the shareability of a resource. It specifies the activities allowed to share a specific resource.

The Considered Allocation Mechanism. Although the model proposed in [10] presents a considerable improvement compared to having the resource allocations hard-coded in an ad-hoc manner, it still presents a number of issues. In fact, the proposed modelling approach is informal and the operators should be described formally. Furthermore, even though the elasticity operator is used to set an elasticity type for each of the provided resources, it does not actually allow the modelling of the desired behaviour rigorously. As a matter of fact, the interaction between the Cloud resources provider and the different business processes' activities cannot be depicted using such operators.

In our work, we approach Cloud resource allocation in a way that is inspired from the work in [10] and yet treats it formally and more elaborately. We propose a model that separates the Cloud provider from the requesting activities and therefore is able to represent the interaction between the two parties. This model allows the modelling of parallel requests as well as exclusive requests which correspond to an AND- and XOR-typed assignment operator respectively. Basically, a Cloud provider offers a number of resources. Each resource is characterized by a type, a provided capacity and a number of provided instances. An activity makes a request for a number of instances of some resource of a certain type indicating the requested capacity. Elasticity in our work is not a mere description of a resource property. It is considered at three different levels:

1. Elasticity at the level of the request: an elastic request is a request for a non fixed number of resource instances that have a possibly varying capacity. In other words, both the number of instances and their requested capacity can vary at each execution within the range of a fixed interval.
2. Elasticity at the level of the provider: a resources administered by a Cloud provider can be elastic, i.e., it may have the capability of scaling-up and down its provided capacity. It can be either horizontally or vertically elastic. A horizontally elastic resource scales-up/down by creating/destroying resource instances. A vertically elastic resource scales-up/down by increasing/decreasing the capacity of its instances.
   Two strategies can be considered when dealing with elasticity. The first is a *proactive* strategy: the provider predicts the need to scale-up/down a resource before the scaling becomes a necessity. This can be triggered by reaching capacity thresholds or a certain usage percentage. The second is a *reactive*

strategy: the provider proceeds to scale-up when an elastic resource is no longer able to satisfy an incoming request.
3. Elasticity at the level of the requester: an activity's needs may vary during the execution of the process, and consequently, it can choose to ask for more or less resources at runtime.

3.2 Coloured Petri Net

Petri nets have proven to be one of the best formalisms to model and analyse concurrent systems. Nevertheless, the basic Petri net model is not suitable for the modelling of many systems encountered in IT. In fact, trying to describe a real system using Petri nets usually results in a very large and complex model that may even be inaccurate. Moreover, the tokens in a Petri net are often mapped into objects or resources in the modelled system. However, a simple Petri net token does not make a suitable representation for an object with attributes. To solve these problems many authors propose extensions of the basic Petri net model. Several authors have extended the basic Petri net model with *coloured* or *typed tokens* [22] which have values. A large Petri net model can therefore be represented in a much more compact and manageable manner using a *Coloured Petri net*.

Coloured Petri Nets (CP-nets or CPNs) [12] is a graphical language designed to construct models of concurrent systems and analyse their properties. It combines the capabilities of Petri nets, which serve as basis for the graphical notation, with the capabilities of the programming language CPN ML, which is based on the high-level functional programming language Standard ML [15], to define data types. CP-nets are generally used to model systems where concurrency and communication are key characteristics, such as business processes and workflows.

We refer the reader to [12] for a formal definition of Coloured Petri nets.

4 A Coloured Petri Net Formal Modelling of Cloud Resource Allocation in BP

In this section, we detail the provider's and requester's sub-models separately while highlighting the way we represent elasticity at its different levels. Then we explain the way our model can be integrated with the BP's control flow model.

4.1 Overview of the Model

In this work, we propose a coloured Petri net model to represent the cloud resources perspective in BPs. As shown in Fig. 1, our model separates the requester from the provider, resulting in two sub-models communicating asynchronously through three buffers: buffer_request, buffer_response and buffer_release.

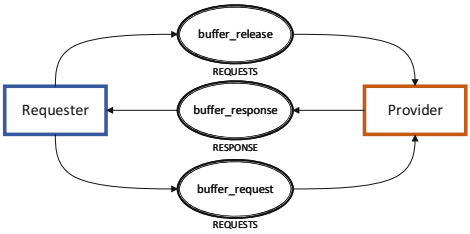

Fig. 1. Global view on the model

Colour Sets. To be able to represent the specific aspects of Cloud resource allocation, we create the following main colour sets which we use in our model:

- id_p: an integer representing the process identifier.
- id_i: an integer representing the process's instance identifier.
- *id*: a couple (p: id_p, i: id_i) to identify a unique process instance.
- id_a: an integer representing the activity's instance identifier.
- id_r: an integer representing the resource identifier.
- *restype*: an enumeration to specify the type of a resource (a compute, storage or network resource).
- *capacity*: an integer representing the capacity of a resource.
- *instnumber*: an integer representing the number of instances of a resource.
- *elasticity*: an enumeration to specify the type of resource elasticity (none, vertical or horizontal).
- *instance*: a couple (*cap*: *capacity*, *inb*: *instnumber*) to represent the number of resource instances with a capacity *cap*.
- *resource*: an octuple (r: id_r, t: *restype*, *init_cap*: *capacity*, *init_inst*: *instnumber*, *total_cap*: *capacity*, *total_inst*: *instnumber*, *inst_list*: *list* (*instance*), *el*: *elasticity*) where *init_cap* is the resource instances initial capacity, *init_inst* is the initial number of provided resource instances, *total_cap* is the total provided capacity, *total_inst* is the total number of provided instances, *inst_list* is a list of the resource's instances and *el* is the resource's elasticity type.
- *demand*: an octuple (r: id_r, t: *restype*, *inst_min*: *instnumber*, *inst_max*: *instnumber*, *inst*: *instnumber*, *cap_min*: *capacity*, *cap_max*: *capacity*, *cap*: *capacity*) where *inst_min* and *inst_max* are the lower and upper bounds of the possible number of requested instances, *inst* is the actual number of requested instances, *cap_min* and *cap_max* are the lower and upper bounds of the possible requested capacity and *cap* is the actual requested capacity.
- *request*: a couple (a: id_a, *ld*: *list demand*) where id_a identifies the requesting activity and *list demand* specifies a set of requested resources.
- *requests*: a couple (i: *id*, *req*: *request*) where *id* identifies the process as well as the instance and *request* is a list of requested resources by a specific activity.
- *response*: a couple (*reqs*: *requests*, b: *bool*) indicating whether a certain request has been satisfied.

For the sake of simplifying the expressions used in our model later on, we denote by l_R the list of resource identifiers of the elements of a *list* of *resources* l.

4.2 The Provider's Sub-model

Upon the reception of a request, and depending on the availability of resources, the provider either allocates the requested resources to the requesting activity or sends back a response indicating the rejection of the request. Upon the release of resources, the provider takes them back into consideration as available resources.

We propose the model in Fig. 2 to represent the behaviour of the provider.

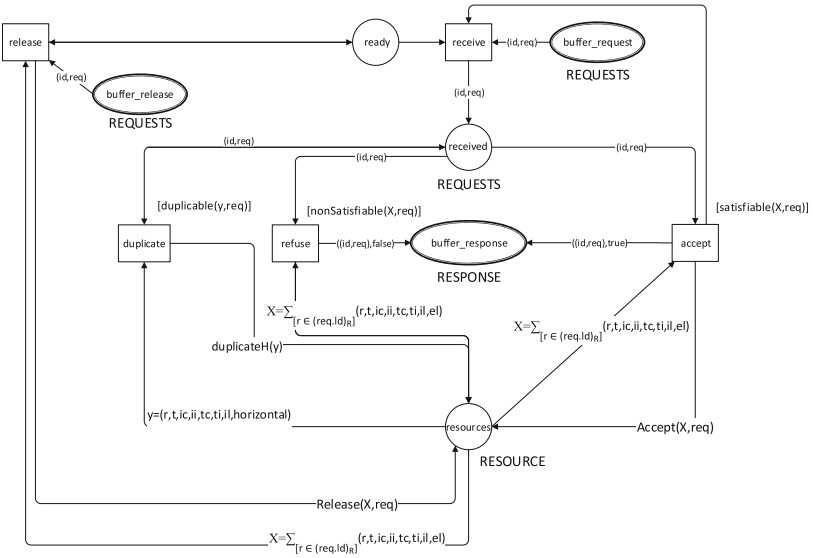

Fig. 2. The provider's model with reactive elasticity

The presence of a request in the buffer_ request enables the *receive* transition. If that request is satisfiable (enough resources are available in the place *resource*) a positive response is deposited in the buffer_ response and the provided resources are updated accordingly through the $Accept(X, req)$ function (the requested number of instances is subtracted from the corresponding resource's provided instances), otherwise if the request is unsatisfiable (not enough instances available of some non elastic requested resource) a negative response is sent through the buffer_ response. The case where resources need to be duplicated for the request to become satisfiable (i.e., reactive elasticity) is treated through the *duplicate* transition.

Released resources are communicated via the buffer_ release. The firing of the *release* transition updates the values of the the corresponding resource from the *resources* place by adding the released resource's instances to the provided resource's instances list.

Proactive Elasticity for the Provider's Sub-model. As previously mentioned in Sect. 3.1, a Cloud provider may consider a proactive strategy to manage the elasticity of its resources. To do so, the model in Fig. 3 needs to be composed with the one in Fig. 2 by merging the *resources* place of the two models. Conditions 1 to 4 are used to trigger the duplication/consolidation actions. Transitions *duplicateH* and *consolidateH* are responsible for the management of horizontally elastic resources. Transitions *duplicateV* and *consolidateV* are responsible for the management of vertically elastic resources.

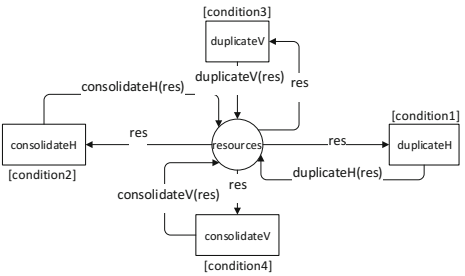

Fig. 3. Proactive elasticity for the provider

4.3 The Requester's Sub-model for Parallel Requests

In need of resources to accomplish its task, an activity sends a request to the provider indicating the type of resources it needs as well as the number of instances and capacity required. If it gets a positive response for all the requested resources, the activity proceeds to execute and releases those resources upon completion. Otherwise, the activity fails to execute. This behaviour corresponds to the use of the assignment operator with its *type* parameter set to AND.

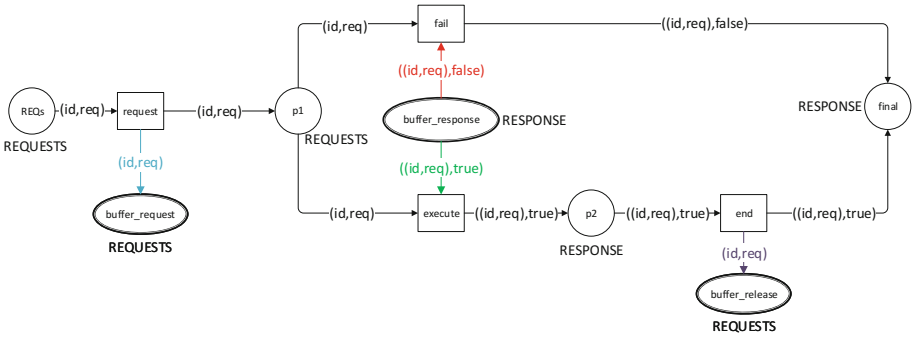

Fig. 4. The requester's model for parallel requests

We propose the model in Fig. 4 to represent the behaviour of the requester when making a demand for multiple resources. In order to request some resources, the task has to go through a three-step procedure:

1. announce its request to the provider (the blue arc) via buffer_ request and wait for its response;
2. start its execution and consume the acquired resources (the green arc) as soon as a positive response to the request is available in buffer_ response;
3. and lastly release those resources (the purple arc) through buffer_ release and finish its execution successfully.

In case the request had not been met (a corresponding negative response is present in buffer_ response), the task fails (the red arc), and skips to a final state that indicates its failure. We note that we use lists to represent a request for multiple resources. It is worth mentioning that to model the assignment of a single resource, we simply use lists containing one element.

4.4 The Requester's Sub-model for an Exclusive Choice of Requests

An activity may need to make an exclusive choice between a number of requests and leave the decision for runtime. This behaviour corresponds to the one expressed by the assignment operator with an *XOR type*.

To model this using CPN, we propose the model in Fig. 5. This model is similar to the previous one, with the few following changes: the place *REQs* and transition *request* are duplicated as many times as the number of exclusive requests. A place *pXOR* is added to make sure that only one *request* transition is fired, and therefore only one request is made at runtime.

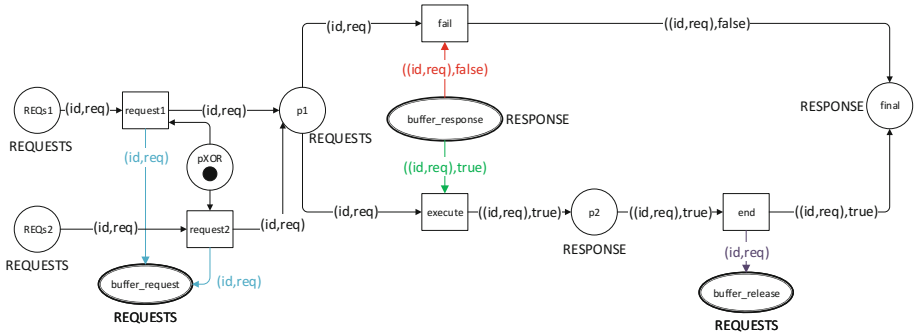

Fig. 5. The requester's model for requests with an exclusive choice

The Requester's Sub-model with Support for Elasticity. As previously mentioned in Sect. 3.1, a requester may need more or less resources than it had already requested. To be able to represent this behaviour, an elasticity mechanism that allows a requester to update its request just before its execution

needs to be implemented. Figure 6 shows the model we propose to support such a property. To avoid a more complex model, we use $req3$ to refer to a request smaller than the initial request req and $(req + req2)$ to refer to a request larger than the initial request req.

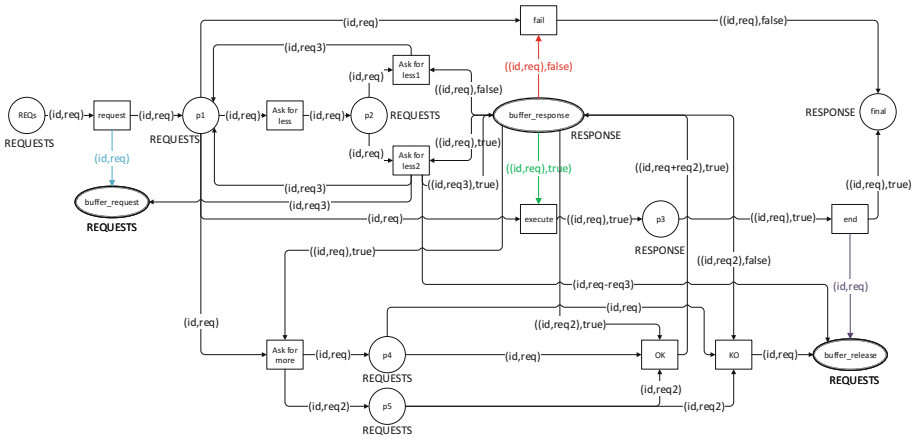

Fig. 6. The requester's model for requests with support for elasticity

4.5 Composing the Control Flow and the Resource Perspective

As soon as knowledge of resource requests is provided, the pattern for the *requester* is duplicated and the following steps are performed to create the link between the control flow model and that of the *requester*:

1. A token representing resource requests coming from the requesting activity of the process instance in question is placed in $REQs$. If an exclusive choice is required for these requests, each demand is placed in a separate $REQs$ place following the requester's model for requests with an exclusive choice.
2. The *request* transition is merged with the transition representing the activity after which the requests are made known. In case the requests are made at the beginning of the process (i.e., before any activity has started) the *request* transition is placed after the start place along with an additional place to maintain the correct structure of the Petri net. In case of exclusive choice, as many *request* transitions as the number of exclusive requests are created.
3. A place typed *request* is added between the newly created transition in step 2 and the transition of the requesting activity with the corresponding request inscribed on both its in and out arcs. In the case of an exclusive choice, this is done for each *request* transition.
4. The *execute* transition of the requester's model is merged with the transition representing the requesting activity in the control flow model.

It is worth mentioning that our proposed model supports multi-tenancy at multiple levels. As a matter of fact, it allows the modelling of multiple resources used by a single activity. Besides, it allows the modelling of, not only multiple processes but also multiple instances, sharing the same pool of Cloud resources. This is achieved by using the same provider model composed with multiple requester models pertaining to different activities of different processes' instances.

5 Evaluation

In order to evaluate our work, we established the corresponding model for a case study from France Telecom/Orange labs [3] which, due to the lack of space, we thoroughly present on our web page[1].

5.1 Case Study

The model of our case study has been implemented and validated using CPN Tools which offer a palette to analyse the net's state space. In this section we discuss some of the results indicated by the resulting state space analysis report. The full CPN model along with the generated report file can be found on our web page[1] where we present our case study with different provided resources to showcase errors detection when the allocation requests are unsatisfiable and lead to a deadlock. In fact, the allocations' correctness amounts to having a corresponding CPN model whose dead markings correspond to final markings only. The presence of a dead marking that does not correspond to a final marking translates the fact that some made request is unsatisfiable and prevents a complete execution of the business process. This can be deducted by inspecting the report file of the CPN model in question.

5.2 Checking Behavioural Properties

A major strength of Petri nets is their support for the analysis of many properties and problems associated with the modelled systems [16]. Those properties can be classified into two categories: those which depend on the initial marking, and those which are independent of the initial marking. The former type of properties is referred to as marking-dependent or behavioural properties, whereas the latter type of properties is called structural properties.

In this section, we are interested in basic behavioural properties. The properties *boundedness*, *reversibility* and *liveness* are independent of each other.

Boundedness. The state space report shows that all of the places of our illustrating example's model are bounded. This translates the fact that the number of tokens in every place does not exceed some finite number for any marking

[1] http://www-inf.it-sudparis.eu/SIMBAD/tools/CLoudResourceBP.

reachable from the initial marking. We are particularly interested in the boundedness of the three buffer places which are shown to be bounded by the number of initially made requests. By verifying this property, it is guaranteed that there is no overflows in the buffers, no matter what firing sequence is taken.

Reversibility. To be able to derive this property from the model of our proposed example, we use a short-circuited version of the model where we added a transition *repeat* that retrieves answers from the $final$ place and restores them as requests in the $REQs$ place in order to simulate a continuous flow of requests.

The state space report indicates that all of the markings are home markings which translates the fact that for each marking M reachable from the initial marking M_i, M_i is reachable from M. Thus, we can always get back to M_i.

Reachability. Reachability is a fundamental basis for studying the dynamic properties of any system [16]. The *reachability problem* for Petri nets is the problem of finding if a given marking M in a net (P, M_i) is reachable from the initial marking M_i. This property can be relaxed if we are only interested in the markings of a subset of places. In this case, we talk about a *submarking reachability problem*. This problem can be defined as the problem of finding whether M' is reachable from M_i, where M' is any marking whose restriction to a given subset of places agrees with that of a given marking M.

In our case, it is interesting to make sure that the initial marking of the resource perspective's model is reachable. This is proved by the fact that all of the markings (including the initial marking) are home markings (see Sect. 5.2).

Liveness/Deadlock-Freedom. The concept of liveness is closely related to the complete absence of deadlocks in operating systems. In fact, a live Petri net guarantees a deadlock-free operation, regardless of the firing sequence.

The resulting state space analysis report for our illustrating example's model (the short-circuited version) indicates that all of our model's transitions are live, which means that every transition remains fireable for any firing sequence. Therefore we can say that the whole CPN model is live and can deduct that it is deadlock-free. In other words, a final marking (a marked "final" place) is always reachable. In case of a faulty allocation, this property cannot be satisfied and a deadlock is indeed detected.

If we limit this property to the requester's and provider's sub-models only, without taking into account the control flow's sub-model, we find that it is always satisfied which means that our model makes sure that, even if a request cannot be satisfied and the requesting activity fails, which may result in the failure of the whole BP, the model can still manage incoming requests from other BPs. In other words, our proposed model supports multi-tenancy at multiple levels.

6 Conclusion

Business process deployment in the Cloud has been a hot topic over the last years. The Cloud resource perspective however, has rarely been broached in the context of BPM and the need to formalize this perspective has yet to be studied. To address this issue, we proposed in this paper a Coloured Petri Net formal model for Cloud resource allocation in Business Processes.

The CPN model that we propose can be manipulated to map the BPMN extension proposed in [10]. However, we chose to represent an allocation mechanism that is richer and more detailed. Therefore, BPMN cannot be used to represent the Cloud resource allocation and elasticity aspect as we treat it using coloured Petri net in this work. From a designer's point of view, it would be easier to use a more business-related language than to model a business process using Petri net. Consequently, we will be proposing in future work a BPMN extension that includes all the relevant information needed to perform an automatic transformation into our CPN model which can then be used to verify the initial model.

This formalization that we propose is a first step towards the verification of Cloud resource allocation in business processes. We have so far worked on the verification of generic properties such as liveness and deadlock-freedom of the model. In the future work, we will detail how to use our model to verify other properties which are more specific to resource management and therefore improve the correctness verification of the allocations at design-time. To do so, we consider proving domain-specific properties like the shareability of Cloud resources. Such properties and others can be expressed in LTL (Linear Temporal Logic) and verified using tools as Helena [9] and CPN Tools.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases **14**(1), 5–51 (2003)
2. Amziani, M., Melliti, T., Tata, S.: Formal modeling and evaluation of service-based business process elasticity in the cloud. In: 2013 IEEE 22nd International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), pp. 284–291. IEEE (2013)
3. Assy, N., Yongsiriwit, K., Gaaloul, W., Yahia, I.G.B.: A framework for semantic telco process management - an industrial case study. In: 14th International Conference on Intelligent Systems Design and Applications, ISDA 2014, 28–30 November 2014, Okinawa, Japan, pp. 44–49 (2014). https://doi.org/10.1109/ISDA.2014.7066276
4. Bersani, M.M., Bianculli, D., Dustdar, S., Gambi, A., Ghezzi, C., Krstic, S.: Towards the formalization of properties of cloud-based elastic systems. In: Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems, pp. 38–47. ACM (2014)
5. Boubaker, S., Mammar, A., Graiet, M., Gaaloul, W.: An event-B based approach for ensuring correct configurable business processes. In: IEEE International Conference on Web Services, ICWS 2016, 27 June – 2 July 2 2016, San Francisco, CA, USA, pp. 460–467 (2016). https://doi.org/10.1109/ICWS.2016.66

6. Cabanillas, C., Knuplesch, D., Resinas, M., Reichert, M., Mendling, J., Ruiz-Cortés, A.: RALph: a graphical notation for resource assignments in business processes. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 53–68. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_4
7. Cabanillas, C., Norta, A., Resinas, M., Mendling, J., Ruiz-Cortés, A.: Towards process-aware cross-organizational human resource management. In: Bider, I., et al. (eds.) BPMDS/EMMSAD -2014. LNBIP, vol. 175, pp. 79–93. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43745-2_6
8. Chiotti, O., Stroppi, L.J.R., Villarreal, P.: Extending the WS-humantask architecture to support the resource perspective of BPEL processes (2014)
9. Evangelista, S.: The Helena Petri net tool (2013). http://www.lipn.univ-paris13.fr/~evangelista/helena/
10. Hachicha, E., Assy, N., Gaaloul, W., Mendling, J.: A configurable resource allocation for multi-tenant process development in the cloud. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 558–574. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_34
11. Hachicha, E., Gaaloul, W.: Towards resource-aware business process development in the cloud. In: 29th IEEE International Conference on Advanced Information Networking and Applications, AINA 2015, 24–27 March 2015, Gwangju, South Korea, pp. 761–768 (2015)
12. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems, 1st edn. Springer, Heidelberg (2009). https://doi.org/10.1007/b95112
13. Jrad, A.B., Bhiri, S., Tata, S.: STRATModel: elasticity model description language for evaluating elasticity strategies for business processes. In: Panetto, H. (ed.) OTM 2017. LNCS, vol. 10573, pp. 448–466. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_29
14. Klai, K., Tata, S.: Formal modeling of elastic service-based business processes. In: 2013 IEEE International Conference on Services Computing (SCC), pp. 424–431. IEEE (2013)
15. Milner, R., Tofte, M., Harper, R.: Definition of Standard ML. MIT Press, Cambridge (1990)
16. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
17. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., Edmond, D.: Workflow resource patterns: identification, representation and tool support. In: Pastor, O., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 216–232. Springer, Heidelberg (2005). https://doi.org/10.1007/11431855_16
18. Stroppi, L.J.R., Chiotti, O., Villarreal, P.D.: Extending BPMN 2.0: method and tool support. In: Dijkman, R., Hofstetter, J., Koehler, J. (eds.) BPMN 2011. LNBIP, vol. 95, pp. 59–73. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25160-3_5
19. Stroppi, L.J.R., Chiotti, O., Villarreal, P.D.: Extended resource perspective support for BPMN and BPEL. In: CIbSE, pp. 56–69 (2012)
20. Van Der Aalst, W.M.: Workflow verification: finding control-flow errors using Petri-net-based techniques. Bus. Process. Manag. **1806**, 161–183 (2000)
21. Van Der Aalst, W.M., et al.: Soundness of workflow nets: classification, decidability, and analysis. Form. Asp. Comput. **23**(3), 333–363 (2011)

22. Van Hee, K., Verkoulen, P.: Integration of a data model and high-level Petri nets. In: Proceedings of the 12th International Conference on Applications and Theory of Petri Nets, Gjern, pp. 410–431 (1991)
23. Woitsch, R., Utz, W.: Business process as a service (BPaaS). In: Janssen, M., et al. (eds.) I3E 2015. LNCS, vol. 9373, pp. 435–440. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25013-7_35
24. Yataghene, L., Ioualalen, M., Amziani, M., Tata, S.: Using formal model for evaluation of business processes elasticity in the cloud. In: Drira, K., et al. (eds.) ICSOC 2016. LNCS, vol. 10380, pp. 33–44. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68136-8_3