Resource Provisioning Strategies for BPMN Processes: Specification and Analysis using Maude

Francisco Durán

ITIS Software, University of Málaga, Málaga, Spain

Camilo Rocha

Pontificia Universidad Javeriana, Cali, Colombia

Gwen Salaün

Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000 Grenoble France

Abstract

Business process optimization is a strategic activity in organizations because of its potential to increase profit margins and reduce operational costs. One of the main challenges in this activity is concerned with the problem of optimizing allocation and sharing of resources. Companies are continuously adjusting their resources to their needs following different strategies. However, the dynamic provisioning strategies are hard to compare. This paper proposes an automatic analysis technique to evaluate and compare the execution time and resource occupancy of a business process relative to a workload and a provisioning strategy. Four different strategies are presented, which are guided –respectively– by recent resource usage, recent resource request, predicted behavior, and a combination of available strategies. Such analysis is performed on models conforming to an extension of BPMN with quantitative information, including resource availability and constraints. Within this framework, the approach is fully mechanized using a formal and executable specification in the Maude rewriting logic framework, which relies on existing techniques and tools for simulating probabilistic and real-time specifications. The paper includes results on the extensive experimentation that has been carried out for validation purposes.

Key words: Business processes, BPMN, resource provisioning, rewriting logic, automated verification, Maude, simulation-based analysis.

Preprint accepted to JLAMP *December 17, 2021*

1. Introduction

A business process is a collection of structured activities or tasks that produce a specific product and fulfil a specific organizational goal for a customer or market. The aim of a process is to model activities, and their causal and temporal relationships by defining specific business rules. Process instances then have to comply with such a description once they are deployed. The Business Process Model and Notation (BPMN) [14] is a graphical modeling language for specifying business processes, which has become the common notation for designing business processes. Several industrial platforms have been developed during the last 10 years to support the modeling and management of BPMN processes. Nowadays, organizations are making efforts to use such platforms to define their organizational processes, with the goal of achieving better control over the processes when they are deployed.

Business process optimization is a strategic activity in organizations because of its potential to increase profit margins and reduce operational costs [11]. Resource allocation and provisioning is one of the main challenges in order to minimize execution times, optimize resource usage, improve sharing, and detect bottlenecks with the final goal of improving processes' executions. However, providing automated techniques for analyzing and optimizing BPMN processes is a challenging problem. It requires a model of the process including execution time of tasks and flows, as well as an explicit description of resource usage requirements. A solution to this problem would take such a process as input and compute a set of metrics (e.g., process execution time, waiting times, resource occupancy) as output. These measures could then be used as part of a further analysis stage with the goal of optimizing the process relative to a cost model.

As it is shown in the related work (Section 7), there have been different proposals to analyze and optimize the assignment and scheduling of resources. By having a better allocation of resources, it is possible to reduce the overall process time and/or costs when it is finally deployed. However, the assignment of resources is seldom static, rendering the optimization problem more interesting. Modern enterprises and systems have access to resource repositories and to the possibility of acquiring/releasing them with great flexibility. Thus, in addition to having a pool of resources that can be directly used, they can provision and release resource instances as needed, which makes the problem not only more interesting, but also much more challenging. Since the analysis procedure involves complex computations and lengthy simulations, it is highly convenient to be able to perform resource analysis in a fully automated way, especially at design stages,

before the processes are finally deployed.

Consider, for example, a company that uses cars for some of its tasks. The company may have some cars on lease, which may be used to carry on the different tasks in its processes. Indeed, having enough resources available at all times may be key to minimize execution times. But it also affects costs. Even if inactive, their leases are costing money to the company. For this reason, to be as efficient as possible, it would be ideal to always have the pool of resources as fit to the given needs as possible. Hence, at any time, it may be decided to get new cars or terminate the lease of some of the cars in the current pool. That is, continuously, or at least periodically, the amount of instances in this pool must be re-evaluated.

The criteria for adding or removing resources to a pool may depend on many different variables, as well as on the type of resource or its specific use. The current use of the resources, the number of times a resource was not available, or the prediction of the resource usage in some near future can help in defining such criteria. For instance, having a number of customer orders greater than the maximum that could be served in adequate conditions can help in deciding to hire additional employees, or lease additional cars and drones. Conversely, if the cars are being underused, a good decision may be to finish the lease of some of them or simply not renewing their leases. The key question to ask is how can it be decided what the right or best strategy is?

Instead of focusing on the allocation of a fixed set of available resources, this paper presents a solution for the analysis of alternative provisioning strategies for the *dynamic adaptation* of resource assignments in process models. The approach presented here focuses on the analysis of quantitative properties associated to BPMN processes. Although it encompasses a broad selection of quantitative measures, this paper uses execution time (i.e., the time it takes to execute a process) and resource occupancy (i.e., the percentage of usage of any or all replicas of a resource) to validate the approach. The final goal is thus to use such analyses to streamline a process by reducing its operational costs in relation to execution time and resources, which can be directly inferred from the expected execution times and resource usage. These measures and resulting values can also be used to dynamically adjust the number of resources at runtime.

The approach relies on a formal specification in rewriting logic of BPMN processes. The specification is given in the rewriting logic based language Maude [4] and serves as an executable semantics of the BPMN language under consideration. Since it is *executable*, it has the advantage of enabling the use of Maude's verification tools for computing a number of metrics of processes with a clear mathematical meaning. More precisely, given a process description, and taking as

parameters the workload and the provisioning strategy, exhaustive Monte Carlo simulations of the business process are executed in order to provide detailed information on the evolution of execution times, use of resources, and therefore costs, which altogether enable the comparison of different strategies for resource provisioning in a dynamic environment. The approach proposed in this paper is illustrated and discussed on two case studies with dynamic allocation of resources.

The current paper is an extended and improved version of [8]. Please, see the discussion in Section 7 for details on the contributions of this paper with respect to that work. The current Maude specification builds on the one developed by the same authors in previous related work [6, 7] for different forms of analysis of business processes. These other works are also discussed in Section 7. The reader is referred to `https://github.com/narudocap/maude-bpmn` for details on the formal specification, experiments, and additional examples.

The organization of the rest of the paper is as follows. Section 2 presents the BPMN notation extended with the annotations supporting the proposed approach. Section 3 overviews rewriting logic and Maude, and its use for the specification of object-oriented, real-time and probabilistic systems. Section 4 presents the specification of the annotated BPMN extension in Maude's rewriting logic, which serves as a semantics for the language and makes automated analysis possible using Maude's tools. Section 5 introduces several strategies for dynamically allocating resources as well as the experimental environment. Section 6 shows and discusses on the experimental results, which illustrate how the number of resources evolve, and how comparison of provisioning strategies helps to reduce costs and time. Finally, Section 7 presents a discussion on related work and Section 8 concludes the paper.

2. Annotated BPMN

BPMN is a graphical notation for modeling business processes as collections of related tasks that produce specific services or products. In BPMN, processes are modeled using graphical representations for tasks, events, and gateways, which are connected through flows. In this work, the focus is on the core features of BPMN (i.e., its control flow constructs); it supports the most common types of tasks, events, and gateways. The initiation and finalization of processes are represented by initial and final events. Events are also used to represent the sending of messages and the firing of timers. A task represents an atomic activity that has exactly one incoming and one outgoing flow. A sequence flow describes two nodes executed one after the other, i.e., imposing an execution order between these

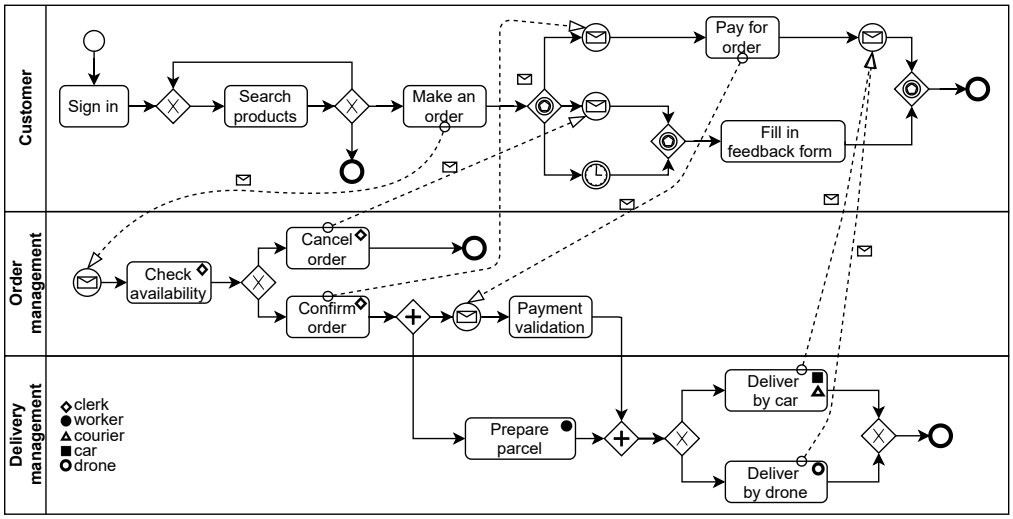

Figure 1: Running example: parcel delivery.

nodes. Tasks may send messages, which in such a case activate the corresponding message flows.

Gateways are used to control the divergence and convergence of the execution flows. In this work, *exclusive*, *inclusive*, *parallel*, and *event-based* gateways are supported. Gateways with one incoming branch and multiple outgoing branches are called *splits* (e.g., split inclusive gateway). Gateways with one outgoing branch and multiple incoming branches are called *merges* (e.g., merge parallel gateway). An exclusive gateway chooses one out of a set of mutually exclusive alternative incoming or outgoing branches. For an inclusive gateway, any number of branches among all its incoming or outgoing branches may be taken. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches. For an event-based gateway, it takes one of its outgoing branches or accepts one of its incoming branches based on events.

In addition to the description of specific tasks and their sequencing, collaboration diagrams also involve *pools* and *lanes*, which are structuring elements that split processes into pieces.

To introduce and illustrate the use of the supported BPMN constructs, and the analysis techniques presented in this work, the process depicted in Figure 1 is used; it describes a parcel ordering and delivery. The process consists of three lanes, namely, one for customers, one for the order management, and one for the

delivery management. In this process, the client first signs in and then repeatedly looks for products. Eventually, the client can decide to give up (i.e., termination) or to make an order by submitting it to the order management lane. The client then waits for a response (i.e., acceptance or refusal of this order). However, the client waits for a response for a maximum amount of time, as is represented by a timer-event branch. If the order can be completed, then the parcel is received and the client pays for it. Otherwise (i.e., timeout or order refused), the client fills in a feedback form. As far as the management lane is concerned, the first task aims at verifying whether the goods ordered by the client are available. If they are not available, then the order is canceled; otherwise, the order is confirmed. The order management takes care of the payment of the order whereas the delivery lane is triggered to prepare the parcel to be delivered. The delivery may be carried out by car or by drone.

In BPMN, each lane in a collaboration diagram corresponds to a specific role or resource. However, other resources may also be involved, and tasks could require multiple resources or instances of the same resource. Therefore, instead of implicitly associating resources to lanes, resources are explicitly defined at the task level. Hence, a task that requires resources for its execution can include, as part of its specification, the required resources. To do it graphically, symbols are associated to each resource type, and these symbols are depicted inside the corresponding tasks. Notice that resources can refer to humans (e.g., employee, cashier, executive) as well as non-human ones (e.g., robot, virtual machine, drone, tool). For example, the process in Figure 1 relies on clerks for the handling of customers' orders, workers for parcel packing, and couriers for car delivery. In addition, cars and drones are used to deliver the parcels. For instance, the diamonds at the right-top corners of the Check availability, Cancel order, and Confirm order tasks indicate that one instance of the clerk resource is required for the execution of the tasks. Task Deliver by car requires instances of the car and courier resources. In general, besides multiple resources, specific amounts of them can be specified. For example, some weight of flour to prepare a recipe or some amount of money to purchase some product could be specified. To avoid dealing with multiple units of measurement, resources are counted as instances or replicas, and if more than one instance of a certain resource type is required, they are depicted as a number of icons in the task.

The process evolves by successively executing its tasks. However, the execution of a task requires the specified amounts of resources, which may lead to a competition for such resources. Notice that multiple tasks may require the same resource, and multiple instances of the process may also run concurrently. In our

running example, e.g., clerks are used in several tasks, and multiple customers may be trying to simultaneously purchase products. Then, if there are enough resource instances available, the execution of the task can proceed. Otherwise, the task remains suspended until enough instances become available.

It will be seen later in this paper that the proposed analysis techniques rely on simulation. To simulate the execution of processes, the process specifications are enriched with quantitative information. This additional information is added as annotations to the process model. Specifically, durations and delays associated to tasks and flows are expressed as stochastic expressions. Similarly, alternative constructs (split exclusive and inclusive gateways) are extended with probabilities associated to outgoing flows.

Process discovery [10, 24] aims at inferring a structured representation of a process from actual executions, i.e., event logs. In this work, it is assumed that there is at disposal an abstract description of the process model, since branching constructs (exclusive and inclusive split gateways) do not come with any additional information. Similarly, durations associated to flows and tasks are not necessarily provided. Therefore, this work proposes to learn branching probabilities, durations, and delays from actual execution traces [29, 17]. Given a sufficient number of execution traces, these mechanisms enable an accurate generation of quantitative process models. Alternatively, these parameters may be provided by the experts that specify the business process. In any case, these real-time probabilistic models are then used, not only to simulate the behavior of the processes, but also to predict its expected behavior by looking in advance at the following potential steps of the simulation. Figure 2 shows the process given in Figure 1 enriched with such information.

Data-based conditions for split gateways are modeled using probabilities associated to outgoing flows of exclusive and inclusive split gateways. For instance, notice the exclusive split after the Search products task in the customer lane of the running example, which has outgoing branches with probabilities 0.6, 0.2, and 0.2, specifying the likelihood of following each corresponding path. The probabilities of the outgoing flows in an exclusive split must sum up to 1, while each outgoing flow in an inclusive split can be equipped with a probability between 0 and 1 without a restriction on their total sum.

The timing information associated to tasks and flows (durations or delays) is described either as a literal value (a non-negative real number) or sampled from a probability distribution function according to some meaningful parameters. The probability distribution functions currently available include exponential, normal/Gauss, and uniform (see, e.g., [30]). To simplify the reading of the

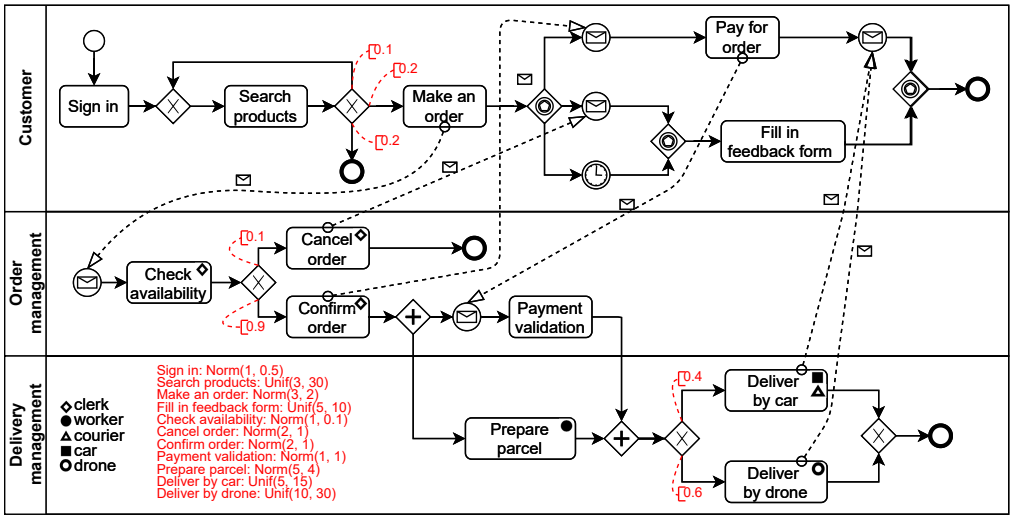

Figure 2: Running example: parcel delivery with durations and probabilities.

process in Figure 1, the specification of task durations has been placed apart from the process description, at the bottom-left corner. In the modelling tool, these parameters would be specified as properties of the corresponding elements. For instance, the duration of the Sign in task is specified as Norm(1, 0.5), which means that it follows a normal distribution with mean 1 and variance 0.5, and the Search products task follows a uniform distribution in the interval $[3, 30]$, that it is specified as Unif(3, 30). Also to simplify the specification of the process, the delays in all flows are set to Norm(1.0, 0.2) to express that it takes some time to move from one task to the following one(s).

3. Rewriting Logic and Maude in a Nutshell

This section provides an overview of rewriting logic [19] and Maude [4], rendering its focus on their object-like, real-time, and probabilistic features.

3.1. Rewriting logic and Maude

Rewriting logic [19] is a logic of change that can naturally deal with state and with highly nondeterministic concurrent computations. A rewrite logic theory is a tuple $(\Sigma, E \uplus B, R)$, where $(\Sigma, E \uplus B)$ is a membership equational logic [3] theory with Σ its signature, E a set of conditional equations and sort membership axioms, B a set of equational axioms (e.g., associativity, commutativity and identity) so that rewriting is performed modulo B, and R is a set of labeled conditional rules.

Maude [4] is a high-level language and a high-performance interpreter that supports membership equational logic and rewriting logic specification and programming of systems. Thus, Maude integrates an equational style of functional programming with rewriting logic computation. Thanks to its efficient rewriting engine and its metalanguage capabilities, Maude turns out to be an excellent tool for creating executable environments of various logics, models of computation, theorem provers, and even programming languages.

A functional specification must be terminating, confluent, and sort-decreasing. Computation in a functional module is accomplished by using the equations as simplification rules from left to right until a canonical form is found. Some equations, such as those expressing the commutativity of binary operators, are, however, not terminating. Nonetheless, they are supported by means of *operator attributes*, so that Maude performs simplification modulo the equational theories provided by such attributes, which can be associativity (assoc), commutativity (comm), identity (id), and idempotency (idem). The above properties must therefore be understood in the more general context of simplification *modulo* such equational theories.

In Maude, a distributed system is axiomatized by a rewrite theory describing its states as an algebraic data type (an equational sub-specification) and a collection of conditional rewrite rules specifying its *behavior*. Rewrite rules are written crl $[l] : t \Rightarrow t'$ if C, with l the rule label, t and t' terms, and C a guard or condition. Rules describe the local, concurrent transitions that are possible in the system, i.e., when a part of the system state fits the pattern t, then it can be replaced by the corresponding instantiation of t'. The guard C acts as a blocking precondition: a conditional rule can only be fired if its condition is satisfied. Sometimes rules are given without label or condition (which can be assumed to be true). Rewrite specifications are not required to be terminating nor confluent.

In the Maude language, object-oriented systems can be specified by object-oriented modules in which classes and subclasses are declared, with the usual support for inheritance, dynamic binding, etc. A class is declared with syntax class $C \mid a_1 : S_1, \ldots, a_n : S_n$, where C is the name of the class, a_i are attribute identifiers, and S_i are the sorts of the corresponding attributes. The objects of a class C are then record-like structures of the form $< O : C \mid a_1 : v_1, \ldots, a_n : v_n >$, where O is the name of the object and v_i are terms of corresponding sorts S_i that represent the current values of its attributes.

In a concurrent object-oriented system, the concurrent state, which is called a *configuration*, consists of a multiset of objects and messages. Rewrite rules then define transitions between such configurations. These transitions represent the

different actions that may occur in the system. For instance, there will be rules modeling the effects of events, or the synchronous and asynchronous communication events of objects and messages. The general form of a rewrite rule `r` is the following:

```
crl [l] :
  < O1 : C1 | atts1 > ... < On : Cn | attsn >
  M1 ... Mm
  => < Oi1 : C'i1 | atts'i1 > ... < Oik : C'ik | atts'ik >
     < Q1 : C''1 | atts''1 > ... < Qp : C''p | atts''p >
     M'1 ... M'q
  if Cond .
```

where l is the rule label, $M_1...M_m$ and $M'_1...M'_q$ are messages, $O_1...O_n$ and $Q_1...Q_p$ are object identifiers, $C_1...C_n$, $C'_{i_1}...C'_{i_k}$ and $C''_1...C''_p$ are classes, $i_1...i_k$ is a subset of $1...n$, and *Cond* is a Boolean condition (the rule's *guard*). The result of applying such a rule is that: (i) messages $M_1...M_m$ disappear, i.e., they are consumed, (ii) the state, and possibly the classes of objects $O_{i_1}...O_{i_k}$ may change, (iii) all the other objects O_j vanish, (iv) new objects $Q_1...Q_p$ are created, and (v) new messages $M'_1...M'_q$ are created, i.e., they are sent.

3.2. Real-time and probabilistic specifications

The executable specification of BPMN presented in the following sections is a probabilistic rewrite theory $\mathcal{R} = (\Sigma, E \uplus B, R)$, where $(\Sigma, E \uplus B)$ is a membership equational logic theory [1]. The equational subtheory offers the infrastructure for defining a process in the sublanguage of BPMN described in Section 2, including the timing behavior for tasks and flows, resource dynamics, and probabilities for outgoing flows of split gateways. The real-time aspects are modeled using Real-Time Maude [22], which supports the formal specification and analysis of *real-time systems*. Specifically, the probabilistic rewrite rules R axiomatize how time advances and probabilistic choices are made in this infrastructure, in order for a given process to transition from an initial to a final state.

Real-Time Maude provides a sort `Time` to model the time domain, which can be either discrete or dense. Time advancement is modeled with *tick rules*, e.g.,

```
crl [l] : { t, T } => { t', T + τ } if C .
```

where t and t' are system states (a process in execution in the case of this paper), T is the global time and τ is a term of sort `Time` that denotes the *duration* of the

rewrite, affecting the *global time elapse*. Since tick rules affect the global time, in Real-Time Maude time elapse is usually modeled by one single tick rule and the system dynamic behavior by instantaneous transitions. Although there can be many sampling strategies, in this work time elapse is modeled with a single tick rule with the help of two functions: the delta function, that defines the effect of time elapse over every model element, and the mte (maximal time elapse) function, that defines the maximum amount of time that can elapse before any action is performed (see [22] for additional details).

In a standard rewrite theory, the conditions of rewrite rules are assumed to be purely equational. As explained above, a rewrite rule $l(\vec{x}) \rightarrow r(\vec{x})$ *if* $\phi(\vec{x})$ specifies a pattern $l(\vec{x})$ that can match some fragment of the system's state t if there is a substitution θ for the variables $\vec{x}$ that makes $\theta(l(\vec{x}))$ equal modulo B to that state fragment, changing it to the term $\theta(r(\vec{x}))$ in a local transition if the condition $\theta(\phi(\vec{x}))$ is true. In a probabilistic rewrite theory [1], rewrite rules can have the more general form $l(\vec{x}) \rightarrow r(\vec{x}, \vec{y})$ *if* $\phi(\vec{x})$ *with probability* $\vec{y} := \pi(\vec{x})$, where some new variables $\vec{y}$ are present in the pattern r on the right-hand side. Because the pattern $r(\vec{x}, \vec{y})$ may have new variables $\vec{y}$, the next state specified by such a rule is not uniquely determined: it depends on the choice of an additional substitution ρ for the variables $\vec{y}$. In this case, the choice of ρ is made according to the family of probability functions π_θ: one for each matching substitution θ of the variables $\vec{x}$. In Section 4, it will be seen how this is realized in practice through an eval operation implementing different probability distributions. Therefore, a probabilistic rewrite theory can express both non-deterministic and probabilistic behavior of a concurrent system.

3.3. A simple example on bank accounts

This section introduces a simple example illustrating the ideas introduced in the previous subsections, with the purpose of helping the reader to better grasp the contents of the rest of the paper.

The module in Figure 3 shows the definition of a simple Account class with a single balance attribute, for which a single transfer message is defined. The effect of this message is represented in the rule in the module. Upon the reception of a message transfer(A, B, M), the balance of account A is decreased in M units at the time the balance of account B gets increased in the same amount.

The example in Figure 4 shows a very simple definition of a Bank class. The Bank class is defined as a subclass of the Account class (line 14). In addition to the attribute balancethat the class inherits from Account, Bank instances will have attributes timer and account. According to the module BANKING-SYSTEM, banks

```
 1 omod ACCOUNT is
 2   protecting INT .
 3   vars A B : Oid .
 4   vars M Bal Bal' : Int .
 5
 6   class Account | balance : Int .          ---- Account class
 7   msg transfer : Oid Oid Int -> Msg .      ---- transfer message
 8  crl < A : Account | balance : Bal >
 9      < B : Account | balance : Bal' >
10      transfer(A, B, M)
11   => < A : Account | balance : Bal - M >
12      < B : Account | balance : Bal' + M >
13   if M <= Bal .
14 endom
```

Figure 3: An object-oriented module defining an account class.

make a periodic transfer of money to some fixed account. Attribute account keeps the object identifier (type Oid) of the addressee of the transfers, and attribute timer is a timer that indicates the time remaining until the next transfer. Every time the timer becomes zero, a new transfer message is sent from the bank object to its pre-established target account for 100 units, and the timer is set to some random value between 29 and 31.

In Maude, random values are defined by a pseudo-random number generator. The eval function takes a stochastic expression and an index (a Nat value), and returns a pair $[T, N]$, with T a value of sort Time and N a value of sort Nat. Specifically, eval(Unif(29, 31), *Idx*) returns a pair $[T', Idx']$, where T' is the *Idx*-th Time value in the uniform distribution between 29 and 31. Since complex stochastic expressions may require multiple indexes, the eval function also returns the next index Idx' to be used.

In addition to objects (Account and Bank) and messages, a banking system needs to maintain the current index — for random number generation — and the global time. The declaration in line 11 defines a constructor $\{\_,\_,\_\}$ for such terms. Finally, as pointed out in the previous section, there is a global time and a unique tick rule. The generic tick rule is shown in lines 20–23. The rest of the module presents typical definitions of the mte and delta functions. Note that since the only timer in the system is the one in Bank objects, and there are no other timers, clocks or delays, the definitions of these functions are straightforward. The maximum time elapse for some configuration is given by the smallest of the timers of the bank objects, and the effect of time elapse results in the decrementation of these timers.

Figure 5 shows a sample execution of a banking system with an account with

```
omod BANKING-SYSTEM is
  inc ACCOUNT .
  inc STOCHASTIC-EXPRESSION .
  vars A B : Oid .
  vars T T' : Time .
  vars Idx Idx' : Nat .
  var  Conf : Configuration .
  ops a b : -> Oid .

  sort System .
  op {_,_,_} : Configuration Nat Time -> System .          ---- System constructor

  class Bank | timer : Time, account : Oid .               ---- Bank class
  subclass Bank < Account .

 crl { < B : Bank | timer : 0, account : A > Conf, Idx, T } ---- periodic transfer
  => { < B : Bank | timer : T', account : A > transfer(B, A, 100) Conf, Idx', T }
  if [T', Idx'] := eval(Unif(29, 31), Idx) .

 crl [tick] :                                              ---- tick rule
     { Conf, Idx, T }
  => { delta(Conf, T'), Idx, T + T' }
  if T' := mte(Conf) /\ T' =/= 0 .

  op mte : Configuration -> Time .                         ---- mte function
  eq mte(< B : Bank | timer : T > Conf) = min(T, mte(Conf)) .
  eq mte(Conf) = INF [owise] .

  op delta : Configuration Time -> Configuration .         ---- delta function
  eq delta(< B : Bank | timer : T > Conf, T')
   = < B : Bank | timer : T monus T' >
     delta(Conf, T') .
  eq delta(Conf, T) = Conf [owise] .
endom
```

Figure 4: A simple banking example to illustrate the use of time and probabilities in Maude.

```
Maude> frew [100] { < a : Account | balance : 1000 >
                    < b : Bank | timer : 0, account : a, balance : 1000000 >,
                    0, 0 } .
result System: { < a : Account | balance : 4300 >
                 < b : Bank |
                   balance : 996700,
                   timer : 4321657017174495/140737488355328,
                   account : a >
                 transfer(b, a, 100),
                34,
                13918168045755612 3/140737488355328}
```

Figure 5: A sample execution of the banking system.

balance 1 000 and a bank with balance 1 000 000. The rewriting is limited to 100 steps, leading to a balance of 4 300 in the account and 996 700 in the bank. Note that time is represented by rational numbers.

4. Executable Specification of BPMN

This section presents an overview of the executable Maude specification formalizing the timed and probabilistic extension of BPMN presented in Section 2. It omits the presentation of the dynamic strategies for the provisioning of resources, which is the focus of Section 5.

The formalization of real-time probabilistic BPMN builds on recent results [7]. Its Maude specification consists of two parts: the process structure, as an equational specification — a membership equational logic theory $Spec_{\text{BPMN}}$ so that a process model P is an element of the (ground) term algebra $\mathcal{T}_{Spec_{\text{BPMN}}}$ — and its evolution semantics using rewrite rules — the rewrite theory RT_{BPMN} extends $Spec_{\text{BPMN}}$ and defines the behavior of BPMN processes by providing some additional definitions and rules specifying such a behavior. The following sections are devoted to each of these two parts.

4.1. Process description

In the Maude specification of BPMN, a process is represented as an object having sets of nodes and flows as attributes. The representation of each node type includes the necessary information to describe its structure and to contribute to the overall process analysis. For instance, a task node involves an identifier, a description, two flow identifiers (input and output), a stochastic function modeling its duration, a set of resources required for its execution, and a set of messages to be delivered after its completion. A split node includes a node identifier, a gateway

type (exclusive, parallel, inclusive, or event-based), an input flow identifier, and a set of output flow identifiers. A merge node includes a node identifier, a gateway type, a set of input flow identifiers, and an output flow identifier. The representation of a flow includes a probability distribution function specifying its delay, and an optional message or timer. The message blocks the flow until it is received, whereas the timer represents a delay after which the execution is triggered.

Nodes, flows, and events are assigned unique identifiers. Given the labeling shown in Figure 6 for the running example, this process can be specified as shown in the excerpt in Figure 7. It shows how a Process object has attributes with the definition of its nodes and flows connecting them. For example, the exclusive split id("n005") (lines 5–6) has id("f004") as incoming flow, and id("f005"), id("f006"), and id("f007"), with associated probabilities 0.6, 0.2, and 0.2, respectively, as outgoing flows. Furthermore, the event-based split gate id("n007") (line 7) has id("f008") as incoming flow, and id("f009"), id("f010"), and id("f011") as outgoing flows. Note the definition of these flows in lines 17–19; after the corresponding delay, they become active upon the reception of the corresponding messages or by the id("timeout") timer firing. Finally, note that the specifications of tasks and flows also include their duration or delays as stochastic functions. For example, the duration of the Prepare parcel task follows a normal distribution with average 5 and variance 4, which is specified by the term Norm(5.0, 4.0). All flows are specified with Norm(1.0, 0.2) as second argument, stating that they all have a delay that follows a normal distribution with given parameters.

Modelling of BPMN processes can be performed using available tools for BPMN 2.0 (e.g., Activiti, Bonita, Signavio). The transformation from the BPMN diagrammatic representation of processes to the corresponding Maude representation is implemented by using the VBPMN platform [15, 16]. The additional quantitative information (e.g., durations, delays, resource information) that is not part of the standard is manually added to the Maude code previously generated. The development of a web application providing a user-friendly UI for modelling BPMN and its quantitative extensions, for calling the proposed verification techniques, and for presenting the results in a readable way will be part of future work.

4.2. Execution semantics

The operational semantics of BPMN is defined using rewrite rules, which mainly model how *tokens* evolve through a process. Each move of a token inside a BPMN process is modeled as a rewrite rule. E.g., when a token arrives at an event-based split gateway, the token is made active with its optional timer. In that rule, if there is an outgoing flow with a timer, an event is added to the set

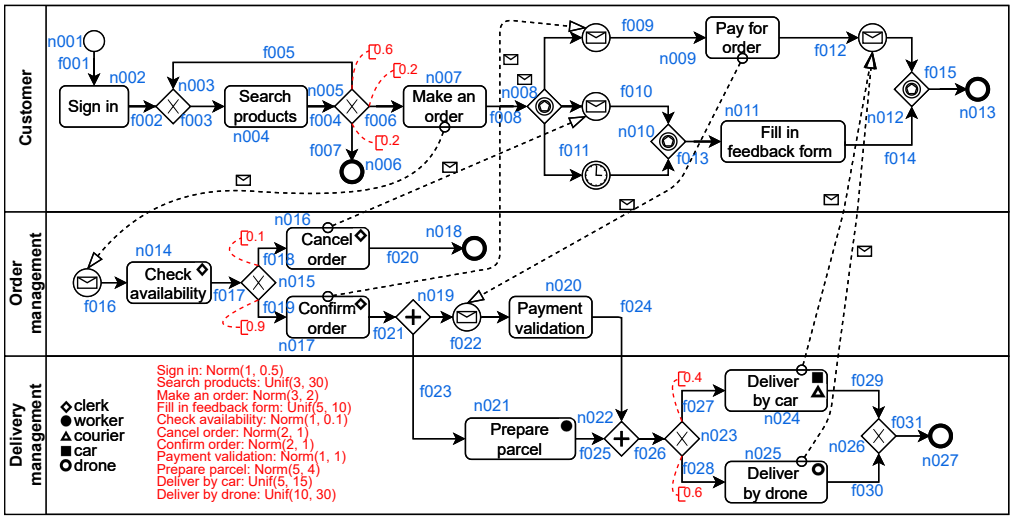

Figure 6: Running example: parcel delivery with node/flow identifiers.

```
< pid : Process |
    nodes :
      (start(id("n001"), id("f001")),
       merge(id("n003"), exclusive, (id("f002"), id("f005")), id("f003")),
       split(id("n005"), exclusive, id("f004"),
         ((id("f005"), 0.6) (id("f006"), 0.2) (id("f007"), 0.2))),
       split(id("n007"), eventbased, id("f008"), (id("f009"), id("f010"), id("f011"))),
       task(id("n020"), "Prepare parcel", id("f023"), id("f025"), Norm(5.0, 4.0),
         id("worker"), empty),
       task(id("n023"), "Deliver by car", id("f027"), id("f029"), Unif(5.0, 15.0),
         (id("car"), id("courier")), id("parcel delivered")),
       task(id("n024"), "Deliver by drone", id("f028"), id("f030"), Unif(10.0, 30.0),
         id("drone"), id("parcel delivered")),
       ...),
    flows :
      (flow(id("f001"), Norm(1.0, 0.2)), ...,
       flow(id("f009"), Norm(1.0, 0.2), message(id("order confirmed"), "order confirmed")),
       flow(id("f010"), Norm(1.0, 0.2), message(id("order canceled"), "order canceled")),
       flow(id("f011"), Norm(1.0, 0.2), timer(id("timeout"), 60)),
       ...) >
```

Figure 7: Running example: Maude representation of the parcel delivery process.

```
class Simulation |
  tokens : List{Token},                ---- scheduler
  gtime : Time,                        ---- global time
  resources : Set{Resource},           ---- resources in the system
  events : Map{Id, Set{Event}},        ---- events in each execution
  process-execs : Map{Id, Time},       ---- execution time of each execution
  sync-times : Map{Id, Map{Id, Time}}, ---- sync. time of each gate in each execution
  task-times : Map{Id, Map{Id, Time}}, ---- task execution times
  ...
```

Figure 8: Declaration of the Simulation class (partial, please, note the ellipsis).

of available events with the corresponding time. Another rule specifies the case of an outgoing flow with a message in the set of events. In that case, this flow is activated by adding one token for that specific flow. Additional objects of classes Workload and Supervisor are in charge of, respectively, modeling the workload of the process, and provisioning resources depending on the whereabouts of the process execution.

In general, rewrite rules operate on systems comprising a Process object, a Simulation object, a Workload object, and a Supervisor object.

Simulation. While process objects represent static processes, and they do not change along simulations, all the information on process execution is kept in simulation objects. Specifically, a simulation object stores a collection of tokens (in a scheduler, see below), a global time (gtime), a set of events (messages and timers), and a set of resources. It also keeps track of the metrics being computed. Figure 8 presents the definition of the Simulation class. Intuition on how these values get updated in the rule is given in Figure 10. For analysis purposes, during the execution of a process some information is collected: time stamps, task durations, and waiting time at parallel and inclusive merge gateways. This information is necessary for guiding the execution of the process and for summarizing results of interest (e.g., waiting times) to the user for possible optimizations.

Tokens. Tokens are used to represent the evolution of the workflow under execution. Since there may be several simultaneous executions of a process, each execution is identified with a unique identifier, and use it to associate tokens to executions. Thus, a token is represented as a term token(Tid, Id, T), where Tid is the execution instance the token belongs to, Id is the identifier of the flow or node it is attached to, and T represents a timer, of sort Time, modeling a delay of the token, which represents the duration of a task or the delay associated to a flow. Once its timer becomes 0, a token can be consumed.

Scheduling. Tokens are stored in a *scheduler* — tokens attribute of the Simulation object in Figure 8 — implemented as a priority queue, so that tokens are processed according to their due time. However, even if a token is at the front of the queue with timer 0, it may be required to delay its execution. For example, consider a task that requires some resource that is not available, or a parallel merge for which some incoming flow is not yet active. To avoid deadlocks, the scheduler implements a *shifting* mechanism that identifies the first active token to the front of the queue in case the current head needs to be delayed.

Events. A message event may be associated to a flow, which is blocked until the message is received. A timer event may also be associated to a flow. When a token arrives at a timer event, its countdown is started: once the countdown is completed, the token moves to the outgoing flow. Both message and timer events are usually associated to event-based gateways, but this is not always the case (see, e.g., the initial flow for the order management lane in the process in Figure 1). Asynchronous events are modeled using an event set in the Simulation object (see line 5 in the code of Figure 8): when a message is dispatched, a corresponding event is added to the set. Flows and gateways waiting for specific messages use this set to check whether messages have arrived. Only inter-lane events are possible; to consider environment events, the environment may be added to the simulation model.

Dynamic resources. For each resource type, a number of instances or replicas are provisioned. At each moment during a simulation, some of these instances can be in use and others can be available for tasks to use them. Section 5 will present different strategies for the dynamic provisioning and releasing of resource instances along executions.

Given a number of provisioned resources, if a running task requires resources and they are available, it blocks them and initiates execution immediately. Indeed, whenever a task requires several resource types, it *atomically* picks them, or waits for all of them to be available. If the required resources are not all available, resource requests are submitted, and the task remains blocked until its requests are satisfied. To support this, each resource type keeps a queue of requests.

Eventually, new resources may be provisioned, i.e., added to the pool of available resources, or released, i.e., removed from such pool. Resource provisioning is not instantaneous however. There might be some delay in the allocation of new instances; this time is referred to as *allocation time* (AT).

Each resource type is represented by a resource operator that gathers all required information (see Figure 9): an identifier, the minimum and maximum num-

```
 1 sort Resource .
 2 op resource :
 3      Id                          ---- Id of the resource
 4      Nat Nat                     ---- range on the number of resources (min, max)
 5      Time                        ---- allocation time
 6      Nat                         ---- total number of replicas
 7      List{2-Tuple{Float, Nat}}   ---- total number of replicas along time
 8      Nat                         ---- number of available replicas
 9      List{2-Tuple{Float, Nat}}   ---- number of available replicas along time
10      Bag{Replica}                ---- replicas of the resource
11      Nat                         ---- size of the resource's queue
12      List{2-Tuple{Float, Nat}}   ---- evolution of the size of the queue along time
13      List{2-Tuple{Float, Float}} ---- resource usage along time
14      Time                ---- time all replicas are in use (reset at supervisor's checks)
15      -> Resource [ctor] .
```

Figure 9: Definition of resources.

ber of allocatable replicas (0, if unlimited), its allocation time, the total number of allocated replicas, the number of available replicas, the total amount of time the replicas of this resource type have been in use, and some historical information on resource usage, request queues, etc., which are handy for analysis purposes.

Tasks. The execution of a task is modeled with two rules. The first rule, the initTask rule shown in Figure 10, represents the task initiation, which is applied when a token with zero time is available at the incoming flow (line 5). If all the resources required by this task are available, which is checked with the allResourcesAvailable function (line 8), then a new token is generated with the task identifier and the task duration (line 12). Otherwise, the shifting mechanism is invoked (line 20) — note the ellipsis. If available, all required resources are removed from the resource set (grabResources function, line 18). Note also that rules update the information on execution times, task durations, etc. (see, e.g., the update of the task-tstamps attribute, lines 13–16).

A second rule, which models task completion, is triggered when there is a token for that task with zero time. In that case, the token is consumed and a new one is generated for the outgoing flow. All resources are released, and all the message events associated to that task, if any, are added to the set of events.

Gateways. Split gateways are triggered when a token arrives in its incoming flow. Depending on the type of gate, tokens are placed in one of the outgoing flows, in several of them, or on all of them. The decision of which flows to activate is taken at random. For event-based gates, when a merge gateway is triggered, the incoming tokens are removed, a new token is added to the scheduler for the

```
 1 rl [initTask] :
 2    < PId : Process |
 3       nodes : (task(NId, TaskName, FId1, FId2, SE, RIds, SEI), Nodes), Atts >
 4    < SId : Simulation |
 5       tokens : (token(TId, FId1, 0) Tks),
 6       task-tstamps : TTSs, gtime : T, resources : Rs, Atts1 >
 7    < CId : Counter | counter : N >
 8 => if allResourcesAvailable(RIds, Rs)
 9    then < PId : Process |
10           nodes : (task(NId, TaskName, FId1, FId2, SE, RIds, SEI), Nodes), Atts >
11         < SId : Simulation |
12           tokens : insert(Tks, token(TId, NId, time(eval(SE, N)))),
13           task-tstamps : if TTSs[TId][NId] == undefined
14                          then insert(TId, insert(NId, T, TTSs[TId]), TTSs)
15                          else TTSs
16                          fi,  ---- for loops, stamps get overwritten
17           gtime : T,
18           resources : grabResources(RIds, Rs, time(eval(SE, N)), T), Atts1 >
19         < CId : Counter | counter : int(eval(SE, N)) >
20    else ...                    ---- if necessary, the scheduler is updated
21    fi .
```

Figure 10: Task initiation rule.

outgoing flow, and simulation information is updated with synchronization times. For inclusive gateways, the semantics of BPMN 1.0 and 2.0 are both supported in this specification. A detailed description of the different rules handling the different types of gateways can be found in [7].

Supervisor. The Simulation object is in charge of collecting the data on the chosen metric for the specified window of time (history length). The supervisor then analyzes the collected information and, if necessary, decides to update (increase or decrease) the number of resource instances. More details on the supervisor class and its resource-handling strategies are discussed further in Section 5.

Workloads. Simulation-based analysis techniques are typically parameterized by the workload. A workload defines the rate at which new instances of a given process are executed. The rule in Figure 11 specifies the behavior of closed workloads, that is, the workload declares a fixed number of tokens or works to be injected in the process, corresponding to the number of times the process is to be executed. Given a number of works and a stochastic expression SE describing the inter-arrival time (kept in the rate attribute), the rule generates a new work after the specified amount of time until all works have been created. Note that the timer attribute of the Workload object is initialized with the result of evaluating the stochastic expression (line 5). The rule is applicable when the timer becomes 0, and then a new token in the initial node is inserted in the scheduler (line 7). The

```
 1 rl [Workload] :
 2    < WId : Workload | timer : 0, rate : SE, works : s W >
 3    < SId : Simulation | tokens : Tks, events : ME, Atts1 >
 4    < CId : Counter | counter : N >
 5 => < WId : Workload | timer : time(eval(SE, N)), rate : SE, works : W >
 6    < SId : Simulation |
 7        tokens : insert(Tks, token(token(s W), initial, 0)),
 8        events : (token(s W) |-> empty, ME),
 9        Atts1 >
10    < CId : Counter | counter : int(eval(SE, N)) >
```

Figure 11: Workload rule.

evaluation of stochastic expressions is done by the eval operation.

Throughout the specification, random numbers are generated using a pseudo-random number algorithm. As presented in Section 3.3, Maude provides such a generator as a mapping from natural numbers to integers. A Counter object (lines 4 and 10 in Figure 11) appropriately increases the numbers to be used as keys.

5. Dynamic Resource Allocation

During the execution of a process, resource instances are taken from a pool of resources on demand. Note that the specific number of instances in this pool may change along time in order to accommodate to specific needs. That is, the amount of instances in this pool must be continuously, or at least periodically, re-evaluated. The instances in this pool can then be in use by some of the tasks in the process, or available and ready to be used. This section presents several possible strategies for resource provisioning. Then, Section 6 provides mechanisms to simulate these provisioning strategies, thus enabling a quantitative comparison between them.

The size of the pool of provisioned resources may or may not be limited. For instance, consider a cloud application. At any time, any number of virtual machines may be provisioned: there is a potentially unlimited number of virtual machines at disposal in case of need. That is, new virtual machines may be incrementally provisioned for their use. The only limit to this number, theoretically at least, is the amount of money to be spent on them. However, the number of cars in the pool may be limited by the parking space or by the provider. Similarly, the number of employees may be limited by the number of counters or office space on premises.

The development presented next assumes that, independently of the criteria used for provisioning/releasing resources, the amount of resources is periodically

re-evaluated. The decision of whether increasing or decreasing the number of available resource instances is evaluated every TBC time units (time between checks). TBC does not need to be fixed because each strategy decides on how frequently to carry on these checks. Provisioning strategies may also take the length of the history of the resource to be considered in the evaluation as a parameter (HL). For instance, to avoid spurious decisions, based on one-off events, the decision on how to proceed may be based on the average of several recent events. By setting HL to zero, strategies are forced to operate without considering the execution history. Negative values are not valid for HL. Other strategies may as well base their decisions on predictions instead of using the resource history (this is illustrated below).

Given this general approach, the presentation takes place as follows. In section 5.1, an abstract class Supervisor defines a general scheme for the definition of strategies. Then, different strategies are defined by providing different subclasses of this one. Although there is no constraint on the definition of new strategies, other than operating on data collected along the simulations, a similar scheme is followed (see sections 5.2–5.5). Thus, in order to define a new strategy, the user needs to add a new subclass of Supervisor. This class should define additional attributes to keep the information necessary for such strategy, e.g., thresholds, buffer bounds, times, etc. Then, a rule will periodically evaluate the amount of resources and will act accordingly. A similar pattern is followed by all the strategies: an update operation is invoked for defining how resources are allocated.

5.1. A generic supervisor

The definition of the Supervisor class below shows the generic structure of supervisors with three attributes: the first attribute defines the time between two consecutive checks, the second one is a timer to trigger the next check, and the third one defines the size of history to be considered in the decision.

```
class Supervisor | time-between-checks : Time,
                   time-to-next-check : Time,
                   check-interval : Time .
```

The general procedure for the provisioning and releasing of resource instances is decided in accordance to some given thresholds, which are also provided as parameters. The algorithm periodically performs the following check: if the value of the considered property is greater that the upper-bound threshold, a new instance of the resource is allocated to the set of available resources; if it is smaller than a lower bound, an instance is removed so that it is no longer available for use. These thresholds are provided by the user before initiating the analysis, depending on the

specific strategy selected and the overall goals. For instance, if a strategy makes decisions in accordance to the resource usage, as the one in Section 5.2, at the beginning of the analysis it can be specified that a new replica of a resource needs to be added if its usage percentage goes over 95%, or remove a replica if it goes under 50%.

Resources are not allocated instantly: an additional parameter, allocation time (AT), indicates the time required to allocate a new instance. Thus, once a strategy takes the decision of allocating a new instance of a given resource, a lapse of time AT passes before such new instance becomes available.

The following subsections present four different specializations of the Supervisor class to control the allocation of resources. Sections 5.2 and 5.3 present strategies that are guided by the usage of resources and by the sizes of the queues of pending requests, respectively. Section 5.4 presents a strategy that bases its decisions on predictions performed using the probabilistic model of the process under consideration. The strategy in Section 5.5 shows how to combine several strategies, particularly illustrating with the joint use of usage and queue-based strategies.

5.2. Usage-based strategy

The *usage-based strategy* takes into account the recent usage of a resource to decide on whether to allocate new instances of such resource or release some of the allocated ones. Specifically, the following check is periodically performed on every resource type. If the average usage of the given resource is above the given upper-limit threshold and the maximum number of instances allowed has not been reached, then a new instance is allocated for that specific resource. Similarly, if the average is under the given lower-limit threshold, there is an available instance, and the number of instances is above the minimum number of instances, then one instance is released.

The SupervisorUsage class extends the class Supervisor. It defines a new attribute thresholds for storing the data on thresholds as a map that associates a pair with the lower and upper values triggering the strategy to each resource identifier. As an example, if a specific resource defines $[50\%, 70\%]$ as thresholds, it means that if the average usage for the last period of time CI is less than 50%, a replica can be released. Conversely, if, in average, the replicas are used above 70%, another replica should be added.

```
class SupervisorUsage | thresholds : Map{Id, Tuple{Float, Float}} .
subclass SupervisorUsage < Supervisor .
```

```
 1 rl [supervisor] :
 2    < SId : Simulation | resources : Rs,
 3                         gtime : T,
 4                         Atts1 >
 5    < Sup : SupervisorUsage | time-between-checks : TBC,
 6                              time-to-next-check : 0,
 7                              check-interval : CI,
 8                              thresholds : Thds,
 9                              Atts2 >
10 => < SId : Simulation | resources : update(Rs, Thds, CI, T),
11                         gtime : T,
12                         Atts1 >
13    < Sup : SupervisorUsage | time-between-checks : TBC,
14                              time-to-next-check : TBC,
15                              check-interval : CI,
16                              thresholds : Thds,
17                              Atts2 >
```

Figure 12: Usage-based strategy supervisor rule.

Figure 12 shows the rule governing the strategy. Every TBC time units, the supervisor object updates the number of resource instances (line 10) according to the state of the resources (Rs), the thresholds (Thds), the interval of time to consider (CI), and the current global time (T). Notice that the rule is applied when the time-to-next-check attribute has value 0 (line 6) and it is set to the value of TBC on the right-hand side of the rule thus scheduling its next application (line 14). The actual update of the resources is the result of the update operation (line 10). Also notice that the remaining attributes of the Simulation and SupervisorUsage objects (represented by variables Atts1 and Atts2, respectively) are not updated.

Note that the scheduling of a next check for the usage-based strategy, as specified in the rule in Figure 12, is done for a fixed amount of time. This does not need to always be the case. For example, a strategy may change the time between its checks depending on its log. The TBC is just an attribute of these objects. If there have been some time without changes, it may increase its TBC, or decrease it in times of instability.

The specification of the update operation for the usage-based strategy is shown in Figure 13. It performs the previously described check on each resource type. Each resource keeps track of the amount of time its instances have been in use. This information, however, is collected in two different places for the precise computation of the values of usage. Since the usage percentage is calculated as the quotient between the total amount of time all instances are in use and the total amount of elapsed time, it is required to keep separate values every time the number of instances of a resource changes. The last argument of the resource operator (see Figure 9) accumulates the total amount of time all the resource instances have

been in use since the last check or change in the number of instances. However, since resource checks may happen while instances are in use, the total amount needs to be added with the time instances have been in use. For that purpose, each resource type keeps an explicit representation of its instances (line 10 in Figure 9), which is either idle (i.e., available) or has the form replica(TId, NId, T), where TId is the identifier of the current execution, NId the identifier of the task currently using the resource, and T is the time at which the replica was allocated to be used by such task, or the time at which the last resource check happened. The total amount of time since the last check is then computed in line 32; the used operator gathers the amount of time each replica has been in use. This value is reset to the current time using the reset operation (lines 21 and 26).

The sequence of usage values along time is kept as a list of tuples (time, usage). The current sequence is matched to SeqUsage (LUpd, LUsg) in line 4, what allows us to get the time of the last update (LUpd) and its usage value (LUsg). These values are then used to compute the usage in the last period (line 33) and the average use in the considered window of time with the averageUsage operation (lines 34–35).

Given the calculated value for the average usage (AvgUsage), the resources may then be updated. If the average usage of the given resource is above the given upper-limit threshold and the maximum number of instances allowed has not been reached, then a new instance is allocated for that specific resource (lines 11-16). Note that a new instance is allocated with delay AT (delayed(idle, AT)). The actual creation of the instance takes place when this timer goes off. If the average is under the given lower-limit threshold, there is an instance available, and the number of instances is above the minimum number of instances, then an instance is released (lines 17–24). Then, the total number of instances and the number of available instances is decremented (lines 19–20), and the idle replica is removed using the rmIdle operation (line 21). Otherwise, the usage value is added to the sequence of values and the replicas are reset (lines 25–28).

Note that the increment or decrement of the number of replicas is carried out one by one, and one may have to wait until the next check for a modification. This does not have to always be the case. An alternative may be, for example, such that changes are carried out depending on the degree of non-satisfaction of certain requirement. For example, it could define an alternative update operation in which, if the increment of AvgUsage is between 0% and 10% it increases the number of replicas in 1, but if it is in the range 10–20% the increment is in 2 units, etc.

When the delayed-creation timer gets off, the equation in Figure 14 does not

```
1  op update : Set{Resource} Map{Id, 2-Tuple{Float, Float}} Time Time Time -> Set{Resource} .
2  ceq update(
3        (resource(RId, Min, Max, AT, Ttl, SeqTtl, Avl, SeqAvl, Replicas, QSize, SeqQSize,
4                  SeqUsage (LUpd, LUsg),
5                  T),
6         Rs),
7        ((RId |-> (ULL, UUL)), Thds),
8        TBC,
9        CI,
10       T')
11    = if (Max == 0 or Ttl < Max) and-then AvgUsage > UUL
12      then resource(RId, Min, Max, AT, Ttl, SeqTtl, Avl, SeqAvl,
13             (Replicas ; delayed(idle, AT)),
14             QSize, SeqQSize,
15             SeqUsage (LUpd, LUsg),
16             T)
17      else if (Avl > 1 and Ttl > Min) and-then AvgUsage < ULL
18           then resource(RId, Min, Max, AT,
19                         Ttl - 1, SeqTtl (float(T'), Ttl - 1),
20                         Avl - 1, SeqAvl (float(T'), Avl - 1),
21                         rmIdle(reset(Replicas, T')),
22                         QSize, SeqQSize,
23                         SeqUsage (LUpd, LUsg) (float(T'), Usage),
24                         0)
25           else resource(RId, Min, Max, AT, Ttl, SeqTtl, Avl, SeqAvl,
26                         reset(Replicas, T'), QSize, SeqQSize,
27                         SeqUsage (LUpd, LUsg) (float(T'), Usage),
28                         0)
29           fi
30      fi,
31      update(Rs, Thds, TBC, CI, T')
32    if T'' := (T + used(Replicas, T'))
33    /\ Usage := float(100 * T'') / (float(Ttl) * (float(T') - LUpd))
34    /\ AvgUsage := averageUsage(SeqUsage (LUpd, LUsg) (float(T'), Usage),
35                     Ttl, Avl, T' monus CI) .
36 eq update(empty, Thds, TBC, CI, T) = empty .
```

Figure 13: Specification of the update operation for the usage-based strategy.

only remove the delayed operator (see lines 3 and 12), but it also updates the total and available resources, computes the new usage value, inserts a new pair in the sequence of usage values (line 14), and resets the rest of the instances (line 12).

5.3. Queue-based strategy

Recall that every time a token arrives at a task, it requests the necessary resources for its execution. If the necessary instances of the required resources are available, they are directly taken. Otherwise, requests for such resources are enqueued and the task is waiting to be executed. When the resources become available, the instances are taken by the task for its execution and the requests

```
ceq < SId : Simulation |
        resources : (resource(RId, Min, Max, AT, Ttl, SeqTtl, Avl, SeqAvl,
                             (delayed(idle, 0) ; Replicas),
                             QSize, SeqQSize, SeqUsage (LUpd, LUsg), T),
                     Rs),
        gtime : T',
        Atts >
  = < SId : Simulation |
        resources : (resource(RId, Min, Max, AT,
                             Ttl + 1, SeqTtl (float(T'), Ttl + 1),
                             Avl + 1, SeqAvl (float(T'), Avl + 1),
                             (idle ; reset(Replicas, T')),
                             QSize, SeqQSize,
                             SeqUsage (LUpd, LUsg) (float(T'), Usage),
                             0),
                     Rs),
        gtime : T',
        Atts >
   if T'' := (T + used(Replicas, T'))
   /\ Usage := float(100 * T'') / (float(Ttl) * (float(T') - LUpd)) .
```

Figure 14: Actual allocation of a new instance.

are removed from the corresponding queues. The queuing procedure for resource requests allows us to have information on the resource demand.

The *queue-based strategy* follows a scheme very similar to the usage-based one. Indeed, the rule governing its execution completely mimics the one in Figure 12. The update operator, however, has a different definition in this case. Although its structure is quite similar, the update operator for this strategy checks whether the average number of pending requests on the given resource is inside the thresholds range, for every resource type.

The SupervisorQueues class extends the Supervisor class and defines a new attribute thresholds. In this case, the threshold values are of type Nat.

```
class SupervisorQueues | thresholds : Map{Id, 2-Tuple{Nat, Nat}} .
subclass SupervisorQueues < Supervisor .
```

5.4. Prediction-based strategy

Making decisions based on the analysis of risks and predictions is common in today's businesses. This section presents a strategy that decides on whether increasing or decreasing the number of allocated instances by looking to a prediction of the state of its resources after some time. The amount of time to look ahead into the future is provided as an additional parameter of the analysis (look-ahead time, LAT).

As for other strategies, the need for provisioning or releasing resources is evaluated periodically. Intuitively, in this case the evaluation consists in predicting and analyzing the state of the resources after some time, in order to deduce the need of resources in the future. As explained in Section 2, the annotated process models are used, not only to simulate the behavior of the processes, but also to predict its expected behavior by looking in advance at the following potential steps of the simulation.

This is achieved by making a copy of the process and simulation environment, and by running this copy for the specified amount of time. Then, the need of resources may be evaluated using different criteria. In this case, it is key to identify the usage of the resource; as for other strategies, upper and lower-limit thresholds are used. Consider some specific resource, for which the thresholds given are $(20\%, 95\%)$. If the maximum number of instances has not been reached and in the predicted state the usage of the resource is over 95%, a new instance is scheduled for allocation. Similarly, an instance is released if there is an idle instance, the minimum number of instances has not been reached, and in the predicted state the usage of the resource is under 20%.

The look-ahead time and the thresholds for each resource are kept as attributes of the SupervisorPrediction class, which is defined as a subclass of the Supervisor class as other strategy classes.

```
class SupervisorPrediction |
        thresholds : Map{Id, Tuple{Float, Float}},   ---- usage thresholds
        look-ahead-time : Time,
        forked-state : Maybe{System} .
subclass SupervisorPrediction < Supervisor .
class Timer | remaining-time : Time .
```

Notice that the class also has an attribute forked-state of type Maybe{System}, which corresponds to the copy of the whole system. The Maybe{System} is a parameterized type, which is defined as a supersort of System, so that it is either null or a term of sort System. Class Timer defines a timer used to stop the execution of the predictive simulation at the specified time.

The strategy is then defined by the two rules shown in Figure 15. As for the other strategies, the periodic evaluation of the resources happens when the time-to-next-check attribute has value 0 (line 6). The supervisor-initiate-prediction rule (lines 0–23) models the action of initiating the simulation by making a copy of the current state of the simulation in its forked-state attribute (lines 17–21). Notice that in addition to the Simulation, Process, Counter, and Workload objects, a Timer object is added (line 21), with remaining-time set to the look-ahead time of the supervisor object. The supervisor-prediction-completed rule (lines 25–47) is

applied when the prediction is completed, i.e., the timer gets to zero (line 34). As for other strategies, an update operator with appropriate arguments is in charge of implementing the above-described corresponding actions.

5.5. Combined strategy

The interest of combining different strategies is to use different sources of information (e.g., usage, queues, prediction), and thus provide more accurate choices when deciding to add or release some specific instances of resources. Even using the same common scheme for all the strategies presented in this work, there are several indicators that can be combined in different ways. The decision of updating resources can be taken when these indicators are in agreement, when at least one of them recommends an update in the number of instances, or when a majority recommends an update. In the rest of this section, for illustration purposes, a combination of the strategies introduced in Sections 5.2 and 5.3 is presented. In this case, a decision is taken if both strategies are in agreement. Note also that this combination aims at providing additional stability, gathering the advantages of both strategies, and reducing their disadvantages. Using rules similar to the one in Figure 12, the update operator is redefined by combining the computations for both indicators. More precisely, this operator computes the averages of the resource usages and request queues' sizes for the given check intervals and compares them against the thresholds for each of the criteria.

The SupervisorUsageQueues class extends the class Supervisor and defines two new attributes corresponding to the thresholds required for the usage and queue-based strategies, respectively.

```
class SupervisorUsageQueues |
        uthresholds : Map{Id, Tuple{Float, Float}},   ---- usage thresholds
        qthresholds : Map{Id, 2-Tuple{Nat, Nat}} .    ---- queues thresholds
subclass SupervisorUsageQueues < Supervisor .
```

The next section presents results on the use of this and the previously described strategies, compares their results, and provides some insights on their use.

6. Evaluation

Whilst the usage-based strategy is guided by the recent use of the resources, the queue-based strategy bases its decisions on the state of the requests queues. At any moment in time, the queues' sizes provide indirect information on the resource demand: it is reflected in the usage of the resources in a timespan after that moment. However, although current values of the demand may provide insights

```
crl [supervisor-initiate-prediction] :
   < SId : Simulation | Atts1 >
   < PId : Process | Atts2 >
   < CId : Counter | Atts3 >
   < WId : Workload | Atts4 >
   < Sup : SupervisorPrediction |
       time-to-next-check : 0,     ---- check is due
       forked-state : null,
       look-ahead-time : T,
       Atts5 >
=> < SId : Simulation | Atts1 >
   < PId : Process | Atts2 >
   < CId : Counter | Atts3 >
   < WId : Workload | Atts4 >
   < Sup : SupervisorPrediction |
       time-to-next-check : 0,
       forked-state :              ---- a copy of the state is used to predict
         { < SId : Simulation | Atts1 >
           < PId : Process | Atts2 >
           < CId : Counter | Atts3 >
           < WId : Workload | Atts4 >
           < t : Timer | remaining-time : T > }, ---- prediction time
       look-ahead-time : T,
       Atts5 > .

rl [supervisor-prediction-completed] :
    < SId : Simulation | resources : Rs, gtime : T, Atts1 >
    < Sup : SupervisorPrediction |
        time-between-checks : TBC,
        time-to-next-check : 0,
        check-interval : CI,
        thresholds : Thds,
        forked-state :
          { < SId : Simulation | resources : Rs', gtime : T', Atts3 >
            < t : Timer | remaining-time : 0 >
            Conf },
        Atts2 >
 => < SId : Simulation |
        resources : update(Rs, Rs', Thds, TBC, CI, T, T'),
        gtime : T,
        Atts1 >
    < Sup : SupervisorPrediction |
        time-between-checks : TBC,
        time-to-next-check : TBC,
        check-interval : CI,
        thresholds : Thds,
        forked-state : null,
        Atts2 > .
```

Figure 15: Prediction-based strategy supervisor rule.

on the evolution of the demand, they may turn out to be irrelevant for future allocation. A good prediction on the behavior of the processes and resources can lead to a significant gain regarding resource allocation.

Despite the initial intuition that has been developed on the expected results when using each of the strategies, the experiments presented in this section provide a finer understanding of how they really work in practice. There are different parameters that influence their effectiveness. Furthermore, there are different constraints that may be key to consider, as the objective function, the cost of the resources, the allocation time for them, or the workloads. Certain choice of parameters may work well for some of the strategies and badly for others. This section includes experimental results with the goal of acquiring a better comprehension on how the strategies behave in practice. Analyzing the results obtained on specific examples enables a better understanding of the results of the simulations in other cases, and thus ultimately help to decide what strategy should be used in a specific situation. The strategies for which concrete parameters are extensively explored, are then used and compared on two different examples; insights are developed to suggest how the processes could be improved.

As far as automation of the evaluation is concerned, once all the parameters necessary as input have been defined (workload, allocation time, costs, resource ranges, and some other parameters related to the strategies), the results of the evaluation are computed in a fully automated way. A Python script takes the outputs of the simulation generated by Maude and transforms them into CSV format, which can then be processed to graphically depict them.

6.1. Understanding the different strategies and their parameters

Any business process operates under certain conditions, like the prices of the resources and their allocation times — lapse of time between a new instance is requested and it becomes available — and a certain workload. In the following, results obtained from the analysis of the strategies implemented for the delivery example are presented. These results are used to discuss the advantages and disadvantages of each of the strategies, and on the impact of the different parameters on its use on the case study.

The first focus is on the time between checks. Intuitively, continuously checking the situation should result in better results. However, this might not lead to the optimal solution. Decisions on resource provisioning need time to have its intended effects; also, taking measures too often may result in more operations than required, and even in higher costs. In some cases, it may be convenient to let

the system stabilize before making further changes. This is even more important if the provisioning of new instances is not instantaneous.

Something similar may happen with the check interval. Considering a larger window in the decisions may be beneficial for the system's stability, but it also may bring delays in the implementation of required adjustments if it is too large. Then, what are the appropriate values for these and the other parameters of the different strategies?

Let us consider the delivery process presented in Section 2. In the process, five different resources are used, which are identified as clerk, worker, courier, car, and drone. Their allocation times are: 5 time units for clerks, 4 for workers, 3 for couriers, 2 for cars, and 3 for drones. Their costs are: 60 units (let us say euros) per time unit for clerks, 50 for workers, 40 for couriers, 20 for cars, and 30 for drones. In some cases, simulations are used for gathering information on the current conditions, with explicit constraints; in other cases, the proposed analysis tools are used to evaluate potential improvements for the enterprise. This first experiment considers both constrained and unconstrained resource ranges for each of the resources. When constrained, the ranges used are $[1, 3]$ for clerks, $[1, 2]$ for workers, $[1, 4]$ for couriers, $[1, 8]$ for cars, and $[1, 4]$ for drones. When unconstrained, ranges are specified as $[1, 0]$. It is assumed that the workload follows an exponential distribution with average $\lambda = 5$. Each simulation has been run on 1000 instances.

In addition to the aforementioned context-given parameters, in this first experiment, the usage-based strategy is instantiated with the following parameters: TBC = 10, CI = 10, unlimited instances, and thresholds $(40, 80)$. The results produced by the simulations indicate that the average execution time was 73.10, with a variance of 2.54. The total cost amounts to 989 754.58€, with usage percentage per resource type of 47.41% for clerks, 48.80% for workers, 43.20% for couriers, 42.51% for cars, and 53.27% for drones. The tool also provides graphical representations of the evolution of the usage for each of the resources, of the evolution of the sizes of the requests queues, and of the evolution of the number of instances along time. For illustration purposes, the charts in Figure 16 show the number of instances (on the left) and usage percentage (on the right) along time for each resource type.

It can be observed in Figure 16, that the process is using either one or two clerks most of the time, sporadically taking a third one, and only once using an extra fourth one. This evolution matches the graph on its right. Since the thresholds are defined as $(40, 80)$, a new instance is allocated when the resource usage goes over 80%. Notice that, about time 1 800, the demand becomes higher and,

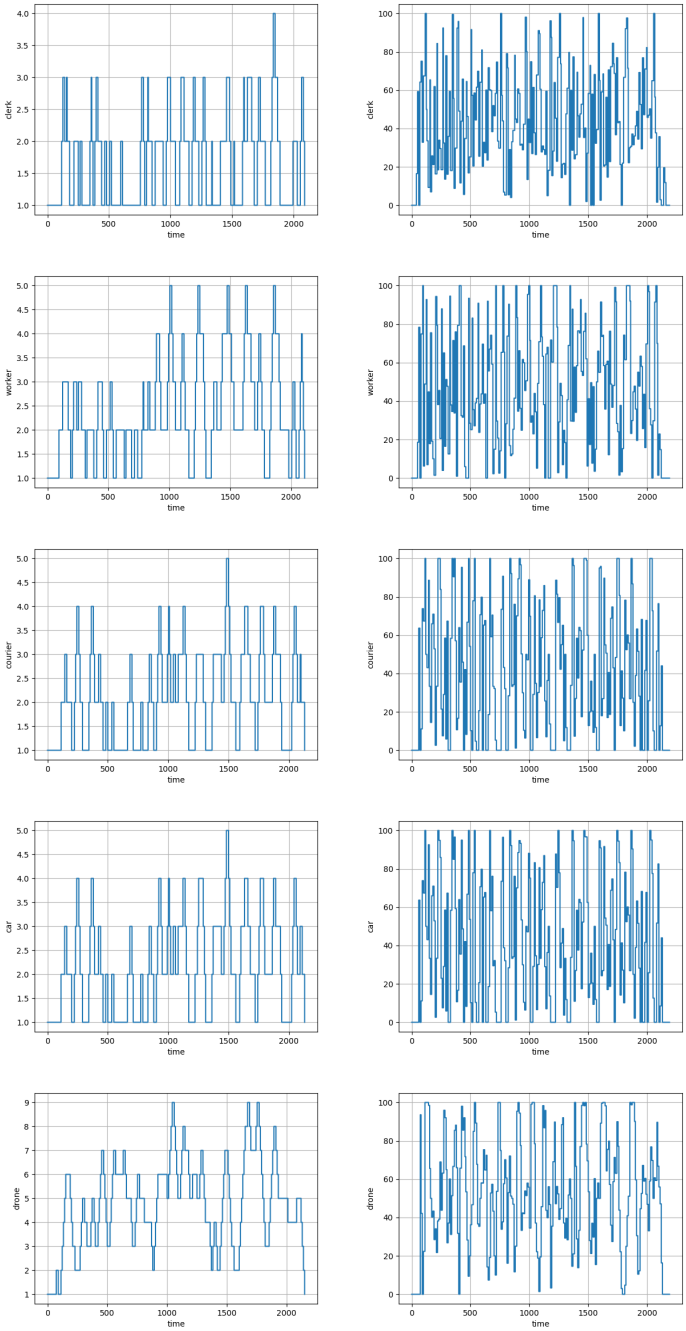

Figure 16: Number of instances (left) and usage percentage (right) for each resource type for a simulation with the usage strategy, TBC=10, CI=10, and Thds=(40, 80).

then, a fourth instance of clerk is added for a while. The lower limit works similarly, by removing instances when the usage percentage goes below 40%. The demand and number of active clerks are the two factors that make the number of instances continuously go down to one and back up again to two and sometimes three. In the case of the worker resource, it goes up to five instances several times, but immediately goes down again to four instances.

Recall that even if a new instance is allocated because of a real need, its allocation takes some time, meaning that when the instance becomes available it may no longer be required, and thus it is immediately released. This suggests that waiting a little bit to take the decision or directly preventing it, may improve its usage numbers. To do so, either the threshold values or limit the number of instances can be adjusted. Increasing parameter CI might also provide a better stability. However, there are other considerations to take into account. There is a trade-off between execution time and cost: Although by getting better usage percentages, costs may be reduced, but execution times may increase.

When looking at the other resources, similar conclusions can be reached. What is clear in any case is that charts suggest too much instability: newly allocated instances are used for very short periods of time before releasing them.

6.2. Towards the best combination of parameters

Let us consider now the following second experiment. The charts in Figure 17 correspond to the experiment that uses the predictive-usage strategy, with TBC=5, LAT=10, and thresholds (60, 90). As showcase below, this simulation is also included in the comparison presented in Table 2. In this case, the usage percentages are between 43% and 70%, with an average execution time of 67.91 and a total cost of 673 013€. It is worth noting that other strategies provide higher percentages of resource usage, but with longer execution times and higher total costs. Even though the time between checks is established at 5, the look-ahead time does not only provide information on what resources will be needed in the future, but also prevents from the allocation of excessive instances. For example, one or two clerks are necessary most of the time, this number increases to three clerk instances only in four occasions during the entire simulation. Similarly, a more cautious allocation applies for the other resources as well, all of them exhibiting a lower number of instances and better usages. The fact that the threshold values are greater also helps: instances are released only if usage goes below 60% and new ones are allocated only if usage goes over 90%.

Although experience playing with these parameters is key to find good parameters, finding the best ones is complex without a systematic search. This is a

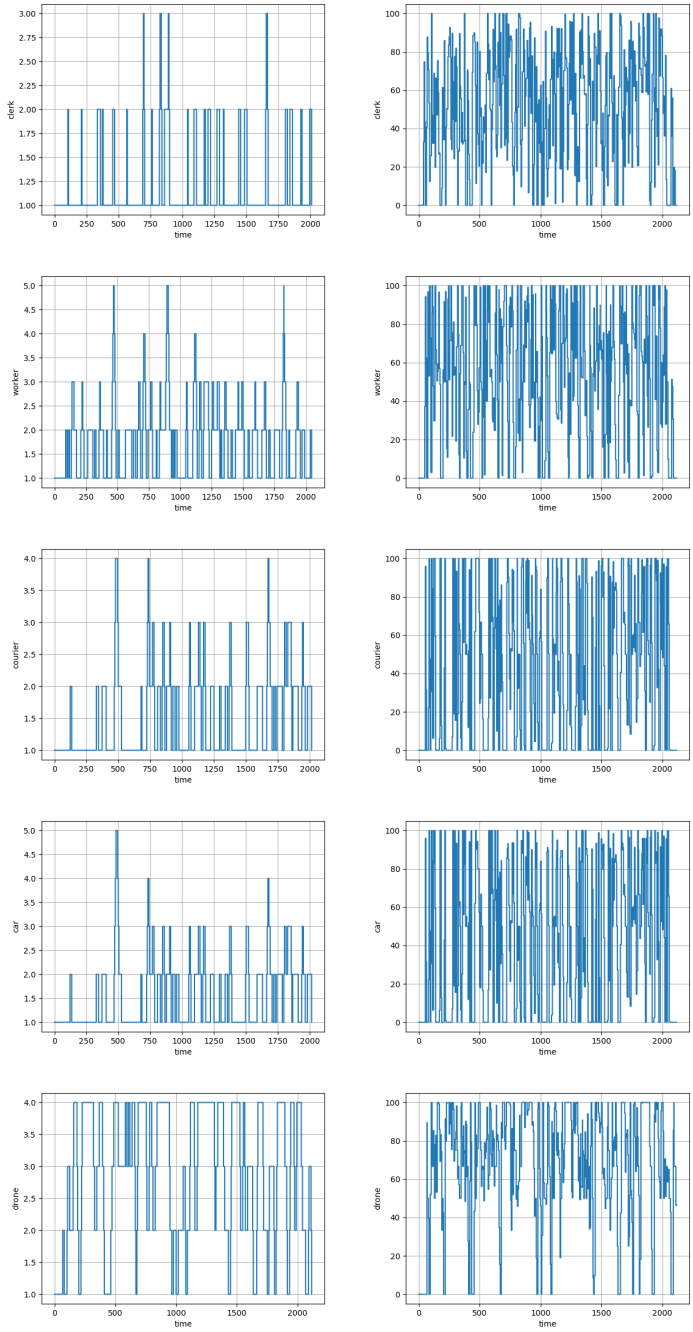

Figure 17: Number of instances (left) and usage percentage (right) for each resource type for a simulation with the predictive-usage strategy, TBC=5, LAT=10, and Thds=(60, 90).

	Execution time (h)		Cost (€)
	Avg	Var	
average	76.78	0.31	993 102.08
minimum	67.91	0.20	569 268.35
maximum	223.42	4.03	37 134 644.69
median	74.25	0.26	858 280.50

Table 1: Average, minimum, maximum and median values of the simulations for the delivery example.

classical multi-objective optimization problem and it has to be faced as such. A total of 6 912 simulations have been run. The usage-based strategy has been run with 768 combinations of parameters: TBC values 5, 10 and 15, and check intervals of 5 and 10. For each resource, an unlimited number of instances and a fixed limit are considered. Specifically, 0 and 3 for clerks, 0 and 2 for workers, 0 and 4 for couriers, 0 and 8 for cars, and 0 and 4 for drones. Finally, the same thresholds are considered for all resource types, picked between 40 and 60 as lower-bound threshold, and 70 and 90 as upper bound. The same combinations are considered for the queue-based strategy, although in this case the alternatives for threshold bounds are 1 and 2, and 3 and 5. Notice that this also gives 768 combinations. The same values are considered for the usage-queues strategy, but since in this case thresholds for both usage and queue sizes must be considered, the number of combinations goes up to 3 072. As for the predictive-usage strategy, the same values are considered, but notice that in this case only thresholds for usage values are considered, no history is considered, and an additional parameter specifying the look-ahead time (LAT) must be considered. The values for the LAT parameter used as alternatives are 5, 10, and 15. This gives a total of 2 304 different combinations. Table 1 shows the minimum, maximum and average values of the execution time average, variance and cost of all the simulations, including the results for all four strategies and all combinations of parameters above described. Note that although there is a great difference between the minimum and the maximum values both for the execution times and costs, the average remains quite close to the minimum value.

6.3. A multi-objective problem

Given all these simulations and corresponding results, the problem of finding the best solution is then modeled as an optimization problem. Given the terms $Cost^{min}$ and $Cost^{max}$ that denote the minimum and maximum values of cost for the different parameter combination (and correspondingly $ExcTime^{min}$ and

$ExcTime^{max}$ for the execution times), the utility functions normalize the corresponding values. Then, the following objective function can be used to select the candidate combination of parameters that miminizes the aggregated utilities:

$$\sum \left(\omega_{Cost} * \frac{Cost - Cost^{min}}{Cost^{max} - Cost^{min}} + \omega_{ExcTime} * \frac{ExcTime - ExcTime^{min}}{ExcTime^{max} - ExcTime^{min}} \right)$$

Table 2 shows the results for some significant combinations of parameters. Although the predictive strategy in general shows the best results, the two best results for each of the strategies are selected, with the goal of discussing the advantages of each of them. Notice for example that the best execution time is obtained by the predictive strategy (row 1). However, the best usage percentages are shown by the simulations using the queue and usage-queues strategies (see rows 3 and 8). This is translated into lower costs: the simulation in row 8 provides the lowest global cost of all the performed simulations. The best solutions to the multi-objective problem (using weights 0.6 and 0.4) are provided by the predictive-usage strategy, with the usage-based one showing the second best results. This means these strategies, for appropriate combinations of parameters, are able to provide optimal execution times showing good usage percentages and therefore costs.

To sum up, for this particular process, the prediction-based strategy provides the best results regarding execution times, followed by the usage-based strategy. As far as costs are concerned, it is the queue-based strategy that delivers the best results, followed by the usage-queues strategy. The reason for this is that they manage to get a better use of the resources. However, when looking at execution times and costs jointly, the prediction-based strategy is definitively the one giving the best results.

6.4. Recruitment Example

The BPMN model in Figure 18 specifies the recruitment process of some company. Candidates fill up requested forms, have a medical check-up, and, if necessary, apply for a visa. Then, they submit the required documents. If rejected, they can re-apply. Minor problems on the presented documentation may be amended by submitting additional documentation. The presented documentation is checked by human resources (HR) officers, who decide to either reject, accept or partially accept the presented documentation. If accepted, either through a direct accept or after completing the pending documentation, some wages are anticipated, a welcome kit is prepared, and the corresponding information in the company's DB are updated. Regarding resources, all tasks related to documentation checking and

	Strat	TBC	CI	LAT	Resources					Usage	Exec time (h)		Cost (€)
					Name	Range	AT	Cost	Thrd	(%)	Avg	Var	
1	predictive-usage	5	-	10	clerk	[1, 0]	5	60	(60, 90)	52.84	67.91	2.04	673 013
					worker	[1, 0]	4	50		57.18			
					courier	[1, 4]	3	40		44.33			
					car	[1, 0]	2	20		43.28			
					drone	[1, 4]	3	30		70.45			
2	predictive-usage	5	-	15	clerk	[1, 0]	5	60	(40, 70)	46.49	67.93	2.04	853 492
					worker	[1, 0]	4	50		47.24			
					courier	[1, 0]	3	40		38.49			
					car	[1, 0]	2	20		37.32			
					drone	[1, 0]	3	30		51.28			
3	queues	10	5	-	clerk	[1, 0]	5	60	(2, 5)	67.48	82.01	3.51	586 781
					worker	[1, 2]	4	50		71.60			
					courier	[1, 0]	3	40		76.34			
					car	[1, 0]	2	20		76.32			
					drone	[1, 4]	3	30		85.59			
4	queues	5	5	-	clerk	[1, 0]	5	60	(2, 5)	65.80	81.97	3.51	574 150
					worker	[1, 2]	4	50		77.50			
					courier	[1, 0]	3	40		63.01			
					car	[1, 0]	2	20		63.01			
					drone	[1, 4]	3	30		85.35			
5	usage	5	10	-	clerk	[1, 3]	5	60	(40, 70)	42.03	68.71	2.11	1 015 811
					worker	[1, 0]	4	50		44.91			
					courier	[1, 0]	3	40		40.27			
					car	[1, 0]	2	20		39.05			
					drone	[1, 0]	3	30		48.32			
6	usage	5	5	-	clerk	[1, 0]	5	60	(60, 90)	49.99	68.55	2.10	831 476
					worker	[1, 0]	4	50		56.12			
					courier	[1, 0]	3	40		48.61			
					car	[1, 0]	2	20		47.45			
					drone	[1, 0]	3	30		64.25			
7	usage-queues	15	15	-	clerk	[1, 0]	5	60	(60, 90) (2, 5)	67.15	87.98	4.25	569 365
					worker	[1, 0]	4	50		79.15			
					courier	[1, 0]	3	40		87.64			
					car	[1, 0]	2	20		87.64			
					drone	[1, 0]	3	30		82.96			
8	usage-queues	5	10	-	clerk	[1, 0]	5	60	(60, 90) (2, 8)	68.76	89.33	4.44	569 268
					worker	[1, 2]	4	50		80.44			
					courier	[1, 0]	3	40		84.57			
					car	[1, 0]	2	20		84.51			
					drone	[1, 4]	3	30		85.61			

Table 2: Outputs for some of the simulations carried out for the delivery example.

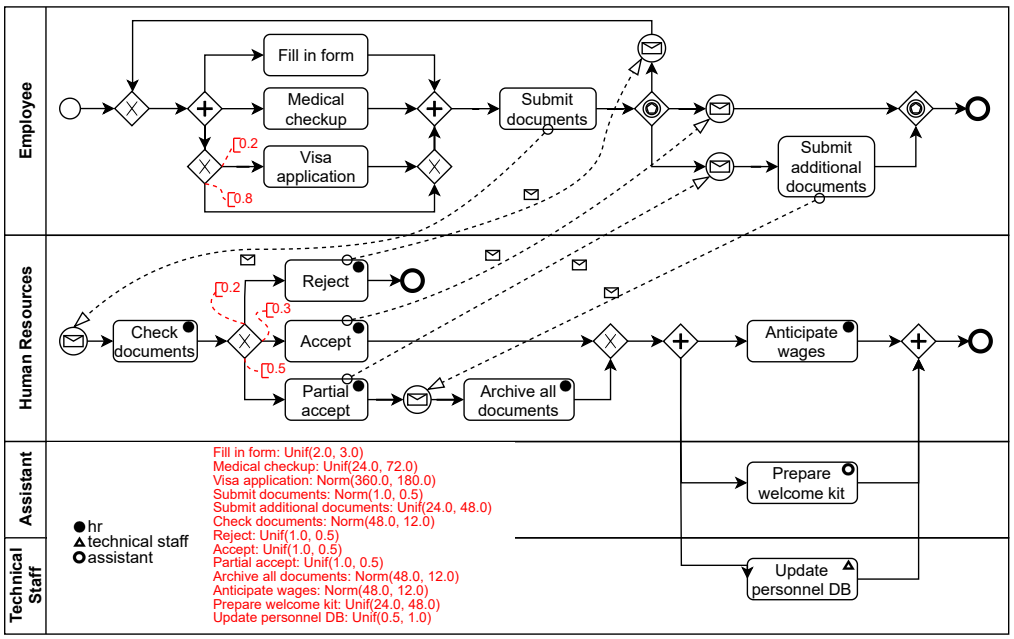

Figure 18: Recruitment process.

wages anticipation are carried out by HR officers, welcome kits are prepared by assistants, and DB updates are carried out by technical staff (IT technicians).

The duration of tasks Check documents, Archive all documents, and Anticipate wages is given by a stochastic expression Norm(48, 12), and the task Prepare welcome kit is given by an expression Unif(24, 48). The rest of the tasks that require resources have a small duration compared to these ones. Overall, the recruitment process specified in this way is highly intensive in manual work.

The conditions in which the process is assumed to be used are also very different. In this case, it is assumed that the office space is limited, being able to accommodate a maximum of 100 HR officers, 25 assistants, and 2 IT technicians. The allocation times, i.e., the hiring times, are 10 for HR officers, 2 for assistants, and 5 for IT technicians. Finally, their salaries amount to 90€ per hour for HR officers, and 50 and 70 for assistants and IT technicians, respectively.

As for the delivery example, the process has been analyzed to get information on the resource usage and to evaluate whether the different strategies fit to the specific process. Given the above restrictions, simulations have been carried out using the four different strategies for TBC values of 5, 10, and 15, and depending

	Execution time (h)		Cost (€)
	Avg	Var	
average	450.46	17.56	21 382 282.64
minimum	298.35	0.01	1 926 935.24
maximum	682.84	94.40	26 707 878.24
median	388.74	3.51	21 209 387.87

Table 3: Average, minimum, maximum and median values of the simulations for the recruitment example with bounds 100, 25 and 2.

on the strategy, CI values of 5 and 10, and LAT values of 5, 10, and 15. Regarding thresholds, bounds $(40, 70)$ and $(60, 90)$ have been considered for usage percentages, and $(2, 5)$ and $(2, 8)$ for request-queue sizes. A total of 54 different combinations have been analyzed, showing some global numbers in Table 3. These numbers suggest that the election of one strategy over the other may lead to significant differences. For these simulations, execution times range between 298.35 and 682.84, with an average of 450.46 and median 388.74. However, costs range between 1.9 million and 26 millions, with an average around 21 millions.

Table 4 shows details of a selection of these simulations. Specifically, row 2 shows the results for the simulation showing the lowest execution time, row 3 shows the results for the simulation showing the lowest cost, and rows 1 and 2 show the best results for the optimization problem with weights 0.6-0.4. Even though the combination of values of row 3 shows a surprisingly low cost, its execution time doubles the time obtained for other combinations. Despite its better use of resources — 76.43% of HR officers and 72.92% of assistants — the excessive execution time and the high variance suggest a problem in the timing of these resources. The fact that it operates with greater TBD and CI values suggest that stability is being rewarded regarding cost, while the increase in execution time may be coming from the delay in the response.

6.5. Information for process improvement

These numbers do not only suggest which strategies and parameters could be the most convenient for this particular process, they also bring some light on some possible improvements on the process restrictions. Although the analysis may consider a greater variety of possibilities before suggesting a final improvement, the focus of this section is on the evolution of the resources for the best of these combinations.

Figure 19 shows the evolution of the number of instances and corresponding usage percentages along the simulation for each of the resource types involved in

	Strat	TBC	CI	LAT	Resources					Usage (%)	Exec time (h)		Cost (€)
					Name	Range	AT	Cost	Thrd		Avg	Var	
1	predictive-usage	5	-	10	HR off. assist. IT tech.	[1, 100] [1, 25] [1, 2]	10 2 5	90 50 70	(60, 90)	69.30 65.23 24.47	303.48	9.92	19 233 280
2	usage	5	5	-	HR off. assist. IT tech.	[1, 100] [1, 25] [1, 2]	10 2 5	90 50 70	(60, 90)	64.70 62.56 24.18	298.36	10.16	19 792 565
3	usage	10	10	-	HR off. assista. IT tech.	[1, 100] [1, 25] [1, 2]	10 2 5	90 50 70	(60, 90)	76.43 72.92 26.46	621.97	74.48	1 926 935

Table 4: Outputs for some of the simulations carried out for the recruitment example.

this example. Regarding IT technicians, from Table 4 and Figure 19, one clear conclusion is that they are underused. They follow a pattern of use very different to those of HR officers and assistants, with one instance most of the time, getting a second one several times for very short periods of time. The fact that its cost is insignificant, compared to the cost of the other resources, possibly makes harder to optimize its use. This suggests that allowing the use of different strategies or parameters for each resource type may improve the execution time and cost of processes.

Since the simulations begin with only one instance per resource type, and given the high demand for both HR officers and assistants from the very beginning, they both show a very steep initial growth until they stabilize. However, note that although the usage is at 100% for some time for HR officers, that does not happen for assistants. This is undoubtedly due to the fact that most of the tasks assigned to HR officers go before the ones assigned to assistants. It is interesting to see that despite the high demand on the assistant resource, between 60% and 100% usage, the number of instances does not go over 25, which is the given maximum number of allocated instances.

Let us consider the possibility of raising the upper limit for the number of assistants. Table 5 shows the outputs for the recruitment example with the parameters of row 1 in Table 4 for different ranges of assistants. It can be seen that even though the average execution time is slightly improved for value 26, further increasing the upper limit makes it worst. The total cost increases when passing from 25 to 26 and 27, but it goes down for 28 and 29. Overall, looking at the objective function, with weights 0.6-0.4, it can be observed how the value for $[1, 26]$ is the best of the results. It can be concluded that by hiring an extra assistant the

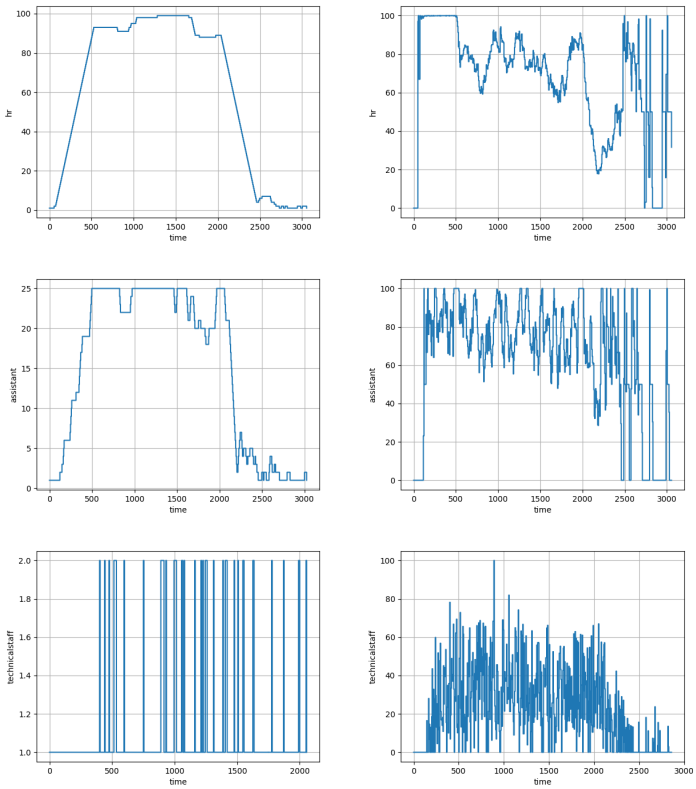

Figure 19: Number of instances (left) and usage percentage (right) for each resource type for a simulation with the predictive-usage strategy, TBC=5, LAT=5, and Thds=(60, 90).

Assistant's range	Exec time (h)		Cost (€)	Goal function
	Avg	Var		$\omega_{ExecTime} = 0.6, \omega_{Cost} = 0.4$
[1, 25]	303.48	9.92	19 233 280	0.19635
[1, 26]	301.98	9.63	19 353 864	0.19203
[1, 27]	306.80	10.59	19 790 926	0.59817
[1, 28]	314.82	12.31	18 950 316	0.52846
[1, 29]	316.55	12.70	19 148 931	0.69451
[1, 30]	311.68	11.62	19 340 055	0.58453

Table 5: Outputs for the recruitment example with the parameters of row 1 in Table 4 for different ranges of assistants.

goal function can be slightly improved because of the reduction on the average execution time, despite a slight increase in the global cost.

7. Related Work

Several works on the analysis and scheduling of resources can be found in the literature. In this section, an overview is presented of those more closely related to the contributions in the current manuscript.

Schmig and Rau [26] use colored stochastic Petri nets to specify and analyze business processes in the presence of dynamic routing, simultaneous resource allocation, forking/joining of process-control threads, and priority-based queuing. In their work, each resource is equipped with properties grouped in a role defining if the resource is eligible to perform a certain activity. Li *et al.* [18] introduce *multidimensional workflow nets* to model and analyze resource availability and workload. Oliveira *et al.* [21] use generalized stochastic Petri nets for correctness verification and performance evaluation of business processes. In their work, an activity can be associated to multiple roles and the completion of an activity can use a portion of the resources available for a role. They also propose metrics for evaluating process performance such as: the minimum number of resources needed for a role in order to complete a process, the expected number of activity instances when completing a process under the assumption of sufficient resources, and the expected activity response time. Colored Petri Nets are used in [20] for understanding how bounded resources can impact the behavior of a process. They introduce the notion of "flexible resource allocation" as a way to assign resources associated to a given role based on priorities. In their approach, alternative strategies are used to better allocate a fixed number of available resources. Havur *et al.* [12] study the problem of resource allocation in business processes management systems where constraints can be assigned to resources

(e.g., time of availability) and have dependencies. Their technique is based on the answer set programming formalism and is capable of deriving optimal schedules. Sperl *et al.* [28] describe a stochastic method for quantifying resource utilization relative to structural properties of processes and past executions.

The work presented in this paper encompasses the specification and analysis features for BPMN covered by the aforementioned papers. The encoding of the BPMN syntax and execution semantics in rewriting logic supports an expressive subset of BPMN consisting of: activity and collaboration diagrams, several types of gateways, timed flows and tasks, probabilities for exclusive and inclusive split gateways, unbalanced workflows, looping behavior, and resource provisioning. The rewriting logic specification presented here support the specification and verification of dynamic allocation of resources, several related measures (execution time, resource occupancy), and the impact of resources evolution on the costs associated to a process. None of the aforementioned works attempts at providing analysis techniques or tools for the dynamic allocation of resources with respect to response time and resource usage, as the proposed approach does.

In [31], a solution is presented to optimize resource allocation by focusing on the structure of the process, and more precisely on dependencies between resources and tasks. The approach then proposes a solution to adapt the structure of the business process to better fit the resources available in the enterprise. The authors in [5] focus on the specification and verification of concurrently running processes, operating in time-critical scenarios and having assigned a limited amount of resources. The authors propose to use a fragment of first-order logic to capture process fragments along the timeline and to combine them in a sound model, by observing constraints defined on both activity durations and resource availability. In [23], a contribution to the field of business process simulation is made by providing a new simulation engine, which supports advanced resource specificities such as queuing mechanisms, resource dependencies, or simulation parameters. A conceptual model supports these features and a prototype implementation of this conceptual model are proposed. Incorporating these features also allows for more accurate simulation of the processes and obtaining more relevant performance metrics. Finally, [13] presents a framework to integrate optimized resource allocation in business processes by adding a new component called *resource manager*. It is responsible for maintaining all relevant information concerning the availability of resources and for allocating resources to a process instance. The process designer can specify resource requirements within the business process model through dedicated resource-allocation activities.

Compared to these papers on resource allocation and process optimization,

the main difference of the work presented in the current paper is that it supports various strategies for adding/removing resources based on different criteria such as usage or queuing analysis. In contrast, the aforementioned papers propose a single solution for dynamic allocation of resources. Moreover, beyond proposing several strategies for resource allocation, the proposed approach fully automates simulation-based techniques for comparing these allocation strategies with respect to several quantitative values (e.g., execution time, resource occupancy, process total cost).

There are many tools supporting the design and management of business processes (e.g., Arena, ARIS10, iGraphx, Signavio, BPMOne, BIMP, Camunda), of which a subset supports the analysis and optimization of processes. For instance, this is the case of Signavio [27], which packs tools such as the Signavio Process Intelligence for process optimization. It automatically mines process models from currently running systems and monitors those processes with the purpose of collecting data that enables end-users to make decisions for process improvement. The proposal here takes a different approach, since the focus is on predicting the behavior of designed models given resource provisioning strategies: thus, the approach presented in this work supports the decision making at design time, even before a process is deployed. Recall that the development in this work allows resources, given a resource allocation strategy, to be dynamically provisioned and released, and meeting the constraints specified as parameters to the process model.

This work is part of a long term project with the goal of developing different tools for the analysis of BPMN processes. In [9], basic BPMN processes were specified. This first work provides operations for the estimation of execution times, and uses model-checking techniques to verify reachability problems and LTL properties. In [6], a model very similar to the current one was proposed, although for a much smaller subset of features; it was used for stochastic analysis using the statistical model checker PVeStA [2]. In [7], Maude is used to model and analyze the resource allocation of business processes. In that work, optimal allocation is presented as a multi-objective optimization problem, where response time and resource usage are minimized. Similar techniques were later added to the Signavio tool.[1] None of these works face the more ambitious challenge of dealing with dynamic sets of resources.

This paper is a significant extension of [8] in the following directions:

[1] Tools for the simulation-based analysis of resources were added to the Signavio Process Manager in its Version 13.9.0, dated in November 2019: Signavio Process Manager.

- the Maude specification of BPMN processes has been extended and refined to improve its efficiency and to consider allocation times and a wider variety of strategies;
- the specification is now ready to be extended with additional strategies by considering the properties for which data is being collected; for example, a new strategy could be engineered for taking into account the average execution time of tasks involving resources;
- two additional strategies, following completely new approaches, have been proposed; one is based on predictions of the whereabouts of the process and the other one that enables the combination of existing criteria (specifically, this paper has presented a strategy that takes decisions based on the future usage of the resources, and another one that combines the usage-based and the queue-based strategies);
- the approach has been applied to several case studies for evaluation purposes and extensive experimentation has been carried out challenging all the described strategies; and
- the paper has not been only extended with new contributions and text, the entire paper has been revised, by taking the aforementioned extensions and improvements into account, and posing a more in-depth presentation of the motivation, specification, analysis, and discussion.

8. Concluding Remarks

This paper focuses on the problem of dynamic resource allocation using BPMN as modeling language for business processes. It presents an extension of BPMN with annotations for describing the duration of task execution and delays of flows, probabilities in split gateways, and additional information about resources. Given such a process specification, automated techniques are proposed for analyzing its behavior and dynamically adjusting the number of necessary resources following some given adaptation strategy. In this paper, several strategies for resource provisioning were presented and illustrated on concrete examples showing how parameters (namely, time between checks, history length, resource ranges, and adaptation thresholds) could be adjusted. The automatic approach was able to handle analyses about the response time and total cost associated to the process. These results were possible thanks to an encoding of the annotated BPMN language into rewriting logic and by using Maude's tools for automating all checks on the concurrent executions of the processes.

This work comes to complement other previous tools also built using a Maude representation of BPMN processes in which model checking, statistical model checking, and simulation techniques have been used for different purposes, including the optimal scheduling and provisioning of resources, and the verification of reachability, stochastic, and LTL properties of the models.

Regarding future work, providing mechanisms to automatically find the best parameters for given strategies is the first direction. This is a multi-objective problem, which is restricted by the concrete nature of the process at hand. Another aim would be at designing and implementing more precise modeling support for the provisioning/releasing procedure, by taking into account aspects such as releasing costs, resource families, etc. It would be interesting to consider the possibility of using a broader form of resources, and to support resource patterns that are not currently covered such as the chain and pile-based execution patterns (see [25]). It is also part of the plan to add identities to resource replicas, which can enable access-control constraints such as, e.g., BoD (binding of duties, forcing the assignment of one same replica to multiple activities) or SoD (separation of duties, stating that some specific resource cannot be used in different tasks). Alternative prediction mechanisms need to be considered too, as those proposed in [10], in which neural networks and other machine learning techniques are used to predict the future steps in the execution of processes. Finally, as far as tool support is concerned, the transformation from BPMN to Maude is automated, but currently requires manual annotation with quantitative information (e.g., probabilities, duration, resources). To make the analysis tool more accessible, a web application providing a user-friendly UI for modeling BPMN and its quantitative extensions, invoking the verification techniques, and presenting the results in a readable way may be built.

Acknowledgements. The authors would like to thank the editor and reviewers for carefully reading the manuscript; their comments have been of great help in improving its quality and clarity. The first author was partially supported by projects PGC2018-094905-B-I00 (Spanish MINECO/FEDER) and UMA18-FEDERJA-180 (J. Andalucía/FEDER). The second author was partially supported by the ECOS-NORD MinCiencias C19M03 project "FACTS: Foundational Approach to Computation in Today's Society".

References

[1] Agha, G.A., Meseguer, J., Sen, K., 2006. PMaude: Rewrite-based specification language for probabilistic object systems, in: 3rd Workshop on Quanti-

tative Aspects of Programming Languages (QAPL), Elsevier. pp. 213–239.

[2] AlTurki, M., Meseguer, J., 2011. PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool, in: 4th International Conference on Algebra and Coalgebra in Computer Science (CALCO), Springer. pp. 386–392.

[3] Bouhoula, A., Jouannaud, J.P., Meseguer, J., 2000. Specification and Proof in Membership Equational Logic. Theoretical Computer Science 236, 35–132.

[4] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C., 2007. All About Maude - A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic. volume 4350 of *LNCS*. Springer.

[5] Combi, C., Sala, P., Zerbato, F., 2018. A logical formalization of time-critical processes with resources, in: Business Process Management Forum (BPM Forum), Springer. pp. 20–36.

[6] Durán, F., Rocha, C., Salaün, G., 2018. Stochastic analysis of BPMN with time in rewriting logic. Science of Computer Programming 168, 1–17.

[7] Durán, F., Rocha, C., Salaün, G., 2019. A rewriting logic approach to resource allocation analysis in business process models. Science of Computer Programming 183.

[8] Durán, F., Rocha, C., Salaün, G., 2020. Analysis of the runtime resource provisioning of BPMN processes using Maude, in: 13th International Workshop on Rewriting Logic and Its Applications (WRLA), Springer. pp. 38–56.

[9] Durán, F., Salaün, G., 2017. Verifying timed BPMN processes using Maude, in: 19th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION), Springer. pp. 219–236.

[10] Francescomarino, C.D., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A., 2017. An Eye into the Future: Leveraging A-priori Knowledge in Predictive Business Process Monitoring, in: International Conference on Business Process Management (BPM), Springer. pp. 252–268.

[11] Frappaolo, C., Keldsen, D., 2008. Business Process Management (BPM) – Leveraging Competencies and Streamlining Processes to Achieve Operational Excellence. Technical Report. Risetime, AIIM.

[12] Havur, G., Cabanillas, C., Mendling, J., Polleres, A., 2016. Resource allocation with dependencies in business process management systems, in: Business Process Management Forum (BPM Forum), Springer. pp. 3–19.

[13] Ihde, S., Pufahl, L., Lin, M., Goel, A., Weske, M., 2019. Optimized resource allocations in business process models, in: Business Process Management Forum (BPM Forum), Springer. pp. 55–71.

[14] ISO/IEC, 2013. International Standard 19510, Information technology – Business Process Model and Notation.

[15] Krishna, A., Poizat, P., Salaün, G., 2017. VBPMN: Automated Verification of BPMN Processes, in: 13th International Conference on Integrated Formal Methods, Springer. pp. 323–331.

[16] Krishna, A., Poizat, P., Salaün, G., 2019. Checking business process evolution. Science of Computer Programming 170, 1–26.

[17] Leemans, S., Fahland, D., van der Aalst, W., 2018. Scalable Process Discovery and Conformance Checking. Software & Systems Modeling 17, 599–631.

[18] Li, J., Fan, Y., Zhou, M., 2004. Performance Modeling and Analysis of Workflow. IEEE Transactions on Systems, Man, and Cybernetics 34, 229–242.

[19] Meseguer, J., 1992. Conditional Rewriting Logic as a Unified Model of Concurrency. Theoretical Computer Science 96, 73–155.

[20] Netjes, N., van der Aalst, W., Reijers, H., 2005. Analysis of Resource-Constrained Processes with Colored Petri Nets, in: Proc. of CPN, pp. 251–266.

[21] Oliveira, C., Lima, R., Reijers, H., Ribeiro, J., 2012. Quantitative Analysis of Resource-Constrained Business Processes. Trans. on Syst., Man, and Cybern. 42, 669–684.

[22] Ölveczky, P.C., Meseguer, J., 2007. Semantics and Pragmatics of Real-Time Maude. Higher-Order and Symbolic Computation 20, 161–196.

[23] Peters, S.P.F., Dijkman, R.M., Grefen, P.W.P.J., 2018. Advanced simulation of resource constructs in business process models, in: Business Process Management Forum (BPM Forum), Springer. pp. 159–175.

[24] Polato, M., Sperduti, A., Burattin, A., de Leoni, M., 2018. Time and activity sequence prediction of business process instances. Computing 100, 1005–1031.

[25] Russell, N., ter Hofstede, A., Edmond, D., van der Aalst, W., 2004. Workflow resource patterns. BETA Working Paper Series, WP 127. URL: http://www.workflowpatterns.com/.

[26] Schömig, A.K., Rau, H., 1995. A Petri Net Approach for the Performance Analysis of Business Processes. Technical Report 116. Universität Würzburg. Würzburg, Germany.

[27] Signavio, 2019. Available: https://www.signavio.com.

[28] Sperl, S., Havur, G., Steyskal, S., Cabanillas, C., Polleres, A., Haselböck, A., 2017. Resource utilization prediction in decision-intensive business processes, in: 7th International Symposium on Data-driven Process Discovery and Analysis (SIMPDA), CEUR-WS.org. pp. 128–141. URL: http://ceur-ws.org/Vol-2016/paper10.pdf.

[29] van der Aalst, W., De Masellis, R., Di Francescomarino, C., Ghidini, C., 2017. Learning Hybrid Process Models from Events - Process Discovery Without Faking Confidence, in: International Conference on Business Process Management (BPM), Springer. pp. 59–76.

[30] Walck, C., 2007. Hand-Book on Statistical Distributions for Experimentalists. Technical Report SUF–PFY/96–01. Universitet Stockholms. Stockholm.

[31] Xu, J., Liu, C., Zhao, X., 2008. Resource allocation vs. business process improvement: How they impact on each other, in: 6th International Conference on Business Process Management (BPM), Springer. pp. 228–243.