Misurare Processi di Business

Roberto Bruni[1], Andrea Corradini[1], Gianluigi Ferrari[1], Tito Flagella[2],
Roberto Guanciale[1], Giorgio Oronzo Spagnolo[1]
[1] *Dipartimento di Informatica, Università di Pisa*
Largo B. Pontecorvo 3, 56127 Pisa
[bruni,andrea,giangi,guancio,spagnolo]@di.unipi.it
[2] *Link.it*
Via San Martino 51, 56125 Pisa
tito@link.it

La notazione BPMN si sta affermando come standard per la descrizione di processi di business ed è supportata da molti strumenti di sviluppo. La distanza tra la modellazione astratta dei processi in BPMN e la loro realizzazione concreta necessita di strumenti per l'analisi di conformità (tra implementazione e modello) e delle prestazioni (per evidenziare le criticità e guidare la reingegnerizzazione dei processi). In questo articolo descriviamo alcune tecniche per il monitoraggio e la valutazione di processi di business dopo il loro deployment. L'analisi viene effettuata sui log di esecuzione e si basa sul modello formale delle reti di Petri. L'uso delle reti di Petri come modello intermedio tra log e processo BPMN permette di proiettare i risultati sul modello astratto, rendendoli di immediata comprensione per gli analisti. In particolare, presentiamo delle estensioni originali di algoritmi noti che offrono una descrizione più fedele dei fenomeni osservati durante l'esecuzione. Gli algoritmi presentati sono disponibili come plug-in di ProM.

1. Introduzione

Il Business Process Management (BPM) è una disciplina di recente affermazione che stabilisce un cambio di paradigma, da una gestione centrata sui dati a una centrata sui processi, ovvero da "cosa" a "come". Infatti, BPM affronta la modellazione, l'organizzazione, l'applicazione e l'ottimizzazione delle attività necessarie per raggiungere un determinato obiettivo (es. offrire un determinato servizio, oppure produrre un certo manufatto). In BPM, i processi vengono rappresentati attraverso formalismi grafici, permettendo di comunicare in modo non ambiguo le regole di business, e quindi discuterle o modificarle, tra gli svariati ruoli coinvolti che vanno dagli esperti del dominio di business o del settore, agli architetti software e sviluppatori.

Negli ultimi anni sono state proposte diverse notazioni grafiche, spesso sponsorizzate da grandi consorzi industriali, supportate da varie piattaforme e integrate nei principali ambienti di sviluppo. Tra le notazioni più diffuse citiamo

Event-driven Process Chains (EPC) [Scheer, 1993], Business Process Execution Language (BPEL) [OASIS, 2007] e Business Process Modeling Notation (BPMN) [OMG, 2011], che è diventato recentemente lo standard più diffuso. Il successo di queste notazioni è spesso legato alla loro facilità di utilizzo, alla comprensione intuitiva delle forme grafiche offerte, all'assenza di vincoli strutturali e alla maggiore flessibilità. Come controparte, per favorire questi aspetti si tollera la possibilità di tracciare diagrammi inconsistenti o con ambiguità semantiche.

Dato che i modelli di processi vengono poi tradotti in applicazioni software, la distanza tra il modello astratto e la sua realizzazione è tale da richiedere strumenti di monitoraggio e analisi per monitorare la conformità dell'implementazione al modello concettuale e le prestazioni raggiunte. A questo scopo, diversi modelli formali sono stati sviluppati ed adottati per fornire una semantica rigorosa agli standard industriali (come il π-calculus [Milner et al, 1992], ASM [Börger e Thalheim, 2008], le reti di Petri [Petri, 1962] e in particolare le Workflow Net [van der Aalst, 1998]). Infatti, la descrizione non ambigua del comportamento di un processo è un prerequisito fondamentale per ogni algoritmo o tecnica di analisi.

In questo contributo ci focalizziamo su una specifica fase del BPM, che comprende il *monitoraggio* e la *valutazione*. L'obiettivo di questa fase consiste nel verificare la corretta esecuzione dei processi e misurarne le prestazioni dopo il loro deployment. I risultati di questa fase sono prerequisiti di eventuali ottimizzazioni o re-ingegnerizzazioni dei processi, permettendo di individuare discrepanze tra il processo pianificato e la sua realizzazione e di calcolare parametri (come la latenza e i tempi di sincronizzazione) necessari per dimensionare le risorse di business (per esempio i gruppi di lavoro allocati per le attività). I dati necessari a queste analisi sono di solito forniti dall'infrastruttura di supporto attraverso i log di eventi generati a tempo di esecuzione. Dal punto di vista metodologico, queste analisi richiedono che 1) sia disponibile un modello formale del processo; 2) tutte le attività (di business) importanti siano registrate nei log; 3) gli eventi dei log siano correlati se riguardano la stessa istanza di processo o tenuti separati se riguardano istanze differenti; 4) gli eventi dei log siano ordinati temporalmente.

In questo contributo adottiamo BPMN come linguaggio per descrivere i processi e le reti Petri come modello formale. L'uso delle reti di Petri è giustificato dalla ampia disponibilità di strumenti e tecniche di analisi esistenti. In particolare abbiamo adottato ed esteso la tecnica chiamata *log replay* e implementato le nostre tecniche come plug-in di ProM [Eindhoven Univ. of Technology, 2011]. Nella prossima sezione riassumiamo i principali costrutti di BPMN attraverso l'introduzione di un esempio; inoltre, presentiamo una trasformazione di modelli BPMN in reti di Petri, necessaria ad abilitare le analisi formali. Gli algoritmi di analisi per reti di Petri e la proiezione dei risultati sui modelli BPMN di partenza sono presentati nella sezione 3. Il contributo innovativo è riassunto nelle conclusioni.

Le attività di ricerca qui descritte sono state realizzate nell'ambito del progetto RUPOS (*Ricerca sull'Usabilità delle Piattaforme Orientate ai Servizi*, [Link.it, 2011]) con il parziale supporto della Regione Toscana. La rilevanza di queste tematiche è testimoniata dalla recente istituzione di una Task Force dell'IEEE dedicata a process analysis e process mining [IEEE, 2011].

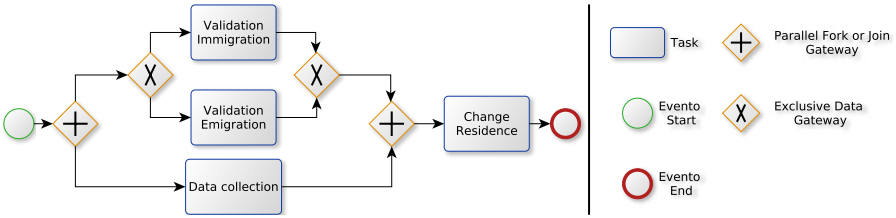

Fig. 1 - modello BPMN per il cambio di residenza

2. Supporto alla modellazione tramite BPMN

BPMN è un linguaggio per modellare processi di business attraverso un'astrazione grafica. In questo articolo non introduciamo formalmente tutti i costrutti di BPMN, ma ne presentiamo i principali attraverso un esempio. Il processo BPMN in Fig. 1 sintetizza le regole necessarie per ottenere un cambio di residenza. Il processo coinvolge quattro distinte **attività** (o **task**): *Validation: Immigration*, *Validation: Emigration*, *Data collection* e *Change Residence*. I cerchi con bordo sottile rappresentano l'inizio del processo, quelli con bordo di spessore maggiore la sua terminazione. I rombi etichettati con + sono chiamati **fork/join gateway**. Nell'esempio, il primo rombo descrive un fork dell'esecuzione; *Data collection* è eseguito concorrentemente alle attività di validazione. Il secondo rombo + rappresenta invece un join: le attività concorrenti devono essere completate prima di attivare *Change Residence*. I rombi etichettati con X sono chiamati **split/merge gateway** e rappresentano una scelta tra due esecuzioni alternative: la procedura di validazione è differente se il cittadino richiede di immigrare o di emigrare. Nel seguito useremo questo esempio per presentare le tecniche sviluppate.

Molti algoritmi di analisi (come quelli discussi nelle prossime sezioni) non sono applicabili direttamente a modelli BPMN, ma sono definiti su reti di Petri. Pertanto definiamo una trasformazione da modelli BPMN a reti di Petri, e successivamente affrontiamo il problema di riportare i risultati di queste analisi sul modello BPMN di partenza. Per trasformare (un sottoinsieme di) modelli BPMN in reti di Petri adottiamo la metodologia presentata in [Dijkman et al, 2008]: le regole di trasformazione corrispondenti sono presentate in Fig. 2. Abbiamo esteso la trasformazione per trattare esplicitamente l'attivazione e la terminazione di attività BPMN. La regola di trasformazione introdotta rappresenta questi eventi attraverso due distinte transizioni. La rete di Petri in Fig. 4 (vedere anche Fig. 7) è ottenuta applicando queste trasformazioni al diagramma BPMN di Fig. 1.

Nel rappresentare la trasformazione abbiamo usato due simboli differenti per due classi di transizioni: (i) le transizioni relative ad attività BPMN sono rappresentate da quadrati non riempiti e sono chiamate **transizioni visibili**; (ii) le transizioni generate per modellare gli altri elementi di BPMN sono rappresentate da rettangoli riempiti e sono chiamate **transizioni invisibili**. Nel proseguo dell'articolo assumiamo che i log dei sistemi contengano solamente gli eventi correlati all'attivazione ed al completamento delle attività.

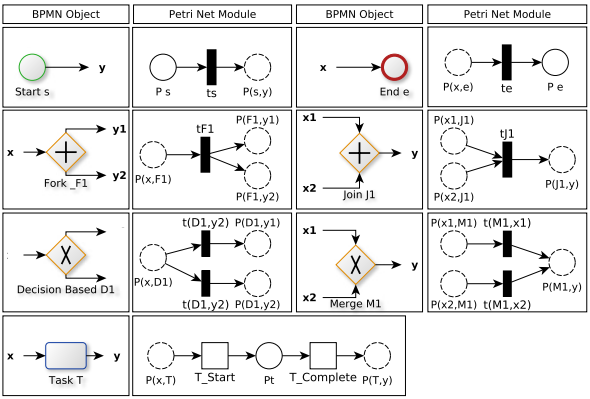

Fig. 2 - Mapping da BPMN a reti di Petri

3. Analisi basata su reti di Petri e log di esecuzioni

Le tecniche presentate in questa sezione permettono di analizzare proprietà di un processo (espresso come rete di Petri) attraverso l'elaborazione di un log di esecuzione. Un log è costituito da una sequenza di eventi, cioè di coppie del tipo $e = (a, t)$ dove a è un'azione registrata dal sistema e t è il suo timestamp (che indicheremo a volte con $\theta(e)$). Gli eventi che appartengono alla stessa istanza di processo sono raggruppati in tracce. Formalmente, una traccia T è una sequenza di eventi $T[1], \ldots, T[n]$ tale che $\theta(T[i]) \leq \theta(T[i+1])$ per tutti gli $i \in [1, n)$. Un log L è un insieme di tracce, cioè la registrazione delle attività svolte da un sistema durante un numero finito di esecuzioni di un processo. Assumiamo che tutte le tracce siano istanze dello stesso processo e che per ogni azione a esista una transizione visibile con lo stesso nome nella rete.

L'algoritmo chiave della nostra metodologia si chiama **log-replay** [Rozinat e van der Aalst, 2008]. Per ogni traccia T, l'algoritmo parte con un token nella piazza iniziale delle rete e per ogni evento (a, t) in T esegue la corrispondente transizione a *in modo non bloccante*, aggiornando il marking della rete di conseguenza. Il termine *"non bloccante"* significa che se la transizione a non è abilitata, i token mancanti vengono creati artificialmente e chiamati **missing token**.

Come descritto nella sezione 2, assumiamo che i log contengano solamente gli eventi correlati all'inizio e al completamento delle attività del processo BPMN. Questo comporta che le transizioni invisibili non siano associate a nessun evento registrato nel log. Di solito, le transizioni invisibili sono considerate *lazy* [van der Aalst et al, 2005], cioè la loro esecuzione è ritardata finché non diventa necessaria all'abilitazione di una transizione visibile corrispondente ad un evento della traccia. Noi introduciamo invece una gestione *eager* delle transizioni invisibili.

Il risultato del log-replay di una traccia T può essere rappresentato da una lista R di coppie (tr, i), ognuna delle quali rappresenta l'esecuzione della transizione tr per mimare l'evento $T[i]$. La presenza di transizioni invisibili può risultare in più transizioni eseguite per replicare un singolo evento, mentre la presenza di ricorsione può risultare in più occorrenze della stessa transizione.

3.1 Analisi di performance

Sfruttando i timestamp associati agli eventi, la tecnica di log-replay può essere usata per misurare le prestazioni del sistema [van der Aalst e van Dongen, 2002]. L'idea chiave è quella di calcolare l'intervallo di tempo tra produzione e consumo di token in ogni piazza della rete. Questa tecnica può essere applicata solo per le tracce che non richiedono la produzione di missing token, perché tali token non possono avere informazioni temporali. Durante il log-replay possono essere calcolate, per ogni traccia, le seguenti metriche riferite a ciascuna piazza: il tempo di soggiorno ($\bowtie$), cioè l'intervallo di tempo tra l'arrivo e la partenza di un token; il tempo di sincronizzazione ($\ltimes$), cioè l'intervallo di tempo tra l'arrivo di un token in una piazza e l'abilitazione di una transizione nel suo post-set; e il tempo di attesa ($\rtimes$), cioè l'intervallo di tempo tra l'attivazione e l'esecuzione di una transizione nel post-set della piazza (in generale vale $\ltimes + \rtimes = \bowtie$).

3.1.1 Log-replay con transizioni invisibili "eager"

Individuate queste misure da analizzare, e motivati dalla necessità di proiettare tali misure sul modello BPMN di partenza, abbiamo sviluppato un nuovo algoritmo per valutare le performance. Intuitivamente, prima di associare ad ogni piazza le misure, effettuiamo una rielaborazione della sequenza di transizioni ottenuta dal log-replay, in modo da eseguire le transizioni invisibili *il prima possibile*. Di seguito chiamiamo questo tipo di sequenza **eager**. In generale una sequenza è eager se per ogni transizione invisibile $t1$, l'ultima transizione visibile $t2$ che precede $t1$ è necessaria per abilitare $t1$. Il seguente algoritmo trasforma una generica sequenza R di coppie (tr, i) ottenuta dal log-replay in una corrispondente sequenza eager. Con la notazione $R \downarrow k$ indichiamo l'indice (tra 1 e $k-1$) dell'ultima transizione visibile che precede la posizione k, oppure 0 se tale transizione non esiste.

for j **from** 2 **to** size(R) **do**
  $tr, i \leftarrow R[j]$
  if tr *is invisible* **then**
    $k, done \leftarrow j-1, false$
    while $k > 0 \ \wedge \neg done$ **do**
      $tr_{prev}, i_{prev} \leftarrow R[k]$
      if ${}^\bullet tr \cap tr_{prev}^\bullet \neq \emptyset$ **then**
        $done \leftarrow true$
      else
        $R[k+1] \leftarrow (tr_{prev}, i_{prev})$
        $R[k] \leftarrow (tr, max(R \downarrow k, 1))$
        $k \leftarrow k-1$
      end if
    end while
  end if
end for

Per esemplificare le metriche calcolate da questa tecnica usiamo la rete di Petri in Fig. 4 e la traccia T in Fig. 3.

Gestendo le transizioni invisibili come *lazy*, il log-replay produce la sequenza $R1 = [(t1,1),(t2,1),(DCS,1),(t3,1),(IS,2),(IC,3),(DCC,4),(t5,4),(t7,4),(CRS,5),(CRC,6)]$.

Benché la transizione $t3$ sia abilitata dopo l'evento *DataCollection*, la sua esecuzione è ritardata finché non diventa necessaria per l'attivazione della transizio-

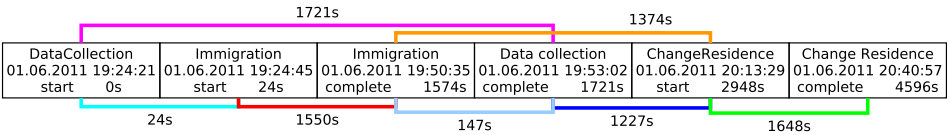

Fig. 3 - Esempio di Log per l'analisi di performance

ne visibile *Immigration*, quindi il token resta nella piazza $p1$ per $24s$ prima che la transizione $t3$ sia eseguita. Analogamente, nonostante la transizione $t5$ sia abilitata dopo l'evento *Immigration*, la sua esecuzione è ritardata finché non diventa necessaria per l'attivazione della transizione visibile *ChangeResidence*, quindi il token resta nella piazza $p4$ per $1374s$ prima che la transizione $t5$ sia eseguita. L'unico tempo di sincronizzazione non nullo è quello in $p6$ ($1227s$) e misura l'intervallo tra l'esecuzione delle transizioni DCC e CRS, che non sono le due attività concorrenti. È nostra opinione che queste misure non descrivano l'evoluzione del processo in modo preciso.

La sequenza $R1$ viene trasformata dal nostro algoritmo nella sequenza eager $R2 = [(t1, 1), (t2, 1), (t3, 1), (DCS, 1), (IS, 2), (IC, 3), (t5, 3), (DCC, 4), (t7, 4), (CRS, 5), (CRC, 6)]$. La valutazione della performance viene eseguita inserendo un token nella piazza iniziale al tempo 0.

- $t1$ e $t2$ sono le prime due transizioni eseguite e sono entrambe associate all'evento $T[1]$. Quindi, al tempo $\theta(T[1]) = 0$ il token nella piazza iniziale viene consumato e due token prodotti (uno in $p1$ e uno in $p2$) per attivare due esecuzioni concorrenti.
- La transizione $t3$ consuma il token in $p1$ e ne produce uno in $p3$ al tempo $\theta(T[1]) = 0$, attivando una delle due possibili validazioni. Quindi $\rtimes(p1) = 0$;
- La transizione DCS, anch'essa eseguita al tempo $\theta(T[1]) = 0$, consuma il token in $p2$ e produce un token in pdc.
- Al tempo $\theta(T[2]) = 24s$ viene eseguita la transizione IS, consumando il token in $p3$ e producendone uno in pi. Quindi, $\rtimes(p3) = 24s - 0s = 24s$
- Al tempo $\theta(T[3]) = 1574s$ viene eseguita la transizione IC, consumando il token in pi e producendone uno in $p4$. Quindi, $\rtimes(pi) = 1574s - 24s = 1550s$;
- Sempre al tempo $\theta(T[3]) = 1574s$ viene eseguita la transizione invisibile $t5$, consumando il token in $p4$ e producendone uno in $p5$. In questo momento $p5$ non è preset di nessuna transizione attiva ($t7$ dipende dalla piazza vuota $p6$) quindi il suo tempo di sincronizzazione sarà maggiore di zero.
- Al tempo $\theta(T[4]) = 1721s$ viene eseguita la transizione DCC, consumando il token in pdc e producendone uno in $p6$. Quindi, $\rtimes(pdc) = 1721s - 0s = 1721s$. In questo momento la transizione $t7$ viene abilitata ed è possibile calcolare i tempi di sincronizzazione dei preset: $\rtimes(p5) = 1721s - 1574s = 147s$ e $\rtimes(p6) = 1721s - 1721s = 0s$
- L'analisi delle ultime tre transizioni permette di calcolare le misure di performance per le rimanenti piazze.

In Fig. 4 è possibile vedere le misure di performance calcolate. Tutte le piazze in bianco hanno tempo di sincronizzazione e di attesa uguale a zero. Il tempo di

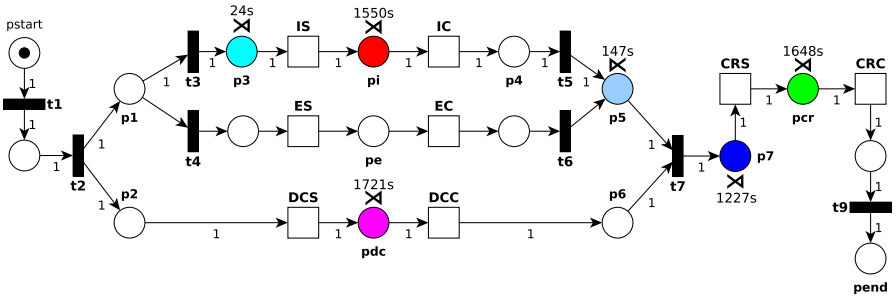

Fig. 4 - Rete di Petri del cambio di residenza con annotata l'analisi di performance

attesa delle piazze pi, pdc, pcr, rappresentano il tempo di esecuzione delle corrispondenti attività BPMN. La sola piazza con tempo di sincronizzazione maggiore di zero è il $p5$ $(147s)$. Questo tempo misura l'intervallo tra l'esecuzione della transazioni IC e DCC. È importante notare che il tempo di sincronizzazione nella piazza $p5$ è ottenuto grazie alla trasformazione della sequenza $R1$ nella sequenza eager $R2$ (che sposta la transizione $t5$ immediatamente dopo IC).

3.1.2 Proiezione dei risultati di analisi sul modello BPMN

La trasformazione delle sequenze del log-replay in sequenze eager semplifica la proiezione sul modello BPMN dei risultati calcolati, Se una transizione invisibile è abilitata questa viene immediatamente eseguita. Per questo motivo, i tempi di attesa delle piazze che hanno solo transizioni invisibili nei post-set sono sempre nulli. Facendo riferimento alle regole della Fig. 2, le piazze che possono avere tempi di attesa diversi dallo zero sono $P(x,T)$ e Pt, cioè le piazze usate per tradurre le attività BPMN. Allo stesso modo, il tempo di sincronizzazione di una piazza può essere maggiore di zero solo se almeno una transizione nel suo post-set dipende da un'altra piazza. Ne segue che le piazze in Fig. 2 che possono avere tempi di sincronizzazione diversi da zero sono solo $P(x1,J1)$ e $P(x2,J1)$, cioè quelle coinvolte nella modellazione del join gateway.

Sulla base di queste considerazioni si possono riportare le misure di performance della rete di Petri sul modello originale BPMN come segue:

- per ogni task T il tempo di esecuzione è $\rtimes(Pt)$ e il tempo di attivazione è $\rtimes(P(x,T))$
- per ogni ramo concorrente ($i \in [1,2]$) racchiuso da un fork gateway ($F1$) e un join gateway ($J1$): (i) il tempo di sincronizzazione è $\ltimes(P(xi,J1))$, ovvero il tempo impiegato per attendere il completamento delle attività concorrenti (ii) i tempi di esecuzione (di seguito $\rtimes(F1i)$)) sono la somma di $\bowtie$ di tutte le piazze raggiungibili attraversando il grafico a partire da $P(Fi,yi)$ senza aver visitato $P(xi,Ji)$. Si noti che$\ltimes(P(xi,J1))+\rtimes(F1i)$ è costante per ogni ramo di un fork/join. In seguito chiamiamo questa misura come il "tempo di esecuzione" del fork, e lo indicheremo come $(\bowtie(F1,J1))$.

La Fig. 5 mostra la proiezione delle analisi di performance sul modello BPMN. Sottolineiamo che le considerazioni appena fatte non valgono per le implemen-

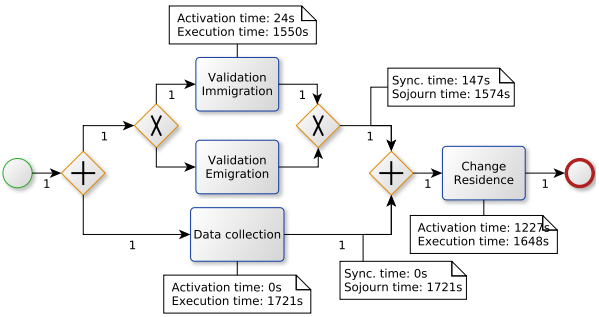

Fig. 5 - BPMN con annotata l'analisi di performance

tazioni preesistenti dell'analisi di performance (come quella di ProM 5.2) che non transformano le sequenze del log-replay in sequenze eager. In particolare, tali implementazioni permettono di valutare i tempi di sincronizzazione delle piazze coinvolte dai gateway join ($(P(x1, J1)$ e $P(x2, J1))$) solo se sono immediatamente precedute da una transizione visibile.

3.2 Analisi di conformance

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

Fig. 6 - Esempio di Log

I problemi di conformità possono essere diagnosticati analizzando i token creati artificialmente dal log-replay (i *missing token*) ed i token che non sono stati consumati (i *remaining token*). Il log rappresentato in Fig. 6 fornisce un esempio di esecuzione errata rispetto al processo di Fig. 1. La traccia contiene l'esecuzione di attività mutualmente esclusive (*Immigration* ed *Emigration*) e non contiene gli eventi relativi al completamento dell'attività *DataCollection*. Le misure di conformità prodotte dal log-replay per questa traccia sono mostrate nella Fig. 7. Nella piazza $p3$ non è presente alcun token, tuttavia almeno un token è necessario per il replay della transizione ES che non è abilitata perché il ramo alternativo *Immigration* è già stato eseguito ed ha consumato il token. Nella piazza pdc rimane un token: questa situazione avviene perché nel log manca la terminazione dell'attività *DataCollection*. Dunque la piazza $p7$ non riceve alcun token durante il log-replay, la sincronizzazione $t7$ fallisce non essendo mai abilitata e quindi resta un token nelle piazze $p4$ e $p5$. Il token mancante nella piazza $p8$ è necessario per il replay della transizione CRS.

Descriviamo ora come riportare i dati di conformance sul modello originale. Il log-replay produce artificialmente i token mancanti solo quando sono necessari per eseguire le transizioni visibili. Pertanto i token mancanti possono essere generati solamente in quelle piazze che hanno nel post-set almeno una transizione visibile. Nella Fig. 2 queste piazze sono $P(x, T)$ e Pt. Inoltre il log-replay esegue le transizioni invisibili solo se la loro esecuzione è necessaria per attivare una

transizione visibile. Per esempio, per ogni esecuzione della transizione iniziale ts il log-replay esegue necessariamente una transizione visibile che consuma il token nella piazza $P(s, y)$. La stessa considerazione si applica a tutte le transizioni invisibili che producono un solo token. Quindi, le sole piazze in cui possono rimanere token sono: (i) quelle nel post-set di una transizione visibile; (ii) quelle nel post-set di una transizione invisibile che produce più di un token. Nella Fig. 2 queste piazze sono Pt, $P(T, y)$, $P(F1, y1)$ e $P(F1, y2)$, cioè quelle coinvolte nella modellazione di una singola attività BPMN e del fork gateway.

In base a queste considerazioni, le metriche di conformità della rete di Petri sono riportate sul diagramma BPMN come segue:

- Per ogni task T un eventuale token mancante in $P(x, T)$ viene definito come "un'esecuzione errata"; la mancanza di un token in Pt viene definita come "un fallimento interno"; la presenza di un token rimanente in Pt viene definita come "un completamento mancante"; e un token rimanente in $P(T, y)$ viene definito come "un'esecuzione interrotta".
- Per ogni ramo ($i \in 1, 2$) di un fork $F1$, i token rimanenti in $P(F1, yi)$ sono detti "rami interrotti". Si noti che per ogni esecuzione di $tF1$ un token può rimanere in $P(F1, y1)$ oppure in $P(F1, y2)$, ma non in entrambe le piazze.

Nella Fig. 8 i dati di conformance sono proiettati sul modello BPMN. La figura evidenzia: (i) il mancato completamento di *DataCollection* (ii) l'esecuzione sbagliata di *Emigration*, causata dall'attivazione dell'altro ramo del gateway esclusivo (iii) le esecuzioni interrotte di *Emigration* e *Immigration*, causate dalla mancata sincronizzazione del gateway join (iv) l'esecuzione errata di *ChangeResidence*.

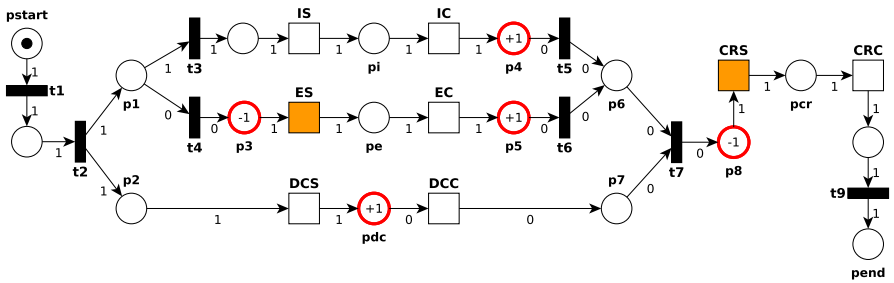

Fig. 7 - Rete di Petri con annotata l'analisi di conformance

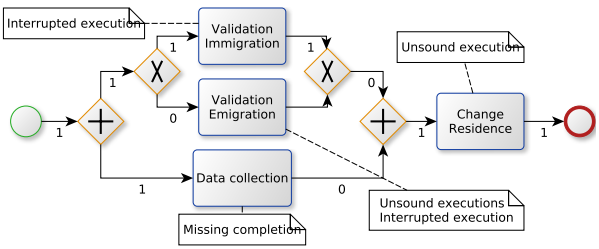

Fig. 8 - BPMN con annotata l'analisi di conformance

4. Conclusioni

Abbiamo presentato l'estensione dell'algoritmo esistente di valutazione di performance in modo da gestire le transizioni invisibili come "eager". La nostra estensione permette la valutazione dei tempi di sincronizzazione anche per le reti di Petri complesse ottenute dai modelli BPMN e semplifica la proiezione delle analisi sul modello di partenza. La possibilità di analizzare modelli BPMN e la presentazione dei risultati come annotazione dei modelli stessi permettono, ai vari attori coinvolti nella gestione dei processi, di trarre beneficio dalle diverse tecniche formali in modo trasparente.

Le tecniche presentate sono state implementate come quattro nuovi plug-in di ProM 6: (i) PerformanceAnalysis utilizza il log-replay esistente, ne trasforma gli output in sequenze eager e valuta i tempi di attesa e sincronizzazione; (ii) BpmnToPetri trasforma modelli BPMN in reti di Petri; (iii) ConformanceToBPMN e (iv) PerformanceToBPMN annotano i modelli BPMN con gli artifact necessari per rappresentare le misure di performance e conformance.

Stiamo attualmente estendendo il framework per integrare tecniche di data mining. In particolare ci stiamo concentrando sulla possibilità di fornire strumenti per individuare le classi di processo che causano colli di bottiglia. Il nostro obiettivo è quello di integrare i linguaggi esistenti per la definizione di Service Level Agreement e verificarne la conformità dei processi.

Riferimenti bibliografici

Börger E., Thalheim B. Modeling workflows, interaction patterns, web services and business processes: The ASM-based approach. In *ABZ*. LNCS, 5238, 2008, 24–38.

Dijkman R. M., Dumas M., Ouyang C. Semantics and analysis of business process models in BPMN. Information & Software Technology, 50-12, 2008, 1281–1294.

Eindhoven Univ. of Technology, 2011. *ProM 6.1*. http://www.processmining.org/.

IEEE, 2011. *Task Force on Process Mining*. http://www.win.tue.nl/ieeetfpm/doku.php.

Link.it, 2011. *Progetto RUPOS*. http://www.link.it/projects/rupos/.

Milner R., Parrow J., Walker D. A calculus of mobile processes, I. Inf. Comput., 100-1, 1992, 1–40.

OASIS, 2007. *WSBPEL 2.0*. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html.

OMG, 2011. *BPMN 2.0*. http://www.omg.org/spec/BPMN/.

Petri C. A. Fundamentals of a theory of asynchronous information flow. In *IFIP Congress*, 1962, 386–390.

Rozinat A., van der Aalst W. M. P. Conformance checking of processes based on monitoring real behavior. Inf. Syst., 33-1, 2008, 64–95.

Scheer A.-W. Architecture of integrated information systems (ARIS). In *DIISM*. IFIP Transactions, B-14, 1993, 85–99.

van der Aalst W. M. P. The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers, 8-1, 1998, 21–66.

van der Aalst W. M. P., van Dongen B. F. Discovering workflow performance models from timed logs. In *EDCIS*. LNCS, 2480, 2002, 45–63.

van der Aalst W. M. P., de Medeiros A. K. A., Weijters A. J. M. M. Genetic process mining. In *ICATPN*. LNCS, 3536, 2005, 48–69.