Applying Process Analysis to the Italian eGovernment Enterprise Architecture*

Roberto Bruni[1], Andrea Corradini[1], Gianluigi Ferrari[1], Tito Flagella[2], Roberto Guanciale[1], and Giorgio Spagnolo[1]

[1] Dipartimento di Informatica, Università di Pisa, Italy
[2] Link.it, Pisa, Italy

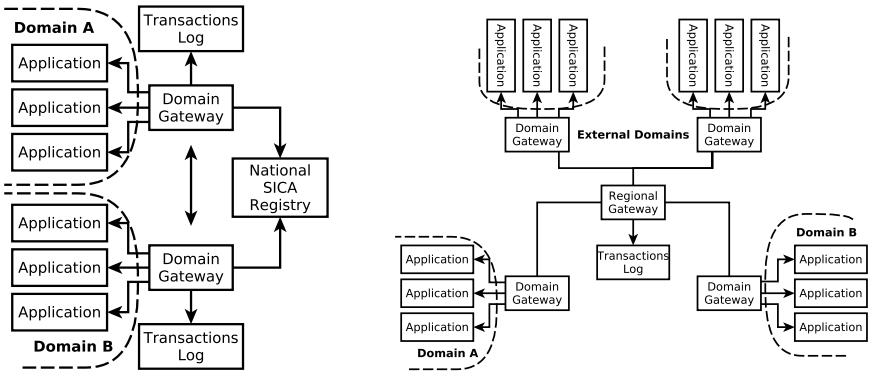
Abstract. We report our experiences gained when integrating process analysis activities into a regional gateway of the Italian eGov platform to promote real-time process monitoring within a Service Oriented Architecture. We exploit ProM, a state-of-the-art suite providing several analysis algorithms for business processes. First, we outline our technological integration efforts, focusing on the architectural changes and implementation strategies to make ProM tools available at runtime for monitoring the gateway. Next we improve an existing performance algorithm with a new approach to deal with invisible transitions when evaluating the synchronization times of complex nets. Finally, we introduce a methodology to transform high level process notations, like BPMN, to Petri Nets in order to enable the analysis techniques and convey back their results.

1 Introduction

In 2003 the CNIPA (National Center for IT in Public Administration, now DigitPA) began the specification of an Enterprise Architecture for ensuring interoperability among the software applications of the Italian Public Administrations. The resulting architecture is called SPCoop (*Public Cooperative System*) and it is based on a Service Oriented Architecture model. It was designed for standardizing both the service agreements (*contracts*) between applications of different domains and the communication protocol used for application interoperability: the former contain both formal (XSD, WSDL, ...) and semiformal documentation concerning service interoperability details; the latter is an extension of the SOAP 1.1 envelope, called *e-Gov Envelop* which consists of a custom SOAP header carrying various message addressing information.

According to the SPCoop specification, every Public Administration must offer its application services through a unique entity named *Porta di Dominio* (Domain Gateway), which also acts as a proxy for invocations of remote services. Thus the Domain Gateway interoperates with gateways of other domains for accepting or delivering messages to and from application services. Domain

* Research partially funded by Regione Toscana through project RUPOS *(Ricerca sull'Usabilità delle Piattaforme Orientate ai Servizi)* and by the European Integrated Project 257414 ASCENS.



(a) Standard Logging (b) Centralized Logging

Fig. 1: Business Transactions Logging in SPCoop

Gateways act according to service agreements recorded in a central registry named SICA (*Interoperability and Application Cooperation Service Registry*).

After some years of experimentation, on February 2005 the Italian Government issued the Law Decree n. 42 [15] which establishes the conceptual architecture and the technical and governance rules of SPCoop, giving in this way legal value to interactions among SOA applications built over the standard. We refer to [5] for a comprehensive description of the SPCoop architecture.

In 2005 Link.it, a spin-off of the Computer Science Department in Pisa, started an open-source project, named *OpenSPCoop* [10], having the goal of implementing all the infrastructural components required by the SPCoop architecture. Among them, the *Domain Gateway* and the *Service Registry*. The software developed within the project quickly became the reference SPCoop implementation and is widely adopted in the main national eGov projects. More details about OpenSPCoop can be found, in Italian, in [7].

As well explained in [5], one of the hardest challenges in the SPCoop architecture is to define metrics for measuring the service level agreements (SLAs) and to design a system for monitoring the SLAs. In fact SPCoop can host potentially thousands of service providers, each having potentially different SLAs for each client, even for the same service. Nevertheless, as a consequence of the availability of a standard product managing all eGov business transactions, a uniform repository of all service transaction logs is available in organizations adopting SPCoop, as shown in Fig. 1a. The case of geographical federations of Public Organizations, e.g. Regional Entities, is particularly interesting. In this case, SPCoop requires to have a centralized regional gateway, logging both intra-regional and inter-regional application communications, as shown in Fig. 1b.

This scenario makes it possible and extremely interesting enabling business process analysis on a Domain Gateway as a mean to monitor SLAs. In this paper we report about the current activities aimed at extending the OpenSPCoop platform with monitoring functionalities based on the exploitation of system's logs for the performance and conformance analysis of business process models.

After introducing the main concepts concerning Business Process Management in Section 2, in Section 3 we present the Monitoring Framework designed to extend OpenSPCoop. In particular we describe how the ProM framework [9] has been integrated in the platform in order to exploit some of its analysis algorithms. In Section 4 we discuss the analysis algorithms currently available in the platform, and in Section 5 we present the ongoing activity aimed at providing automated support to the translation of BPMN model into Petri nets, and to read back the analysis results on the original BPMN model. We assume the readers have some familiarity with Petri nets' basics and notation.

2 Business Process Management

Business Process Management (BPM) is a young discipline related to the understanding, design, organization, enactment and improvement of the tasks to be performed to carry out some specific goal. BPM calls for a paradigm shift, from the data-centered to the process-centered view. The idea is to develop suitable artifacts, called *process representations*, that can be used to coordinate a generic software system that enacts the business process. As the process representation must be agreed upon by different stakeholders, ranging from the business domain expert and knowledge workers to the system architect and developers, graphical (workflow-like) languages are the best suited candidates. Over the years, several notations have been proposed to the purpose, often pushed by large industrial consortia, supported by various platforms and integrated in mainstream development environments. Some successful examples are Event-driven Process Chains (EPC) [2], the Business Process Execution Language (BPEL) [13] and the Business Process Modeling Notation (BPMN) [14], which has recently become a widely adopted standard. There are three main categories of flow objects in BPMN: 1) events denote something that happens during the course of a business process; 2) activities denote units of work to be accomplished during the course of a business process; 3) gateways denote the splitting and joining of workflow paths. Events are represented as circles, activities as rounded boxes and gateways as diamond shapes. Different kinds of decorations are introduced to clarify the nature of the flow object. For example, there can be start, intermediate and end events and different symbols are used for them.

Albeit the syntax of process notations is always defined very precisely, e.g. as XML schemes, their semantics is often described only verbally. Inevitably, any formal analysis for business processes must go through a rigorous semantics and in the recent literature several models have been used to address the issue (e.g. π-calculus [11], ASM [6], Petri nets [16] and in particular workflow nets [1]).

In this paper, we exploit Petri nets as underlying formal support to analyze process representations. We argue that Petri nets are a convenient solution, because several tools are available for their analysis, and their graphical notation can serve to report the outcome of the analysis in a form that is relatively close to the original artifact. Roughly, BPMN events are encoded as places, tasks as transitions and gateways as net fragments involving both places and transitions.

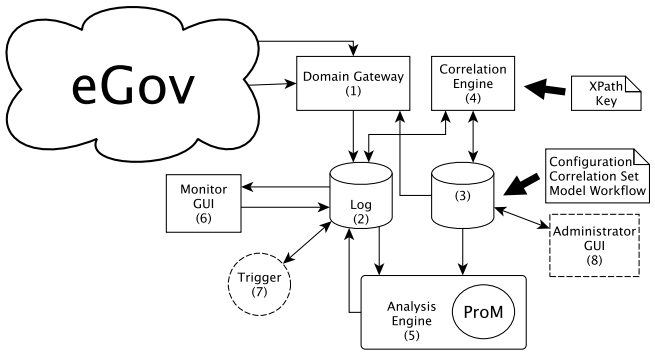


Fig. 2: The OpenSPCoop Business Process Monitoring Framework

Hereafter, we will focus the formalization on the so called *evaluation phase*, i.e. the business activity monitoring phase, that can provide relevant feedback about the effectiveness of the business process after deployment. The runtime data to be evaluated are provided by the middleware in terms of system events, that are collected in log files. Typical questions to be addressed are the *conformance checking* and the *performance evaluation*. The conformance checking allows one to spot the discrepancies between the planned process and its realization. The main technique used for conformance checking is the so-called *log replay* (see Section 4). The performance evaluation, instead, computes quantitative values allowing us to measure the deployed process. For example, timestamps can be used to calculate important parameters (latency, synchronization time) for dimensioning e.g. message buffers.

From a methodological perspective, to enable these analysis it must be the case that: 1) a formal model of the process is available; 2) all relevant activities are logged, 3) activities in the log must be correlated if they belong to the same instance of the process, or kept separated if they belong to different instances (or to different processes), 4) activities in the log are temporally ordered, e.g. using timestamps. It is quite common that for each activity, the beginning and the end are registered separately in the log, with different timestamps.

3 The Process Monitoring platform

In order to equip the Italian eGovernment Enterprise Architecture with a business process monitoring system, we extended the architecture of OpenSPCoop as shown in Fig. 2. The Domain Gateway (1) is the existing software infrastructure intercepting all communications from and to a Public Administration. In perspective, the idea is to instantiate the framework to a Regional Gateway, in order to exploit the centralized logging at regional level: preliminary experimentation are carried on within a project funded by Regione Toscana and are likely to be extended to other Italian regions.

As mentioned in Section 2, in order to enable process analysis techniques, audit logs must be grouped into *traces* in the log repository (2), each of them

representing a unique process instance. To this aim, the Domain Gateway has been enriched with XPath support: a database (3) is used to configure the XPath expressions that have to be evaluated by the Domain Gateway on the intercepted messages in order to extract the information required to identify the proper business process instance. The Correlation Engine (4) is responsible to structure the Domain Gateway logs and to group isolated events into process instances. The configuration database stores the correlation sets required to link the XPath expressions extracted by the domain gateway. Decoupling the Domain Gateway and the Correlation Engine allows us to reconstruct the process instances with minimal effect on the message throughput.

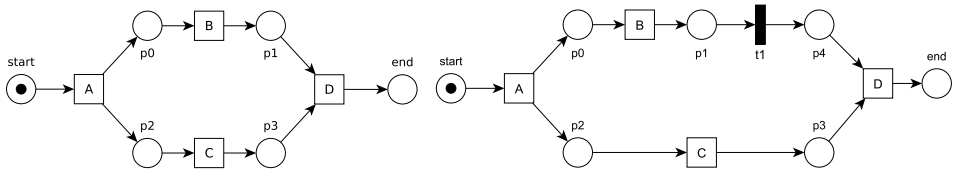
The Analysis Engine (5) is responsible for loading the traces from the log database, applying the analysis algorithms to them, and recording their results. Driven by the application domain, we started focusing on two kinds of analysis, namely *conformance* and *performance analysis* of log traces representing process instances with respect to the process model formalized as a Petri net.

As described below, the Analysis Engine exploits algorithms that are executed in ProM [9]. This is an extensible, integrated framework that supports a wide variety of process mining and analysis techniques in the form of plug-ins. We adopted Version 6 of ProM because its well-structured and modular architecture, featuring a careful separation between analysis algorithms and graphical user interface, simplified the task of integrating it into the Analysis Engine. On the negative side, not all plug-ins of the previous major version, ProM 5.2, have been ported (yet) to ProM 6. For example, a ProM 6 plug-in is already available [12] to handle the conformance analysis, but there was no plug-in to evaluate the performance of Petri Nets. Therefore we implemented a new ProM 6 plug-in for performance analysis taking inspiration from the techniques adopted in ProM 5.2 but, as explained in the next section, with a different handling of the "invisible transitions".

The log database can be used by external components to retrieve execution traces and the corresponding analysis results, enabling the framework to further extensions including user interface components (6) that graphically represent the results (e.g. a standalone GUI, or a web-based monitoring console) and supervisor components (7) that trigger computations whenever some constraints are violated (e.g. alert the system administrator if a process does not respect a SLA or performs unexpected execution). The whole system is equipped with an administration GUI (8) that manages the configuration database including XPath expressions and correlation sets, as well as the formal business process representation in the form of Petri nets or more abstract formalisms like BPMN.

4 Formal Analysis based on Petri nets

The current version of the Analysis Engine of the OpenSPCoop monitoring platform supports conformance and performance analysis of the logs against a model of the business process represented as a Petri net.



(a) Only visible transitions (b) One invisible transition, $t1$

Fig. 3: Two Petri nets with concurrent branches

The basic building blocks of logs are events. An event e can be seen abstractly as a pair $e = (a, t)$ representing an action a recorded by the Domain Gateway and the corresponding timestamp t. In the following we denote by $\alpha(e)$ and $\theta(e)$ the action and timestamp of e, respectively. Events that belong to the same process instance are grouped into *traces*. Formally, a trace T is a finite sequence of events $T[1], \ldots, T[n]$ such that $\theta(T[i]) \leq \theta(T[i+1])$ for all $i \in [1, n)$. A *log* L is a set of traces, recording the activity performed by a system during a finite number of process executions. We assume here that all traces are instances of the same process and that for each action there exists a corresponding transition in the net, that for simplicity will be denoted with the same name of the action.

The key algorithm exploited to analyze the Petri net model with respect to the log is the *log replay* algorithm [17]. For each trace in the log, the log replay starts by placing one token in the start place of the net. For each event in the trace the corresponding transition is fired in a *non-blocking way*, and the marking of the net is updated. "Non-blocking replay" means that if the log replay requires to fire a transition that is not enabled, the missing tokens are artificially created.

In general, there could exist transitions of the Petri net that are not associated with any logged action: such transitions are called *invisible*. This can happen in several cases: for example, the transition models an internal choice that is not visible in the system, or it is used to implement a connector of a more abstract modeling language (as we shall see later for BPMN). Usually, invisible transitions are considered to be lazy [4], i.e., when firing a visible transition t corresponding to an event of the trace, only then the invisible transitions enabling t are fired. For example, considering the net in Fig. 3b, if the log does not contain actions corresponding to transition $t1$ then it is handled as invisible, and its firing is delayed until an action corresponding to D is found in the trace. Further details on the log replay algorithm can be found in [17].

The output of the log replay of a trace T can be represented as an ordered list R of pairs (tr, i), representing that the transition tr has been fired to mimic the event $T[i]$. Note that the presence of invisible transitions can result in several transitions fired to mimic a single event. In presence of recursion several occurrences of the same transition are also possible.

The result of the log replay can be used to evaluate conformance and performance of the Petri net model. Conformance problems can be discovered by analyzing the tokens that have been artificially created (the *missing tokens*) and the tokens that were not consumed (the *remaining tokens*). For example, let us

consider the net in Fig. 3a and the traces $T = [(A, t_1), (B, t_2), (C, t_3), (D, t_4)]$, $T' = [(A, t_1)]$ and $T'' = [(B, t_1)]$. Trace T is compliant with the net: the log replay does not identify any missing token, terminates with a marking containing one token in the end place, and returns the sequence $[(A, 1), (B, 2), (C, 3), (D, 4)]$. The log replay for T' does not produce any missing token but the remaining tokens are $\{p0 \to 1, p2 \to 1\}$: the trace is a partial execution and B and C should be executed to continue the process. The log replay for T'' terminates with a missing token $\{p0 \to 1\}$ and remaining tokens $\{start \to 1, p1 \to 1\}$: the action A was skipped by the execution.

Since the logs contain timestamps, the log replay can be used to compute performance measures of the system [3]. The idea is to calculate the time interval between production and consumption of tokens in each place. This technique can be applied only to traces that do not require the production of missing tokens, because such tokens cannot have time information. During the log replay the following metrics can be computed for catch trace and each place:

- *sojourn time* ($\bowtie$): the time interval between arrival and departure of a token;
- *synchronization time* ($\ltimes$): the time interval between arrival of a token in the place and enabling of a transition in the post-set of the place;
- *waiting time* ($\rtimes$): the time interval between enabling of a transition in the post-set of the place and token departure (thus $\ltimes + \rtimes = \bowtie$).

To clarify the metrics evaluated by this technique we exploit the Petri net depicted in Fig. 3a and the traces in Table 1a. The net starts with the transition A, concurrently fires B and C and terminates after the transition D. When applying the log replay to trace T_1 we cannot associate performance metrics to the starting place, since the trace does not carry information about the start time of the process. The first event in the trace records the firing of transition A at time $1s$, thus tokens arrive at $p0$ and $p2$ at time $1s$. Since both $p0$ and $p2$ have an enabled transition at time $1s$, their synchronization times are $\ltimes(p0) = \ltimes(p2) = 0s$. The next event of the trace is $(B, 2s)$: the token in $p0$ is consumed and a token is produced in $p1$. The sojourn time of $p0$ is then evaluated as $\bowtie(p0) = 2s - 1s = 1s$. The log replay continues with the third event recorded: the firing of C at time $4s$. The token in $p2$ is consumed and a new token is produced in $p3$. The sojourn time of $p2$ is evaluated as $\bowtie(p2) = 4s - 1s = 3s$. Notice that now $(4s)$ transition D is enabled and thus the synchronization times of its preset are $\ltimes(p1) = 4s - 2s = 2s$ and $\ltimes(p3) = 4s - 4s = 0s$. While processing the last recorded event $(D, 8)$ the log replay consumes the tokens in $p1$ and $p3$, and their sojourn times are $\bowtie(p1) = 8s - 2s = 6s$ and $\bowtie(p3) = 8s - 4s = 4s$.

Table 1b reports performance details for three traces and their averages. In this example $\ltimes(p1)$ is usually greater than $\ltimes(p3)$, namely, transition B is fired almost always before C. Moreover, the transition that waits more time to be fired after its activation is D.

We extend the previous example to stress the role of invisible transitions, referring to the net of Fig. 3b. Handling invisible transitions as lazy, the log replay for trace T_1 yields the sequence $[(A, 1), (B, 2), (C, 3), (t1, 4), (D, 4)]$. Notice that, even if transition $t1$ is enabled after the event $T_1[2]$, it is delayed until its

T_1	T_2	T_3
(A, 1s)	(A, 3s)	(A, 9s)
(B, 2s)	(B, 5s)	(C, 10s)
(C, 4s)	(C, 8s)	(B, 11s)
(D, 8s)	(D, 11s)	(D, 15s)

(a) System Log

	T_1		T_2		T_3		Average	
	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$
p0	1	0	2	0	2	0	1.67	0
p1	6	4	6	3	3	0	5	2.33
p2	3	0	5	0	1	0	2	0
p3	4	0	3	0	5	1	4	0.33

(b) Results for net of Fig. 3a

	T_1		T_2		T_3		Average	
	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$
p0	1	0	2	0	2	0	1.67	0
p1	6	0	6	0	4	0	5.33	0
p2	3	0	5	0	1	0	2	0
p3	4	4	3	3	5	5	4	4
p4	0	0	0	0	0	0	0	0

(c) Net of Fig. 3b (lazy)

	T_1		T_2		T_3		Average	
	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$
p0	1	0	2	0	2	0	1.67	0
p1	2	0	3	0	0	0	2.67	0
p2	3	0	5	0	1	0	2	0
p3	4	0	3	0	5	1	0	0.33
p4	4	0	3	0	4	0	0	0

(d) Net of Fig. 3b (ProM 5.2)

	T_1		T_2		T_3		Average	
	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$	$\bowtie$	$\ltimes$
p0	1	0	2	0	2	0	1.67	0
p1	0	0	0	0	0	0	0	0
p2	3	0	5	0	1	0	2	0
p3	4	0	3	0	5	1	0	0.33
p4	6	4	6	3	3	0	5	2.33

(e) Net of Fig. 3b (eager)

Table 1: Traces and performance results with different evaluation methods

activation is required by the visible transition D. Evaluating performance of this Petri net using this approach yields the results in Table 1c. Notice that both $p1$ and $p4$ have no synchronization time and that the synchronization time of $p3$ is greater than zero even when C is fired after B: in our opinion these measures do not interpret faithfully the process instance.

A slightly different strategy (used in ProM 5.2) generates the same sequence of transitions, but instead of associating the invisible transition with the triggering event, it uses the last replayed event. Thus for trace T_1 the sequence $[(A,1),(B,2),(C,3),(t1,3),(D,4)]$ is returned, and the corresponding performance measures are reported in Table 1d. This approach allows us to correctly evaluate synchronization times whenever C precedes B, but fails otherwise.

Implementing the new performance plug-in for ProM 6 we exploited a different strategy, motivated not only by the measures just analysed, but also by the need of projecting the performance measures from the net back to the BPMN model, as discussed later. Intuitively, before associating with places the performance measures, we rearrange the sequence of transitions returned by the log replay in order to fire invisible transitions as soon as possible. We call the resulting sequences *eager* and define them formally as follows.

Definition 1 (last visible transition). *Let $R = [(tr_1, i_1), (tr_2, i_2), ..., (tr_n, i_n)]$ be a sequence of transitions and corresponding event indexes, and let $j \leq n$. We define the* last visible transition *$R \downarrow j$ of R before the jth transition as*

$$R \downarrow j = \begin{cases} -1 & \text{if } j \leq 0 \\ j-1 & \text{if } j \in [2,n] \text{ and } R[j-1] \text{ is visible} \\ R \downarrow (j-1) & \text{otherwise} \end{cases}$$

```
for j from 2 to size(R) do
  tr, i ← R[j]
  if tr is invisible then
    k, done ← j − 1, false
    while k > 0 ∧ ¬done do
      tr_prev, i_prev ← R[k]
      if •tr ∩ tr•_prev ≠ ∅ then
        done ← true
      else
        R[k + 1] ← (tr_prev, i_prev)
        R[k] ← (tr, max(R ↓ k, 1))
        k ← k − 1
      end if
    end while
  end if
end for
```

Fig. 4: Conversion of a generic sequence R to a corresponding eager sequence

Definition 2 (eager sequence). *Let* $R = [(tr_1, i_1), (tr_2, i_2), ..., (tr_n, i_n)]$ *be a sequence as above. Then* R *is* eager *if for all* $j \in [1, n]$ *one of the following holds*[3]

- tr_j *is visible*
- tr_j *is invisible and either* $R \downarrow j = -1$ *or* ${}^\bullet tr_j \cap tr^\bullet_{R\downarrow j} \neq \emptyset$

Intuitively, a sequence is eager if for each invisible transition t the last preceding visible transition enables t, if it exists. Converting a generic sequence R obtained from a log replay to a corresponding eager sequence can be done easily with the algorithm in Fig. 4.

For example, by applying the algorithm to the trace T_1 we obtain the eager sequence $[(A, 1), (B, 2), (t1, 2), (C, 3), (D, 4)]$, and the corresponding performance measures are reported in Table 1e. Notice that now only places $p3$ and $p4$ can have a synchronization time greater than zero: in our opinion these measures describe more faithfully the evolution of the net. Section 5 further motivate our approach, by describing advantages of interpreting invisible transitions as "eager" when the performance measures of the Petri Net must be projected back to business process models at the higher level of abstraction.

4.1 Using invisible transitions to improve performance analysis

As described above, performance analysis associates suitable measures with the *places* of a net. But often one is rather interested in measures related with the *visible transitions*, which represent system activities, like their *waiting time* or *duration*. In the nets of Fig. 3 the waiting time of each transitions can be identified with the largest among the waiting times of the places in their pre-set, but this is not always true. The Petri net in Fig. 5a describes a process where action A precedes the execution of either B or C. In this case the waiting time

[3] As usual, by ${}^\bullet t$ we denote the *preset* of transition t, and by $t^\bullet$ its *postset*.

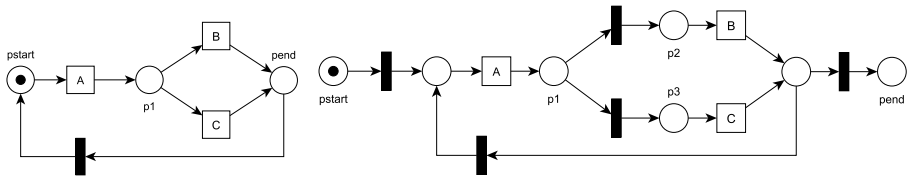


Fig. 5: Improve performance analysis by introducing invisible transitions

of $p1$ is determined by the time interval between the timestamp of action A and the timestamp of the subsequent action (either B or C). However, from this measure we cannot determine if B and C have different waiting times, i.e. if the average waiting times of $p1$ is different when either B or C occurs. This issue is even more serious due to presence of recursion. This problem can arise only when a place in the pre-set of a visible transition is also in the pre-set of another transition. We call *isolated* a visible transition such that this does not happen.

Definition 3 (isolated transition). *A visible transition t of a Petri net is called* isolated *if* $\forall p \,.\, p \in {}^\bullet t \Rightarrow p^\bullet = \{t\}$.

By introducing suitable invisible transitions we can transform a net into an equivalent one (i.e., with the same traces of visible transitions) where all visible transitions are isolated. For example, the net of Fig. 5b is obtained through this transformation from that of Fig. 5a: notice that now we can interpret the waiting times of places $p2$ and $p3$ as the waiting times of B and C, respectively.

Another measure that is often useful when analysing a system is the duration of certain activities. Since the events recorded in a log are considered as instantly executed (they have execution timestamps recorded, but not their duration) it is not possible to infer how long the corresponding action ran. Thus in order to evaluate the execution time of business process activities each of them has to be modeled by at least two actions, representing its start and its termination, that have to be recorded by the system in the log with the corresponding timestamps. Clearly, depending on the application, it might be convenient to represent an activity with a more complex Petri, for example exploiting three transitions to model the activity start, successful completion and failure respectively.

5 Supporting BPMN modeling

Several standards for business process modeling (e.g. BPMN and JBPM) provide much richer graphical primitives than those available for Petri nets, and allow to model a system at a higher abstraction level, easier to understand for the stakeholders. In our setting, in order to exploit the algorithms presented in Section 4 to analyze processes expressed in richer process description languages, a straightforward strategy is to define a model transformation that starting from the more abstract specification yields a Petri net. The resulting net can be analyzed with the presented algorithms to verify conformance of executions and to

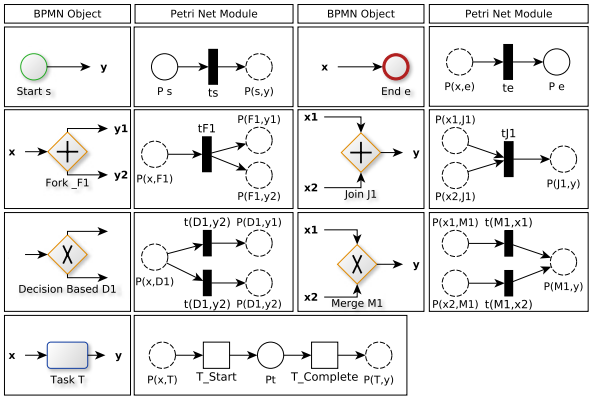


Fig. 6: From BPMN to Petri net

evaluate their performance. The critical issue is then to project back the analysis result to the starting abstract model, as discussed below.

As a business process modeling formalism our platform will support BPMN. We adopt essentially the methodology presented in [8] to transform (a subset of) BPMN models into Petri nets. Even if the subset of BPMN considered here is quite minimal, on one hand we think that it is sufficient to illustrate the advantages of our approach, on the other hand we have concrete evidence that it can be extended seamlessly to other BPMN constructs like events and messages.

The transformation rules for our BPMN subset are presented in Fig. 6 with a slight change in the rule transforming a BPMN task, for which we generate a net with two transitions, as discussed above. It is worth stressing that these transformation rules guarantee that all visible transitions in the resulting net are isolated, which allows us to exploit the performance algorithm to infer the execution time of each BPMN task. Let us discuss now, on the basis of these translation rules, how the analysis results obtained on a net can be projected back to the BPMN model.

Performance analysis. Recall that in the algorithm proposed in Section 4 invisible transition are treated as eager: if such a transition is fired by the log replay, it is considered to be fired as soon as it is enabled. For this reason, the waiting times for places having no visible transition in their post-sets are always zero. Referring to the rules in Fig. 6, the only places that can have non-zero waiting times are $P(x, T)$ and Pt, i.e. the places that model BPMN activities. Similarly, the synchronization time of a place can be greater than zero only if at least one transition in its post-set depends on another place. The places in Fig. 6 that can have non-zero synchronization times are just $P(x1, J1)$ and $P(x2, J2)$, i.e. the places involved in modeling join gateways. Based on these considerations we can project the performance measures of a net obtained from the translation to the original BPMN model as follows:

– For each task T the execution time is $\rtimes(Pt)$ and the activation time is $\rtimes(P(x, T))$

– For each concurrent branch ($i \in [1, 2]$) enclosed by a fork ($F1$) and a join ($J1$) gateways: (i) the synchronization time is $\ltimes(P(xi, J1))$, namely the time spent to wait the termination of the other concurrent activities (ii) the execution time (referred to as $\rtimes(F1i)$) is the sum of $\bowtie$ of all places reachable by traversing the graph starting from $P(Fi, yi)$ without visiting $P(xi, Ji)$. Notice that $\ltimes(P(xi, J1)) + \rtimes(F1i)$ is constant for each branch of a fork/join pair. We call this time the *fork* execution time ($\bowtie(F1, J1)$).

We stress that the considerations discussed above do not hold for the original ProM 5.2 performance plug-in. In particular, places involved by the join gateways ($P(x1, J1)$ and $P(x2, J1)$) can have synchronization time greater that zero only if their presets contain at least one visible transition. Hence, the algorithm can infer synchronization times of join gateways only if the concurrent branches terminate with a BPMN task.

Conformance analysis. We exploit a similar reasoning to project back data of conformance results. Log replay artificially produces missing tokens only to fire visible transitions. Hence only places in the pre-set of at least one visible transition can have missing tokens. In Fig. 6 these places are $P(x, T)$ and Pt. Moreover, log replay fires invisible transitions only if their execution is required to activate a visible transition. For example, for any execution of the start transition ts the log replay fires a visible transition that consumes the token in the place $P(s, y)$. The same consideration holds for all invisible transitions that produce only one token. Hence, only places in the post-set of a visible transition or of an invisible transition spawning several tokens can have remaining tokens. In Fig. 6 these places are Pt, $P(T, y)$, $P(F1, y1)$ and $P(F1, y2)$, that are involved to model a single BPMN activity and the fork gateway. Thus, we project the conformance metrics of the Petri net to the starting BPMN model as follows:

– for each task T missing tokens in $P(x, T)$ are referred as "unsound executions", missing tokens in Pt are referred as "internal failures", remaining tokens in Pt are referred as "missing competition" and remaining tokens in $P(T, y)$ are referred as "interrupted executions"
– for each branch ($i \in [1, 2]$) of a fork $F1$, the remaining tokens in $P(F1, yi)$ are referred as "interrupted branches". Notice that for each execution of the transition $tF1$ a token can remain in either $P(F1, y1)$ or $P(F1, y2)$, but not in both places.

6 Citizen migration

In this section we present a real life Italian eGov business process. The BPMN model in Fig. 7 summarizes the main activities required to perform a change of residence of an Italian citizen (the annotations can be ignored for the moment). In order to change the residence, the request must be previously verified and the necessary sensible data collected by the eGov platform. Note that two

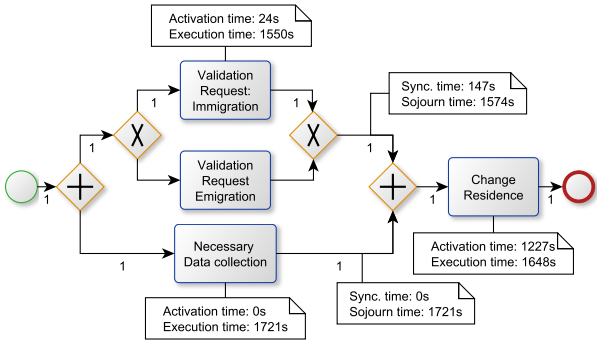


Fig. 7: BPMN model for citizen migration (annotated with performance analysis)

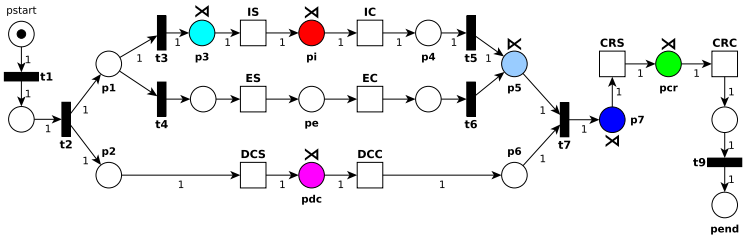


Fig. 8: Petri Net for citizen migration (annotated with performance analysis)

different validation tasks can be executed, depending on the citizen being emigrating or immigrating. The Petri Net in Fig. 8 has been obtained by applying the model transformation described in Section 5. Note that for each BPMN task (e.g. *ChangeResidence*) the net has two transitions (e.g. CRS and CRC) representing the start and completion of the activity. The trace in Fig. 9a records a possible execution of the Business Process.

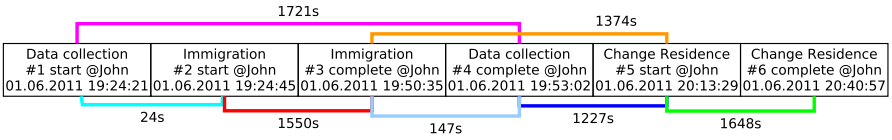
An example of performance analysis. In order to evaluate performance measure of the Petri Net, our plug-in applies the log replay to obtain the sequence

$R1 = [(t1, 1), (t2, 1), (DCS, 1), (t3, 1), (IS, 2), (IC, 3), (DCC, 4), (t5, 4), (t7, 4), (CRS, 5), (CRC, 6)]$

Our implementation considers invisible transitions fired immediately. It transform the transition sequence into the eager sequence:

$R2 = [(t1, 1), (t2, 1), (t3, 1), (DCS, 1), (IS, 2), (IC, 3), (t5, 3), (DCC, 4), (t7, 4), (CRS, 5), (CRC, 6)]$

Figure 8 includes the performance metrics evaluated by our plug-in. All white places have zero synchronization time and zero waiting time. Waiting times of places pi, pdc and pcr represent the *execution times* of the corresponding BPMN activities. Waiting time of place $p3$ is $24s$. In fact, execution of transitions $t1$, $t2$ and $t3$ are associated to the event index 1, while the transition IS is associated to the event index 2. The only place having non-zero synchronization time is $p5$ ($147s$). This time is equal to the interval among execution of the transitions IC and DCC. We stress that the synchronization time of place $p5$ results from the transformation of the sequence $R1$ into the eager sequence $R2$ (migrating transition $t5$ immediately after IC). We already discussed in Section 5 that the ProM 5.2 implementation cannot infer synchronization time for the place



(a) Sound execution

Immigration #1 start @John 01.06.2011 17:14:56	Immigration #2 complete @John 01.06.2011 17:31:19	Emigration #3 start @John 01.06.2011 18:24:33	Emigration #4 complete @John 01.06.2011 18:37:10	Data collection #5 start @John 01.06.2011 18:25:27	Change Residence #6 start @John 01.06.2011 18:35:21	Change Residence #7 complete @John 01.06.2011 18:42:41

(b) Unsound execution

Fig. 9: Execution Logs

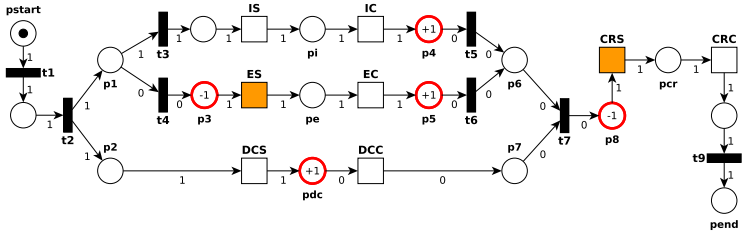


Fig. 10: Petri Net (annotated with conformance analysis)

$p5$, because its preset contains only invisible transitions. ProM 5.2 yields non-zero synchronization time for $p6$ (1227s). Figure 7 shows the projection of the performance analysis back to the BPMN model.

An example of conformance analysis. The trace log depicted in Fig. 9b provides an example of wrong execution. In fact, the trace records the execution of both the exclusive tasks (*Immigration* and *Emigration*) and does not carry events regarding the completion of the activity *DataCollection*.

The conformance measures yielded from log replay for this trace are depicted in Fig. 10. Place $p3$ misses one token: it is required to replay the transition ES that is not enabled because the alternative branch *Immigration* already started. Place *pdc* contains one remaining token: it depends on the missing termination of *Data Collection*. Since termination of this activity is not recorded, place $p7$ does not receive a token during the log replay and the transition $t7$ is never enabled. Hence, the synchronization fails resulting in remaining tokens in both $p4$ and $p5$. The failed synchronization is also the cause of the missing token in place $p8$, required to replay the transition CRS.

Figure 11 projects the conformance data back to the BPMN model. The figure reports: (i) the missing completions of *Data Collection*; (ii) the unsound execution of *Emigration*, because the other branch of the exclusive gateway already activated; (iii) the interrupted execution of both *Immigration* and *Emigration*, because the join gateway does not synchronize; (iv) the unsound execution of *Change Residence*, caused by the failed synchronization.

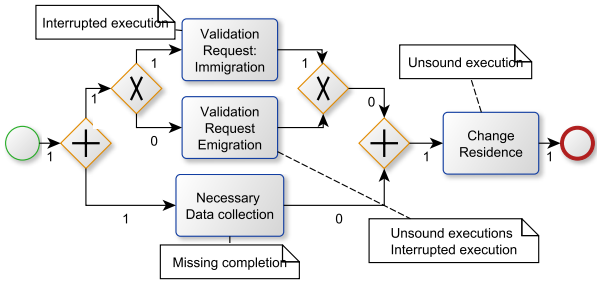


Fig. 11: BPMN (annotated with conformance analysis)

7 Integrating ProM in the Process Monitoring platform

ProM 6 exploits the concept of *context* to allow plug-ins to access to the surrounding environment. A context implements the management of resources, e.g. thread management, parameter resolution, progress monitoring and process interruption. In order to start the ProM framework, the *Boot.boot* static method has to be invoked on the chosen context implementation. The *Boot* factory is delegated to create the proper context and to invoke its entry point after the framework initialization. The integration strategy focuses on the development of a specific context (*ServiceContext*) that is Swing-free, but that allows the environment to interact with the ProM plug-ins, for example by triggering the execution of a specific analysis algorithm whenever a process instance terminates. Moreover the context should keep a ProM instance alive to satisfy several requests and to reuse allocated resources (like thread pools). The implementation of the ServiceContext allows us to allocate thread pools directly or to delegate their management to the surrounding environment (e.g. to the application server JBoss). Moreover, the context exposes the ProM algorithms via an API: the analysis results are returned as serializable objects.

In order to fulfill the lack of a performance analysis plug-in for ProM 6, we provided a new implementation exploiting the formal approach described in Section 4. Our implementation exploits some functionalities provided by existing plug-ins (e.g. ETConformance makes available the log-replay). This way we can benefit from further enhancements of the ProM framework. We developed three other plug-ins: (i) *BpmnToPetri* transforms a BPMN model into a Petri net; the plug-in currently support the subset of BPMN described in Section 5. (ii) *ConformanceToBPMN* and (iii) *PerformanceToBPMN* annotate the BPMN model with suitable artifacts to represent the conformance and performance measures.

8 Concluding Remarks

We have presented a business process monitoring platform for the Italian eGovernment Enterprise Architecture. The main contributions of this paper are (i) the integration of the ProM framework into a Service Oriented Architecture, allowing the stakeholders to take benefit transparently from several formal methods, (ii) an updated performance evaluation algorithm that manages invisible

transaction as eager, and (iii) a methodology to apply the analysis to high level process models. Our monitoring platform can be extended to support several new features. We are developing a web based monitoring interface to represent the analysis results graphically. We are also implementing the transformation tools required to handle BPMN models taking inspiration from the methodology presented in this paper. A further research effort is focused on integrating data mining techniques into the analysis engine. The interplay between process analysis and data mining should help discovering, for example, clusters of messages that cause bottlenecks. Finally, we plan to implement a supervisor component to trigger suitable actions whenever a SLA constraint is violated.

References

1. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
2. van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. Information and Software Technology pp. 639–650 (1999)
3. van der Aalst, W.M.P., van Dongen, B.F.: Discovering workflow performance models from timed logs. In: EDCIS. LNCS, vol. 2480, pp. 45–63. Springer (2002)
4. van der Aalst, W.M.P., de Medeiros, A.K.A., Weijters, A.J.M.M.: Genetic process mining. In: ICATPN. LNCS, vol. 3536, pp. 48–69. Springer (2005)
5. Baldoni, R., Fuligni, S., Mecella, M., Tortorelli, F.: The italian e-government enterprise architecture: A comprehensive introduction with focus on the SLA issue. In: ISAS. LNCS, vol. 5017, pp. 1–12. Springer (2008)
6. Börger, E., Thalheim, B.: Modeling workflows, interaction patterns, web services and business processes: The ASM-based approach. In: ABZ. LNCS, vol. 5238, pp. 24–38. Springer (2008)
7. Corradini, A., Flagella, T.: OpenSPCoop: un Progetto Open Source per la Cooperazione Applicativa nella Pubblica Amministrazione. In: AICA'07 (2007)
8. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and analysis of BPMN process models (2007), `http://eprints.qut.edu.au/7115/`
9. Eindhoven Univ. of Technology: ProM, `http://www.processmining.org/`
10. Link.it: OpenSPCoop, `http://www.openspcoop.org`
11. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, I. Inf. Comput. 100(1), 1–40 (1992)
12. Muñoz-Gama, J., Carmona, J.: A fresh look at precision in process conformance. In: BPM. LNCS, vol. 6336, pp. 211–226. Springer (2010)
13. Oasis: Web Services Business Process Execution Language Version 2.0 (2007), `http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html`
14. OMG: Business Process Model and Notation, `http://www.omg.org/spec/BPMN/`
15. Parlamento Italiano: Istituzione del sistema pubblico di connettività e della rete internazionale della pubblica amministrazione, D.L. n. 42 del 28/02/2005, G.U. n. 73 del 30/03/2005, `http://www.parlamento.it/parlam/leggi/deleghe/05042dl.htm`
16. Petri, C.A.: Fundamentals of a theory of asynchronous information flow. In: IFIP Congress. pp. 386–390 (1962)
17. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)