Expert Systems with Applications 42 (2015) 5366–5379

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

# A quality-driven approach for resources planning in Service-Oriented Architectures

Marcelo Teixeira [a,*], Richardson Ribeiro [a], Cesar Oliveira [b], Ricardo Massa [b]

[a] Department of Informatics (Dainf), UTFPR, Pato Branco, Paraná, Brazil
[b] Center of Informatics (Cin), UFPE, Recife, Pernambuco, Brazil

ARTICLE INFO

*Article history:*
Available online 5 March 2015

*Keywords:*
Petri Nets
Service-Oriented Architectures
Service orchestration
Modeling
Simulation

ABSTRACT

Service-Oriented Architecture (SOA) is a paradigm for software development based on the concept of service. In SOA, the Quality of Services (QoS) impacts on the status of a business and on the relationship between service customers and providers. As customers expect to receive services with quality no less than they have paid for, it is usual to stress a SOA application in order to measure its QoS levels, which can be expensive and time consuming. This paper shows that the behavior of a SOA system can be modeled by Petri Nets and, from the model, QoS levels (performance and availability) can be estimated. In this way, the analysis can be conducted without necessarily implementing the real system, which tends to be valuable in the design phase of SOA. Additionally, we present a methodology to implement the model findings, which allows to verify its accuracy in practice. As a final contribution we associate our modeling approach to the *Service Level Agreements* (SLA) composition, which allows to discover and prevent bottlenecks delaying the system and to anticipate potential SLA violations. Two examples illustrate our results.


## 1. Introduction

Service-Oriented Architecture (SOA) is a principle for software structuring based on the concept of *service*. A service is an independent block of code that can be naturally reused and composed with other services to create a system. The main motivation for developing software based on SOA can be summarized in a single word: *decoupling*. In fact, SOA systems can be incrementally built by adding and modifying services, or by switching service providers. Due to a well controlled coupling mechanism, all these tasks can be done safely, at low cost, and within a short period of time.

As services are independent entities created to provide specific business functionalities, an external element is required to compose them and to orchestrate their execution flow inside a SOA application (Jula, Sundararajan, & Othman, 2014; Li et al., 2011). It is common to employ XML-based languages for this task, such as WS-BPEL – *Web Services Business Process Execution Language* (Andrews et al., 2003) (or simply BPEL). BPEL descriptions are interpreted by an orchestration (or workflow) engine that coordinates the communication flow, correlates messages exchanged by partners, supports persistence of long-running business transactions and provides exception handling. BPEL workflows can be automatically generated and functional requirements can be guaranteed by construction (Palomo-Duarte, García-Domínguez, & Medina-Bulo, 2014).

Since the concept of service is the SOA kernel, the *Quality of Service* (QoS) has a commercial effect on the status of a business and on the relationships between service customers and providers. Therefore, improving the service execution also enhances the business quality (Casati, Shan, Dayal, & Shan, 2003). However, typical QoS metrics in SOA, such as network bandwidth, XML processing, workload, etc., can impose enormous variability. It has been reported that the ratio of the load variation for Internet applications can reach an order of 300% (Chase et al., 2001)Such instability challenges the quality management in SOA (Almeida & Menasce, 2001). Moreover, it is natural for a SOA workflow to be in constant modification, with new services being introduced, updated or removed (Baresi & Guinea, 2008). Then, it becomes essential to estimate the impact of such modifications on the QoS levels before actually changing the system. The usual way to do this so, is by the two-step process that follows: (1) update the application; and (2) stress it to measure its quality. These steps can be expensive and time consuming, since an executable version of the system has to be available to be checked.

A less expensive alternative is to simulate the behavior of SOA applications. A number of modeling approaches have been

* Corresponding author.
*E-mail addresses:* marceloteixeira@utfpr.edu.br (M. Teixeira), richardsonr@utfpr.edu.br (R. Ribeiro), calo@cin.ufpe.br (C. Oliveira), rmfl@cin.ufpe.br (R. Massa).

http://dx.doi.org/10.1016/j.eswa.2015.02.004
0957-4174/

proposed (Hwang, Wang, Tang, & Srivastava, 2007; Rud, Schmietendorf, & Dumke, 2007; Bruneo, Distefano, Longo, & Scarpa, 2010; Xia, Liu, Liu, & Zhu, 2012) but, in general, they fail to respond as fast and accurately as the businesses demand, specially due to the lack of support for their quick conversion into practical tools.

In this paper, the behavior of SOA applications is modeled by Petri Nets (PNs) (Reisig & Rozenberg, 1996). PNs are grounded on a solid mathematical foundation combined to an intuitive high-level user interface. This allows to construct formal, yet simple, descriptions of systems characterized by concurrency, synchronization, mutual exclusion, etc., which appear quite often in SOA (Li, Fan, & Zhou, 2004). We adopt a timed extension of PNs, the *Generalized Stochastic Petri Nets* (GSPNs) (Marsan, Balbo, & Conte, 1984; Marsan, Balbo, & Conte, 1995), to analyze resource consumption and service levels degradation in SOA-based systems with different workloads and orchestration models. Since we exploit the design level of SOA, implementation details are irrelevant and such analysis can be conducted without a complete version of the real system, which tends to be valuable.

By subsuming our previous results (Teixeira, Lima, Oliveira, & Maciel, 2009; Teixeira, Lima, Oliveira, & Maciel, 2010; Teixeira, Lima, Oliveira, & Maciel, 2011), we answer questions frequently faced by software engineers at the design phase of SOA, such as: *"what is the system performance degradation rate?"*, or *"which service contract could be guaranteed for a given web service?"* or even *"which workload level is supported by the system before it fails?"*, and so on. The answer for these questions are mostly unclear at SOA design step and deriving them using purely empirical evidences may lead to unreliable resources planning.

A further contribution of the paper is a methodology for implementing the model findings, which allows to verify its accuracy in practice. This methodology comprises (i) a *black box* analysis, dedicated to evaluate performance aspects of the entire process; and (ii) a *white box* analysis, which looks inside the process to identify structural problems in the service composition level.

The paper brings yet another contribution by associating the modeling approach to the *Service Level Agreements* (SLA) composition, which allows to discover and prevent bottlenecks delaying the system and to anticipate potential SLA violations. To illustrate our approach, we evaluate two examples of SOA systems. It is shown that the overall response time of a system can be systematically reduced in a way that it becomes suitable for a given SLA.

The remainder of the paper is structured as follows: Section 2 presents the literature overview; SOA concepts are presented in Section 3; Section 4 introduces the proposed models, which are illustrated by two examples in Section 6. Conclusions are discussed in Section 8, along with prospects of future work.

## 2. Related work

From a practical perspective, SOA-based systems can be monitored by automated tools in order to observe and possibly interfere on their dynamic behaviors (IBM, 2008; HP, 2010; Abe & Jeng, 2007; Apache, 2011). The *JMeter* tool (Apache, 2011), for example, allows to simulate different protocols of communication (e.g., HTTP, FTP, SOAP), facilitating a structured QoS evaluation in SOA. BPEL code annotation has been also used to describe QoS aspects, contributing to the application management process (Dambrogio & Bocciarelli, 2007; Marzolla & Mirandola, 2007; Lin & Kavi, 2013).

Performance improvements in SOA can also be approached from a modeling perspective. A modular architecture that runs in parallel with SOA systems is presented in (Solomon & Litoiu, 2011). An *estimator* module reads variables from a monitored process; a *predictor* module refines them to the *simulator* module, which reproduces a behavioral sample of the system. Since this dynamic correlates expected performance indexes, it allows to compose strategies to tune the system at runtime. Parejo, Segura, Fernandez, and Ruiz-Cortés (2014) describe an optimization algorithm for the run-time rebinding of web services according to QoS constraints. Five properties are considered: cost, execution time, availability, reliability, and security. The QoS of a workflow composition is computed based on the QoS properties of each service. To this end, an aggregation function is derived from the workflow design. Each composition structure implies in a different aggregation function for combining the composed blocks. Four composition structures are considered: sequence, branch, fork, and loop. For instance, in a sequential composition, the costs of each block are summed up, while the security of the composed structure has the value of the less secure of all blocks. Since the algorithm focuses on service rebinding, this work does not approach the redesign of the workflow itself. There is also no human intervention in the process.

Working on the configuration level of a SOA process has also shown to be effective to optimize its QoS levels (Hwang et al., 2007; Xiong, Fan, & Zhou, 2008). Finding a service workflow that guarantees certain non-functional requirements has been extensively investigated (Lin & Kavi, 2013; Palomo-Duarte et al., 2014; Jula et al., 2014). It is usually complex to provide appropriate service compositions, as it depends on unclear aspects such as demand, resource limitations, etc., and this has been classified in the literature as a NP-hard optimization problem (Jula et al., 2014; Parejo et al., 2014). Moreover, redesigns structurally modify processes, in a way that operational conflicts become susceptible and have to be worked out (Xiong, Fan, & Zhou, 2010).

In general, improving a SOA system passes by upgrading an already developed application. While the planning phase is usually bypassed by many development teams, for management, on the other hand, this is a fundamental phase that allows the company to negotiate contracts, plan resources provision, etc. Since such negotiation must occur at design time, the predictive business process monitoring has drawn attention. In Metzger et al. (2015), three main classes of predictive monitoring techniques have been compared: machine learning, constraint satisfaction and QoS aggregation. Results show that a particular technique can be appropriate for some cases, yet inappropriate for others, which implies that they have to be combined to effectively improve the prediction process. Alternatively, analytical models can be adopted (Rud, Schmietendorf, & Dumke, 2006; Rud, Kunz, Schmietendorf, & Dumke, 2007; Rud, Schmietendorf et al., 2007). However, for an analytical analysis to be conducted, the system is usually assumed to behave deterministically, which undermines the evaluation of dynamic scenarios (Teixeira et al., 2009).

In this regard, a probabilistic evaluation of business processes has been conducted by Oliveira, Lima, Reijers, and Ribeiro (2012). Using GSPN models, the performance of human-centered workflows is estimated. This approach is naturally extensible to SOA but, to that end, physical infrastructure properties (e.g., networking, processing, etc.) have also to be modeled.

In this paper, we subsume our seminal works on extending GSPN analysis to SOA (Teixeira et al., 2009; Teixeira et al., 2010; Teixeira et al., 2011). Our approach maps BPEL structures to GSPN, also including low level elements, such as networking, buffering, BPEL and service processing, etc. A GSPN decomposition model for bottleneck discovery is also derived. We finally integrate our results to the SLA planning.

In the literature, the modeling of SOA with GSPNs have been used for different purposes. Translation rules for the automatic generation of Petri Nets from BPEL have been addressed in (Xia et al., 2012; Bo, Junliang, & Min, 2012). *Colored Petri Nets* (CPN) and *Stochastically Timed Petri Nets* (STPN) have been adopted to

model services compositions (Wu, Yuan, Yang, & Xiong, 2008). STPNs are used to represent the system uncertainty, while CPNs are adopted to label specific instances of SOA, tracing them along the process. *Non-Markovian Stochastic Petri Nets* (NMSPNs) have been applied to provide QoS guarantees in service compositions (Bruneo et al., 2010), while functional properties of service compositions have been checked using Petri Nets in Yoo, Jeong, and Cho (2010) and Bo et al. (2012).

This paper fits into a gap recognized among the mentioned works. Firstly, it provides a model that is modular, being able to accommodate different infrastructures to connect the web services. Secondly, it allows for the computation of multiple measures using the same basic blocks. It is also supported by a methodology for identifying bottlenecks and for planning SLA contracts.

In order to contextualize our work with respect to the literature, we present a comparative analysis in Table 1. We enumerate the major contributions of this paper and then we compare them to a selected set of related research. Eight contribution are compared: (1) *Performance estimation*; (2) *Availability estimation*; (3) *Resources utilization*; (4) *Bottleneck discovering*; (5) *Support to improvements*; (6) *Support to SLA planning*; (7) *Tooling support*; and (8) *Design aided*. **Y** (*Yes*), **N** (*No*), **P** (*Partially*) and **E** (*Extensible*) respectively denote the coverage of a given approach w.r.t. the compared contribution.

It is important to emphasize that the model proposed in this paper incorporates many essential elements of SOA, such as control flow structures, network communication, and data processing, and gives support for the design-time improvement of workflows. From the best of our knowledge, the existing literature lacks the ability to combine all those aspects together in a single model.

**Table 1**
Comparative analysis of six selected related work.

| Research | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Hwang et al. (2007) | **P** | **N** | **N** | **N** | **N** | **N** | **N** | **N** |
| Oliveira et al. (2012) | **Y** | **N** | **Y** | **Y** | **Y** | **E** | **Y** | **Y** |
| Rud, Schmietendorf et al. (2007) | **Y** | **N** | **Y** | **N** | **N** | **N** | **N** | **Y** |
| Bruneo et al. (2010) | **P** | **Y** | **N** | **N** | **E** | **E** | **Y** | **Y** |
| Xia et al. (2012) | **Y** | **N** | **N** | **N** | **N** | **N** | **Y** | **Y** |
| This paper | **Y** | **Y** | **Y** | **Y** | **Y** | **Y** | **Y** | **Y** |

## 3. SOA related concepts and modeling

This section brings an overview on SOA, including its related concepts, technologies and infrastructure. It is also presented a modeling formalism for SOA-based systems.

### 3.1. Service-Oriented Architecture (SOA)

SOA is fundamentally based on the concept of service (Josuttis, 2008). Services are *loose coupled* components of software that provide one or more functionalities and can be combined to other functionalities, from other services. Even when combined, services operate independently from each other and from the state of other processes or functions. They simply receive a request, process it, and return an answer, usually by a web interface.

*Enterprise Service Bus (ESB)* is the infrastructure that provides interoperability among distributed services. In practice, an ESB implements the way by which clients invoke one or more services provided by third-party suppliers, which are possibly running on heterogeneous platforms of both hardware and software, communicating through distinct protocols, etc. The systematic execution of distributed services characterizes a SOA system and can be programmed using specialized languages such as *Business Process Execution Language* (BPEL), described next.

### 3.2. Business Process Execution Language

BPEL (Oasis, 2007) has been considered the most popular language for service orchestration and execution. Orchestration is the task of shaping and reshaping business processes, preparing prospective services to be converged with other services and external customers. It is implemented in BPEL by the proper design of *atomic* structures that are *composed* to generate complex workflows.

- *BPEL atomic structures:*
  - ⟨Invoke⟩ – sends a request to a web service operation;
  - ⟨Receive⟩ – waits for a message to arrive;
  - ⟨Wait⟩ – delays the process by a period of time;
  - ⟨Assign⟩ – manipulates variables assignments;
  - ⟨Reply⟩ – sends an answer to the requestor.
- *BPEL composite structures:*
  - ⟨Sequence⟩ – models a sequential flow;
  - ⟨If⟩ – chooses a path by testing a *Boolean* condition;
  - ⟨RepeatUntil⟩/⟨While⟩/⟨ForEach⟩ – repeats a cycle while holding a *Boolean* condition;
  - ⟨Flow⟩ – performs parallel paths, synchronizing them at the end or at arbitrary points, using a ⟨Link⟩ command.

In BPEL, services are autonomous and their functional specifications are given by service interfaces. The implementation and the platform over which services operate are transparent to the BPEL engine, which brings portability and offers flexibility to adapt the system according to the organizational needs.

### 3.3. Service Level Agreements

In SOA, legal commitments among customers and providers are expressed by contracts known as *Service Level Agreements* (SLAs) (Sturm, Morris, & Jander, 2000). SLAs express obligations, responsibilities, and rights with respect to the offered QoS levels. It may also prescribe penalties for potential damages caused to customers by delivering a QoS level bellow the promised standard (Raibulet & Massarelli, 2008).

A usual SLA clause regulates metrics such as response time, availability, cost, etc. Since customers are usually required to pay in advance for the contracted QoS levels, they usually check services at run-time, using monitoring tools. Although monitoring is the natural way to determine QoS levels for contracted services, this can be expensive, as a number of services from different providers need to be tested. Besides, in-design SOA applications require to anticipate the impact of a given service on the performance of the whole orchestration.

This paper addresses this issue and allows to check a range of SOA scenarios in a short period of time. GSPNs are the foundation of our proposal and are presented next.

### 3.4. Petri Nets

*Petri Net* (PN) (Reisig & Rozenberg, 1996) is a formalism that combines a solid mathematical foundation with an intuitive modeling language. PNs allow to design and evaluate complex characteristics of a system, such as concurrency, synchronization, mutual exclusion, etc. As these features appear quite often in SOA systems, PNs are naturally chosen as modeling formalism in this paper.

Structurally, a PN is composed by *places* (modeling states), *transitions* (modeling state changes), *and directed arcs* (connecting

places and transitions). To express the conditions that hold in a given state, places are marked with *tokens*.

Extensions of PNs have been proposed to incorporate the notion of time (Murata, 1989). *Generalized Stochastic Petri Net* (GSPN) (Marsan et al., 1984; Marsan et al., 1995), for example, is an extension that represents *time* by exponentially distributed random variables associated to a special type of transition, so called *timed transition*. When time is irrelevant for the behavior that has been designed, it is represented in GSPN by *non-timed* (or *immediate*) transitions. Formally, a GSPN is a 7-tuple $GSPN = GSPN = \langle P, \mathcal{T}, \Pi, I, O, M, W \rangle$, where:

- $P = \{p_1, p_2, \ldots, p_n\}$ is a finite set of places;
- $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions;
- $\Pi : \mathcal{T} \to \mathbb{N}$ is the priority function, where:

$$\Pi(t) = \begin{cases} \geqslant 1, & \text{if } t \in \mathcal{T} \text{ is an immediate transition;} \\ 0, & \text{if } t \in \mathcal{T} \text{ is a timed transition.} \end{cases}$$

- $I : (\mathcal{T} \times P) \to \mathbb{N}$ is the input relation that defines the multiplicities of directed arcs from places to transitions;
- $O : (\mathcal{T} \times P) \to \mathbb{N}$ is the output relation that defines the multiplicities of directed arcs from transitions to places;
- $M : P \to \mathbb{N}$ is the initial marking relation. $M$ indicates the number of tokens[1] in each place, i.e., it defines the current state of a GSPN model;
- $W : \mathcal{T} \to \mathbb{R}^+$ is the weighting function that represents either the immediate transitions weights ($w_t$) or the timed transitions rates ($\lambda_t$), where:

$$W(t) = \begin{cases} w_t \geqslant 0, & \text{if } t \in \mathcal{T} \text{ is an immediate transition;} \\ \lambda_t > 0, & \text{if } t \in \mathcal{T} \text{ is a timed transition.} \end{cases}$$

The relationship between places and transitions is established by the sets ${}^\bullet t$ and $t^\bullet$, defined as follows.

**Definition 1.** Given a transition $t \in \mathcal{T}$, define:

- ${}^\bullet t = \{p \in P | I(t,p) > 0\}$ as the *pre-condition* of $t$;
- $t^\bullet = \{p \in P | O(t,p) > 0\}$ as the *post-conditions* of $t$.

A state of a GSPN changes when an enabled transition fires. Only enabled transitions can fire. *Immediate* transitions fire as soon as they get enabled. The *enabling rule* for firing and the *firing semantics* are defined next.

**Definition 2** (*Enabling Rule*). A transition $t \in \mathcal{T}$ is said to be enabled in a marking $M$ if and only if:

- $\forall p \in {}^\bullet t,\ M(p) \geqslant I(t,p)$.

When an enabled transition fires, it removes tokens from the input to the output places (its *pre* and *post* conditions, respectively).

**Definition 3** (*Firing Rule*). The firing of a transition $t \in \mathcal{T}$, enabled in the marking $M$, leads to a new marking $M'$ such that $\forall p \in ({}^\bullet t \cup t^\bullet), M'(p) = M(p) - I(t,p) + O(t,p)$.

A GSPN is said to be *bounded* if there exists a limit $k > 0$ for the number of tokens in every place. If that is the case, one ensures that the state-space resulting from a bounded GSPN is finite.

When the number of tokens in each input place $p$ of $t$ is $n$ times the minimum needed to enable $t$ ($\forall p \in {}^\bullet t,\ M(p) \geqslant n \times I(t,p)$, where $n \in \mathbb{N}$ and $n > 1$), it enables the transition to fire more than once. In this case, the transition $t$ is said to be enabled with degree $n > 0$. Transition firing may use one of the following dynamic semantics:

- *single-server*: $n$ sequential fires;
- *infinite-server*: $n$ parallel fires;
- *k-server*: the transition is enabled up to $k$ times in parallel; tokens that enable the transition to a degree higher than $k$ are handled after the first $k$ firings.

GSPNs allow to compute metrics both by simulation or state-space analysis. Furthermore, GSPNs allow to combine exponential models to form different time distributions (Desrochers, 1994), which is useful to represent specific behaviors of systems.

## 4. Proposed model

In this section, the proposed GSPN model to estimate response time and availability in SOA-based systems is presented. In SOA, the usual interactions among partners can be characterized as follows: (a) users request operations to be executed in a business process implemented in BPEL and hosted in a remote server (orchestrator); (b) the orchestrator receives each request and invokes web service operations according to the programmed workflow; (c) an answer is replied back by the orchestrator to the user.

One can observe that such interactions define three main classes of computational tasks: (i) network communication; (ii) web services processing; and (iii) XML processing, to orchestrate the process. Therefore, the proposed model is constructed in a way to properly represent these tasks. This is done by designing a set $\mathcal{SOA}$ defined by a collection of submodels $\mathcal{B}_i$, each one covering the modeling of a particular SOA task, such that: $\mathcal{SOA} = \{\mathcal{B}_{up}, \mathcal{B}_{op}, \mathcal{B}_{dw}, \mathcal{B}_{eg}, \mathcal{B}_{rp}\}$, where:

- $\mathcal{B}_{up}$: *models the message flow from BPEL to a web service*;
- $\mathcal{B}_{op}$: *models the web service operation processing*;
- $\mathcal{B}_{dw}$: *designs the message flow from a web service to BPEL*;
- $\mathcal{B}_{eg}$: *represents the XML processing to orchestrate the BPEL process*;
- $\mathcal{B}_{rp}$: *models the message flow from BPEL back to the client.*

Next, we provide details about the modeling of each element in the $\mathcal{SOA}$.

### 4.1. Generic block modeling

In our approach, each model $\mathcal{B}_i \in \mathcal{SOA}$ is interpreted as a GSPN block. Structurally, blocks are identical, which allows us to characterized them by a generic definition and compose them afterwards. As blocks are individually independent, their composition requires to define which *channels* connect them. For example, the BPEL engine block ($\mathcal{B}_{eg}$) communicates with the upload block ($\mathcal{B}_{up}$) through the "engine-upload" channel, or *eg-up* for short. This idea is then taken into account to introduce the *block constructor* operator that follows:

**Definition 4** (*Block Constructor ($\mathcal{C}$)*). Let $b$ be a block label, ch-in and ch-out be channel names. The block constructor operator $\mathcal{C}(b, ch\text{-}in, ch\text{-}out)$ generates a GSPN $G = \langle P_b, \mathcal{T}_b, \Pi_b, I_b, O_b, M_b, W_b \rangle$, such that:

[1] Black dots are used to graphically represent a token in a place.

- $P_b = \{I_b, R_b, S_b, K_b, O_b\}$, where:
  - $M_b(I_b) = M_b(O_b) = M_b(S_b) = 0$;
  - $M_b(R_b) > 0$ and $M_b(K_b) > 0$.

- $\mathcal{T}_b = \{c_{ch\text{-}in}, t_b, c_{ch\text{-}out}, T_b\}$, where:
  - $c_{ch\text{-}in}$, $t_b$, and $c_{ch\text{-}out}$ are immediate transitions;
  - $T_b$ is an infinite server timed transition with delay $d_b \in \mathbb{R}^*_+$.

- Input arcs and weights:
  - $I_b(c_{ch\text{-}in}, R_b)$, such that $W_b(c_{ch\text{-}in}, R_b) = r_b$;
  - $I_b(t_b, I_b)$, such that $W_b(t_b, I_b) = 1$;
  - $I_b(t_b, K_b)$, such that $W_b(t_b, K_b) = K_b$;
  - $I_b(T_b, S_b)$, such that $W_b(T_b, S_b) = 1$;
  - $I_b(c_{ch\text{-}out}, O_b)$, such that $W_b(c_{ch\text{-}out}, O_b) = 1$.

- Output arcs and weights:
  - $O_b(c_{ch\text{-}in}, I_b)$, such that $W_b(c_{ch\text{-}in}, I_b) = 1$;
  - $O_b(t_b, R_b)$, such that $W_b(t_b, R_b) = R_b$;
  - $O_b(t_b, S_b)$, such that $W_b(t_b, S_b) = 1$;
  - $O_b(T_b, O_b)$, such that $W_b(T_b, O_b) = 1$;
  - $O_b(c_{ch\text{-}out}, K_b)$, such that $W_b(c_{ch\text{-}out}, K_b) = K_b$.

Fig. 1 shows the structure assumed by a GSPN block $calB_b$ derived from $\mathcal{C}$.

The presence of a token in $I_b$ models a request waiting to be served (input queue). The arrival of requests in $I_b$ is controlled by the transition $c_{ch\text{-}in}$. The amount of free space available in $I_b$ is denoted by the number of tokens in $R_b$. Thus, the capacity (maximum number of tokens) of $R_b$, denoted by $max(R_b)$, limits the size of the input queue $I_b$.

The presence of $k_b$ or more tokens in place $K_b$ indicates that the server has enough resources to handle new requests. In this case, if a request is waiting in the input queue to be served, the immediate transition $t_b$ fires, removing a token (a request) from the input queue, also removing $k_b$ tokens from $K_b$ (resource allocation), and including a token into place $S_b$ (request is being served). The delay $d_b$, associated to the timed transition $T_b$, models the time spent to serve a given request. After served, requests are removed to the output queue $O_b$. The maximum number of tokens in $K_b$, denoted by $max(K_b)$, limits the number of resources available to process and dispatch requests, which is given by $max(K_b)/k_b$.

### 4.2. Modeling BPEL requests

In this section, we show how the constructor $\mathcal{C}$ can be used to properly integrate blocks belonging to $\mathcal{SOA}$, in a way to model BPEL requests. We consider BPEL request as the five-steps sequence that follows:

1. *upload* ($\mathcal{B}_{up}$): a request message is uploaded from the orchestrator to the server that hosts the web service to be called;
2. *execution* ($\mathcal{B}_{op}$): the operation is performed by the web service;
3. *download* ($\mathcal{B}_{dw}$): a response is downloaded from the service by the orchestrator;
4. *processing* ($\mathcal{B}_{eg}$): the BPEL engine processes the response and determines the next action to be taken;
5. *response* ($\mathcal{B}_{rp}$): when no further invocation remains, a response is sent back to the client and the process is concluded.

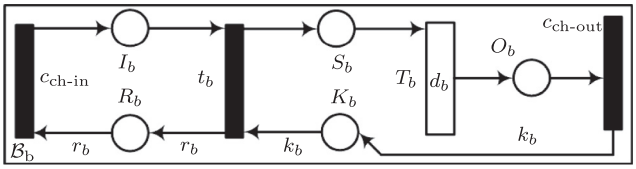


**Fig. 1.** Generic GSPN interface – $\mathcal{B}_b$.

Reproducing such sequence on the GSPN blocks that model each step requires ordering the elements in $\mathcal{SOA}$. For that, they are restructured as follows.

**Definition 5** (*Blocks Reshape*). Let $\mathcal{C}$ be the block constructor operator from Definition 4. For $\mathcal{B}i \in \mathcal{SOA}$, define:

- $\mathcal{B}_{up} = \mathcal{C}(up,\ in\text{-}up,\ up\text{-}op)$;
- $\mathcal{B}_{op} = \mathcal{C}(op,\ up\text{-}op,\ op\text{-}dw)$;
- $\mathcal{B}_{dw} = \mathcal{C}(dw,\ op\text{-}dw,\ dw\text{-}eg)$;
- $\mathcal{B}_{eg} = \mathcal{C}(eg,\ dw\text{-}eg,\ eg\text{-}rp)$;
- $\mathcal{B}_{rp} = \mathcal{C}(rp,\ eg\text{-}rp,\ rp\text{-}out)$.

Now, most blocks share channels. For example, $\mathcal{B}_{op}$ shares a channel with $\mathcal{B}_{up}$ and another with $\mathcal{B}_{dw}$. Fig. 2 shows the graphical version of $\mathcal{SOA}$ after applying $\mathcal{C}$ over its blocks.

When composed, the shared channels amtare merged (the transitions become the same), which connects the corresponding blocks. The result is a single GSPN model that models the five-step sequence of a BPEL request. The composition effect can be obtained by applying a simple union ($\cup$) operation, defined as follows.

**Definition 6** (*GSPN Union* ($\sqcup$)). Let $G_1, G_2$ be two GSPNs such that $G_1 = \langle P_1, \mathcal{T}_1, \Pi_1, I_1, O_1, M_1, W_1\rangle$ and $G_2 = \langle P_2, \mathcal{T}_2, \Pi_2, I_2, O_2, M_2, W_2\rangle$.

$\sqcup : GSPN \times GSPN \to GSPN$ is an operation that generates a new GSPN $G_3 = G_1 \sqcup G_2$ defined as:

(i) $P_3 = P_1 \cup P_2$;
(ii) $T_3 = T_1 \cup T_2$;

$$\text{(iii)}\ I_3(p,t) = \begin{cases} I_1(p,t) & \text{if} \quad p \in P_1 \quad \text{and} \quad t \in T_1 \\ I_2(p,t) & \text{if} \quad p \in P_2 \quad \text{and} \quad t \in T_2 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{(iv)}\ O_3(p,t) = \begin{cases} O_1(p,t) & \text{if} \quad p \in P_1 \quad \text{and} \quad t \in T_1 \\ O_2(p,t) & \text{if} \quad p \in P_2 \quad \text{and} \quad t \in T_2 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{(v)}\ \omega_3(t) = \begin{cases} \omega_1(t) & \text{if} \quad t \in T_1 \\ \omega_2(t) & \text{if} \quad t \in T_2 \end{cases}$$

$$\text{(vi)}\ \Pi_3(t) = \begin{cases} \Pi_1(t) & \text{if} \quad t \in T_1 \\ \Pi_2(t) & \text{if} \quad t \in T_2 \end{cases}$$

$$\text{(vii)}\ M_0^3(p) = \begin{cases} M_0^1(p) & \text{if} \quad p \in P_1 \\ M_0^2(p) & \text{if} \quad p \in P_2 \end{cases}$$

A BPEL invoke operation can finally be modeled by the GSPN in (1).

$$Inv = \mathcal{B}_{up} \cup \mathcal{B}_{op} \cup \mathcal{B}_{dw} \cup \mathcal{B}_{eg} \cup \mathcal{B}_{rp}. \tag{1}$$

We remark that the immediate transition $c_{in\text{-}up}$ has been transformed into a timed transition $T_{ar}$, with delay $d_{ar}$, in order to model the client requests arrival rate. The resulting GSPN is shown in Fig. 3.

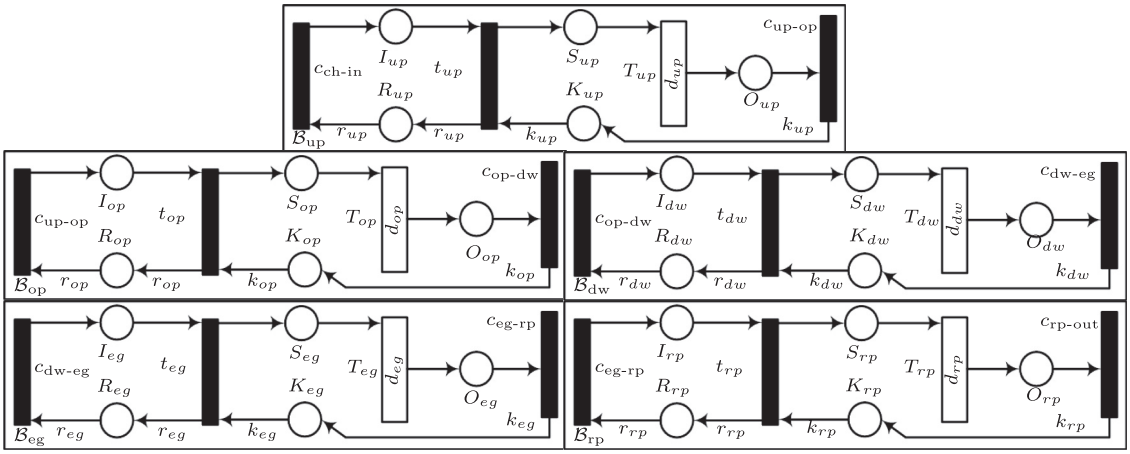


**Fig. 2.** General architecture of the GSPN model $\mathcal{SOA}$.

### 4.3. Model input parameters

Structurally, the GSPN $\mathcal{SOA}$ is already suitable for analysis. However, some parameters must be set up to properly represent the modeled behavior. These parameters are identified in (blue) bold in Fig. 3 and we show next how they can be individually obtained.

#### 4.3.1. Buffering resources

The aim is to find out the number of tokens that properly models the input buffer of each block $\mathcal{B}_i \in \mathcal{SOA}$ and, therefore, should be used to mark $R_i \in P_{\mathcal{B}_i}$. It is also defined the impact caused by the respective arcs $r_i$ on $R_i$. As $\#R_i$ is particular to each block, we illustrate in Fig. 4 an example of a BPEL message flow that contextualizes how these parameters are derived.

The directed flow connects three network buffers located: one in the *customer* node (CN), other in the *BPEL* node (BN) and another in the *service* node (SN). Each buffer has a specific size (*Bs*), which is respectively denoted by $Bs_{\text{CN}}, Bs_{\text{BN}}$ and $Bs_{\text{SN}}$. The communication between two any nodes is given by messages of size (*Ms*), respectively denoted by $Ms_{\text{CN}}, Ms_{\text{BN}}$ and $Ms_{\text{o}_i}$, where $o_i, i = 1, \cdots, n$ represents service operations in the service node (SN).

Now, we are in a better position to derive the parameters for $R_i$ and $r_i$ of each $\mathcal{B}_i \in \mathcal{SOA}$, as presented in Table 2.

Consider, for example, sending a message sized $Ms_{\text{BN}}$ from the BPEL node to a service operation $o_1$. In this case, the GSPN block $\mathcal{B}_{\text{up}}$ models the message departing from BPEL. So, clearly, the marking $\#(R_{up})$ is a function of $Bs_{\text{BN}}$. In the same way, the weights of the arcs $r_{up}$ are derived from $Ms_{\text{BN}}$. A similar idea applies to the other blocks. Particularly for $\mathcal{B}_{\text{op}}$ and $\mathcal{B}_{\text{eg}}$, which do not model network services (and therefore do not dispose of similar buffering systematic), we set up their parameters according to the input and output blocks.

#### 4.3.2. Processing resources

In our model, processing resources are modeled by the places $K_i$. Therefore, we have to derive the following input parameters:

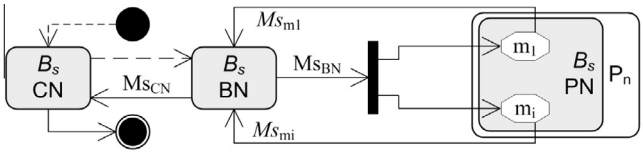


**Fig. 4.** Example of a message flow in BPEL.

**Table 2**
GSPN buffering parameters.

| $\mathcal{B}_i$: | $\mathcal{B}_{\text{up}}$ | $\mathcal{B}_{\text{op}}$ | $\mathcal{B}_{\text{dw}}$ | $\mathcal{B}_{\text{eg}}$ | $\mathcal{B}_{\text{rp}}$ |
|---|---|---|---|---|---|
| $\#(R_i)$ | $Bs_{\text{BN}}$ | $max(K_{up})$ | $Bs_{\text{PN}}$ | $max(K_{dw})$ | $Bs_{\text{BN}}$ |
| $R_i$ | $Ms_{\text{BN}}$ | 1 | $Ms_{\text{o}_i}$ | 1 | $Ms_{\text{BN}}$ |

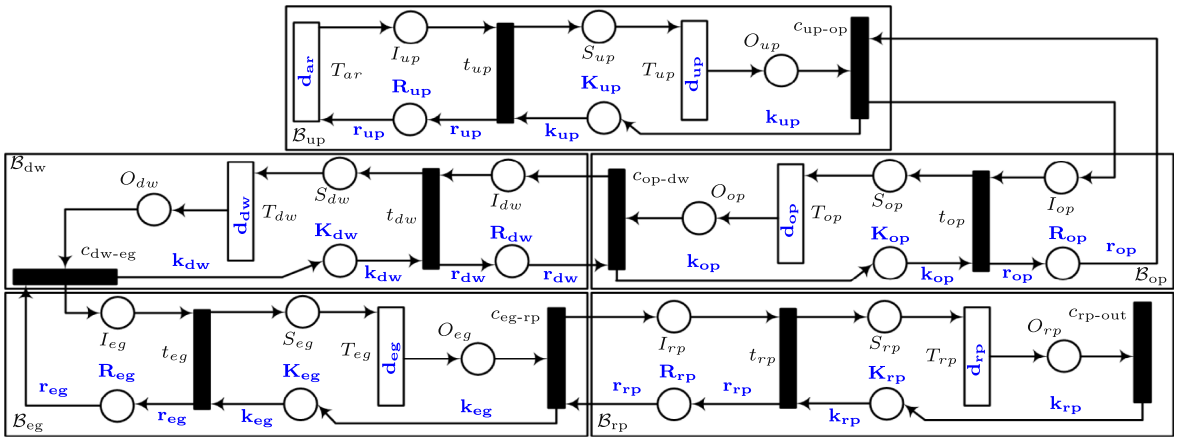


**Fig. 3.** Composed architecture of the GSPN model $\mathcal{SOA}$.

- a marking to the place $K_i$ (the upper bound of $S_i$);
- a weight to the arcs $k_i$ (the amount of resources demanded from/to $K_i$).

In practice, these parameters represent the capacity of each GSPN block for parallel processing. Next, we explain how to define them for each block in $\mathcal{SOA}$.

*Network processing.* Network blocks in $\mathcal{SOA}$ are modeled by $\mathcal{B}_{\text{up}}$, $\mathcal{B}_{\text{dw}}$ and $\mathcal{B}_{\text{rp}}$. Marking the correspondent place $K_i$ requires to know the total capacity of the network bandwidth ($C[n]$) linking the BPEL server to an external node (service provider) $p \in P$, where $P$ is the set of all service providers. Therefore,

$$\#(K_i) = C[n].$$

Tokens available in $K_i$ model the network bandwidth that is still available for communication. Actions on $K_i$ are implemented by weighting the arcs $k_i$, connected to $K_i$. With respect to the real system, these weights model the impact of messages exchanged through the network link, and are given as follows:

$$k_{up} = Ms_{\text{BN}}; \quad k_{dw} = Ms_{\text{o}_\text{i}}; \quad k_{rp} = Ms_{\text{CN}}.$$

Remark that each weight is directly associated to the class of messages released by the corresponding node.

*Service operation processing.* In $\mathcal{SOA}$, service operations are modeled by blocks $\mathcal{B}_{\text{op}}$. Then, we have to mark the place $K_{\text{op}} \in P_{\mathcal{B}_{\text{op}}}$, which contains the number of resources available to concurrently process a particular operation.

Let us first define a particular notation to be used next. For a set of service providers $P = \{p_1, \ldots, p_x\}$, each node $p_i \in P$ may provide a set of services denote by $S[p] = \{s_1, \ldots, s_y\}$. Each service $s_i \in S[p]$ may be composed by a set of operations denote by $O[s] = \{o_1, \ldots, o_w\}$. All services in a node provide, therefore, the set $O[p] = \{o_1, \ldots, o_z\}$.

Now, observe that it is usual for a service $s_i \in S[p]$ to implement a number of internal operations, in a way that $O[s]$ is rarely unitary. Naturally, modeling $s_i$ requires to share the amount of processing resources in $p_i$ among those $w$ operations in $O[s]$ an also among those $z$ operations in $O[p] \supseteq O[s]$.

In GSPN, this is equivalent to merge every single place $K_{op}$, of each block $\mathcal{B}_{\text{op}}$ that models operations provided by a server $p_i$. So, from now denoted by $K_p$ such centralized place. The arcs from/to $K_p$ remain, nevertheless, individual for each operation in $O[p]$.

It remains to be defined the parameters for $\#(K_p)$ and $k_{op}$. For that we measure the response time of every single operation $o_i \in O[p]$. The aim is to estimate how many instances of $o_i$ can be concurrently processed by the server that provides it. In particular, we are interested on the number of instances that can be handled without generating processing queues. Therefore, from samples of $o_i$ we collect:

- $\tau_{\text{o}_\text{i}}$: processing time of $o_i$ before queue formation;
- $\varpi_{\text{o}_\text{i}}$: maximum number of instances of $o_i$ that can be simultaneously processed.

We remark that, to collect $\tau_{\text{o}_\text{i}}$ and $\varpi_{\text{o}_\text{i}}$, we gradually increase the workload over $o_i$ until the point where requests start to accumulate. This can be identified by a perceptible increase in the response time.

We now define $\#(K_p)$ by

$$\#(K_p) = \text{LCM}(\varpi_{o_1}, \varpi_{o_2}, \ldots, \varpi_{o_z}), \tag{2}$$

where LCM is the *lowest common multiple* of each operation $o_i, i = 1, \ldots, z$ that is upheld by a given provider $p_i \in P$.

The use of the LCM idea to mark $\#(K_n)$ allows us to proportionally distribute the amount of processing resources among the operations provided by the same server. To accomplish this so, we have to weight the arcs $k_{op}[o_i]$ in such a way that it might reflect the demand of resources of each operation $o_i$. Eq. (3) shows how to weight $K_{op}[o_i]$.

$$k_{op}[o_i] = \frac{\#(K_n)}{\varpi_{o_i}}. \tag{3}$$

For example, let $o_1$ and $o_2$ be two operations provided by a server $p$, such that $o_1$ is twice more computationally expensive than $o_2$, i.e., $\varpi_{o_2} = 2 \cdot \varpi_{o_1}$. In this case, LCM $(\varpi_{o_1}, \varpi_{o_2}) = 2 \cdot \varpi_{o_1} = \#(K_p)$ and $k_{op}[o_1] = 2$ and $K_{op}[o_2] = 1$.

*BPEL processing.* The block $\mathcal{B}_{\text{eg}}$ aims to model the time spent by BPEL to orchestrate and execute the hole service composition. In $\mathcal{B}_{\text{eg}}$, marking the place $K_{eg}$ requires to somehow identify the capacity of the BPEL server for parallel processing. The way we propose to derive $\#(K_{eg})$ is the following. From BPEL, we send requests to each operation $o_i \in O[p]$ and collect:

- $P_{kg}Out_{o_i}$: the size of the package sent to $o_i$ (bytes);
- $P_{kg}In_{o_i}$: the size of the package replied by $o_i$ (bytes);
- $\varpi_{Pkg_{o_i}}$: maximum number messages simultaneously handled during this communication, before queue formation.

Now, we merge the places $K_{eg}$, of each blocks $\mathcal{B}_{\text{eg}}$ in $\mathcal{SOA}$, into a single place $K_e$, in order to share the BPEL processing resources. Then, $\#(K_e)$ can be derived as follows:

$$\#(K_e) = \text{LCM}(\varpi_{Pkg_{o_1}}, \varpi_{Pkg_{o_2}}, \ldots, \varpi_{Pkg_{o_z}}). \tag{4}$$

Without loss of generality, (4) can be extended to all $p \in P$. The resources consumption in $\#(K_e)$ can be now defined by

$$K_{eg}[o_i] = \frac{\#(K_e)}{\varpi_{Pkg_{o_i}}}. \tag{5}$$

#### 4.3.3. Delay parameters

We now deal with the delay of each block $\mathcal{B}_{\text{i}}$, which consists in finding a *real* number to be assigned to the parameter $d_i$ of each timed transition $T_i$.

*Network delays.* Consider a network linking a BPEL server to a node $p \in P$. The delays for network blocks are derived as in (6)

$$d_i = L[p] + \frac{Ms_{\text{j}}}{J[p]}, \tag{6}$$

where:

- $L[p]$ is the latency (ping time) of the network;
- $J[p]$ is the mean throughput of the network; and
- $Ms_{\text{j}}$ is the message size obtained such that:
  - $j = BN$ when $\mathcal{B}_{\text{i}} = \mathcal{B}_{\text{up}}$;
  - $j = o_i$ when $\mathcal{B}_{\text{i}} = \mathcal{B}_{\text{dw}}$; and
  - $j = CN$ when $\mathcal{B}_{\text{i}} = \mathcal{B}_{\text{rp}}$.

That is, $d_i$ is derived from a proper combination of flow direction, message sizes, network's bandwidth and latency.

*Operation delay.* The delay of the transition $T_{op}$ of the block $\mathcal{B}_{\text{op}}$ is defined according to the response time collected when measuring the service operation, i.e., $d_{op} = \tau_{\text{o}_\text{i}}$.

*Engine delay.* To derive the delay of the transition $T_{eg}$ of each block $\mathcal{B}_{\text{eg}}$, we have to first calculate:

- $S_{o_i}$: the amount of XML code (bytes) processed by BPEL to orchestrate an operation $o_i$. $S_{o_i}$ is calculated as in (7).

$$S_{o_i} = \frac{P_{kg}In_{o_i} + P_{kg}Out_{o_i}}{2}. \quad (7)$$

- $D_{o_i}$: the delay $x \in \mathbb{R}$ (seconds) measured to process $S_{o_i}$;

From $S_{o_i}$, $D_{o_i}$ and $\varpi_{Pkg_{o_i}}$ (messages simultaneously handled), one can calculate $d_{eg}$, as in (8).

$$d_{o_i} = \frac{S_{o_i}}{\varpi_{Pkg_{o_i}}} \cdot D_{o_i}. \quad (8)$$

For example, suppose that a service operation $o_1$ is to be invoked by a BPEL server. The messages to be requested and replied are composed respectively by 300 and 200 bytes and the BPEL engine takes 0,7 s to orchestrate them. Besides, consider that up by 4 requests can be concurrently orchestrated. So, in our model, $P_{kg}In_{o_1} = 200$, $P_{kg}Out_{o_1} = 300$, $D_{o_1} = 0,7$, and $\varpi_{Pkg_{o_1}} = 4$. In this way, $S_{o_1} = (200 + 300)/2 = 250$ and $d_{o_1} = (250/4) * 0,7 = 43,75$.

#### 4.3.4. Block response time

Now we can simulate the GSPN model to estimate the metrics of interest, such as the SOA process *response time* ($\Omega$).

Estimating the $\Omega$ of $\mathcal{SOA}$ takes into account the concept of *expectation* ($\xi$), which represents the average of how many tokens are expected in a given place $P_i$, i.e., $\xi(P_i)$. Obtaining $\Omega$ also requires to know the *average rate* $\lambda$ in which tokens arrive in a given block $\mathcal{B}_i$, i.e., $\lambda_{\mathcal{B}_i}$.

From $\xi$ and $\lambda$, one can estimate $\Omega$ as in (9).

$$\Omega(\mathcal{B}_i) = \frac{\xi(\mathcal{B}_i)}{\lambda_{\mathcal{B}_i}} \text{ or, equivalently, } \xi(\mathcal{B}_i) * d_i. \quad (9)$$

In (9), $\xi(\mathcal{B}_i)$ has been generalized to the entire bock $\mathcal{B}_i$, such that $\xi(\mathcal{B}_i) = \xi(I_i) + \xi(S_i) + \xi(O_i)$. The same reasoning can be applied to generalize (9) to all the blocks in $\mathcal{SOA}$, as follows.

$$\Omega(\mathcal{SOA}) = \Omega(net) + \Omega(op) + \Omega(eg), \quad (10)$$

such that,

$$\Omega(net) = \frac{\xi(\mathcal{B}_{\text{up}})}{\lambda_{\mathcal{B}_{\text{up}}}} + \frac{\xi(\mathcal{B}_{\text{dw}})}{\lambda_{\mathcal{B}_{\text{dw}}}} + \frac{\xi(\mathcal{B}_{\text{rp}})}{\lambda_{\mathcal{B}_{\text{rp}}}}, \quad (11)$$

$$\Omega(op) = \frac{\xi(\mathcal{B}_{\text{op}})}{\lambda_{\mathcal{B}_{\text{op}}}} \quad (12)$$

and

$$\Omega(eg) = \frac{\xi(\mathcal{B}_{\text{eg}})}{\lambda_{\mathcal{B}_{\text{eg}}}}. \quad (13)$$

### 4.4. Composite GSPN model

Now we show how BPEL orchestration structures (*sequence*, *if*, *flow* and *loop*) can be modeled in GSPN. We properly adjust the GSPN invocation model *Inv* in (1), in order to allow its composition and redesign. The concepts of *Inv* ((1)) and constructor $\mathcal{C}$ (Definition 4) are required next. Algorithm 1 shows how a ⟨sequence⟩ activity can be constructed in GSPN, as a function of the fusion of *Inv* structures.

**Algorithm 1:** GSPN sequential composition.

**Input**: $\mathcal{C}(b, \text{ch-in}, \text{ch-out})$;
**Output**: $\mathcal{B}_{\text{seq}}$;
Read $n$; Set $i = 1$, $\mathcal{B}_{\text{seq}} = \emptyset$,
$Inv' = \mathcal{C}(up_i, \text{up}_i, \text{op}_i) \cup \mathcal{C}(op_i, \text{op}_i, \text{dw}_i) \cup \mathcal{C}(dw_i, \text{dw}_i, \text{eg}_i)$;
**for** $i = 1 \cdots n$ **do**
  $\mathcal{B}_{\text{seq}} := \mathcal{B}_{\text{seq}} \cup Inv' \cup \mathcal{C}(eg_i, \text{eg}_i, \text{up}_{i+1})$;
  i:=i+1;
**end**
**return** $\mathcal{B}_{\text{seq}}$;

Note that blocks $\mathcal{B}_{\text{up}}$, $\mathcal{B}_{\text{op}}$ and $\mathcal{B}_{\text{dw}}$ are merged in sequence and the output of $\mathcal{B}_{\text{eg}}$ is adapted to restart a new invocation, as it is merged to the input of $\mathcal{B}_{\text{up}}$, characterizing a sequential flow. Fig. 5 shows the resulting GSPN.

It remains to compose a final block $\mathcal{B}_{\text{rp}}$ to $\mathcal{B}_{\text{seq}}$, in order to model the reply to the initial requestor. Thus, $\mathcal{B}_{\text{seq}} \cup \mathcal{C}(rp, \text{up}_n, \text{out})$ is the final form of a BPEL ⟨sequence⟩ activity modeled in GSPN.

Different structures of flow in BPEL are also possible to be modeled in GSPN. This only requires to properly arrange ⟨sequence⟩ activities before replying a request. We show in Fig. 6 how (a) alternative path (⟨If⟩), (b) parallel (⟨Flow⟩) and (c) iteration ((c1) ⟨While⟩ and ⟨ForEach⟩; (c2) ⟨RepeatUntil⟩) activities can be modeled in GSPN.

Note that each GSPN structure designs sequential invocations. Naturally, those structure can also be recombined with each other and we assume that this is an engineering task that can be properly addressed.

#### 4.4.1. Resources sharing

When composing GSPN structures, it becomes important to define how they concur by resources, as they aim to model systems that naturally share resources. This is addressed next.

*Network resources.* It is assumed that the network link connecting the BPEL server to external links (services and customers) is a two-way channel where download and upload have individual bandwidth. Thus, messages networked through a given direction share the same bandwidth. In GSPN this effect is modeled as follows.

- *Upload link*: in GSPN, $\mathcal{B}_{\text{up}}$ and $\mathcal{B}_{\text{rp}}$ are the blocks modeling network uploads. To be consistent to the two-way channel policy, these blocks are required to share the same structure $K_i$ of resources, i.e., their places $K_{up}$ and $K_{rp}$ have to be merged.
- *Download link*: in GSPN, $\mathcal{B}_{\text{dw}}$ is the block modeling network downloads from services to BPEL. Therefore, every place $\mathcal{B}_{dw}$ has to be merged.

*Processing resources.* It is considered that a node $p \in P$ equivalently shares its computational power among the threads that are to be processed. To reproduce similar effect in GSPN, we merge the places $K_{op}$ of every block $\mathcal{B}_{\text{op}}$ that models an operations in $p$.

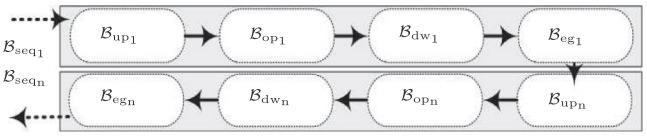


**Fig. 5.** Sequential GSPN composition.

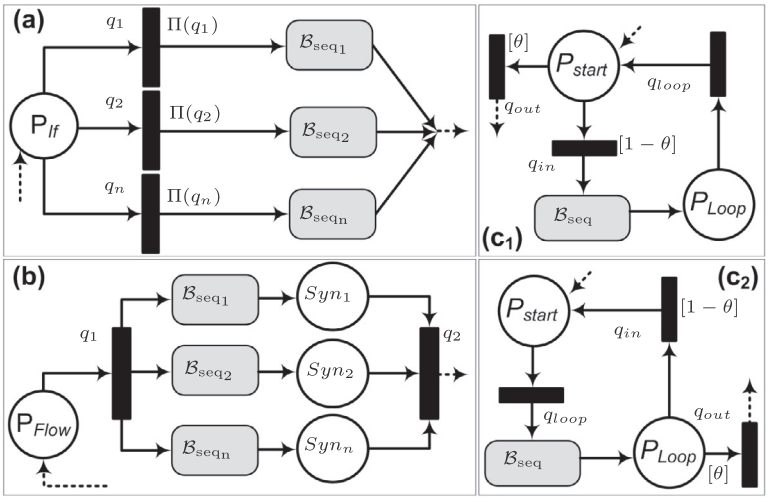


**Fig. 6.** GSPN composition rules.

*BPEL engine resources.* Since the orchestration task is entirely handled by BPEL, the engine model merges the places $K_{eg}$ of every block $\mathcal{B}_{eg}$.

### 4.5. GSPN composition analysis

Now we show how to collect performance metrics from the GSPN structures in Fig. 6. Given the set of GSPN blocks $\mathcal{SOA} = \{\mathcal{B}_{up}, \mathcal{B}_{op}, \mathcal{B}_{dw}, \mathcal{B}_{eg}, \mathcal{B}_{rp}\}$, let

$$\xi(\mathcal{B}_i) = \xi(I_i + S_i + O_i)$$

be the expectation of tokens into a block $\mathcal{B}_i \in \mathcal{SOA}$. Also, for $\mathcal{B}_{seq_i} = \mathcal{B}_{up_i} \cup \mathcal{B}_{op_i} \cup \mathcal{B}_{dw_i} \cup \mathcal{B}_{eg_i}$, let

$$\xi(\mathcal{B}_{seq_i}) = \xi(\mathcal{B}_{up_i} + \mathcal{B}_{op_i} + \mathcal{B}_{dw_i} + \mathcal{B}_{eg_i})$$

be the expectation of tokens into the GSPN block $\mathcal{B}_{seq_i}$. The response time $\Omega$, of each structure presented in Fig. 6, can be estimated as follows.

**Definition 7** (*Sequential flow*). For $i = 1, \ldots, n$, let $\mathcal{B}_{seq} = \{\mathcal{B}_{seq_1}, \ldots, \mathcal{B}_{seq_n}\}$ be a set of blocks $\mathcal{B}_{seq_i}$ sequentially arranged as in Fig. 5. Define:

- Arrival rate: $\lambda = \lambda_{\mathcal{B}_{seq_i}} = \lambda_{\mathcal{B}_{seq_n}}$;
- Expectation in $\mathcal{B}_{seq}$:

$$\xi(\mathcal{B}_{seq}) = \sum_{i=1}^{n} \xi(\mathcal{B}_{seq_i}); \tag{14}$$

- Mean response time of $\mathcal{B}_{seq}$:

$$\Omega(\mathcal{B}_{seq}) = \frac{\xi(\mathcal{B}_{seq})}{\lambda}. \tag{15}$$

**Definition 8** (*Conditional flow*). For $i = 1, \ldots, n$, let $\mathcal{B}_{if} = \{\mathcal{B}_{seq_1}, \ldots, \mathcal{B}_{seq_n}\}$ be a set of blocks $\mathcal{B}_{seq_i}$, conditionally arranged as in Fig. 6(a), and let $\Pi(q_i)$ be the priority function associated to the transitions $q_i$ (possible paths). Define:

- Arrival rate: $\lambda_i = \lambda \cdot \Pi(q_i)$;
- Expectation in $\mathcal{B}_{if}$:

$$\xi(\mathcal{B}_{if}) = \left(\sum_{i=1}^{n} \xi(\mathcal{B}_{seq_i})\right), \tag{16}$$

- Mean response time of $\mathcal{B}_{if}$:

$$\Omega(\mathcal{B}_{if}) = \frac{\xi(\mathcal{B}_{if})}{\lambda}. \tag{17}$$

**Definition 9** (*Parallel flow*). For $i = 1, \ldots, n$, let $\mathcal{B}_{flow} = \{\mathcal{B}_{seq_1}, \ldots, \mathcal{B}_{seq_n}\}$ be a set of blocks $\mathcal{B}_{seq_i}$, concurrently arranged as in Fig. 6(b). Define:

- Arrival rate: $\lambda_i = \lambda$;
- Expectation in $\mathcal{B}_{flow}$:

$$\xi(\mathcal{B}_{flow}) = \frac{1}{n}\left(\sum_{i=1}^{n} \xi(\mathcal{B}_{seq_i}) + \xi(Syn_i)\right), \tag{18}$$

- Mean time for synchronization in $\mathcal{B}_{flow}$:

$$\Omega(Sync) = \frac{1}{n \cdot \lambda} \sum_{i=1}^{n} \xi(Syn_i); \tag{19}$$

- Mean response time in $\mathcal{B}_{flow}$, including synchronization:

$$\Omega(\mathcal{B}_{flow}) = \frac{\xi(\mathcal{B}_{flow})}{\lambda}. \tag{20}$$

**Definition 10** (*Iterative flow*). Let $\mathcal{B}_{seq}$ be a sequential GSPN block iteratively disposed as in Fig. 6 $c_1$ and $c_2$. Define:

- Arrival rate: $\lambda = \frac{\lambda}{\theta}$, where $\theta$ defines the probability of the exit loop transition $q_{out}$ to be true.
- Expectation in $\mathcal{B}_{loop}$: $\xi(\mathcal{B}_{loop}) = \xi(\mathcal{B}_{seq})$;
- Mean response time of $\mathcal{B}_{loop}$:

$$\Omega(\mathcal{B}_{loop}) = \frac{\xi(\mathcal{B}_{loop})}{\lambda}. \tag{21}$$

## 5. Availability model

An important complement for the GSPN performance model is introduced in this section. The availability model, described next, has been constructed to be combined to the performance model in such a way that it becomes possible to estimate the percentage of requests overcoming a given condition. In practice, this condition can be associated to a contractual clause of a SLA. The

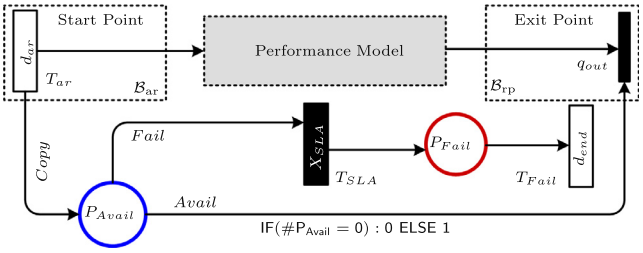


**Fig. 7.** GSPN availability model.

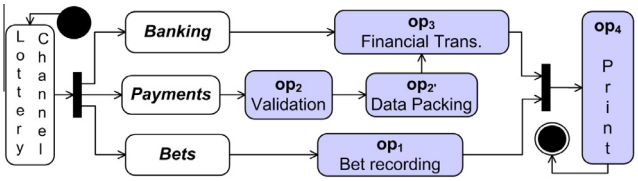


**Fig. 8.** Lottery process – activity diagram.

combination of performance and availability estimations has proved to be suitable to support advanced SLA planning in SOA.

The basic idea behind this new concept is to encapsulate the performance model into the scope of a timeout mechanism. In this way, besides to estimate mean response time, we can also calculate whether or not a request violates a given SLA clause, as well as the percentage of requests violating contracts. The structure of the availability model is depicted in Fig. 7.

When the transition $T_{ar}$ fires, it sends a token to the performance model and it also generates a copy of each token (arc *Copy*), storing this copy into a place named $P_{Avail}$. The deterministic timed transition $T_{SLA}$ defines a delay $X_{SLA}$ which, in practice, represents the time to be waited until recording a failure.

The behavior while tokens wait in $P_{Avail}$ is as follows. If the transition $q_{out}$ (exit point of the performance model) fires before the timeout $X_{SLA}$ is reached, then both tokens (from the performance model and from $P_{Avail}$) are discarded from the model. In this case, the transition $T_{SLA}$ is disabled (as there is no enabling precondition) and no failure is recorded, i.e., failure is not counted for tokens that have been processed within an acceptable MRT.

On the other hand, if the delay $X_{SLA}$ is reached before the transition $q_{out}$ gets enabled (before the performance model finishes), then a token is moved from the place $P_{Avail}$ (arc *Fail*) to the place $P_{Fail}$, where a failure can be counted. From $P_{Fail}$, tokens are discarded by the timed transition $T_{Fail}$, with delay $d_{end} = 1$ and firing semantic *Exclusive Server*. This configuration is necessary for the marking probability to be estimated in the place $P_{Fail}$ which actually leads to the estimation of the failure rate.

Remark that even if a failure is recorded, the performance model remains on track, i.e., it keeps simulating the received tokens. For this to be possible, the arc *Avail* has to have no effect on the enabling condition of $q_{out}$ when a token fails. In our model, we implement this by the formula IF(#P$_{Avail}$ = 0) : 0ELSE1, which changes the enabling condition of the transition $q_{out}$ when a token fails, providing independence between performance and failure models.

## 6. Analysis

The proper combination of performance and failure models can supply essential information for decision makers. To illustrate this, we next evaluate two examples of SOA systems. The examples are inspired on industrial applications that have been adapted to concentrate on the aspects of interest of this work, such as the electronic-centered operations. *Java Technology* has been used to implement the services, *BPEL* language to orchestrate and execute them, and *Timenet* tool (Zimmermann, 2014) for GSPN modeling. Simulations have been performed under a confidence level of 95% and relative error of 10%.

### 6.1. Lottery process

The first example we present is inspired on a SOA system that integrates lottery channels. The system works as illustrated in Fig. 8 and is usually susceptible to intensive demands of operations.

The system comprises three classes of transactions named *Bank*, *payments* and *Bet*. When a transaction is started (in a lottery house) it interacts with remote service operations, in a black box view, until an answer is sent back to the customer. Our focus here is to go inside this black box and model the system behavior in GSPN. We start by implementing and deploying the five operations identified in Fig. 8. Then, we gradually generate workload over them, in a way to avoid queue formation. Then, we measure each operation, collecting the parameters $\tau_{o_i}$, $\varpi_{o_i}$ and $Ms_{o_i}$, which are shown in Table 3.

Afterwards, the process in Fig. 8 is modeled by a GSPN that receives the parameters from Table 3. To simulate the GSPN, we consider four different orchestration models: (1) simulating the whole process (randomly choosing the transactions *Banking*, *Payment* or *Bets* to be performed); (2, 3 and 4) forcing the execution of a particular transaction. We simulate each orchestration under 1, 5, 10, 15 and 20 req/s, although the model is naturally extensible to different workloads. After the simulations, we collect the estimated *mean response time* (MRT). To validate those estimations, we also measure the real system under the same workload conditions. The comparison is presented by Table 4.

Note that, even stressing the system by increasing the workload of requests, the GSPN estimations remain close to the behavior measured from the real application. The observed accuracy of the GSPN estimations, for each orchestration model, has been on the respective order of: *Whole Process* 92%, *Pay* 85%, *Bank* 83% and *Bet* 86%. In average, the GSPN estimations follow the real behavior with a stochastically reasonable accuracy of 86%.

**Table 3**
GSPN input collections.

| Collected metric | Service operation | | | | |
|---|---|---|---|---|---|
| | $op_1$ | $op_2$ | $op_{2'}$ | $op_3$ | $op_4$ |
| $\tau_{o_i}$ (ms) | 480 | 70 | 580 | 210 | 950 |
| $\varpi_{o_i}$ (Simult. Req.) | 7 | 10 | 6 | 5 | 10 |
| $Ms_{o_i}$ (bytes) | 250 | 150 | 275 | 230 | 245 |

**Table 4**
Lottery process: performance evaluation.

| Process orchestration | | Workload level (req/s) | | | | |
|---|---|---|---|---|---|---|
| | | 5 | 10 | 15 | 20 | |
| Whole Process | Measured MRT | 2,79 | 4,22 | 7,43 | 10,84 | 13,78 |
| | Estimated MRT | 3,05 | 4,53 | 6,65 | 10,69 | 15,84 |
| Pay | Measured MRT | 3,77 | 5,04 | 12,43 | 17,22 | 20,78 |
| | Estimated MRT | 3,49 | 4,41 | 10,28 | 22,11 | 24,99 |
| Bank | Measured MRT | 2,07 | 3,89 | 8,83 | 13,42 | 17,25 |
| | Estimated MRT | 2,62 | 3,23 | 9,96 | 17,46 | 20,05 |
| Bet | Measured MRT | 2,10 | 4,18 | 7,75 | 12,68 | 17,74 |
| | Estimated MRT | 2,21 | 3,41 | 7,40 | 16,83 | 20,94 |

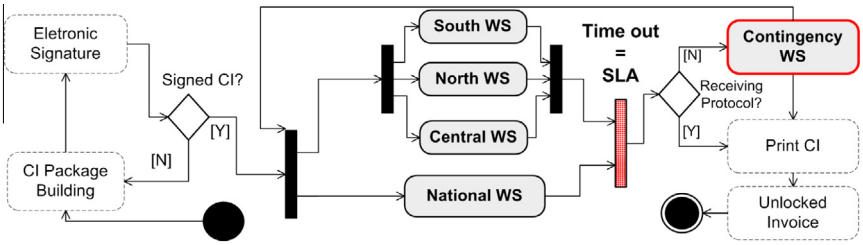


**Fig. 9.** Activity diagram of the evaluated process.

### 6.2. Electronic invoice system

The second example to be presented is inspired on a SOA application that manages taxes information in electronic invoices. The system is composed by five services, orchestrated according to the workflow in Fig. 9.

When the customer emits an invoice, a data package is sent via Internet and an answer is waited for the order to be released. Internally, the orchestrator simultaneously invokes two web services: (i) a regional service, here represented by *South*, ıNorth and *Central*; (ii) a *national* service centralizing requests all around the country. The orchestrator is able to reply the customer only when an answer is received from both services. Usually, a SLA regulates this waiting time and, when it is violated, the customer is replied anyway, using a *Contingency* strategy that works as a fault prevention but is better not to be used.

The question that guides us throughout this section is: "*what is the contingency utilization under a given workload?*", that is, "*how many requests are supported by the system until the contingency service is needed?*". Although this might not be so easily answered using empirical evidences, we show that this can be informed via model. We start from the performance evaluation, as in the previous example. Afterwards, we combine each performance estimation to a range of timeouts that simulate possible SLAs for the process. For the results to be derived next, the proposed availability model plays an essential role.

### 6.3. Process performance evaluation

Firstly, we develop and deploy each service operation.

Secondly, operations are individually measured and metrics for $\tau_{o_i}, \varpi_{o_i}$ and $Ms_{o_i}$ are collected, as presents Table 5.

Remark that these inputs are collected with the highest possible workload before observing queue formation. They are then used to calculate the input parameters for the GSPN that models the process in Fig. 9. To simulate the GSPN, we now consider the following classes of workload levels: *Light:* (1 req/s), *Mid:* (5 req/s) and *Heavy:* (10 req/s). Table 6 presents the estimated MRT under each workload level. It is also presented the real system MRT when running under the same workload.

Last row shows how the system performance behaves upon the workload variation. It changes due to the nondeterminism caused by resources consumption. Even so, the GSPN model follows this changing in a very similar way, independently from the applied workload. The accuracy observed under *Light*, *Mid* and *Heavy* workloads are respectively on the order of 69%, 74% and 96%. In average, the accuracy is on the order of 80%. It worth noticing that the best estimation is unexpectedly obtained under the most variable condition (96% under *Heavy* workload). Presumably, it happens because intensive demands generate high concentrations of queues, which exploits the GSPN model at all.

**Table 5**
GSPN input collections.

| Collected metric | South | North | Central | National |
|---|---|---|---|---|
| $\tau_{o_i}$ (ms) | 1.145 | 1.010 | 750 | 1.550 |
| $\varpi_{o_i}$ (Simult. Req.) | 2 | 3 | 4 | 2 |
| $Ms_{o_i}$ (bytes) | 240 | 230 | 245 | 225 |

**Table 6**
Performance evaluation.

| Workload level: | Light | Mid | Heavy |
|---|---|---|---|
| *Measured* MRT (s): | 1,90 | 5,10 | 7,58 |
| *Estimated* MRT (s): | 2,76 | 6,91 | 7,32 |

### 6.4. Planning SLAs for the mean response time

In addition to the performance evaluation, our model has shown to be also suitable to supply different organizational information. For example, consider the following question.

*Question 1: For the process in Fig. 9, let $W \in \mathbb{N}^*$ be the observed workload level (req/s). Which SLA for the MRT admits at most 10% of violation?*

In order to answer this question, we again simulate the model, this time combining performance and availability evaluations. We start by assigning a given delay $X_{SLA}$ ($x$ seconds) to the transition $T_{SLA}$ of the timeout structure. As a return, we estimate the percentage of failure under $W$. If the performance model simulates faster than $x$, the failure rate tends to be low. However, if we gradually reduce $X_{SLA}$, the failure rate tends to grow and we observe when it overcomes 10%. Remark that it would be complex to empirically determine when violations get close to 10%, although this is essential for the SLA planning.

In Table 7, we present a set of answers to the Question 1. We regressively variate $X_{SLA}$ from 9 to 1 and, for each value of $X_{SLA}$, we also variate $W$, considering $W_1 = Light$, $W_2 = Mid$, $W_2 = Heavy$ as defined before

For example, if $W_1$ is the typical process workload, then up to 5 s are required for the MRT to keep the failure rate less than 10%. However, 5 s is not appropriate for $W_2$, as it would lead to 19% of looses, violating the specification. $W_2$ requires actually a MRT of 7 s, while $W_3$ requires 9 s.

**Table 7**
Mean response time planning.

| Suggested SLA for the process MRT (s) | Estimated failure rate | | |
|---|---|---|---|
| | $W_1$ (%) | $W_2$ (%) | $W_3$ (%) |
| 9 | **3** | **8** | **10** |
| 7 | **6** | **10** | 15 |
| 5 | **8** | 19 | 21 |
| 3 | 13 | 32 | 36 |
| 1 | 37 | 97 | 99 |

Bold and underline: Indicates whether or not the SLA has been achieved for a particular workload $W_i$.

**Table 8**
Workload planning.

| Predefined SLA (process response time) (s) | Tuned workload (req/s) | Estimated failure rate (maximum 10%) (%) |
|---|---|---|
| 2 | **0,274** | 9,999 |
| 4 | **0,909** | 9,996 |
| 6 | **2,623** | 9,999 |
| 8 | **3,345** | 9,997 |
| 10 | **9,913** | 9,413 |
| 12 | **15,385** | 8,849 |

### 6.5. Planning the workload level

Now, we introduce another class of information that our model can assess.

*Question 2: For the process in Fig. 9, assume that a given MRT has been contracted. Now, let $W = \{w_1, \ldots, w_n\}$ be a set of possible workloads to the system. Which $w_i \in W$ would keep the system under the MRT SLA in a way to allow at most 10% of SLA violation?*

In the following, we answer this question for six possibly contracted MRT: $2, 4, 6, 8, 10$, and 12 s. For each MRT, we simulate the model in such a way that the workload is tuned to discover which one produces at most 10% of SLA violation. Table 8 presents the results.

First column outlines the contracted MRT. Second column shows the estimated workload. We stop tuning the simulations when we get a percentage close to (but under) 10% of SLA violation, as shows the third column.

## 7. Process structural analysis

We have shown how to monolithically exploit the GSPN model. This is an end-to-end analysis where the system is seen as a black box. Now, we modularize our approach to get inside the black box and provide internal analysis. This new perspective allows the investigation of specific steps of a business process, facilitating the system expansion planning and the modular resolution of structural problems, such as bottlenecks. The following challenge summarizes the kind of evaluation we aim to provide next.

*Challenge: For the process in Fig. 9, we have estimated a MRT of 2,76 s, 6,91 s and 7,32 s, respectively under $w_i$, $i = Light, Mid, Heavy$ workloads. Under what conditions one could ensure a MRT not superior to 5s, for any $w_i$?*

To address this challenge, we propose the systematic analysis shown in Fig. 10.

The *black box analysis* comprises the performance evaluation we have already presented, while the *white box analysis* can be described by the following phases:

- *Discover: the system is modularly simulated to evaluate the performance of its subsystems;*
- *Unbox: subsystems with low performance profile are opened to evaluate the performance of their internal components;*
- *Plan: alternatives for performance problems are suggested and an the improvements action plan is constructed;*
- *Implement: the improvements action plan is materialized.*

Now, we show how our challenging issue can be addressed by those steps.

### 7.1. Discovering phase (first round)

We start by splitting the SOA system in four subsystems named: *Network (Net)*; *Service processing (Exec)*; *BPEL engine (Eng)*; *Synchronization (Sync)* (as the evaluated process includes parallel steps). We then reproduce the performance evaluation for each subsystem. The goal is to identify saturation points delaying the process. The row *first* in Table 9 presents the results.

Note that *Exec* is clearly the more expensive subsystem, consuming about 75% of the total process time. So, we have identified a saturation point and it remains to unbox it to be internally analyzed in our white box phase.

### 7.2. Unboxing phase (first round)

We open the *Exec* module to collect the time spent by each service operation. Then, we indicate their individual impact on

**Table 9**
Performance evaluation (discovering phase).

| Improvement round | Workload level | Mean response time (m s) | | | |
|---|---|---|---|---|---|
| | | Net | Exec | Eng | Sync |
| First | Light | 275 | **1.800** | 5 | 675 |
| | Mid | 200 | **5.350** | 11 | 1.351 |
| | Heavy | 215 | **6.070** | 6 | 1.025 |
| Second | Light | 275 | **525** | 500 | 175 |
| | Mid | 260 | **3.550** | 200 | 450 |
| | Heavy | 218 | **5.225** | 110 | 500 |
| Third | Light | 400 | **425** | 250 | 200 |
| | Mid | 465 | **2.450** | 200 | 400 |
| | Heavy | 315 | **3.800** | 215 | 425 |

Bold: Indicates the bottlenecker operation.

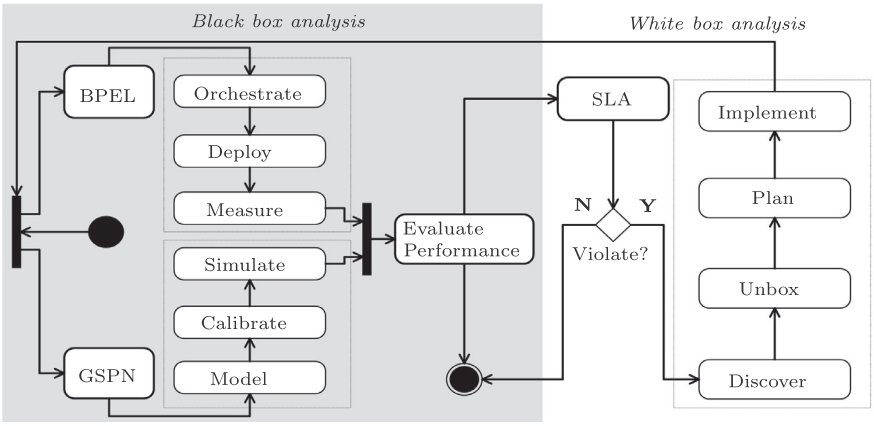


**Fig. 10.** Systematic processes improvement.

**Table 10**
Performance evaluation (unboxing phase).

| Opened module | Workload level | Services (percentage of spent time) | | | |
|---|---|---|---|---|---|
| | | South (%) | North (%) | Central (%) | Nat (%) |
| Exec | Light | 13 | 13 | 9 | **65** |
| | Mid | 19 | 12 | 5 | **64** |
| First round | Heavy | 19 | 16 | 6 | **59** |
| Exec | Light | 21 | 19 | 13 | **46** |
| | Mid | 25 | 11 | 6 | **58** |
| Second round | Heavy | 22 | 17 | 7 | **55** |
| Exec | Light | 25 | 22 | 12 | 32 9 |
| | Mid | 31 | 20 | 6 | 37 6 |
| Third round | Heavy | 30 | 22 | 7 | 37 5 |

Bold: Highlights the percentage of bottleneck concentration when it is not homogeneous.

**Table 11**
Systematic performance evaluation.

| Orchestration scenario | Data source | Response time (s) | | | | | |
|---|---|---|---|---|---|---|---|
| | | Light | | Mid | | Heavy | |
| Original system | Measured | 1,58 | ✓ | 5,10 | × | 7,58 | × |
| | Estimated | 2,76 | ✓ | 6,91 | × | 7,32 | × |
| Improved server | Measured | 1,14 | ✓ | 3,72 | ✓ | 5,20 | × |
| | Estimated | 1,47 | ✓ | 4,48 | ✓ | 6,03 | × |
| Reduced operation granularity | Measured | 1,23 | ✓ | 3,42 | ✓ | 4,52 | ✓ |
| | Estimated | 1,28 | ✓ | 3,52 | ✓ | 4,78 | ✓ |

the time spent by the module. In GSPN, this is equivalent to individually simulate $\mathcal{B}_{op}$ blocks. Table 10, row *first round*, summarizes the unboxing results.

It is shown in bold that the *Nat* operation is the bottlenecker of the *Exec* module. In fact, *Nat* operation is orchestrated in such a way that it receives a copy of every processed request, which makes it susceptible to overloading even under the *Light* workload. Thus, the next step in the white box analysis aims to design an action plan to minimize the *Nat* processing cost.

### 7.3. Planing step (first round)

The first attempting to optimize the *Nat* operation focuses on hardware infrastructure improvements.

### 7.4. Implementation step (first round)

We deploy the *Nat* operation in a server with more processing power. As a result, the response time has been reduced from $\tau_{Nat} = 1550$ ms to $\tau'_{Nat} = 1160$ m s. Then, we repeat all the steps from the black box analysis and the estimations are presented in Table 11 (line *improved server*).[2]

The adopted action plan has substantially reduced the process response time with respect to the first scenario (*original system* row), which proves that the model estimation was indeed correct. Yet, under *Heavy* workload, the process still takes about 6,03 s to answer. As, therefore, we have not yet achieved the challenging goal, we start the second round of the white box analysis.

### 7.5. Second round of improvements

*Discovering Phase:* from Table 9, row *second*, we have that the performance has been balanced under *Light* workload. Nevertheless, for *Mid* and *Heavy* workloads, a similar bottleneck is still concentrated on the *Exec* module.

*Unboxing Phase:* the internal analysis of the *Exec* module reveals that a bottleneck is still concentrated on the *Nat* operation (see row Exec *second round* in Table 10), although it shows to be more stabilized now.

*Planing Phase:* in the second attempting to get the challenging goal addressed, we consider to structurally modify the *Nat* operation. The plan is to reduce its granularity by splitting it in two different operations, so-named *Insert* and *Remake*. As, then, they get enabled for concurrent processing, the same functionalities implemented by the old *Nat* tend to be processed faster.

*Implementing Phase:* we implement the mentioned action plan and the new collections for the two new operations are respectively: $\tau_{Insert} = 975$ m s and $\varpi_{Insert} = 2; \tau_{Remake} = 650$ m s and $\varpi_{Remake} = 6$.

Third row (*reduced operation granularity*) of Table 11 presents the response time analysis for this new version of the system. By taking less than 5*s* to answer, under any $w_i$, the analysis evidences that the challenging goal has been properly addressed.

An idea about how balanced the system stays after the last intervention can be captured by performing again the white box analysis, completing the cycle of iterative tasks in Fig. 10. For example, by proceeding with the next *discovering phase*, we observe that the *Exec* module is still the most time-consuming subsystem (see line 3 in Table 9 – *third*). However, the *unboxing phase* reveals a more homogeneous performance cost among the operations composing the *Exec* module (see line three of Table 10). The local processing balance implicitly suggests a global absence of bottlenecks on the process.

## 8. Discussion and conclusions

This paper describes an innovative approach to estimate performance and availability of SOA systems using GSPN. The main goal is to provide an expert tool for supporting software engineers in the decision-making process at modeling time. There are a number of unique contributions of this work that differentiate it from related work in the field. Firstly, our modeling approach is modular, allowing for setting up different execution environments and easily switching from one configuration to another. Secondly, the same basic structures are reused to construct both the performance and the availability models. Moreover, instead of considering each web service as an isolated entity, this work takes into account the use of physical resources such as network band and processing power, and how the allocation of those resources to different workflow configurations impacts the overall performance of the system. The model also incorporates the notion that web services do not behave deterministically, but have a stochastic nature instead. All those elements are introduced in this work together with a methodology for planning SLAs in corporate applications.

Such contributions help to reduce a gap on existing expert systems literature regarding the support for strategic decisions during the design of SOA applications. Existing methodologies either focus on the rebinding of service endpoints during runtime or try to adapt the workflow using pre-defined execution alternatives. In none of those situations, however, it has been given to the software architect the opportunity to evaluate the cost and performance trade-offs under a strategic perspective. In a strategic decision making process, the designers must consider the real limitations of their infrastructure and plan for creating business value to their application's customers by differentiating their service from competitors (Neves, Oliveira, Lima, & Sabat, 2012). The ability of autonomous systems to reach such high level reasoning is limited. Thus, an adequate tool to support decision making at design time is

[2] ✓ and × indicate respectively whether or not the goal has been achieved.

paramount to allow for an SLA planning that fits the business plans of the organization.

The benefits of the proposed model have been illustrated by two examples representing prototypes of real systems. To validate our results, we compare the behavior estimated by the model against the behavior measured from the respective in-operation system. Overall, the results have shown to be sound. The accuracies observed in the performance evaluation of the examples have been respectively on the order of 86% and 80%, which we consider acceptable. Moreover, the methodology has shown to be consistent, since it has allowed to incrementally improve (second example) the original system to reach a predefined SLA.

We remark that our approach is not appropriate for a deterministic analysis, as we are more interested on the dynamic behavior of SOA. Moreover, we remark that our results can statistically variate under different simulation conditions.

Prospects of future works include the development of computational tools to integrate our model to a SOA platform, in such a way that estimations could be automatically provided, both at design and runtime. We also aim to propose a formal way to synthesize optimal orchestrators for SOA. This approach is different from the literature, as it allows QoS metrics to be ensured by construction, without requiring additional checking. A next step will integrate the synthesis method to the modeling proposed in this paper, such that the resulting orchestrator would naturally reflect the interventions suggested by the model.

## References


Abe, M., & Jeng, J. (2007). Authoring tool for business performance monitoring and control. In *IEEE int. conf. on service-oriented computing and applications. CA, USA*.

Almeida, V. A. F., & Menasce, D. A. (2001). *Capacity planning for web services: Metrics, models, and methods*. Prentice Hall.

Andrews, T. et al. (2003). BPEL4WS, business process execution language for web services version 1.1. IBM. URL <http://download.boulder.ibm.com>.

Apache (2011). jMeter 2.5.1. URL <http://jakarta.apache.org>.

Baresi, L., & Guinea, S. (2008). A dynamic and reactive approach to the supervision of bpel processes. In *Annual india software engineering conference, Hyderabad, India* (pp. 39–48).

Bo, C., Junliang, C., & Min, D. (2012). Petri net based formal analysis for multimedia conferencing services orchestration. *Expert Systems with Applications, 39*, 696–705.

Bruneo, D., Distefano, S., Longo, F., & Scarpa, M. (2010). Qos assessment of ws-bpel processes through non-markovian stochastic petri nets. In *IEEE int. symp. on parallel distributed processing* (pp. 1 –12).

Casati, F., Shan, E., Dayal, U., & Shan, M. (2003). Business-oriented management of web services. In *Communications of the ACM, New York, USA* (pp. 55–60).

Chase, J.S. et al. (2001). Managing energy and server resources in hosting centers. In *Symp. on operating systems principles, Alberta, Canada*.

Dambrogio, A., & Bocciarelli, P. (2007). A model-driven approach to describe and predict the performance of composite services. In *ACM workshop on software and performance, Buenos Aires, Argentina* (pp. 78–80).

Desrochers, A. A. (1994). *Applications of petri nets em manufacturing systems: Modeling, control and performance analysis*. IEEE Press.

HP (2012). HP enterprise technologies. URL <http://www8.hp.com/us/en/business-services>.

Hwang, S., Wang, H., Tang, J., & Srivastava, J. (2007). A probabilistic approach to modeling and estimating the qos of web-services-based workflows. In *Information sciences: An international journal* (pp. 5484–5503). NY, USA: Elsevier Science Inc..

IBM (2012). Service assurance for communication service providers. URL <http://www-01.ibm.com/software/tivoli>.

Josuttis, N. M. (2008). *SOA in practice* (1st ed.). Oreilly.

Jula, A., Sundararajan, E., & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications, 41*, 3809–3824.

Li, Y., Chen, H., Zheng, X., Tsai, C.-F., Chen, J.-H., & Shah, N. (2011). A service-oriented travel portal and engineering platform. *Expert Systems with Applications, 38*, 1213–1222.

Li, J., Fan, Y., & Zhou, M. (2004). Performance modeling and analysis of workflow. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 34*, 229–242.

Lin, C., & Kavi, K. (2013). A qos-aware bpel framework for service selection and composition using qos properties. *International Journal on Advances in Software, 6*, 56–68.

Marsan, M. A., Balbo, G., Conte, G., et al. (1995). Modelling with generalized stochastic Petri nets. In *Wiley series in parallel computing*. New York: Wiley.

Marsan, M.A., Balbo, G., & Conte, G. (1984). A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. In *ACM transactions on computer systems* (Vol. 2, pp. 1–11).

Marzolla, M., & Mirandola, R. (2007). Performance prediction of web service workflows. In *Int. conf. on quality of software architectures, Medford, USA* (pp. 127–144).

Metzger, A., Leitner, P., Ivanovic, D., Schmieders, E., Franklin, R., Carro, M., et al. (2015). Comparing and combining predictive business process monitoring techniques. *IEEE Transactions on Systems, Man, and Cybernetics: Systems, 45*, 276–290.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE, 77*, 541–580.

Neves, A., Oliveira, C., Lima, R., & Sabat, C. (2012). Computing strategic trade-offs in web service deployment and selection. In *2012 IEEE 19th International Conference on Web Services (ICWS)* (pp. 210–217). http://dx.doi.org/10.1109/ICWS.2012.15.

Oasis (2007). Web services business process execution language version 2.0.

Oliveira, C. A. L., Lima, R. M. F., Reijers, H. A., & Ribeiro, J. T. S. (2012). Quantitative analysis of resource-constrained business processes. *IEEE Transactions on Systems, Man and Cybernetics, Part A, 42*, 669–684.

Palomo-Duarte, M., García-Domínguez, A., & Medina-Bulo, I. (2014). Automatic dynamic generation of likely invariants for ws-bpel compositions. *Expert Systems with Applications, 41*, 5041–5055.

Parejo, J. A., Segura, S., Fernandez, P., & Ruiz-Cortés, A. (2014). Qos-aware web services composition using {GRASP} with path relinking. *Expert Systems with Applications, 41*, 4211–4223.

Raibulet, C., & Massarelli, M. (2008). Managing non-functional aspects in soa through sla. In *Int. conf. on database and expert systems application, Turin, Italy*.

Reisig, W., & Rozenberg, G. (1996). Informal introduction to Petri nets. In *Petri nets* (pp. 1–11).

Rud, D., Schmietendorf, A., & Dumke, R. (2006). Performance modeling of ws-bpel-based web service compositions. In *IEEE services computing workshops, CA, USA* (pp. 140–147).

Rud, D., Kunz, M., Schmietendorf, A., & Dumke, R. (2007). Performance analysis in ws-bpel-based infrastructures. In *UK performance engineering workshop*. England: Edge Hill University.

Rud, D., Schmietendorf, A., & Dumke, R. (2007). Performance annotated business processes in serviceoriented architectures. *International Journal of Simulation: Systems, Science & Technology, 8*, 61–71. Special Issue on Performance Modelling of Computer Networks, Systems and Services.

Solomon, A., & Litoiu, M. (2011). Business process performance prediction on a tracked simulation model. In *Proceedings of the third int. workshop on principles of engineering service-oriented systems PESOS '11* (pp. 50–56). New York, NY, USA: ACM.

Sturm, R., Morris, W., & Jander, M. (2000). *Foundations of service level management*. Sams Publishing.

Teixeira, M., Lima, R., Oliveira, C., & Maciel, P. (2009). Performance evaluation of service-oriented architecture through stochastic petri nets. In *IEEE int. conf. on systems, man, and cybernetics, TX, USA*.

Teixeira, M., Lima, R., Oliveira, C., & Maciel, P. (2010). A stochastic model for performance evaluation and bottleneck discovering on soa-based systems. In *IEEE int. conf. on systems, man, and cybernetics, Istanbul, Turkey*.

Teixeira, M., Lima, R., Oliveira, C., & Maciel, P. (2011). Planning service agreements in soa-based systems through stochastic models. In *ACM symp. on applied computing, TaiChung, Taiwan*.

Wu, Z., Yuan, L., Yang, J., & Xiong, W. (2008). A stochastic timed performance model for web service composition. In *IEEE int. symp. on information science and engineering, Jiaxing, China*.

Xia, Y., Liu, Y., Liu, J., & Zhu, Q. (2012). Modeling and performance evaluation of bpel processes: A stochastic-petri-net-based approach. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 42*, 503–510.

Xiong, P., Fan, Y., & Zhou, M. (2008). Qos-aware web service configuration. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 38*, 888–895.

Xiong, P., Fan, Y., & Zhou, M. (2010). A petri net approach to analysis and composition of web services. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans, 40*, 376–387.

Yoo, T., Jeong, B., & Cho, H. (2010). A petri nets based functional validation for services composition. *Expert Systems with Applications, 37*, 3768–3776.

Zimmermann, A. (2014). TimeNET 4.1. <http://www.tu-ilmenau.de/TimeNETs>.