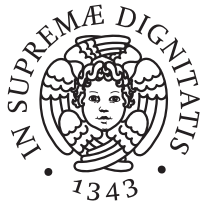


Università degli studi di Pisa
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Triennale in Informatica

Tecniche di data mining per l'analisi di processi di business

Candidato:
Hind Chfouka

Tutori Accademici:
Andrea Corradini
Roberto Guanciale

Anno accademico 2010-2011

Ai miei genitori

Indice

Capitolo 1

Introduzione

Il tirocinio svolto nasce nell'ambito del progetto RUPOS (Ricerca per l'Usabilità delle Piattaforme Orientate ai Servizi) finanziato dalla Regione Toscana nell'ambito del programma di aiuti alle imprese per investimenti in materia di ricerca industriale e sviluppo sperimentale.

Il progetto è stato portato avanti dalla collaborazione fra tre enti: il Dipartimento di Informatica dell'Università di Pisa, presso cui è stato svolto il tirocinio, la società spin-off del medesimo dipartimento Link.it e la società Hyperborea.

L'obiettivo del progetto RUPOS è ricercare, realizzare e sperimentare tecniche innovative per la progettazione, l'esecuzione, il monitoraggio e la gestione di applicazioni distribuite realizzate secondo un'architettura orientata ai servizi (SOA) [20], ed integrare tali tecniche in una nuova piattaforma open source. Tale piattaforma dev'essere inoltre arricchita da avanzati tool di monitoraggio che permettono di ricostruire la dinamica dei processi di business, ricavare indicatori di performance ed individuare eventuali scostamenti ed anomalie rispetto ai comportamenti attesi.

Per il monitoraggio, RUPOS ha realizzato un sistema di tracciamento, archiviazione ed elaborazione dei log che permette di raccogliere grandi quantità di dati sull'esecuzione dei processi di business. Avere molti dati tuttavia non significa disporre di informazioni in grado di spiegare le ragioni legate al manifestarsi di certi fenomeni, oppure in grado di fornire indicazioni strategiche agli esperti del dominio. Per una più profonda comprensione dei processi di business, la disciplina del process mining offre delle potenzialità capaci di ripondere a queste esigenze.

Il tirocinio si colloca proprio nell'ambito della ricerca, condotta dal Dipartimento di Informatica, di tecniche di process mining da attuare a servizio degli esperti del dominio del processo di business. Tali tecniche sono realizzate e sperimentate con l'appoggio di ProM, un sofisticato framework open source di process mining [14].

L'obiettivo del tirocinio consiste nella ricerca di altri metodi di analisi per l'arricchimento delle tecniche già esistenti. In particolare, questa esplorazione deve orientarsi verso la possibilità di utilizzare quelle enormi quantità di dati che sono raccolte nei log, ma che sono considerate solo in parte dalle tecniche attuali. L'idea è stata quella di mettere in atto un approccio basato sia sui risultati delle tecniche già esistenti che su tecniche di data mining. Tale approccio dev'essere in grado di individuare le regole nei dati che influenzano il comportamento dei processi. In particolare, l'approccio deve portare alla scoperta di regole nei dati del processo, in corrispondenza delle quali si verificano gli errori di conformance, ovvero scostamenti rispetto al comportamento atteso. Sono stati sviluppati a questo proposito un insieme di plugin che estendono ProM per sperimentare l'approccio su dei prototipi di processi di business generati sperimentalmente.

Le tecniche esistenti, che costituiscono peraltro il punto di partenza del tirocinio, si distinguono in due categorie: tecniche che si occupano di fornire una rappresentazione rigorsa del modello di processo di business, e tecniche in grado di analizzare il processo, in modo formale, a partire dal modello derivato e dai log registrati. Nella prima categoria rientra lo sviluppo di una metodologia capace di tradurre un modello di processo espresso con un formalismo grafico ad alto livello (BPMN) ad un modello espresso mediante le reti di Petri. Nella seconda categoria invece rientrano tecniche in grado di analizzare un processo di business valutandone la fedeltà del modello, ovvero identificando le discrepanze fra il comportamento da esso previsto e gli eventi effettivamente registrati nei log delle esecuzioni (analisi di conformance). Sempre nella seconda categoria rientrano tecniche di analisi in grado di fornire delle misure di perfomance del processo, ovvero la rilevazione dei tempi di esecuzione di tutto il processo o delle singole attività che lo compongono, e la rilevazione di tempi di sincornizzazione nel caso di attività che procedono in parallelo.

Per potere raggiungere l'obiettivo del tirocinio è stato primariamente fondamentale effettuare uno studio sulle tematiche attinenti al process mining,

con particolare riguardo alle tecniche che costituiscono la base di partenza. Oltre a questo, si è rivelato necessario esplorare le tecniche presenti in letteratura per l'analisi di grandi quantità di dati. In questo senso lo studio si è orientato verso tematiche attinenti al data mining, più specificamente sulla classificazione mediante alberi di decisione.

Questa relazione ha l'obiettivo di documentare il lavoro di tirocinio svolto. Nel capitolo 2 sono presentate le tematiche di process mining che sono stati oggetto di studio, sono presentati quindi i formalismi per la rappresentazione dei processi di business e le tecniche di analisi dei processi. Nel capitolo 3 viene esplorato il problema della classificazione con alberi di decisione. Nel capitolo 5 è fornita invece una descrizione dell'approccio sviluppato, viene mostrato come è possibile trasformare un problema di conformance in uno di classificazione, inoltre, attraverso la descrizione di due prototipi sperimentali di processo, vengono illustrati alcuni benefici che gli analisti di processo possono trarre adottando questo tipo di approccio. I capitoli 4 e 6 assumono invece un carattere più tecnico. In 4 infatti viene fornita una descrizione di tutte le tecnologie impiegate nel lavoro di tirocinio, mentre in 6 è presentato l'insieme dei plugin sviluppati che danno una possibile implementazione del suddetto approccio.

Capitolo 2

Process Mining

Un processo di business è inteso come un insieme di attività da svolgere in modo correlato, con lo scopo di raggiungere un obiettivo prefissato in fase di pianificazione. Nel corso degli anni questo concetto ha assunto una rilevanza sempre maggiore all'interno dei contesti organizzativi, facendo spostare l'attenzione dall'obiettivo vero e proprio alle fasi che portano al suo raggiungimento. Ciò ha portato alla nascita di una serie di discipline, tra le quali il process mining, volte alla facilitazione del trattamento dei processi di business.

Attualmente, tantissime informazioni relative ai processi di business vengono registrate dai sistemi informativi sotto forma di event log. Il process mining è una disciplina giovane nata per sfruttare i dati registrati nei log per estrarre informazioni relative ai processi. Ad esempio, in molti contesti organizzativi i processi sono in continua evoluzione, di conseguenza le persone coinvolte tendono ad acquisire una visione errata (o semplificata) di quello che è diventato il processo di business attuale: il process mining interviene per evitare inconvenienti simili mettendo in atto alcune tecniche per l'analisi dei processi ripetibili. Tra le varie tecniche del process mining, quella di interesse per il tirocinio è rappresentata dall'algoritmo log replay. A partire da un modello di processo e da un event log, il log replay è il principale strumento di analisi del processo.

In questo capitolo vengono introdotti due formalismi per la modellazione dei processi di business: BPMN che è descritto nella sezione 2.1.1 e le reti di Petri presentate in 2.1.2. Il primo formalismo è caratterizzato da una forte espressività, il secondo da un forte rigore formale adatto per essere analizzato con precisione dall'algoritmo log replay. Poichè è più intituivo impiegare un

linguaggio espressivo per la modellazione, nella sezione 2.1.3 sono descritte alcune regole per passare da un formalismo all'altro. Nell'ambito dell'analisi dei processi invece, viene presentato un algoritmo fondamentale a questo scopo e vengono descritte due tipologie di analisi possibili: analisi di conformance e di performance. Infine, sono esplorati alcuni accorgimenti utili per proiettare i risultati ottenuti con l'analisi sul modello di processo preso in considerazione.

2.1 Overview sui processi di business

Un processo di business è definito nella sua accezione più ampia come un insieme di attività correlate, svolte all'interno di un'organizzazione con lo scopo di raggiungere un determinato obiettivo.

In generale, il modo di rappresentare i processi di business è quello basato sui diagrammi di flusso. In questo elaborato, per conformità alle tecniche già esistenti su cui si basa il lavoro di tirocinio, i formalismi usati sono BPMN (Business Process Model and Notation) e le reti di Petri.

2.1.1 Business Process Model and Notation

In questa sede questo formalismo non viene introdotto in modo dettagliato, ma semplicemente viene data una panoramica sui costrutti principali e presentato un esempio significativo.

BPMN è uno standard per la rappresentazione dei processi di business creato nel 2005 dalla Business Process Management Initiative (BPMI) [1] [12]. Esso si pone come obiettivo quello di fornire una notazione comprensibile da analisti che definiscono i processi, sviluppatori coinvolti nella implementazione tecnologica dei processi, nonché da persone del business che gestiscono e tengono sotto controllo i processi.

L'adozione di BPMN permette alle organizzazioni di rappresentare i propri processi tramite una notazione intuitiva (diagrammi di flusso), oltre al fatto che la standardizzazione agevola la comunicazione (si pensi a quando si ha a che fare con degli enti esterni). Occorre tuttavia osservare che BPMN

[1] La BPMI è un'organizzazione nata nel 2000 per promuovere la standardizzazione nell'ambito dei processi di business.

serve a modellare solo i processi e non a rappresentare altri aspetti importanti all'interno di un contesto organizzativo, quali possono essere le strutture organizzative, le decomposizioni funzionali, le regole di business oppure le strategie funzionali.

BPMN è un linguaggio per modellare processi di business attraverso un'astrazione grafica che pone l'enfasi sul controllo di flusso (control-flow). Esso definisce un diagramma detto Business Process Diagram (BPD) simile ad un digramma di flusso che incorpora dei costrutti utili per la modellazione dei processi di business. Un BPD è costruito a partire dagli elementi di BPMN, i cui principali sono "oggetti" e "sequenze di flusso". Un oggetto può essere un evento, un'attività oppure un gateway. Una sequenza di flusso invece può collegare due oggetti in un BPD mostrando il flusso del controllo, ovvero l'ordine di esecuzione.

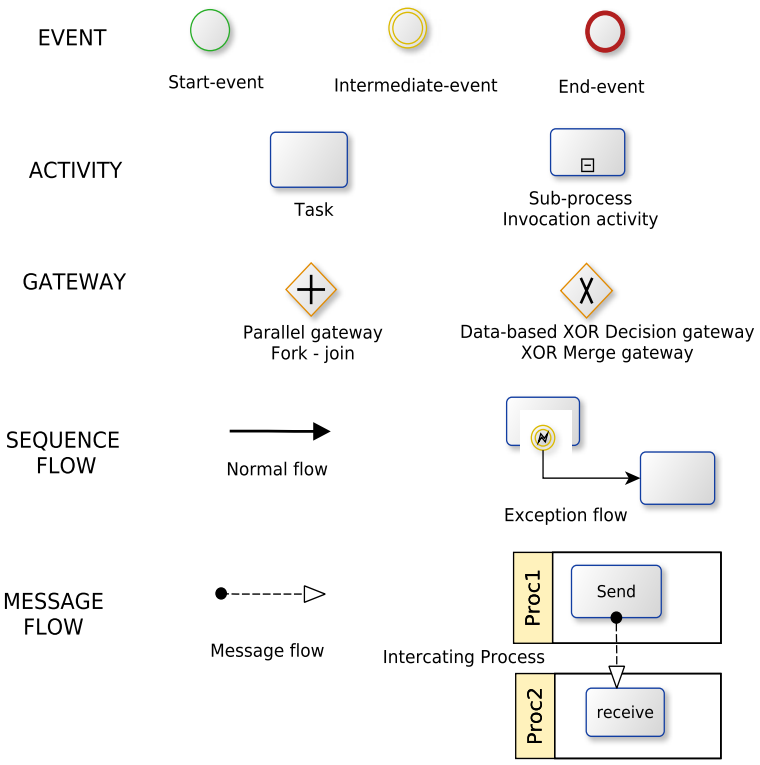


Figura 2.1: Elementi BPMN

Gli eventi si distinguono in:

- evento di inizio: permette di inizializzare un processo.
- evento di fine: permette di segnalare la terminazione di un processo.
- evento intermedio: rappresenta un evento che può verificarsi durante l'esecuzione del flusso di controllo. Un esempio può essere l'arrivo di un messaggio.

Le attività invece considerate in questo nucleo BPMN sono di due tipi:

- Task: rappresenta un'attività, intesa come un'operazione atomica da eseguire all'interno del processo.
- Sub-process Invocation activity: rappresenta l'invocazione di un insieme di attività intese come un sottoprocesso definito a sé stante.

I gateway invece sono di due tipi:

- Parallel gateway: modella un punto in cui il flusso di esecuzione viene sdoppiato in due flussi che procedono in modo parallelo ed indipendente (fork). Parallel gateway è usato anche per modellarare un punto di sincronizzazoione ed unificazione di due flussi di esecuzioni separati, che procedevano in parallelo, in un unico flusso (join).
- XOR gateway: modella un punto di decisione in base ai dati del problema. E' impiegato in contesti in cui esistono varie direzioni su cui il flusso di controllo può proseguire. XOR gateway è utilizzato anche nei punti in cui si vuole fare l'unione (merge) di due flussi separati.

L'esempio in figura 2.2 mostra come può essere modellato con BPMN un semplice processo di business della procedura di cambio di residenza.
Il processo descritto con questo modello può essere sintetizzato come segue:

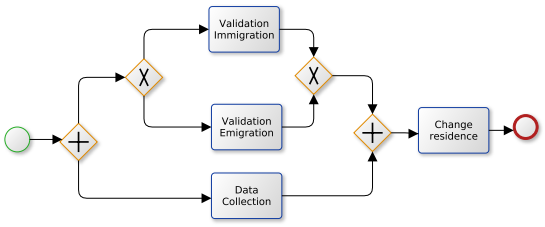


Figura 2.2: Esempio BPMN

Dopo l'evento di inizio "event start", segue il costrutto "Fork" che permette

di proseguire due rami di attività in parallelo in quanto risultano indipendenti; lungo un ramo troviamo il costrutto di scelta "Data Based Decision gateway" che permette di procedere con l'attività "Validation Immigration" oppure (in modo mutualmente esclusivo) l'attività "Validation Emigration". L'altro ramo invece presenta una sola attività da eseguire ed è :"Data collection". Dopo l'esecuzione di queste attività, i due rami paralleli si sincronizzano per ricongiungersi. La sincronizzazione è rappresentata dal costrutto "Join gateway". Il flusso di esecuzione unificato procede con l'esecuzione dell'ultima attività "Change Residence". La terminazione del processo è segnalata con il costrutto di fine "end event".

2.1.2 Reti di Petri

In questo lavoro, le reti di Petri rappresentano il formalismo principale su cui si baseranno tutte le analisi dei processi di business. La capacità delle reti di Petri di rappresentare processi in modo rigoroso e privo di ambiguità rende possibile l'applicazione di tecniche di analisi e di verifica del comportamento del sistema. BPMN invece è un linguaggio meno formale ed in quanto tale non del tutto privo di ambiguità. Dato che BPMN è lo standard de facto per la descrizione di processi, in sezione 2.1.3 vedremo un mapping che permette di trasformare un modello descritto con questo formalismo in uno equivalente interamente espresso con una rete di Petri.

Le reti di Petri, proposte nel 1962 da Carl Adam Petri [13], sono uno strumento per modellare processi ed in particolare per modellare la comunicazione fra processi paralleli. Nella descrizione di un processo spesso si ha la necessità di rappresentare sottoprocessi o attività che possono essere eseguite contemporaneamente fra loro ma non in modo del tutto indipendente l'una dall'altra: potrebbe infatti accadere che un determinato passaggio o una certa fase del processo non possa verificarsi o non possa essere attivata fintanto che altre fasi o attività non sono concluse o fino al verificarsi di determinate condizioni. Un digramma di flusso, come quelli usati nella descrizione di un algoritmo strutturato, non è adatto nella modellazione di situazioni di questo tipo, infatti in quest'ultimo le attività sono rigidamente serializzate. Con le reti di Petri questo vincolo viene superato. Esse introducono anche un formalismo grafico ed una semantica formale che rappresentano un valore aggiunto rispetto ai diagrammi di flusso. Un aspetto ancora importante da non trascurare consiste nel fatto che esse non solo permettono di descrivere globalmente un processo, ma consentono anche di seguirne l'evoluzione per-

mettendo di visualizzare lo stato in cui la rete si trova in un certo istante.

Il modello delle reti di Petri è basato sui concetti di *condizioni e azioni*. Una rete di Petri descrive un sistema specificando l'insieme delle azioni che possono essere compiute dal sistema, le condizioni che rendono possibile ciascuna azione, e le condizioni che diventano vere in seguito all'esecuzione di ciascuna azione. Lo stato del sistema è definito dall'insieme delle condizioni che in un dato istante sono vere, e le transizioni da uno stato all'altro sono causate dall'esecuzione delle azioni (che nel vocabolario delle reti di Petri si chiamano anche transizioni).

Generalità

Una rete di Petri è un grafo orientato bipartito, i cui nodi sono divisi in *piazze* e *transizioni*, ed i cui archi uniscono piazze a transizioni o transizioni a piazze (quindi uniscono due nodi di tipo diffente). Se un arco va da una piazza ad una transizione la piazza è detta di *ingresso*, altrimenti è di *uscita*. Ad ogni piazza p è associato un numero intero $M(p)$, e si dice che p è *marcata* con $M(p)$ token, o che *contiene* $M(p)$ token. La funzione M che assegna un numero di token ad ogni piazza è la *marcatura* della rete; questa funzione serve a modellare le condizioni che rendono possibile lo *scatto* (cioè l'esecuzione) delle transizioni. Graficamente le piazze vengono rappresentate da cerchi, le transizioni da segmenti o rettangoli, gli archi da frecce, ed i token contenuti in un piazza p da $M(p)$ "pallini" dentro il piazza.

Più precisamente, una *rete piazze/transizioni* è una terna:

$$N = \langle P, T, F \rangle \tag{2.1}$$

dove:

- gli insiemi P (piazze) e T (transizioni) sono disgiunti:

$$P \cap T = \emptyset \tag{2.2}$$

- la rete non è vuota

$$P \cup T \neq \emptyset \tag{2.3}$$

- F è una relazione che mappa piazze in transizioni e transizioni in piazze:

$$F \subseteq (P \times T) \cup (T \times P) \tag{2.4}$$

La relazione di flusso F può essere espressa per mezzo di due funzioni $pre()$ e $post()$ che associano, rispettivamente, gli elementi di ingresso e gli elementi di uscita a ciascun elemento della rete, sia esso un posto oppure una transizione. Possiamo considerare separatamente le restrizioni delle due funzioni a due insiemi di nodi, usando per semplicità lo stesso nome della rispettiva funzione complessiva:

- Pre-set di una piazza [2]:

$$pre : P \to 2^T \tag{2.5}$$

$$pre(p) = {\bullet p} = \{t \in T \mid \langle t, p \rangle \in F\} \tag{2.6}$$

- Pre-set di una transizione:

$$pre : T \to 2^P \tag{2.7}$$

$$pre(t) = {\bullet t} = \{p \in P \mid \langle p, t \rangle \in F\} \tag{2.8}$$

- Post-set di un posto:

$$post : P \to 2^T \tag{2.9}$$

$$post(p) = p{\bullet} = \{t \in T \mid \langle p, t \rangle \in F\} \tag{2.10}$$

- Post-set di una transizione:

$$post : T \to 2^P \tag{2.11}$$

$$post(t) = t{\bullet} = \{p \in P \mid \langle t, p \rangle \in F\} \tag{2.12}$$

[2]La notazione 2^T indica *l'insieme delle parti* di T, cioè l'insieme dei suoi sottoinsiemi

La rete così definita è semplicemente la descrizione di una grafo, data elencandone i nodi e gli archi col relativo orientamento (una coppia $\langle x, y \rangle$ è un arco orientato da x a y), quindi la terna $\langle P, T, F \rangle$ rappresenta solo la struttura della rete. Per rappresentare lo stato iniziale del sistema modellato con la rete si introduce la funzione *marcatura iniziale*.

Una *Rete Piazze/Transizioni marcata* è quindi una quadrupla:

$$N = \langle P, T, F, M_0 \rangle \tag{2.13}$$

dove:

- La funzione M_0 è la *marcatura iniziale*:

$$M_0 : P \to \mathbb{N} \tag{2.14}$$

Abilitazione e regola di scatto

La marcatura iniziale descrive lo stato iniziale del sistema: in questo stato, alcune azioni sono possibili ed altre no. Delle azioni possibili, alcune (o tutte) vengono eseguite, in un ordine non specificato (e nemmeno specificabile in questo formalismo). L'esecuzione di queste azioni modifica lo stato del sistema, che viene descritto da una nuova funzione marcatura, ovvero cambia il valore associato a ciascuna piazza (valore chiamato convenzionalmente "numero di token").

Le azioni sono modellate dalle transizioni, quando l'azione corrispondente ad una transizione è possibile nello stato attuale, si dice che la transizione è *abilitata* nella marcatura attuale; quando l'azione viene eseguita si dice che la transizione *scatta (fires)*.

Una transizione t è *abilitata nella marcatura* M se e solo se :

$$\forall\ p \in {\bullet t} :\ M(p) > 0 \tag{2.15}$$

cioè, per ogni piazza in ingresso alla transizione è presente almeno un token. L'abilitazione di una transizione t in una marcatura M si indica con la notazione:

$$M[t\rangle \tag{2.16}$$

che si legge *"t è abilitata in M"*.

Dopo lo scatto di una transizione la rete assume una nuova marcatura M' il cui valore è determinato dalla seguente *regola di scatto*, dove M è la marcatura precedente allo scatto:

$$\forall p \in (\bullet t - t\bullet) : M'(p) = M(p) - 1 \tag{2.17}$$

$$\forall p \in (t\bullet - \bullet t) : M'(p) = M(p) + 1 \tag{2.18}$$

$$\forall p \in (\bullet t \cap t\bullet) : M'(p) = M(p) \tag{2.19}$$

$$\forall p \in P - (\bullet t \cup t\bullet) : M'(p) = M(p) \tag{2.20}$$

La regola di scatto può essere così riassunta:

1. la marcatura di ciascuna piazza in ingresso viene diminuita di uno.
2. la marcatura di ciascuna piazza in uscita viene aumentata di uno.
3. la marcatura di ciascuna piazza di ingresso e di uscita resta invariata .
4. la marcatura delle piazze non collegate direttamente alla transizione (quindi né d'ingresso né d'uscita) resta invariata.

Se, in una data marcatura, più transizioni sono abilitate, ne scatta una sola, scelta (da un ipotetico esecutore della rete) in modo ***non deterministico***. La notazione:

$$M[t\rangle M' \tag{2.21}$$

indica che la marcatura M' è il risultato dello scatto della transizione t abilitata in M.

Si osservi che lo scatto di una transizione ha solo effetti locali, cambiando solo la marcatura delle piazze in ingresso ed in uscita della transizione stessa.

Due transizioni t_1 e t_2 possono scattare in successione se:

$$M[t_1\rangle M' \wedge M'[t_2\rangle M'' \tag{2.22}$$

e si dice allora che *la sequenza di scatti* t_1t_2 è abilitata in M, e si scrive:

$$M[t_1t_2\rangle M'' \tag{2.23}$$

In generale, una sequenza di scatti di lunghezza n viene rappresentata con la seguente notazione:

$$S^{(n)} = t_1....t_n \tag{2.24}$$

e scriviamo:

$$M[t_1...t_n\rangle M^{(n)} \tag{2.25}$$

ovvero:

$$M[S^{(n)}\rangle M^{(n)} \tag{2.26}$$

Situazioni fondamentali

Consideriamo alcune relazioni fra transizioni, determinate sia dalla struttura della rete che dalla marcatura.

sequenza: Due transizioni t ed u si dicono *in sequenza* in una marcatura M se e solo se:

$$M[t\rangle \wedge \neg(M[u\rangle) \wedge M[tu\rangle \tag{2.27}$$

cioè u diventa abilitata solo dopo che è scattata t.

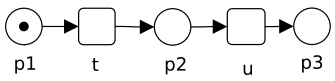


Figura 2.3: Esempio di sequenza

Conflitto: Due transizioni t ed u si dicono in *conflitto strutturale* quando si verifica la condizione: $\bullet t \cap \bullet u \neq \emptyset$, questa corrisponde all'esistenza di piazze di ingresso in comune fra le due transizioni.

Due transizioni t ed u si dicono in *conflitto effettivo* in una marcatura M se e solo se:

$$M[t\rangle \wedge M[u\rangle \wedge \exists p \in {\bullet t} \cap {\bullet u}.\ M(p) < 2 \tag{2.28}$$

Questa condizione significa che tutte e due le transizioni sono abilitate, ma lo scatto di una disabilita l'altra, togliendo dei token dalla piazza di ingresso comune alle due transizioni.

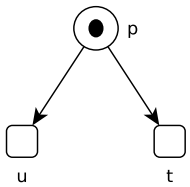


Figura 2.4: Esempio di conflitto

Concorrenza: Due transizioni t ed u si dicono in *concorrenza strutturale* se vale: ${\bullet t} \cap {\bullet u} = \emptyset$, ovvero quando le due transizioni non hanno piazze di ingresso in comune.

Due transizioni t ed u si dicono in *concorrenza effettiva* in una marcatura

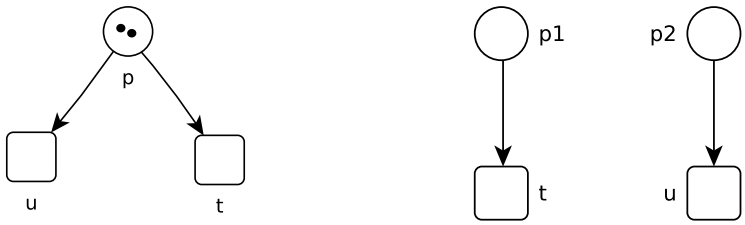


Figura 2.5: Esempio di concorrenza effettiva e strutturale.

M se e solo se:

$$M[t\rangle \wedge M[u\rangle \wedge \forall p \in \bullet t \cap \bullet u.\ (M(p) \geq 2 \tag{2.29}$$

Questa situazione significa che tutte e due le transizioni sono abilitate e possono scattare in qualsiasi ordine, poichè non si influenzano a vicenda.

2.1.3 Mapping BPMN in reti di Petri

In questa sezione viene presentata una trasformazione da modelli BPMN a reti di Petri come presentato in [4]. Questa trasformazione permette di passare da un formalismo ad alto livello che fornisce delle primitive grafiche fortemente espressive (che risultano quindi di facile utilizzo da tutti i soggetti coinvolti in un contesto organizzativo), ad un formalismo di basso livello (presenta pochi costrutti con una semantica formale ben precisa ed è adatto per l'esecuzione degli algoritmi di analisi).

La trasformazione, le cui regole sono presentate in figura 2.6, si applica agli elementi BPMN presentati nella sezione 2.1.1.

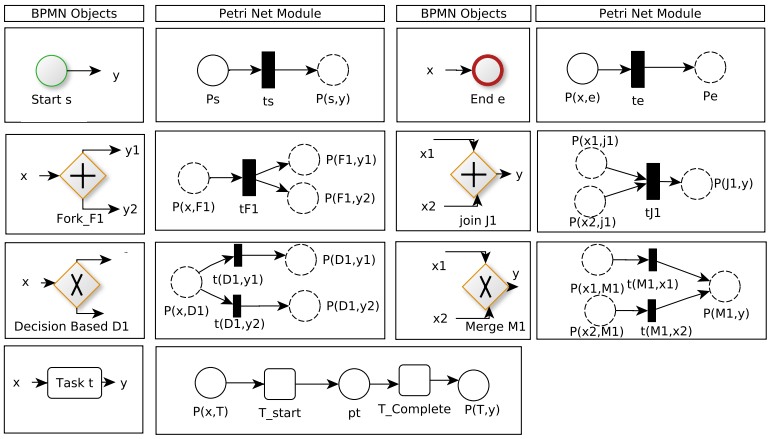


Figura 2.6: Mapping BPMN in reti Petri

Nella trasformazione sono stati utilizzati due simboli differenti per rappresentare due classi di transizioni:

- transizioni invisibili: vengono generate per modellare alcuni elementi di BPMN e sono rappresentate con dei rettangoli riempiti. Tali transizioni sono impiegate nella figura 2.6 per la trasformazione del Parallel

gateway (tF_1 e tJ_1), del XOR gateway ($t(D, y_1)$, $t(D, y_2)$, $t(M_1, x_1)$, $t(M_1, x_2)$), dello Start event (ts) e dell'End event (te).

- transizioni visibili: sono transizioni che corrispondono ad attività BPMN e vengono rappresentate con dei quadrati non riempiti. In particolare, come è mostrato nella figura 2.6, un'attività T viene tradotta con due transizioni visibili: T_start rappresenta l'inizio dell'attività mentre $T\_complete$ rappresenta la terminazione.

2.2 Overview sul process mining

Il process mining cerca di estrarre informazioni non banali ed utili dagli event log. Un event log è una sequenza di eventi che un sistema generico registra mentre è attivo. Il process mining è applicabile ad un insieme ampio di sistemi informativi che hanno incorporato il concetto di processo di business al loro interno e che tengono traccia degli event log, come può essere un sistemi ERP (Enterprise Resources Planning). Questa disciplina comprende delle tecniche di analisi a posteriori che sfruttano l'informazione registrata nei log. Tipicamente questi approcci ipotizzano che è possibile sequenzializzare gli elementi che costituiscono l'event log, in modo tale che ognuno riferisca una attività appartenente ad una determinata istanza di processo (ovvero una particolare esecuzione del processo).

Il process mining focalizza l'attenzione sul fatto che chi ha modellato il processo di business, o se vogliamo il proprietario di processo, ha una visione limitata su quello che sta realmente accadendo all'interno della realtà organizzativa in cui il processo trova esecuzione. In pratica, esiste spesso un divario tra quello che è stato prefissato e quello che accade realmente. Soltanto una valutazione della realtà organizzativa, che è proprio quel che il process mining cerca di offrire, può aiutare nella verifica del modello di processo di business, e quindi permettere eventualemtente una reingegnerizzazione del processo apportando con ciò un miglioramento nel contesto organizzativo.

L'idea del process mining è la scoperta, il monitoraggio ed il miglioramento dei processi reali (quindi non dei modelli di processo) tramite l'estrazione di conoscenza dai log. L'interesse principale quindi è quello di osservare, analizzare i processi così come sono eseguiti.

La figura 2.7 estratta da [6] mette in evidenza gli aspetti principali del process mining. In un contesto reale in cui sono impiegate entità software

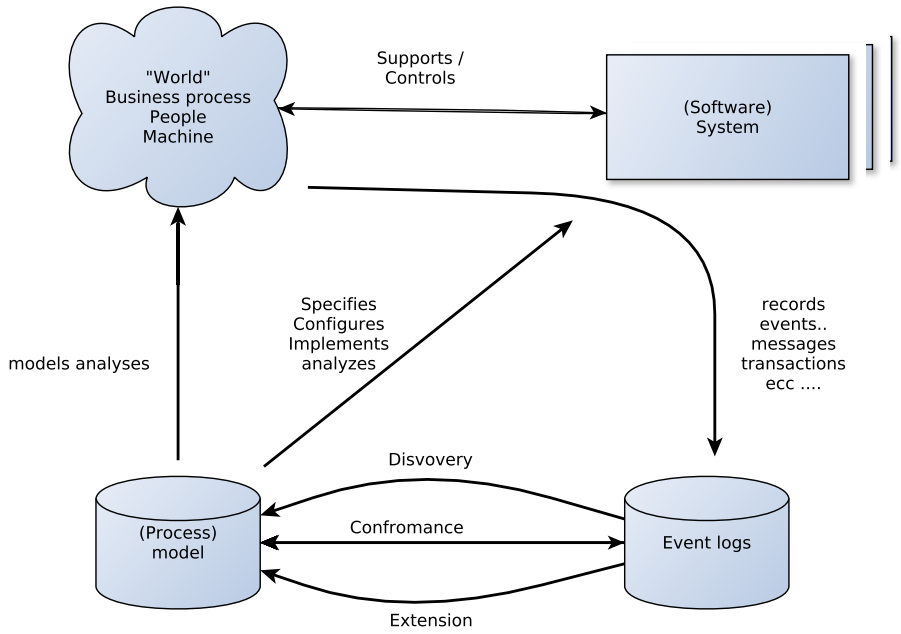


Figura 2.7: Process Mining

per supportare e controllare lo svolgimento dei processi di business, ed in cui sono previsti meccanismi di registrazione di event log, è possibile individuare tre tipologie principali di process mining:

- Scoperta: non esiste nessun modello a priori, basandoci soltanto sugli event logs è possibile inferire un possibile modello usando ad esempio l'algoritmo Alpha [19].
- Conformance: esiste un modello a priori ed è usato per verificare se la realtà corrisponde a ciò che è stato modellato. Viene quindi effettuato un confronto tra il modello del processo e le sue istanze. Le discrepanze identificate possono essere analizzate successivamente.
- Extension: esiste un modello a priori ed è esteso con un nuovo aspetto o prospettiva; l'obiettivo non è quello di controllare le discrepanze ma di arricchire il modello originario. Un esempio è l'estensione con dati sulla performance.

2.2.1 Analisi dei processi di business

Gli aspetti di process mining di interesse per il lavoro di tirocinio sono riconducibili alla conformance ed all'extension (figura 2.7). In particolare, tramite l'analisi dei processi di business e degli event log associati, è possibile ottenere informazioni utili sia per confrontare quanto modellato con la realtà (ed estendere eventualmente il modello), che per avere misure sulle performance del processo.

L'analisi di un processo di business avviene tramite un'analisi formale della rete di Petri che lo rappresenta. Partendo da un modello espresso in questo formalismo ed un log relativo alle esecuzioni di quel particolare processo (istanze di processo), vengono condotti fondamentalmente due tipi di analisi: *analisi di conformance* e *di performance*.

L'unità di base di cui un log si compone è l'evento. Un evento e può essere visto in modo astratto come una coppia $e = (a, t)$ che rappresenta un'azione a registrata dal sistema ed etichettata con un *timestamp* t. L'azione a che corrisponde all'evento e verrà denotata con $\alpha(e)$, mentre il timestamps t verrà denotato con $\phi(e)$. Gli eventi che appartengono alla stessa istanza di processo sono raggruppati in *tracce*. Formalmente una traccia T è una sequenza finita di eventi $T[1], ..., T[n]$ tale che $\phi(T[i]) \leq \phi(T[i+1])$ per ogni $i \in [1, n)$. Un log L è un insieme di tracce, ognuna di esse registra una serie di attività eseguite dal sistema durante un numero finito di esecuzioni di processo. In tutto questo elaborato verranno fatte le seguenti assunzioni:

- Tutte le tracce corrispondono a varie istanziazioni del medesimo processo.
- Per ogni azione esiste una transizione corrispondente nella rete di Petri; questa per semplicità ha lo stesso nome dell'azione.
- I log dei sistemi contengono solamente gli eventi correlati all'attivazione ed al competamento dell'attività come risulta dalla figura 2.6 in sezione 2.1.3.

2.2.2 Algoritmo Log Replay

Il log replay [11] [7] è lo strumento principale per l'analisi di un processo di business. A partire da un log L e dalla rete di Petri PN che modella il

processo, i risultati che l'algoritmo fornisce vengono usati per la valutazione della conformance e della performance del processo. Di seguito verranno presentati gli aspetti principali che lo caratterizzano.

Per ogni traccia nel log, il log replay inizia posizionando un token nella piazza iniziale della rete. Per ogni evento nella traccia viene effettuato lo scatto della corrispondente transizione in modo non bloccante. L'esecuzione non bloccante significa che se ad un certo punto dell'esecuzione dell'algoritmo lo scatto di una transizione non abilitata è necessario per il proseguimento del replay, i token mancanti vengono creati artificialmente e prendono il nome di *missing token*.

In generale, una rete di Petri contiene delle transizioni che non sono associate a nessuna azione registrata nei log: si tratta delle transizioni invisibili precedentemente descritte (figura 2.6). Esistono due modi possibili per trattare quest'ultime. In una gestione "lazy" l'esecuzione di una transizione invisibile $t1$ è ritardata finché non diventa necessaria all'abilitazione di una transizione visibile $t2$ corrispondente ad un evento della traccia considerata. Al contrario, in una gestione "eager", la tendenza è quella di eseguire le transizioni invisibili il prima possibile; questo significa che una transizione invisibile $t1$ viene eseguita non appena l'ultima transizione visibile $t2$ necessaria per abilitare $t1$ viene eseguita.

L'implementazione dell'algoritmo considerata in questo tirocinio effettua una visita di un grafo generato partendo dallo stato $\langle 0, TR, M, N\rangle$ dove: N è una rete di Petri con una marcatura M, una traccia $TR = \alpha.T$ ed un costo iniziale $c = 0$. Il grafo viene generato ricorsivamente applicando le regole (1,2,3) presentate di seguito. L'algoritmo ad ogni passo applica queste regole e sceglie lo stato con costo totale della visita minimo, finché non esaurisce tutti gli eventi della traccia TR.

Le regole utilizzano una funzione parziale t che dato un evento nella traccia, restituisce la transizione nella rete di Petri corrispondente all'evento. Le transizioni per le quali la funzione t non è definita sono le transizioni invisibili.

1.

$$\frac{M[t(\alpha)\rangle M'}{c, \alpha.T, M, N \xrightarrow{t(\alpha)} c, T, M', N} \tag{2.30}$$

Questa regola significa che se la rete di Petri ha una marcatura M in cui la transizione relativa all'evento in testa alla traccia, ovvero $t(\alpha)$, è abilitata, si effettua lo scatto della transizione con costo zero.

2.

$$\frac{M[\tau\rangle M'}{c, \alpha.T, M, N \xrightarrow{\tau} c + 10, \alpha.T, M', N} \tag{2.31}$$

Con questa regola si esprime il fatto che se la rete ha una marcatura M in cui è abilitata una transizione invisibile τ, viene effettuato lo scatto di questa transizione con un costo pari a 10.

3.

$$\frac{M + M_1[t(\alpha)\rangle M'}{c, \alpha.T, M, N \xrightarrow{t(\alpha)} c + 100, T, M', N} \tag{2.32}$$

Se la rete ha raggiunto una marcatura M in cui, aggiungendo altri token in alcune piazze, quindi aggiungendo un'altra marcatura M_1 si riesce ad abilitare la transizione $t(\alpha)$, viene effettuato lo scatto ad un costo pari a 100.

L'output del log replay relativo ad una traccia T può essere rappresentato con una lista ordinata di coppie (tr, i); ognuna di esse sta a significare che la transizione tr è stata eseguita allo scopo di 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.

I problemi di conformance possono essere rilevati tramite l'analisi dei token aggiunti artificialmente (missing tokens) e quelli che non sono stati consumati (remaining tokens) come vedremo nelle prossime sezioni, ma prima di fare ciò di seguito verrà presentato un esempio che illustra l'esecuzione del log replay su una traccia di processo.

Esempio: Consideriamo il processo P definito in notazione BPMN nella figura 2.8, applicando le regole di trasformazione viste in 2.1.1. La rete di Petri risultante è mostrata nella figura 2.9. Partendo dal log presentato in figura 2.10, l'output che si ottiene, applicando l'algoritmo log replay con una

gestione eager delle transizioni invisibili, consiste nella seguente lista ordinata di coppie:

$$R = \{(t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2), (t4, 2), (t6, 2), (C\_start, 3), (C\_complete, 4)\} \tag{2.33}$$

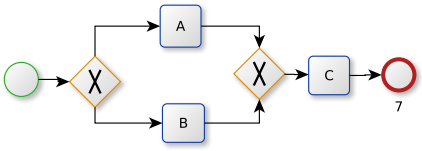


Figura 2.8: Modello BPMN del processo

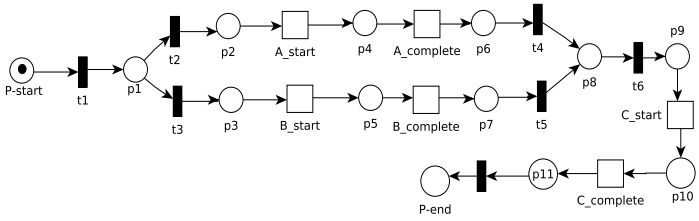


Figura 2.9: reti di Petri del processo

A #1 start 02.12.2011 9:30:50	A #1 complete 02.12.2011 10:30:00	C #1 start 02.12.2011 10:35:25	C #1 complete 02.12.2011 11:05:20

Figura 2.10: Esempio log conforme

2.2.3 Analisi di conformance

Partendo da un modello di processo espresso con una rete di Petri PN e da una traccia T che rappresenta un'istanza di quel processo, fare analisi di conformance significa sostanzialmente verificare se la traccia T soddisfa ciò che è stato modellato con la rete PN. In altre parole, si tratta di verificare se nella realtà il sistema si comporta come è stato prefissato, oppure esistono

casi in cui questo non accade.

I problemi di conformità, possono essere rilevati analizzando i token creati artificialmente dal log replay insieme ai token che non sono stati consumati. Il log presentato nella figura seguente fornisce un'esecuzione errata rispetto al processo presentato nell'esempio della sezione 2.2.2. La traccia contiene in-

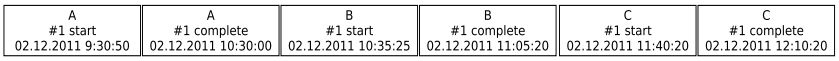


Figura 2.11: Esempio log non conforme

fatti l'esecuzione di due attività mutualmente esclusive (A e B). L'esecuzione del log replay su questa traccia si sintetizza in questi passi:

1. Estrazione del primo evento della traccia $T[1] = A\_start$: Per poter permettere lo scatto della transizione visibile A_start occorre abilitarla. Per fare ciò, in base alle regole (1,2,3) presentate nella sezione 2.2.2, esistono due modi: creare un token artificiale nella piazza $p2$ con costo 100, oppure eseguire lo scatto della transizione invisibile t_1 (che risulta abilitata nella marcatura attuale) ad un costo pari a 10. Dovendo scegliere la strada che comporta il minor costo, viene eseguita la transizione invisibile $t1$, consumato il token nella piazza iniziale e prodotto un altro nella piazza $p1$. L'algoritmo quindi, applicando un ragionamento sui costi analogo a quanto appena esposto, procede con l'esecuzione della transizione $t2$, ciò porta al consumo del token nella piazza $p1$ e alla produzione di un altro nella piazza $p2$. A questo punto la transizione A_start è abilitata, viene dunque eseguita, consumato il token e prodotto un altro nel piazza $p4$.
   La lista di output R risulta intanto: $R = (t1, 1), (t2, 1), (A\_start, 1)$

2. Estrazione dell'evento $T[2] = A\_complete$: La transizione visibile corrispondente è abilitata nella marcatura raggiunta finora dalla nostra rete, viene dunque effettuato lo scatto, consumato il token e prodotto un altro nella piazza $p6$.
   A questo punto si ha: $R = (t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2)$

3. Estrazione dell'evento $T[3] = B\_start$: si osserva che, nel ramo della rete su cui giace la transizione B_start, non è presente alcun token dal momento in cui questo è stato consumato dal branch alternativo per permettere lo scatto delle transizioni corrispondenti agli eventi estratti

ai passi precedenti. Tuttavia, per consetire la prosecuzione del log replay, viene creato un token artificiale nella piazza $p3$. Questo rende possibile lo scatto della transizione B_start, viene consumato il token artificiale e prodotto un altro nella piazza $p5$. Lo stato dell'output pertanto è:
$R = \{(t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2), (B\_start, 3)\}$.

4. Estrazione dell'evento $T[4] = B\_complete$: la transizione corrispondete risulta abilitata nella marcatura raggiunta dalla nostra rete, viene effettuato quindi lo scatto, consumato il token e prodotto un altro nella piazza $p7$. Lo stato della lista R diventa:

$$R = \{(t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2), (B\_start, 3), (B\_complete, 4)\} \tag{2.34}$$

5. Estrazione dell'evento $T[5] = C\_start$: per mimare questo evento viene effettuato lo scatto della transizione invisibile ed abilitata $t4$ (in realtà risulta abilitata anche $t5$, l'algoritmo sceglie in modo non deterministico quale fra i due eseguire), viene anche consumato il token presente nella piazza $p6$ e prodotto un altro nella piazza $p8$. Viene eseguita anche la transizione invisibile $t6$ consumato il token e prodotto un altro nella piazza $p9$. Raggiunto questo punto, la transizione visibile C_start risulta abilitata, si effettua lo scatto, si consuma il token e si produce un altro nella piazza $p10$. L'output risulta essere:

$$R = \{(t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2), (B\_start, 3), (B\_complete, 4), (t4, 5), (t6, 5), (C\_start, 5)\} \tag{2.35}$$

6. Estrazione dell'evento $T[6] = C\_complete$: la transizione visibile corrispondente risulta abilitata nella marcatura raggiunta dalla nostra rete, viene effettuato quindi lo scatto, consumato il token e prodotto un altro nella piazza $p11$. L'algoritmo si ferma a questo punto avendo consumato tutti gli eventi registrati nel log. L'output finale risulta essere:

$$R = \{(t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2), (B\_start, 3), (B\_complete, 4) (t4, 5), (t6, 5), (C\_start, 5), (C\_complete, 6)\} \tag{2.36}$$

Nell'esmepio, la mancanza del token nella piazza $p3$ (missing token) si interpreta come una esecuzione "errata" dell'evento per il quale è stato creato il token artificiale, ovvero dell'evento $T[3] = B\_start$. Infatti, dal modello di processo presentato sopra, si desume l'esistenza di una scelta all'inizio del processo in seguito alla quale viene eseguita l'attività A *oppure* (nel senso di un or esclusivo) l'attività B. I token che non sono stati consumati invece (remaining token), come quello nella piazza $p7$, viene interpretato come una "esecuzione interrotta".

Proiezione dei risultati sul modello BPMN

Facendo alcune considerazioni sul comportamento dell'algoritmo log replay su una rete di Petri ottenuta con una trasformazione da un modello espresso in BPMN, è possibile trarre alcune metriche di conformità che valgono in generale e che possono essere proiettate sul modello BPMN di partenza.

- L'algoritmo 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 figura 2.6 che riporta le regole di mapping, queste piazze sono del tipo $P(x, T)$ e Pt.

- 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, l'algoritmo 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:

  - Piazze che si trovano nel post-set di una transizione visibile.
  - Piazze che si trovano nel post-set di una transizione invisibile che produce più token

  Facendo riferimento alla figura 2.6 queste piazze sono: Pt, $P(T, y)$, $P(F1, y1)$ e $P(F1, y2)$, ovvero quelle coivolte nella modellazione di una singola attività BMPN e del fork gateway.

In base alle osservazioni fatte sopra è possibile dare le seguenti interpretazioni ai token mancanti ed a quelli rimanenti:

- 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 definito invece come "un completamento mancante". Infine, un token rimanente in $P(T, y)$ viene interpretato 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.

Da queste informazioni, è possibile apportare delle modifiche al modello di processo, oppure andare ad esplorare quali sono le cause che portano ad esecuzioni non conformi come verrà descritto nel capitolo 5.

2.2.4 Analisi di performance

Partendo da un log e sfruttando in particolare i timestamp riportati dai vari eventi registrati, la tecnica del log replay può essere impiegata per misurare le prestazioni del sistema. 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 dal momento che tali token non possono avere informazioni temporali. Durante il log-replay possono essere calcolate, per ogni traccia, le seguenti metriche riferite ad ogni piazza:

- Il tempo di soggiorno in una piazza p: indicato con $tsg(p)$, è dato dall'intervallo di tempo tra l'arrivo e la partenza di un token.

- Il tempo di sincronizzazione: indicato con $tsc(p)$, è dato dall'intervallo di tempo tra l'arrivo di un token in una piazza, e l'abilitazione di una transizione nel suo post-set.

- Il tempo di attesa: indicato con $ta(p)$, è dato dall'intervallo di tempo tra l'attivazione e l'escuzione di una transizione nel post-set della piazza.

In generale vale la seguente relazione:

$$tsg(p) = tsc(p) + ta(p) \tag{2.37}$$

Con riferimento alla rete di Petri in figura 2.9 presentata nella sezione 2.2.2, ed al log in figura 2.12, il risultato del log replay è il seguente:

$$R = \{(t1, 1), (t2, 1), (A\_start, 1), (A\_complete, 2), (t4, 2), (t6, 2), (C\_start, 3), (C\_complete, 4)\} \tag{2.38}$$

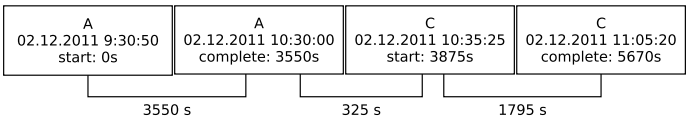


Figura 2.12: Log per la performance

La valutazione delle performance viene eseguita inserendo un token nella piazza iniziale P_start al tempo 0:

1. Le transizioni invisibili $t1$ e $t2$ vengono eseguite per prime e sono entrambe associate all'evento $T[1]$. Quindi al tempo $\phi(T[1]) = 0$ il token nella piazza iniziale viene consumato e ne viene prodotto un altro nella piazza $p1$. La transizione $t2$ consuma il token in $p1$ e ne produce uno in $p2$ attivando una delle due possibili attività. Vale quindi: $ta(p1) = 0$.

2. La transizione visibile A_start viene anch'essa eseguita al tempo $\phi(T[1])$ consumando il token in $p2$ e producendo un token in $p4$. Anche in questo caso si ha: $ta(p2) = 0$.

3. Al tempo $\phi(T[2]) = 3550s$ viene eseguita la transizione visibile $A\_complete$ che consuma il token in $p4$ e ne produco uno in $p6$. Vale: $ta(p4) = 3550s - 0s = 3550s$.

4. Sempre al tempo $\phi(T[2] = 3550)$ viene eseguita la transizione invisibile $t4$ consumando il token in $p6$ e producendo uno in $p8$. Si ha: $ta(p6) = 3550s - 3550s = 0s$.

5. Sempre al tempo $\phi(T[2]) = 3550s$, viene eseguita la transizione $t6$ che consuma il token in $p8$ e ne produce uno in $p9$. Anche qui abbiamo che: $ta(p8) = 3550s - 3550s = 0s$.

6. Al tempo $\phi(T[3]) = 3875s$ viene eseguita la transizione C_start che consuma il token in $p9$ e produce un token in $p10$. Vale: $ta(p9) = 3875s - 3550s = 325s$.

7. Al tempo $\phi(T[4]) = 5670s$ avviene l'esecuzione della transizione $C\_complete$ consumando il token in $p10$ e producendone uno in $p11$. Vale: $ta(p10) = 5670s - 3875s$.

Si osservi che il tempo di attesa nella piazza $p4$ fornisce il tempo di esecuzione dell'attività BPMN "A" per l'istanza di processo rappresentata dal log in figura 2.12, mentre il tempo di attesa in $p10$ rappresenta il tempo di esecuzione dell'attività "C". Per quanto riguarda invece il tempo di sincornizzazione, questo risulta nullo per tutte le piazze della nostra rete, di conseguenza il tempo di soggiorno è uguale al tempo di attesa per tutte le piazze.

Proiezione dei risultati sul modello BPMN

Anche per l'analisi di performance, facendo delle osservazioni a partire anche dall'esempio presentato di sopra, è possibile dedurre alcune metriche di performance valide in generale e che possono essere proiettate sul modello BPMN di partenza:

- Nella gestione "eager" discussa in precedenza, se una transizione invisibile è abilitata, questa viene immediatamente eseguita. Perciò, i tempi di attesa delle piazze che hanno solo transizioni invisibili nel loro post-set sono sempre nulli. Ne consegue che le piazze che possono avere tempi di attesa diversi dallo zero sono $P(x, T)$ e Pt, cioè piazze usate per tradurre le attività BPMN.

- Il tempo di sincornizzazione di una piazza può essere maggiore di zero solo se almeno una transizione nel suo post-set dipende da un'altra piazza. Ne consegue che le piazze che possono avere tempi di sincronizzazione diversi dallo zero sono solo: $P(x1, j1)$ e $P(x2, j1)$, ovvero quelle coinvolte nella modellazione del join gateway.

In base alle considerazioni fatte, è possibile dare la seguente interpretazione dei tempi calcolati in modo che possano essere proiettate su un diagramma BPMN.

- Per ogni attività T, il tempo di esecuzione è dato da $ta(pt)$, mentre il tempo di attivazione è dato da $ta(P(x, T))$.

- Per ogni ramo concorrente ($i \in \{1, 2\}$) racchiuso da una fork gatweay ($F1$) ed un join gateway ($J1$):
  - Il tempo di sincronizzazione è dato da $tsc(P(xi, J1))$, ovvero il tempo impiegato per attendere il completamento delle attività concorrenti.
  - I tempi di esecuzione (di seguito indicati con $ta(F1i)$) sono la somma di tutti i tempi di soggiorno di tutte le piazze raggiungibili attraverso il grafico a partire da $P(Fi, yi)$ senza avere visitato $P(xi, Ji)$. Si noti che $tsc(P(xi, J1) + ta(F1i)$ è costante per ogni ramo di un fork/join. Questa misura viene chiamata come "il tempo di esecuzione" del fork, ed è indicato con: $tsg(F1, J1)$.

(2.39)

Capitolo 3

Data mining: il problema della classificazione

In questo capitolo vengono introdotti i concetti e le basi teoriche del data mining, relativi in particolare alla classificazione, che sono stati utili al raggiungimento dello scopo finale del tirocinio. Per ulteriori approfondimenti si rimanda a [17].

Il data mining consiste in un insieme di tecniche e metodologie che hanno l'obiettivo di estrarre, in modo non banale, informazioni potenzialmente utili, implicite e precedentemente sconosciute, partendo dai dati. Questa disciplina nasce dall'esigenza di rendere utile quella enorme quantità di dati che viene raccolta e registrata quotidianamente dai sistemi informatici (dati derivanti dal web, dalle transazioni bancarie e molto altro). Un altro fattore che ha contribuito all'evoluzione di questa materia è lo sviluppo di tecniche di analisi nel campo dell'apprendimento automatico e dell'intelligenza artificiale.

I metodi del data mining possono essere classificati secondo questi due criteri:

- Metodi per la predizione: a partire dai dati già esistenti, questi metodi permettono di fare un'analisi per costruire modelli in grado di predire i valori sconosciuti o futuri di alcune variabili.
- Metodi descrittivi: sempre in base ad un insieme di dati già esistenti, questi metodi cercano invece di scoprire gli schemi (pattern) che caratterizzano i dati di partenza.

Le tecniche sviluppate da questa disciplina sono molteplici, alcune di tipo predittivo altre invece di tipo descrittivo. Durante il tirocinio la tecnica maggiormente esplorata è stata la classificazione, questa risulta essere sia un metodo predittivo che descrittivo.

3.1 Classificazione

La classificazione consiste nell'assegnare un oggetto ad una fra diverse categorie predefinite. E' un problema comune a diverse applicazioni: ad esempio il rilevamento dei messaggi di spam in base all'intestazione ed il contenuto di una e-mail, la ricerca di cellule maligne e benigne in base alle scansioni di risonanza magnetica, l'identificazione delle galassie in base alle loro forme.
In questa sezione vengono introdotti i concetti principali della classificazione, con un maggiore riguardo per gli alberi di decisione che sono stati usati nel tirocinio.

3.1.1 Concetti preliminari

In un problema di classificazione, i dati sono una collezione di record. Ogni record, detto anche istanza, è caratterizzato da una tupla $(\mathbf{x}, y)$, dove $\mathbf{x}$ è un insieme di attributi, mentre y è un attributo speciale denominato "l'attributo target" ed ha la funzione di indicare la classe a cui l'istanza appartiene. La tabella 3.1 rappresenta un insieme di dati usato per decidere (classificare) se uscire a giocare o meno in base alle condizioni meteo. Gli attributi dell'esempio rappresentano proprietà meteorologiche: la condizione del cielo, la temperatura, l'umidità ed il vento. Gli attributi possono essere sia di tipo discreto (come ad esempio il cielo) che di tipo continuo (la temperatura). L'attributo target invece deve essere sempre di tipo discreto.

La classificazione ha l'obiettivo di insegnare ad una funzione target f ad associare ad ogni insieme di attributi $\mathbf{x}$, una etichetta di classe predefinta y. La funzione target f viene comunemente chiamata: "modello di classificazione" o semplicemente "classificatore".

Un modello di classificazione è utile per i seguenti obiettivi:

- Descrizione: fornire uno strumento di esplorazione per distinguere oggetti appartenenti a categorie differenti. Con riferimento all'insieme di dati presentato in figura 3.1, può risultare utile capire in quali casi è possibile uscire a giocare o meno.

cielo	temperatura	umidità	vento	giocare
sole	85	85	no	no
sole	80	90	si	no
coperto	83	86	no	sì
pioggia	70	96	no	sì
pioggia	68	80	no	sì
coperto	64	65	si	no
sole	72	95	no	no
sole	69	70	no	sì
pioggia	75	80	no	sì
sole	75	70	sì	sì
coperto	72	90	sì	sì
coperto	81	75	no	sì
pioggia	71	91	sì	no

Tabella 3.1: Esmepio data set

- Predizione: predire qual'è la classe di appartenenza di nuovi record che si presentano. Un classificatore in questo contesto è da immaginare come una scatola nera in grado di restituire automaticamente in output un'etichetta di classe quando gli viene passato un record non classificato. Supponiamo di avere un record descritto dalla figura 3.2.

cielo	temperatura	umidità	vento	giocare
sole	70	90	sì	?

Tabella 3.2: Record non classificato

Potremmo usare il classificatore costruito a partire dai dati presentati in tabella 3.1 per determinare se, nel caso le condizioni meteo fossero come descritto dal record in 3.2, fosse possibile andare a giocare o meno.

3.1.2 Approcci generali per i problemi di classificazione

Una tecnica di classificazione è un approccio sistematico volto alla costruzione di un classificatore a partire dai dati di input (detti anche data set). Queste tecniche impiegono strumenti che traggono la loro origine dalla disciplina dell'apprendimento automatico, come ad esempio gli alberi di decisione e le reti neurali. Tali tecniche si basano su un algoritmo di apprendimento al fine di identificare un modello di classificazione che meglio riconosce la

relazione tra gli attributi ed il target. Il modello di classificazione ottenuto dovrebbe sia adattarsi ai dati di input che, allo stesso tempo, riuscire a predire correttamente qual'è l'etichetta di classe di record nuovi ad esso sconosciuti. Perciò, un obiettivo chiave di un algoritmo di apprendimento è la costruzione di modelli di classificazione con una buona capacità di generalizzazione, ovvero modelli che predicono con accuratezza la classe di record nuovi.

In generale, ad un algoritmo di apprendimento vengono forniti due insiemi di dati:

- Training set: è un insieme di dati su cui il classificatore verrà addestrato. Esso consiste in un insieme di record di cui l'etichetta di classe è nota.
- Test set: è un insieme di record separato di cui il classificatore ignora la classe di appartenenza. Su di esso verrà applicato, per motivi di verifca, il modello ottenuto dall'addestramento.

Valutazione di performance di un classificatore

La performance di un classificatore dipende dal numero dei record del test set che sono stati classificati correttamente. Per analizzare la performance viene unsata una tabella che prende il nome di **matrice di confusione**. La tabella 3.3 illustra la matrice di confusione relativa ad un problema di classificazione binaria (ovvero dove le etichette di classe sono di tipo binario).

Classe attuale	Classe predetta	
	Casse = 1	Classe = 0
Classe = 1	f_{11}	f_{10}
Classe = 0	f_{01}	f_{00}

Tabella 3.3: Matrice di confusione per una classificazione binaria.

Ogni entrata f_{ij} in questa tabella denota il numero dei record appartenenti alla classe i che sono stati classificati come appartenenti alla classe j. In base alla matrice di confusione, il numero totale delle predizioni corrette fatte dal classificatore è dato da $(f_{00} + f_{11})$, mentre il numero delle predizioni errate è dato dalla somma $(f_{10} + f_{01})$.

Sebbene la matrice di confusione apporti informazioni significative per valutare le performance di un classificatore, riassumere queste informazioni in un unico indice facilita la valutazione, specialmente in caso di confronto tra classificatori differenti. Per queste ragioni, viene definito un indice di **accuratezza** calcolato in questo modo:

$$Accuratezza = \frac{Numero\ delle\ predizioni\ corrette}{Numero\ totale\ delle\ predizioni} = \frac{f_{11} + f_{00}}{f_{11} + f_{00} + f_{01} + f_{10}} \tag{3.1}$$

Equivalentemente, le performance di un modello possono essere misurate anche in termini del **tasso di errore**, che è definito come:

$$Tasso\ di\ errore = \frac{Numero\ delle\ predizioni\ errate}{Numero\ totale\ delle\ predizioni} = \frac{f_{10} + f_{01}}{f_{11} + f_{00} + f_{01} + f_{10}} \tag{3.2}$$

Molti algoritmi di apprendimento tendono a costruire il modello che abbassa il tasso di errore, o equivalentemente massimizza l'indice di accuratezza.

3.1.3 Alberi di decisione

Gli alberi di decisione sono uno strumento di classificazions semplice e di ampio utilizzo. Per illustrarne il funzionamento, prendiamo in considerazione il problema di classificazione presentato nelle sezione 3.1.1. Supponiamo che in una determinata giornata caratterizzata da alcune condizioni meteorologiche, vogliamo decidere se andare a giocare o meno. Un approccio consiste nel porre una serie di domande sulle condizioni del tempo. La prima domanda potrebbe riguardare la condizione del cielo: se questo è piovoso allora non potremmo uscire a giocare. In caso contrario, possiamo controllare la temperatura e rinunciare al gioco nelle giornate troppo calde.

L'esempio illustrato prima ci mostra come potremmo risolvere un problema di classificazione, facendo una serie di domande riguardanti i valori degli attributi del record che si vuole classificare. Ogni volta che otteniamo una risposta, viene posta un'altra domanda fintanto che non si ottiene l'etichetta di classe di appartenza del record. Le domande e le relative risposte possono essere organizzate sotto forma di un albero di decisione, questo è una struttura gerarchica che consiste in un insieme di nodi e di archi diretti. La figura 3.1 illustra l'albero di decisione relativo all'esempio.

Un albero di decisione ha tre tipi di nodi:

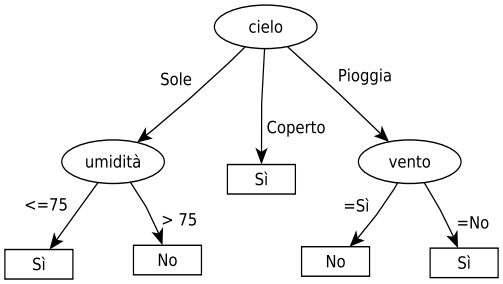


Figura 3.1: Albero di decisione

- Nodo radice: non ha nessun arco in ingresso, mentre ne ha uno o più in uscita.
- Nodi interni: ognuno di questi ha un solo arco in ingresso, mentre ne può avere più di uno in uscita.
- Nodi terminali (foglie): ognuno di essi ha esattamente un arco in entrata e nessuno in uscita.

Ad ogni foglia è assegnata un'etichetta di classe. I nodi non terminali, ovvero la radice ed i nodi interni, contengono dei test condizionali sugli attributi che permettono di separare i dati che hanno caratteristiche differenti. Ad esempio il nodo radice nella figura 3.1 utilizza l'attributo "cielo" per separare i record in base al valore che quest'ultimo assume; se il cielo è coperto allora è sempre possibile andare a giocare a prescindere dai valori assunti dagli altri attributi. Negli altri casi (pioggia e sole) occorre fare ulteriori test sugli altri attributi, di conseguenza vengono creati altri nodi intermedi sui quali vale un ragionamento analogo a quello appena esposto.

Una volta costruito l'albero di decisione, classificare un nuovo record diventa semplice ed immediato: partendo dal nodo radice, vengono effettuati i test condizionali sul record, ed in base al risultato ottenuto, si segue l'arco appropriato. Ad ogni passo, questa operazione porta su un altro nodo intermedio, in corrispondenza del quale un altro test viene effettuato. Quando viene raggiunto un nodo finale, la corrispondente etichetta diventa la classificazione del record considerato.

Differenti algoritmi, di differente complessità, sono stati proposti in letteratura per costruire un albero di decisione a partire da un dataset. Solitamente questi impiegano strategie di tipo greedy, effettuando ad ogni passo

una decisione su quale attributo scegliere per partizionare i dati. Uno di questi è l'algoritmo di Hunt che rappresenta la base di una serie di altri algoritmi per la costruzione di alberi di decisione (ad esempio il C4.5 [16] che è stato impiegato durante il tirocinio).

L'algoritmo di Hunt: [17] l'albero di decisione viene costruito in modo ricorsivo partizionando il training set in insiemi sempre più piccoli. Sia D_t l'insieme dei record del training set associati al nodo t, e $y = \{y_1, y_2, ..., y_c\}$ l'insieme delle etichette di classe.

La seguente sequenza di passi definisce ricorsivamente l'algoritmo di Hunt:

- Passo 1: Se tutti i record in D_t hanno la stessa etichetta di classe y_t, allora t diventa un nodo terminale etichettato con y_t.
- Passo 2: Se D_t contiene record che appartengono a diverse classi, un **test condizionale** su un attributo viene selezionato per partizionare i record in sottoinsiemi più piccoli [1]. In corrispondenza di ogni possibile risposta al test condizionale, viene creato un nodo ed i record in D_t vengono distribuiti ai nodi figli in base alla risposta del test. L'algoritmo viene quindi applicato ricorsivamente ad ogni nodo figlio.

Così come è stato descritto, l'algoritmo di Hunt non prevede alcuni casi che vanno trattati come descritto di seguito:

- E' possibile che alcuni nodi creati al passo 2 risultino vuoti. Questa situazione si genera quando nel traning set non esiste nessun record con una combinazione di attributi corrispondente a quella del nodo creato. In questo caso il nodo viene dichiarato una foglia con un'etichetta di classe uguale a quella più ricorrente nei record associati al nodo padre.
- Al passo 2, se tutti i record in D_t hanno identici valori negli attributi ad eccezione dell'etichetta di classe, allora non è possibile partizionare questi record ulteriormente. In questo caso, il nodo viene dichiarato una foglia con un'etichetta di classe pari all'etichetta più ricorrente nei record associati ad esso.

[1] Esistono varie misure per la selezione del migliore attributo per la partizione. Si rimanda a [17] per maggiori informazioni.

3.1.4 Analisi dei risultati

Gli errori commessi da un classificatore si possono dividere in due categorie:

- Errori di training: è dato dal numero delle classificazioni errate effettuate sul training set.
- Errori di generalizzazione: è dato dal numero atteso di classificazioni errate effettuate dal modello su un insieme di dati precedentemente sconosciuto.

Un buon classificatore non deve solamente adattarsi al traning set sul quale viene addestrato, ma deve anche avere buone capacità di classificare record ad esso ignoti. In altre parole, un buon classificatore deve avere un errore di training basso così come deve essere basso il suo errore di generalizzazione.

Un classificatore che si adatta eccessivamente al training set, ma non è in grado di generalizzare, si dice che è affetto da **overfitting**. Più precisamente, questo problema si verifica quando il classificatore risulta avere un errore di training basso ma un errore di generalizzazione più alto rispetto ad un classificatore con un errore di traning più alto. E' possibile parlare anche di situazione di **underfitting**, questa si riscontra qualora venissero forniti al modello pochi dati per l'addestramento. In questo ultimo caso si riscontrano degli errori alti sia di training che di generalizzazione, il motivo sta nel fatto che il classificatore non è ancora riuscito a capire la vera struttura dei dati.

Negli alberi di decisione, l'errore di training può essere ridotto incrementando la complessità del modello. Infatti, è possibile espandere l'albero fintanto che non classifichi, con buoni risultati, i record del training set. Tuttavia, per la problematica esposta sopra, questa espansione non sempre si traduce in una riduzione dell'errore di generalizzazione.

L'overfitting è una problematica seria che si riscontra facilmente nell'applicazione di queste tecniche. Nei prossimi paragrafi vengono presentati due cause principali di questo problema.

Overfitting dovuto alla presenza di rumore

Con riferimento all'esempio mostrato in 3.1.1, immaginiamo che tra i dati di training ci sia anche un recod $R = \langle Cielo = sole, Temperatura = 80, Umidita' = 80, Vento = si, Giocare = si \rangle$. Ciò potrebbe portare l'algoritmo di addestramento, che non ha nessun modo di sapere che R presenta

qualche dato spurio, ad espandere maggiormente qualche ramo dell'albero per potere tenere in considerazione anche quei casi che il record R potenzialmente rappresenta. In questo modo si ottiene un modello nuovo con una complessità maggiore rispetto a quello illustrato in fig 3.1. Ovviamente il nuovo albero classificherà i record del training set molto meglio rispetto al semplice albero in fig. 3.1. D'altro canto però, dal momento che la complessità del nuovo modello è una conseguenza del rumore nei dati, è ragionevole pensare che la capacità di generalizzazione del vecchio albero è superiore a quella del nuovo classificatore.

Overfitting dovuto a carenza nei record per l'addestramento

I modelli costruiti in base a pochi record sono anch'essi soggeti ad overfitting. Infatti l'algoritmo di costruzione dell'albero così com'è stato descritto procede con la separazione dei record adattandosi alle situazioni rappresentate dai record del traning set. In questo caso, l'errore di training è basso, ma lo stesso non può dirsi per quello di generalizzazione visto che la capacità del modello di classificare i record sconosciuti è sicuramente più bassa di un classificatore addestrato su un training set più rappresentativo.

Gestione dell'overfitting per alberi di decisione

Le soluzioni per affrontare il problema di overfitting negli alberi di decisione possono essere di due approcci:

- Forzare la terminazione della procedura di costruzione dell'albero prima del suo completamento; si cerca di impedire all'abero di adattarsi eccessivamente al training set.
- Riduzione della complessità dell'albero al termine del processo di costruzione senza forzare la terminazione (potatura dell'albero).

Il vantaggio del primo approccio è quello di evitare un eccessivo adattamento ai dati del training set, tuttavia, risulta alquanto complesso determinare una soglia in corrispondenza della quale forzare la terminazione del processo di costruzione; una soglia troppo altra potrebbe portare alla manifestazione dell'underfitting, mentre una troppo bassa potrebbe non essere sufficiente per evitare l'overfitting. Il secondo approccio tende a dare dei risultati migliori rispetto al primo, ma si caratterizza da una maggiore complessità computazione dal momento che deve mettere in atto delle tecniche aggiuntive per

ridurre la dimensione dell'albero.

Capitolo 4

Tecnologie impiegate

In questo capitolo sono illustrati tutti gli strumenti software che sono stati impiegati nel tirocinio. La trattazione è suddivisa in due parti: la sezione 4.1 spiegherà la strumentazione utilizzata per il process mining, mentre la sezione 4.2 tratterà gli strumenti impiegati per mettere in atto le tecniche del data mining.

4.1 Process mining

In questo tirocinio, la principale tecnologia utilizzata per le tecniche di process mining è il framework ProM 6 [14].

4.1.1 Framework ProM 6

ProM è un framework open-source ed estendibile che supporta una vasta gamma di tecniche per il pocess mining e l'analisi dei process, oltre ad essere una piattaforma utile per la ricerca e la sperimentazione di nuovi algoritmi integrabili sotto forma di plug-ins. E' una piattaforma indipendente dal sistema operativo interamente implementata in linguaggio Java ed è scaricabile da [15]. ProM 6 fornisce un'interfaccia grafica user-friendly (fig. 4.1) tramite la quale si ha accesso agli stumenti di analisi.

L'interfaccia di ProM 6 si compone principalmente da tre tab, ognuo di essi presenta alcune funzionalità che sono esposte di seguito.

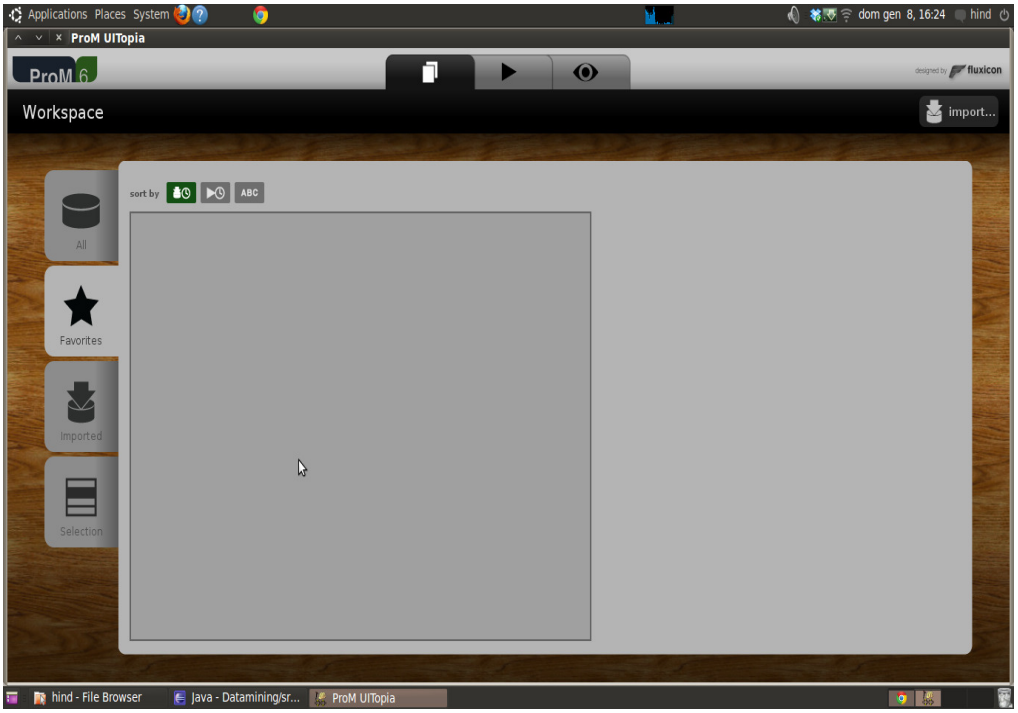


Figura 4.1: Vista workspace di ProM 6.

Vista Workspace

In sostanza in questo tab si fornisce una vista su tutte le risorse. In ProM una risorsa può essere importata dal file system, oppure prodotta come risultato di esecuzione di alcune azioni su altre risorse, come ad esempio l'esecuzione di un plug-in su una risorsa importata. Un esempio di risorsa potrebbe essere un file di log oppure un modello di processo.

In particolare la vista workspace permette di:

1. Importare una risorsa da un file.
2. Visualizzare tutte le risorse presenti in memoria in un dato istante (sia importate che risorse risultato).
3. Selezionare una risorsa per visionarne i dettagli, per la rimozione, per visionarne le risorse padre, ovvero risorse utilizzate nella sua creazione, oppure, analogamente, per visionare le risorse figlio, cioè risorse per le quale la risorsa selezionata è un padre.

4. Esportare una risorsa nel file system.

5. Visualizzare esclusivamente le risorse importate.

6. Visualizzare le risorse etichettate come "preferite".

Vista Action

Sostanzialmente, questa vista visualizza le azioni che si possono eseguire sulle risorse. Per esempio un'azione è l'esecuzione di una certa tecnica di process mining su un file di log, oppure l'esecuzione di un plug-in che implementa le tecniche di mapping discusse in 2.1.3 su un modello BPMN.

In particolare questa vista permette di:

1. Visualizzare quali sono le azioni che si possono eseguire, e quelle che sono in fase di esecuzione.

2. Visualizzare, data un'azione selezionata, qual'è il tipo della risorsa/e richieste in input, ed il tipo della risorsa/e rilasciata/e in output.

3. Data una o più risorse in input, visualizzare le uniche azioni compatibili che possono essere eseguite su quell'insieme di risorse.

4. Data un'azione selezionata, specificare quali sono le risorse sulle quali si vuole fare l'esecuzione. Le risorse ammissibili sono quelle caricate in memoria e gestibili dalla vista workspace come spiegato sopra.

5. Dare il comando di esecuzione per un'azione per cui le risorse di input sono già state individuate.

6. Dare il comando di reset con il quale si ristabilisce la situazione iniziale, ovvero vengono cancellate tutte le risorse di input individuate per ciascuna azione.

Vista View

Questa vista permette di visualizzare una risorsa del workspace per la quale esiste una rappresentazione grafica. Ad esempio è possibile visualizzare il grafo che rappresenta una rete di Petri.

In particolare, le funzionalità fornite da questa vista sono:

1. Visualizzare la rappresentazione grafica di una risorsa del workspace.

2. Aggiornare la rappresentazione di una risorsa visualizzata.

3. Eseguire alcuni comandi come: la stampa, l'impostazione dello zoom.

4. Esportare una rappresentazione sul file system con uno dei formati standard (eps, png, ecc..).

5. Visualizzare più rappresentazioni di risorse diverse contemporaneamente.

6. Rimuovere una rappresentazione visualizzata di qualche risorsa.

Come anticipato in precedenza ProM 6 supporta i modelli di processi di business e li considera delle risorse. I due formalismi di modellazione impiegati sono stati le reti di Petri e BPMN. Le prime sono descritte con lo standard PNML, mentre i diagrammi BPMN sono descritti con lo standard XPDL. ProM supporta inoltre anche i file di log in due formati diversi: XES ed MXML.

4.1.2 Il formato PNML

Petri Net Markup Language (PNML) [1] è un formato di interscambio basato su XML per le reti di Petri. Inizialmente fu concepito come un formato di file per la versiona Java di "Petri Net Kernel", un'infrastruttura per l'implementazione di strumenti di analisi sulle reti di Petri, tuttavia, data la sua semplicità, attualmente sta dando un grande contributo nel tentativo di definire uno standard basato su XML. Una caratteristica principale di questo formato è l'estendibilità; infatti permette di descrivere sia reti di Petri con caratteristiche generali, che reti cha hanno delle specificità particolari.

Dal momento che ProM 6 supporta il formato PNML, è possibile costruire una rete di Petri con qualsiasi tool che genere file in questo formato e poterlo usare come risorsa all'interno del framework. Durante il tirocinio è stato fatto uso dell'editor *Woped* liberamente scaricabile da [18].
Per avere un'idea più concreta, di seguito è presentata la descrizione PNML della semplice rete in figura 4.2

```
<pnml xmlns="http://www.example.org/pnml">
  <net id="n1">
    <place id="p1">
    <arc id="a1" source="p1" target="t1">
    <transition id="t1">
    <arc id="a2" source="t1" target="p2">
```

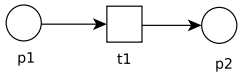


Figura 4.2: Una semplice rete di Petri

```
    <place id="p2">
  </net>
</pnml>
```

Listing 4.1: Descrizione PNML della rete in figura 4.2

4.1.3 Il formato XPDL

XML Process Definition Language (XPDL) può essere visto come il formato di serializzazione di BPMN. Esso è stato definito per dare delle indicazioni su come un diagramma BPMN deve essere memorizzato in modo che sia interscambiabile tra i vari tool che possono farne uso.

XPDL è stato standardizzato dalla "Workflow Management Coalition" [10], un consorzio costituito per definire gli standard di interoperabilità nel campo del workflow management. Esso fornisce un formato in grado di rappresentare tutti gli aspetti di un diagramma BPMN, sia da un punto di vista grafico che semantico.

Anche in questo caso, visto che PROM 6 supporta il formato XPDL, è possibile costruire un modello di processo con qualsiasi tool che memorizza il diagramma in questo formato ed usarlo nel framework come risorsa. Di questi tool ne esistono vari, durante il tirocinio è stato impiegato *sketchpad* [2].

4.1.4 Il formato XES

Extensible Event Stream (XES) è un formato che in questa sezione verrà descritto più in dettaglio ripetto agli altri formati data la sua importanza nello sviluppo dei plug-in realizzati. ProM 6 supporta anche il formato MXML (Mining XML) per gli event log, ma per gli scopi del tirocinio è stato scelto il formato XES perché, a differenza di MXML, fornisce la possibilità di rappresentare il tipo del dato registrato negli event log.

Nella pratica, gli event log possono assumere una moltitudine di forme. Ogni sistema che include al suo interno un meccanismo di registrazione di log può sviluppare un suo formato. XES, come è specificato in [9], è uno standard basato su XML che definisce con un approccio generale un formato per permettere lo scambio di file di log tra diversi tool. E' stato progettato al fine di soddisfare le seguenti caratteristiche:

- Semplicità: sia nel rappresentare l'informazione che nel parsing e nella generazione.
- Flessibilità: anche se è stato progettato per rappresentare gli event log nell'ambito del process mining, dev'essere di facile utilizzo in qualsiasi sistema che prevede meccanismi di registrazione di log, indipendentemente dalle piattaforme e dagli strumenti impiegati.
- Estendibilità: dev'essere facile aggiungere nuove estensioni mantenendo la compatibilità qualora ciò fosse necessario (ad esempio se lo si volesse impiegare in altri ambiti).
- Espressività: anche se è uno standard generico, gli event log espressi in formato XES non devono perdere informazione utile.

Un event log descritto nel formato XES, come si può desumere dal metamodello definito in [9] e presentato in figura 4.3 , ha una struttura gerarchica cui elementi di base sono:

- Log: è l'elemento che si trova a livello più alto e contiene tutti gli eventi registrati relativi ad un singolo processo.
- Trace: un oggetto trace descrive l'esecuzione di una specifica istanza del processo in questione. Un elemento log contiene un numero arbitrario di oggetti trace.
- Event: un oggetto event rappresenta lo stato di un'attività osservata durante l'esecuzione del processo. Ogni trace contiene un numero arbitrario di oggetti event.
- Attribute: un oggetto attribute è descritto da una chiave ed un valore e specifica informazioni relative all'elemento padre (la relazione di parentela è una conseguenza della struttura XML del formato). In un event log, tutti i dati sono memorizzati negli oggetti attribute dato che un oggetto di tipo log, trace oppure event non contiene nessuna

informazione. Un elemento attribute può avere a sua volta un numero arbitrario di elementi attribute che ne descrivono le proprietà. Un attribute può essere di tipo: string, date, int, float o boolean.

Gli altri elementi del metamodello (extensions e classifier) non sono descritti perché non sono rilevanti per gli scopi di questa relazione.

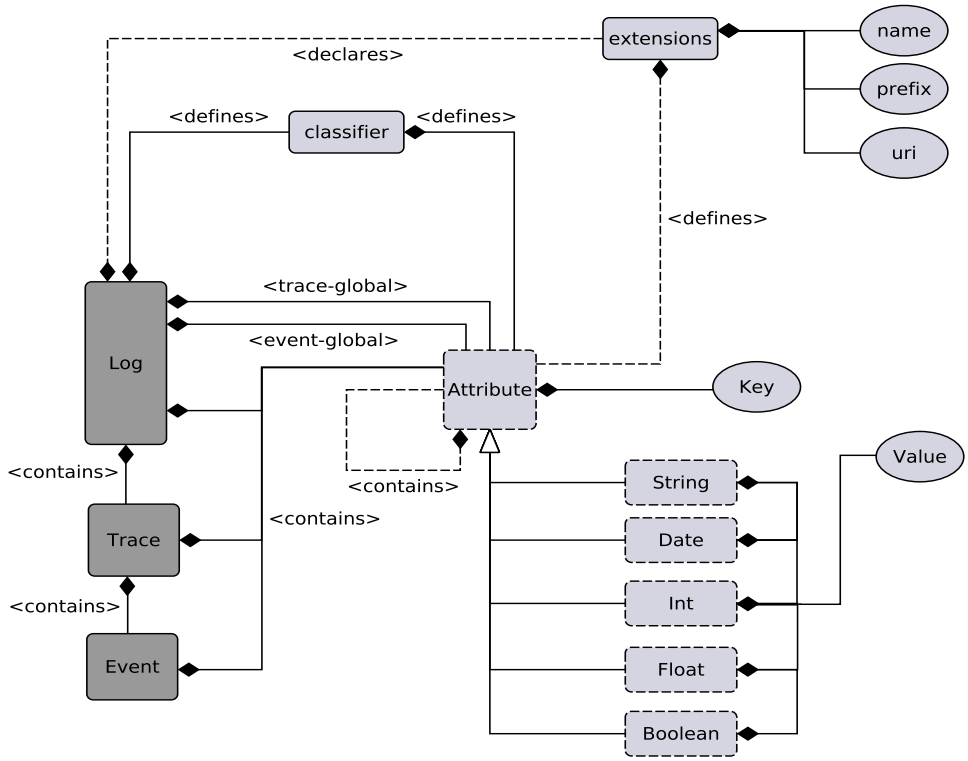


Figura 4.3: Metamodello formato XES

A titolo di esempio, la figura 4.2, mostra una traccia estratta da un event log di un processo che descrive la gestione delle mail ricevute.

```
<trace>
  <string key="concept:name" value="Mail"/>
  <string key="description" value="PaperOne"/>
  <event>
    <string key="Mittente" value="Alessio Rossi"/>
    <int key="ide" value="1"/>
    <string key="org:resource" value="Hind"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="concept:name" value="NotifyMail"/>
    <date key="time:timestamp" value="2012-01-15T16:15:49.000+01:00"/>
  </event>
```

```
  <event>
    <string key="org:resource" value="Hind"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="NotifyMail"/>
    <date key="time:timestamp" value="2012-01-15T17:08:24.000+01:00"/>
  </event>
</trace>
```

Listing 4.2: Event log in formato XES.

Nel tirocinio è stata impiegata la libreria "OpenXES" che rappresenta l'implementazione open-source in Java di quanto specificato dallo standard XES. La libreria e la documentazione associata sono entrambi reperibili in [8].

4.2 Data mining

Esiste una molteplicità di tool che forniscono una piattaforma per l'utilizzo delle tecniche di data mining, quello che è stato scelto per essere impiegato durante il tirocinio è un software Java open source: Weka che risulta essere ampiamente utilizzato in diversi ambiti.

4.2.1 Weka

La sigla Weka sta per "Waikato Environment for Knowledge Analysis", è stato sviluppato all'università di Waikato in Nuova Zelanda e viene rilasciato sotto licenza GNU [5]. Weka consiste in una collezione di algoritmi dedicati all'analisi dei dati tramite l'attuazione di tecniche di data minig tra cui la classificazione. Il progetto è nato nel 1993 ed inizialmente Weka fu sviluppato in linguaggio C. Nel 1997 è stata rilasciata la versione sviluppata interamente in linguaggio Java. Weka, oltre a fornire una API per gli utenti programmatori, mette a disposizione anche un'interfaccia grafica user friendly per facilitarne l'uso.

Ambienti operativi

Dall'interfaccia grafica iniziale (figura 4.4) si può accedere a quattro ambienti operativi:

- SimpleCLI: ambiente a linea di comando da usare per invocare direttamente le funzionalità di Weka.

- Explorer: (figura 4.5) ambiente da utilizzare per caricare gli insiemi di dati da analizzare, visualizzare la disposizione degli attributi, preprocessare i dati ed eseguire algoritmi di analisi per il data mining (ad esempio algoritmi per la classificazione). Questo è l'ambiente che ha fornito le funzionalità impiegate nel tirocinio.

- Experimenter: consente di impostare una serie di analisi su vari insiemi di dati e con vari algoritmi, ed eseguirle tutte insieme. In questo modo è possibile confrontare vari tipi di algoritmi e determinare quale è il più adatto da applicare.

- Knowledge Flow: permette di esprimere le operazioni da eseguire in forma grafica tramite un diagramma che mette in evidenza "il flusso della conoscenza". E' possibile selezionare varie componenti (sorgenti dati, filtri, algoritmi di classificazione) e collegarli in un diagramma "data-flow".

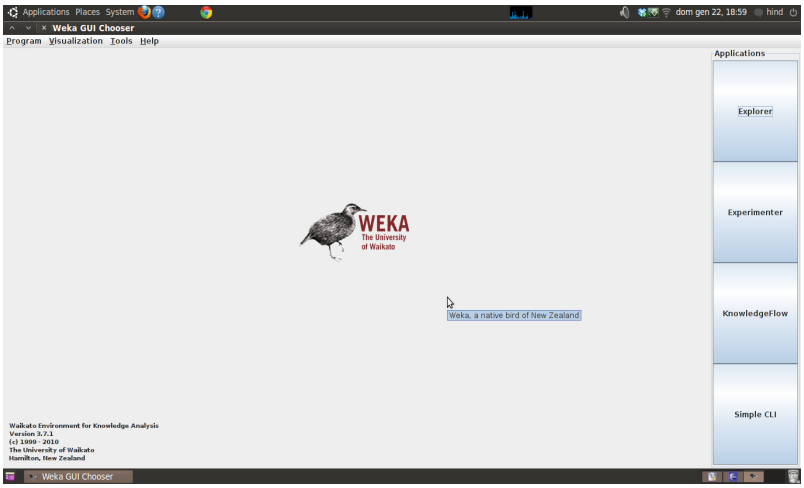


Figura 4.4: Weka GUI Chooser.

Formato dei dati

Weka è in grado di prelevare dati da un file purché sia espresso nel formato ARFF (che è il formato standard per Weka). Un file ARFF è composto da un'intestazione e dal corpo dati vero e proprio. L'intestazione contiene il nome del data set ed una intestazione degli attributi; per ogni attributo è

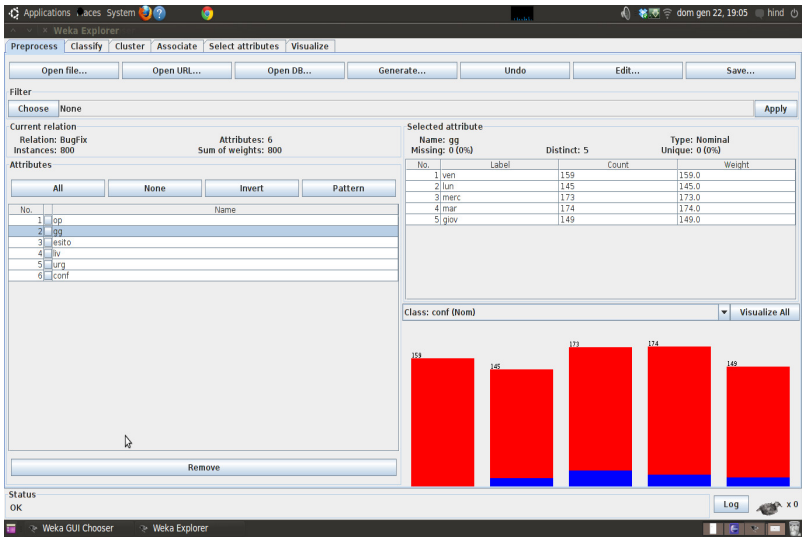


Figura 4.5: Weka Explorer.

possibile specificare il tipo (numerico, nominale, stringa, data). I dati veri sono forniti creando una riga per ogni istanza, e separando i campi con una virgola. un frammento del file ARFF relativo al data set presentato nel capitolo 3 nella figura 3.1 è nella figura 4.3.

```
@relation tempo

@attribute cielo {sole, coperto, pioggia}
@attribute temperatura real
@attribute umidita real
@attribute vento {si, no}
@attribute giocare {si, no}

@data
sole,85,85,no,no
sole,80,90,si,no
coperto,83,86,no,si
pioggia,70,96,no,si
pioggia,68,80,no,si
pioggia,65,70,si,no
coperto,64,65,si,si
sole,72,95,no,no
sole,69,70,no,si
pioggia,75,80,no,si
sole,75,70,si,si
coperto,72,90,si,si
coperto,81,75,no,si
pioggia,71,91,si,no
```

Listing 4.3: data set in formato ARFF.

Si osservi che con la riga `@relation tempo` si specifica un nome per il data set, la riga `@attribute cielo {sole, coperto, pioggia}` specifica che l'attributo `cielo` è di tipo nominale e può assumere i valori `sole`, `coperto` e `pioggia`. La riga `@attribute temperatura real` specifica che l'attributo `temperatura` può assumere valori numerici reali, mentre la riga `@data` specifica l'inizio dei dati veri e propri. E' possibile utilizzare il valore `'?'` come valore di un attributo per indicare un dato mancante. In generale, vale che l'ultimo attributo dichiarato prende il ruolo dell'attributo target (in ogni caso è sempre possibile specificare uno diverso dall'ultimo).

Architettura

Weka è strutturato in una serie di package, di seguito l'attenzione è focalizzata solo su alcuni che risultano di maggiore utilità per il tirocinio:

- weka.core: è il package centrale, contiene classi a cui quasi tutte le altri classi fanno riferimento e di cui le principali sono:
  - Attribute: un oggetto di questa classe rappresenta un attributo e contiene il nome dell'attributo, il tipo, ed i possibili valori nel caso di un attributo nominale.
  - Instance: un oggetto di questa classe contiene i valori degli attributi di una particolare istanza.
  - Instances: un oggetto di questa classe contiene un insieme ordinato di istanze, ovvero rappresenta un data set.
- weka.classifiers: contiene implementazione degli algoritmi più comunemente utilizzati per la classificazione e la predizione. La classe più importante di cui consta questo package è:
  - Classifier: definisce la struttra generale di qualsiasi classficatore e fornisce due metodi fondamentali: buildClassifier() e classifyInstance(). Qualsiasi sottoclasse di Classifier che definisce un algoritmo di apprendimento, deve specificare un'implementazione di questi due metodi.
- weka.filters: permette di fare il preprocessing dei dati. Fornisce la classe Filter che definisce la struttura generale di tutte le classi che contengono algoritmi di filtering. Tali classi sono tutte definite come sottoclassi di Filter.

- weka.gui: contiene le classi per la gestione della parte di interfaccia grafica del software. In particolare contiene il package Weka.gui.arffviewer che permette di visualizzare un file arff e di modificarne gli attributi (preprocessing).

Il package Weka.classifiers contiene a sua volta una serie di altri package, ognuno di essi fornisce le classi che implementano una certa tecnica di classificazione. Durante il tirocinio, è stato impiegato il package "weka.classifiers.tree" che presenta l'implementazione di algoritmi per la generazione di alberi di decisione. In particolare è stata usata la classe J48. Questa rappresenta l'implementazione Weka dell'algoritmo C4.5 [16] che risulta avere uno schema di funzionamente uguale a quello descritto per l'algoritmo di Hunt (sezione 3.1.3).

L'algoritmo J48 viene applicato al training set, se viene specificato anche un test set l'albero di decisione costruito viene testato su di esso. In caso contrario, J48 esegue la tecnica della cross-validation sul traning set. Dato un parametro K, tale tecnica consiste nel suddividere il training set in modo che $(1 - \frac{1}{K})$ dei dati venga usato per costruire il modello ed il rimanente $\frac{1}{K}$ per eseguire il test. Questo processo viene ripetuto K volte in modo che tutti i dati del training set siano usati esattamente una volta nei dati di test.

Capitolo 5

Analisi di processi: un approccio basato sulla classificazione

L'idea di impiegare tecniche di data mining al servizio del process mining è stata sperimentata da Van der Aalst in [3]. In questo articolo viene effettuata un'analisi dei dati registrati negli event log per capire come questi influenzano la direzione che il flusso di esecuzione prende nei punti di decisione. La motivazione che ha incoraggiato questa strada deriva dalla presenza di una quantità enorme di dati registrati negli event log, ma che non vengono sfruttati interamente dalle tecniche di process mining disponibili finora.

In questo tirocinio è stata adottata la stessa metodologia per fornire un supporto aggiuntivo agli analisti di processo. In particolare, è stato sviluppato un approccio in grado di individuare pattern o regole nei dati del processo, in corrispondenza dei quali si verificano gli errori di conformance discussi nella sezione 2.2.3. L'obiettivo è quello di effettuare un'indagine sui dati per individuare le *cause* degli errori, questo infatti è fondamentale per l'attuazione di misure correttive. Data l'enorme quantità di dati registrati durante le varie esecuzioni del processo, l'indagine è effettuata con il supporto della tecnica di classificazione presentata nel capitolo 3.

In questo capitolo vengono presentati due prototipi di processi sperimentali con dei dati esemplificativi, questo ha lo scopo di mostrare le utilità dell'approccio sviluppato: nel primo caso l'analisi dei dati fornisce informazioni utili che portano ad un rimodellamento del processo che si dimostra troppo astratto e non corrispondente alla realtà, mentre nel secondo caso

vengono rilevate esecuzioni errate del processo dovute a imprecisioni umane.

5.1 Caso 1: Modello di processo errato

Supponiamo che all'interno di un'organizzazione viga una determinata procedura per affrontare gli errori (bug) che si verificano in un certo sistema informatizzato. La procedura aziendale consiste in tre fasi:

- Notify Bug: viene notificato il bug da trattare. Durante questa fase semplicemente viene segnalata la presenza del bug agli addetti specializzati e vengono fornite alcune informazioni necessarie: il livello (dal punto di vista dell'architettura del sistema) in cui si verifica il bug ed il livello di urgenza del problema.
- Check Bug: il personale addetto provvede alla verifica del bug eseguendo vari test per l'accertamento.
- Fix Bug: si attuano gli interventi risolutivi da parte del personale tecnico e si provvede alla segnalazione dell'esito dell'operazione.

Questa procedura può essere rappresentata dal processo di business il cui modello BPMN è presentato in figura 5.1. La rete di Petri che risulta dalla trasformazione descritta in sezione 2.1.3 è presentata in figura 5.2.

Figura 5.1: Modello BPMN del processo FixBug.

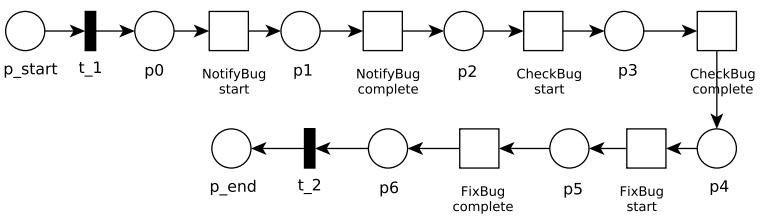


Figura 5.2: rete di Petri del processo FixBug.

L'organizzazione aziendale prevede un meccanismo di registrazione degli event log relativi ai suoi processi. Supponiamo che dai file di log risulti che nell'ultimo anno si siano verificati 1000 segnalazioni di bug, vengono registrati quindi 1000 istanze del processo. Supponiamo anche che in corrispondenza di ognuna di queste vengano registrati i seguenti dati: l'identificatore dell'istanza, il livello del bug, l'urgenza e l'esito della risoluzione. Questi dati sono riassunti nella tabella 5.1 .

Ide	Urgenza	Livello	Esito
0	0	DB	positivo
1	1	IO	negativo
2	0	DB	positivo
3	0	LOGSYS	negativo
4	1	IO	postivo
5	1	DB	negativo
...	...	...	...
...	...	...	...
...	...	...	...
999	0	1	positivo

Tabella 5.1: Dati delle istanze

Con l'impiego del principale strumendo di analisi, ovvero l'algorimto log replay presentato in 2.2.2, ed in base alla rete di Petri in figura 5.2, vengono analizzate le tracce relative alle varie esecuzioni. I risultati dell'analisi di conformance (2.2.3) mostrano che su un totale di 1000 istanze, esistono 505 esecuzioni che non sono conformi al modello, in altre parole, nell'ultimo anno ci sono stati 505 casi che non hanno rispettato la procedura aziendale.

A questo punto, quello che si vuole è cercare di individuare le cause che stanno dietro ai casi di non conformità. Si vuole analizzare se esistono alcuni pattern nei dati in corrispondenza dei quali questi casi si verificano.

Data la quantità dei dati a disposizione, riuscire ad analizzarli individualmente per scoprire eventuali regole non è fattibile se non adottando delle tecniche apposite. Queste ci vengono fornite dal data mining.

5.1.1 Formulazione di un problema di classificazione

Il problema di analizzare i dati derivanti dai file di log può essere trasformato in un problema di classificazione. Dai dati del processo si derivano gli attributi del data set. In particolare, i dati relativi ad ogni istanza di processo determinano un record, questo è caratterizzato dai seguenti attributi: identificatore d'istanza, livello di urgenza, livello del bug e l'esito. La conformance di un'istanza al modello fornisce un ulteriore dato che nel problema di classificazione assume il ruolo dell'attributo target. Tutti gli attributi sono di tipo discreto. In base a questa formulazione, viene costruita la tabella in figura 5.2 che rappresenta il data set del problema di classificazione.

Ide	Urgenza	Livello	Esito	Conformance
0	0	DB	positivo	si
1	1	IO	negativo	no
2	0	DB	positivo	si
3	0	LOGSYS	negativo	si
4	1	IO	postivo	no
5	1	DB	negativo	no
...	...	...	...	...
...	...	...	...	...
...	...	...	...	...
999	0	1	positivo	si

Tabella 5.2: Data set

Utilizzando gli strumenti di data mining descritti in 4.2, è possibile costruire un modello di classificazione basato sugli alberi di decisione: dal file di log relativo alle istanze, vengono estrapolati gli attributi e prodotte tutte le risorse di input necessarie agli algoritmi di data mining impiegati. L'albero di decisione risultante è mostrato in figura 5.3 con la relativa matrice di confusione in 5.3. Occorre comunque osservare che i dati del problema sono stati generati appositamente per la sperimentazione.

5.1.2 Interpretazione dei risultati

L'albero restituito, la cui accuratezza è massima, mostra con chiarezza la regola in base alla quale si desume la conformità di un'istanza: se si tratta di

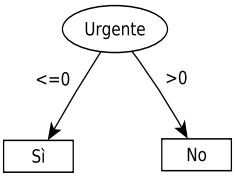


Figura 5.3: Albero di decisione processo FixBug

Classe attuale	Classe predetta	
	Classe = si	Classe = no
Classe = si	495	0
Classe = no	0	505

Tabella 5.3: Matrice di confusione per l'albero di decisione 5.3 .

un bug da risolvere urgentemente la procedura aziendale non viene rispettata.

Per ottenere maggiori informazioni, ci vengono in aiuto i risultati dell'esecuzione del log replay presentati graficamente in figura 5.4 (dove per ogni arco viene specificato il numero delle attivazioni, le piazze colorate di rosso presentano dei token mancanti oppure rimanenti, mentre le transizioni colorate di giallo presentano delle esecuzioni errate). Da questi emerge che in corrispondenza alla piazza p_2 ci sono 505 token rimanenti, mentre nella piazza p_4 ci sono 505 token mancanti.

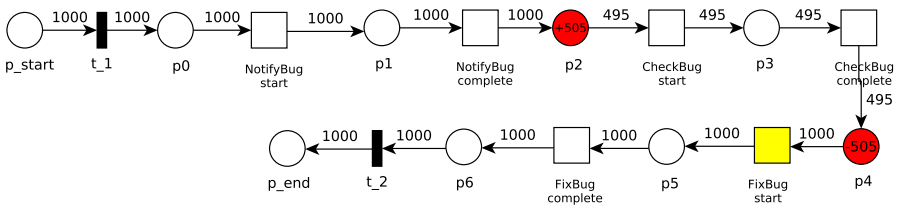


Figura 5.4: Risultati log replay.

In base a quanto spiegato nella sezione 2.2.3, i token mancanti nella piazza p_4 segnalano un'esecuzione errata del task "FixBug". Inoltre, dal momento che ci sono 505 token rimanenti in p_2, la motivazione per cui l'esecuzione del task "FixBug" è errata consiste nell'avere saltato il task "CheckBug", infat-

ti il log replay crea artificialmente i token necessari per mimare gli eventi relativi all'evento "FixBug start" proprio perché la transizione "CheckBug complete" non viene attivata (e nemmeno "CheckBug start" dato che p_2 presenta token rimanenti).

A questo punto le informazioni di cui si dispone si sintetizzano in tre punti:

1. Esistono 505 tracce non conformi (risultato diretto del log replay).
2. Le 505 tracce non conformi corrisondono ad istanze che non rispettano la procedura aziendale perché la fase "Fix Bug" viene attuata senza aver prima effettuato la fase "Check bug" (interpretazione dei risultati del log replay).
3. Le 505 istanze di problema che sono urgenti non rispettano la procedura aziendale (regola individuata dall'albero di decisione).

Dai punti $(1, 2, 3)$ si deduce che quando occorre risolvere un bug urgentemente, la fase del Check Bug non viene effettuata. Questa informazione mostra agli analisti di processo che quanto è stato prefissato non trova una perfetta coincidenza con ciò che accade in pratica nella realtà aziendale. Pertanto, il modello del processo di business in questione è da ritenersi probabilmente troppo astratto, ed un modo per tenere in considerazione le nuove informazioni ottenute è modificarlo come in figura 5.5. In questo modo, con l'aggiunta del "Data-based Decisione Gateway" si permette di prendere una decisione sulla prosecuzione del flusso di esecuzione in base all'urgenza del problema.

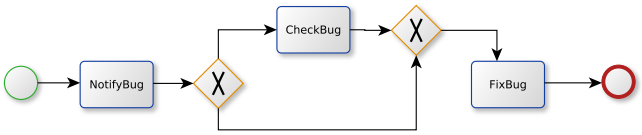


Figura 5.5: Modello BPMN modificato del processo FixBug.

5.2 Caso 2: esecuzioni di processo errate

All'interno di tutte le imprese a carattere commerciale vige una procedura che regolamenta il processo di vendita ai clienti e che consta di diverse fasi. Un frammento di questo processo si compone dalle seguenti attività:

- Notifica ordine: in cui il responsabile commerciale di turno, in seguito alla ricezione di un ordine da parte di un cliente, provvede alla sua notifica sia all'ufficio finanziario che al magazzino.
- Valutazione finanziaria: l'ufficio finanziario effettua un'analisi di tipo economico con particolare riguardo sulla solvibilità del cliente.
- Valutazione di magazzino: il responsabile di magazzino verifica la copertura dell'ordine con le scorte giacenti.
- Notifica cliente: in seguito alla ricezione dei risultati di valutazione sia finanziaria che di magazzino, il responsabile commerciale comunica la conferma o il rifiuto dell'ordine al cliente.

Il modello BPMN relativo al frammento del processo di vendita è presentato in figura 5.6, mentre la rete di Petri derivante dalla trasformazione del modello è in firgura 5.7.

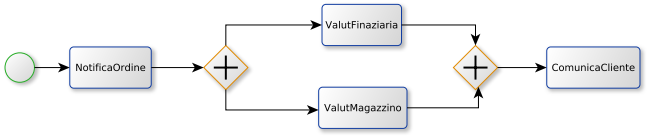


Figura 5.6: Modello BPMN del processo di vendita

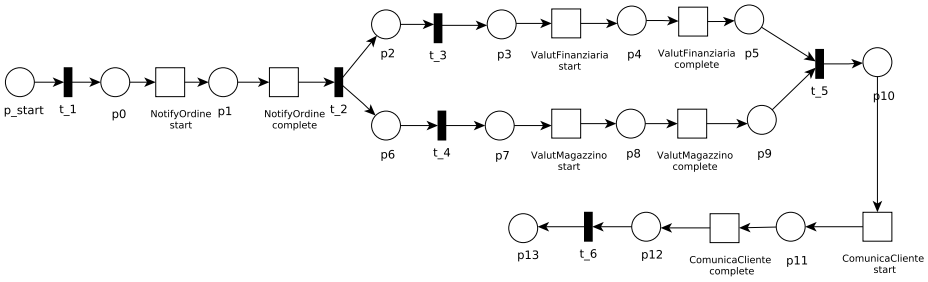


Figura 5.7: rete di Petri del processo vendite.

Supponiamo che in corrispondenza ad ogni ordine ricevuto vengono registrati i dati seguenti: nome del responsabile commerciale che gestisce il cliente, identificatore del cliente, tipologia del cliente (consolidato o nuovo), nome del capo ufficio finanziario, nome del magazziniere, esito della valutazione finanziaria, esito della valutazione di magazzino ed infine l'esito dell'ordine comunicato al cliente. Naturalmente si suppone che l'azienda impieghi

qualche software gestionale nella sua attività ordinaria che preveda i meccanismo di registrazione di event log.

Nell'ultimo periodo di analisi preso in considerazione, sono stati notificati 1000 ordini. I dati relativi a questi ordini derivanti dagli event log sono riassunti in una tabella di cui un frammento è presentato in figura 5.4.

Ide Ordine	Ide Cliente	Categoria Cliente	Resp Commerciale	Resp Finanziario	Magazziniere	Esito Finanziario	Esito Magazzino	Esito Ordine
1	20	consolidato	Alessandro	Alessio	Gianni	positivo	positivo	confermato
2	700	nuovo	Mario	Alessio	Giorgio	positivo	positivo	confermato
3	10	consolidato	Mario	Alessio	Gianni	negativo	positivo	confermato
5	200	nuovo	Mario	Alessio	Gianni	negativo	positivo	negato
6	505	consolidato	Roberto	Alessio	Gianni	negativo	positivo	negato
...	...	...	...	...	...	....	...	....
...	...	...	...	...	...	....	...	....
1000	300	nuovo	Mario	Alessio	Giorgio	negativo	negativo	negato

Tabella 5.4: Dati delle vendite

Le tracce relative alle varie esecuzioni vengono analizzate con l'algoritmo log replay prendendo come modello di riferimento la rete di Petri presentata in figura 5.7. I risultati dell'analisi mostrano che, su 1000 istanze del processo, 173 non sono conformi al modello. Una descrizione dettagliata dei risultati è discussa nelle sezioni successive.

Analogamente al caso 1 in 5.1, a partire dai risultati dell'analisi di conformance e dai dati contenuti negli event log, sfruttando opportunamente la tecnica di calssificazione è possibile condurre un'indagine sulle cause di non conformità.

5.2.1 Formulazione problema classificazione

In corrispondenza di ogni istanza viene individuato un record i cui attributi sono i dati registrati nella traccia relativa a quell'istanza. Il risultato del log replay sulla conformance dell'istanza individua invece l'attributo target. Il data set del problema di classificazione da affrontare è fornito nella tabella 5.5

Ide Ordine	Ide Cliente	Categoria Cliente	Resp Commerciale	Resp Finanziario	Magazziniere	Esito Finanziario	Esito Magazzino	Esito Ordine	Conformance
1	20	consolidato	Alessandro	Alessio	Gianni	positivo	positivo	confermato	si
2	700	nuovo	Mario	Alessio	Giorgio	positivo	positivo	confermato	si
3	10	consolidato	Mario	Alessio	Gianni	negativo	positivo	confermato	no
5	200	nuovo	Mario	Alessio	Gianni	negativo	positivo	negato	si
6	505	consolidato	Roberto	Alessio	Gianni	negativo	positivo	negato	si
...	...	...	...	...	...	....	...	....	...
...	...	...	...	...	...	....	...	....	...
1000	300	nuovo	Mario	Alessio	Giorgio	negativo	negativo	negato	si

Tabella 5.5: Data set delle vendite

Con l'ausilio degli strumenti descritti in 3.1, si applica al data set uno degli algoritmi di data mining per la costruzione di alberi di decisione. Il risultato è fornito nella figura 5.8, mentre in tabella 5.6 è presentata la matrice di confusione per la valutazione dell'accuratezza del classificatore ottenuto.

Classe attuale	Classe predetta	
	Classe = si	Classe = no
Classe = si	827	0
Classe = no	0	173

Tabella 5.6: Matrice di confusione per l'albero di decisione 5.8 .

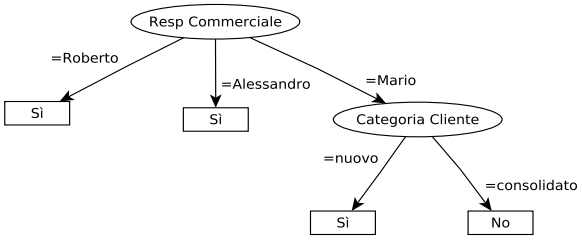


Figura 5.8: Albero di decisione processo Vendita.

5.2.2 Interpretazione dei risultati

Il classificatore descrive i pattern rilevati nei dati in corrispondenza dei quali si verificano errori di conformance: gli ordini gestiti dal responsabile commerciale Mario provenienti dai clienti consolidati dell'azienda non rispettano la procedura standard di vendita. Va tenuto comunque presente che i dati del problema sono stati generati appositamente per la sperimentazione. Per potere risalire a informazioni più significative, occorre mettere in relazione quanto emerso dall'albero di decisione con i risultati di conformance che sono sintetizzati nella rete di Petri in figura 5.9.

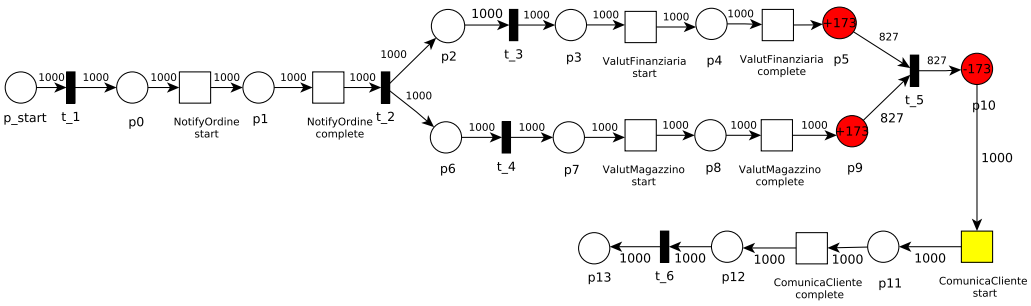


Figura 5.9: Risultati log replay

Dalla rete si desume che si sono verificate 173 istanze non conformi al modello. In particolare, presso la piazza p_5 ci sono 173 token rimanenti così come per la piazza p_9, mentre si sono verificati 173 token mancanti presso p_{10}. Per potere dare una corretta interpretazione ai risultati presentati nella rete, occorre tenere in considerazione che il comportamente adottato dal log replay è non bloccante. Infatti, l'esistenza dei 173 token mancanti nella piazza p_{10}

significa che, per 173 tracce, durante il replay l'algoritmo ha bisogno di mimare l'evento "ComunicaCliente_start" ma la transizione relativa all'evento non risulta abilitata nella marcatura raggiunta dalla rete, ciò può accadere solo nel caso in cui la transizione invisibile t_5 non è abilitata. Ora, se t_5 non è abilitata è solo perché nella piazza p_5 o p_9 (oppure in entrambe) non ci sono token nel momento in cui occorre eseguire il replay dell'evento "ComunicaCliente_start". Tuttavia, la presenza dei token rimanenti nelle piazze p_5 e p_9 dimostra che i due rami del fork sono stati eseguiti in quelle 173 tracce. Di conseguenza, si deduce che l'algoritmo, nel momento in cui deve eseguire il replay dell'evento "ComunicaCliente _start", la transizione "ValutFinanziaria_complete" oppure "ValutMagazzino_complete" (oppure entrambe) non sono state ancora scattate. In altre parole, la non conformità di quelle istanze deriva dal fatto che la comunicazione al cliente dell'esito dell'ordine avviene prima del completamento delle valutazioni previste dal processo aziendale di vendita.

Sintetizzando i risultati di analisi raggiunti fino a questo punto, le informazioni di cui si dispone sono:

1. Esistono 173 istanze non conformi al modello.

2. Le istanze non sono conformi alla procedura aziendale perché l'ufficio commerciale provvede alla comunicazione dell'esito dell'ordine al cliente *prima* del completamento della fase di valutazione finanziaria e/o di magazzino.

3. La gestione degli ordini ricevuti dal responsabile commerciale Mario e provenienti dai clienti consolidati non sono conformi al modello di processo.

E' ragionevole pensare dunque che, per gli ordini effettuati dai clienti consolidati, il responsabile dell'ufficio commerciale Mario provvede a comunicare direttamente l'esito dell'ordine al cliente senza attendere l'esito della valutazione finanziaria e di magazzino, non rispettando in questo modo la procedura di vendita.

In questo caso, gli analisti di processo potrebbero valutare se è opportutno intervenire con un rimodellamento del processo, oppure adottare misure correttive che tendono ad intervenire sui comportamenti errati da parte del personale. In questo senso ad esempio, potrebbe anche risultare utile prevedere i casi di errore per sollecitare il personale interessato onde evitare gli

errori noti.

In generale, vale la pena osservare che il classificatore costruito con questo approccio può essere utilizzato anche in senso predittivo oltre che descrittivo. Infatti, alla ricezione di un event log, per effettuare il controllo di conformance delle istanze, si potrebbe impiegare il classificatore costruito ed eseguire invece l'algoritmo log replay (2.2.2) soltanto per quelle istanze per le quali sono stati predetti errori di conformance. In questo modo si ha il vantaggio di risparmiare tempo visto che il log replay ha dei tempi di esecuzione maggiori rispetto a quelli impiegati da un classificatore per predire la classe di appartenenza di un'istanza.

5.3 Un approccio per l'analisi di performance

L'approccio mostrato nei due casi precedenti, ovvero l'impiego della classificazione a servizio di un'analisi di conformance, può essere esteso anche per effettuare indagini sulle performance del processo di business. Tale estensione risulta però essere oltre gli obiettivi di questo tirocinio, ma è opportuno osservare che si deriva in maniera naturale da quanto presentato nelle sezioni di questo capitolo. Ad ogni modo nei paragrafi successivi viene descritto un possibile modo per attuare tale estensione.

Si consideri un processo in cui sussistono delle attività che possono procedere in parallelo con due flussi di esecuzione indipendenti ma che, ad un certo punto, si devono ricongiungere. Una situazione simile è data proprio dalle attività di valutazione del processo di vendita. In questi casi, potrebbe avere senso scoprire se ci sono delle relazioni nei dati che influenzano i tempi di sincronizzazione. L'utilità di fare ciò potrebbe essere quella di capire sotto quali pattern nei dati si verificano dei rallentamenti nel processo dovuti ad un eccessivo tempo di sincronizzazione ed intervenire magari con dei provvedimenti di ottimizzazione.

Con riferimento al processo di vendita ed alle metriche di performance presentate in sezione 2.2.4 i tempi di sincronizzazione sono dati da: $tsc(p5)$ per il ramo su cui giace l'attività di valutazione finanziaria, e da $tsc(p6)$ per l'attività di valutazione di magazzino. Un possibile modo di trasformare il problema in uno di classificazione è considerare per ogni traccia il massimo tempo di sincronizzazione $max\{tsc(p5), tsc(p6)\}$ come attributo target. La classificazione prevede però solo attributi di tipo discreto come target,

occorre dunque prevedere una fase di discretizzazione. Un modo di fare questo consiste in stabilire delle fasce di appartenenza che caratterizzano i valori temporali di sincornizzazione, questo ovviamente deve avere senso per il dominio in cui il processo si colloca. A titolo di esempio, possiamo giudicare il tempo massimo di sincronizzazione come basso se è compreso in $[0min, 60min]$, accettabile se il valore ricade nell'intervallo $[61min, 120min]$, ed alto se si superano $121min$. Un altro modo consiste invece nel considerare come target di classificazione il valore $tsc(p5) - tsc(p6)$. In questo modo infatti è possibile capire quale ramo risulta più lento in base ai dati, quello su cui giace l'attività di valutazione finanziaria oppure quello relativo alla valutazione di magazzino. La fase di discretizzazione potrebbe semplicemente indicare il ramo più lento: se il valore $tsc(p5)-tsc(p6)$ è positivo allora il ramo più lento è quello su cui giace l'attività di valutazione di magazzino, nel caso contrario è quello relativo alla valutazione finanziaria. Per tutto il resto la trasformazione del problema è analoga a quanto illustrato nella sezione 5.2.1.

Vale la pena osservare che il classificatore ottenuto con questo approccio potrebbe essere usato in senso predittivo oltre che descrittivo. Se è possibile, si potrebbero classificare le istanze di processo ancora in esecuzione per rilevare ad esempio le attività particolarmente lente. In questo caso, si potrebbero cambiare le priorità o l'ordine di esecuzione delle attività per cercare di ottimizzare il tempo totale di esecuzione dell'istanza.

Capitolo 6

Realizzazione del framework di analisi

Questo capitolo assume un carattere più tecnico rispetto ai precedenti. Vengono presentati quali sono i plugin sviluppati per sperimentare l'approccio descritto nel capitolo 5 e riassunti nella tabella 6.1. In particolare questi si distinguono in plugin che implementano la parte di trasformazione di un problema di conformance in uno di classificazione, e plugin che implementano aspetti relativi alla trasformazione di file in risorse gestibili all'interno del framework ProM 6. Ma prima di fare questo la sezione 6.1 descrive come viene sviluppato un generico plugin di ProM.

6.1 Come si scrive un plugin in ProM 6

Per creare un plugin viene creata una classe simile a quella descritta nell'esempio riportato nel listing 6.1.

```
package org.processmining.plugins.helloworld;

import org.processmining.contexts.uitopia.annotations.UITopiaVariant;
import org.processmining.framework.plugin.PluginContext;
import org.processmining.framework.plugin.annotations.Plugin;

public class HelloWorld {
        @Plugin(
                name = "My Hello World Plugin",
                parameterLabels = {},
                returnLabels = { "Hello world string" },
                returnTypes = { String.class },
                help = "Produces the string: 'Hello world'"
        )
        @UITopiaVariant(
```

```
                affiliation = "Pisa University",
                author = "Hind Chfouka",
                email = "chfouka@di.unipi.it"
        )
        public static String helloWorld(PluginContext context) {
                return "Hello World";
        }
}
```

Listing 6.1: Esempio plugin

La parte logica del plugin è contenuta nel metodo `helloworld` che restituisce in output un oggetto di tipo `String` contenente il testo `Hello World`, mentre richiede in ingresso un oggetto di tipo `PluginContext`. Tale oggetto realizza la gestione delle risorse e permette ai plug-in di accedere all'ambiente di ProM. Viene richiesto da tutti i plug-in che estendono ProM.

Il fatto che il metodo `helloworld` costituisce il plugin è dichiarato tramite l'annotazione `@Plugin`. Con questa annotazione, oltre a dare un nome al plugin vengono specificati:

- `parameterLabels`: la lista delle etichette (nomi) degli oggetti in input al plug-in. Nell'esempio risulta essere vuota.
- `returnLabels`: la lista delle etichette degli oggetti in output al plugin. In questo caso un solo oggetto viene restituito ed è etichettato `Hello world string`.
- `returnTypes`: una lista che specifica il tipo degli oggetti in output. Nell'esempio il tipo è `String.class`, infatti l'oggetto restitituto dal plug-in è di tipo `String`.
- `help`: utile per specificare qualche informazione sul plug-in (opzionale).

Con l'utilizzo dell'annotazione `@Plugin`, il framework riconosce che quel metodo annotato costituisce il plugin. Se il metodo è statico, quando viene richiesta l'esecuzione del plugin, il framework invoca il metodo come se venisse effettuata la seguente chiamata:
`HelloWorld.helloWorld(context);`

L'annotazione `@UITopiaVariant` ha il ruolo di informare la GUI del framework ProM 6 dell'esistenza del plugin. Con essa vengono tipicamente specificate le seguenti informazioni:

- `affiliation`: specifica l'affiliazione dell'autore del pulgin.

- `author`: nome dell'autore.
- `email`: l'indirizzo email dell'autore.

6.2 Plugin sviluppati

Di seguito sono presentati i plugin che hanno integrato ProM 6 per permettere di attuare quanto descritto nel capitolo 5. Nella realizzazione è stato adottato un approccio che ha permesso di affrontare le difficoltà in modo incrementale.

6.2.1 Plugin: Weka Instances with Conformance

In sostanza questo plugin ha il compito di preparare il data set necessario per la costruzione del modello di classificazione.

```
public class DMconfPlugin {
    @Plugin(name = "Log to arff",
        parameterLabels = { "Log", "ConformanceResult" },
        returnLabels = { "Weka Instance" },
        returnTypes = { Instances.class }  )
    @UITopiaVariant(
        uiLabel="Weka Instances with Conformance",
        affiliation = "University of Pisa",
        author = "Hind",
        email = "chfouka@di.unipi.it")

    public static Object generateWekaInstances(PluginContext context, XLog
        log,           TotalConformanceResult conformance){

          /*Parte logica del plugin*/

      return Wekainstances;
  }
```

Listing 6.2: Intestazione Weka Instances with Conformance

Come mostra il linsting 6.2, il metodo che implementa la logica del plugin richiede in input, oltre al parametro `context` di tipo `PluginContext` come specificato nella sezione 6.1, due parametri:

1. `log`: di tipo `XLog` che rappresenta l'event log da processare. `XLog` è una classe messa a disposizione dalla libreria OpenXES [8] e fornisce una serie di metodi utili per la gestione di un event log.

2. `conformance`: di tipo `TotalConformanceResult`, rappresenta il risultato dell'analisi di conformance. Ovviamente il risultato del plugin ha senso se la conformance passata come terzo paramentro è relativa all'event log specificato tramite il paramentro `log`.

Il plugin restituisce in output un solo parametro:

1. `Wekainstances`: di tipo `Instances` che rappresenta il data set costruito a partire dai parametri in ingresso. `Instances` è una classe fornita dal package "Weka.core" descritto nella sezione 4.2.1.

Dato un event log, esiste un plugin di ProM già sviluppato che implementa tutta la parte relativa all'analisi di conformance, contiene cioè un'implementazione dell'algoritmo log repaly 2.2.2. Questo plugin prende in input un modello di processo sotto forma di rete di Petri, un event log di tipo `XLog` e restituisce in output un oggetto di tipo `TotalConformanceResutl`.

A partire dall'event log, l'obiettivo del plugin `Weka Instances with Conformance` è capire quali sono gli attributi che caratterizzano il processo le cui istanze sono registrate nel log, risalire a qual'è il risultato di conformance relativo ad ogni istanza e restituire un oggetto di tipo `Instances`. E' opportuno precisare che, per come sono stati generati gli event log, vale la seguente assunzione:

- Dato che gli eventi relativi ad un'attività A possono essere sia di tipo A_start che $A\_complete$, si assume che gli attributi riferiti dall'attività A sono tutti registrati in corrispondenza di A_start.

La difficoltà principale riscontrata è relativa alla gestione degli attributi che si presentano negli event log. In particolare quando si hanno due (o più) attività in corrispondenza delle quali vengono registrati attributi con lo stesso nome. In linea di massima, è possibile riscontrare i seguenti scenari:

1. Le due attività riferiscono lo stesso attributo, ovvero riferiscono la stessa entità della realtà. Si pensi ad un identificatore unico di istanza che viene registrato da ogni attività del processo.

2. Ogni attività sovrascrive l'attributo riferito dall'altra specificando un nuovo valore. Si pensi ad esempio al caso di un attributo di nome `Budget rimanente` il cui valore viene aggiornato da più attività del processo.

3. Gli attributi sono in realtà diversi (denotano entità diverse) anche se portano lo stesso nome. Questo può derivare ad esempio da imprecisioni in fase di studio del dominio del processo, o di progettazione del meccanismo di registrazione di log.

E' ovvio che la gestione di questi casi è strettamente collegata al dominio in cui l'event log è stato registrato e non è possibile fare assunzioni a priori. In questo lavoro è stata adottata la seguente gestione:

- Qualora nella scansione dell'event log venissero riscontrate due (o più) attività che riferiscono attributi con lo stesso nome, questi vengono considerati diversi indipendentemente dall'entità che caratterizzano.

Si consideri un processo in cui esistono due attività sequenziali A e B, ognuna di esse riferisce una serie di attributi tra cui uno che viene denotato con lo stesso nome. Il listing 6.3 mostra un frammento dell'event log che riporta questa situazione.

```
<trace>
  <string key="concept:name" value="esempio"/>
  <string key="description" value="PaperOne"/>
  <event>
    <string key="destinazione" value="Roma"/>
    <int key="importo" value="100"/>
    <string key="org:resource" value="Hind"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="concept:name" value="A"/>
    <date key="time:timestamp" value="2012-01-15T16:15:49.000+01:00"/>
  </event>
  <event>
    <string key="org:resource" value="Hind"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="A"/>
    <date key="time:timestamp" value="2012-01-15T17:08:24.000+01:00"/>
  </event>
  <event>
    <string key="destinazione" value="Roma"/>
    <string key="StazioneArrivo" value="termini"/>
    <string key="org:resource" value="Hind"/>
    <string key="lifecycle:transition" value="start"/>
    <string key="concept:name" value="B"/>
    <date key="time:timestamp" value="2012-01-15T18:15:49.000+01:00"/>
  </event>
  <event>
    <string key="org:resource" value="Hind"/>
    <string key="lifecycle:transition" value="complete"/>
    <string key="concept:name" value="B"/>
    <date key="time:timestamp" value="2012-01-15T19:08:24.000+01:00"/>
  </event>
</trace>
```

Listing 6.3: Frammento event log

L'attributo `destinazione` viene registrato sia in corrispondenza dell'evento "A_start" che in corrispondenza di "B_start". Il plugin in questo caso considera i due attributi diversi e provvede a ridenominarli come `destinazione_1` e `destinazione_2`, ovvero aggiunge una indicizzazione numerica.

Sostanzialmente il plugin, con l'ausilio delle strutture dati elencate di seguito, esegue una scansione mantenendo traccia degli attributi che man mano vengono riscontrati:

- `litAttributesValue`: è una mappa contenente l'associazione tra ogni attributo nominale (considerato senza l'indicizzazione numerica descritta prima), ed i possibili valori che può assumere. Ad esempio contiene l'associazione tra l'attributo `destinazione` e tutti i valori che assume riscontrati nell'event log.
- `litAttributes`: è un insieme contenente gli attributi di tipo nominale considerando l'indicizazzione. Ad esempio contiene gli attributi `destinazione_1`, `destinazione_2`.
- `floatAttributes`: rappresenta l'insieme degli attributi di tipo numerico considerando l'indicizzazione.
- `boolAttributes`: rappresenta l'insieme degli attributi di tipo booleano considerando l'indicizzazione.

Vale la pena osservare che per quanto riguarda gli attributi nominali, vengono mantenute due strutture dati: una mappa che tiene l'associazione tra l'attributo non indicizzato ed i possibli valori, ed un insieme per tenere traccia di tutti gli attributi compresa l'indicizzazione. Questa scelta è dovuta al fatto che i valori possibili per un attributo nominale sono indipendenti dall'indicizzazione, è utile mantenere quindi una mappa separatamente solo per la memorizzazione dei valori.

Durante la scansione, ogni volta che si incontra un attributo ne viene verificato il tipo e poi aggiunto all'insieme appropriato (se l'attributo è di tipo nominale si tiene traccia anche del valore). Una volta raccolti gli attributi, viene creata una struttura dati di tipo `Instances` e nome `Wekainstances`, caratterizzata dagli attributi ottenuti con la scansione e dall'attributo target sulla conformance. `Wekainstances` rappresenta il data set che sarà restituito in output. A questo punto, con una scansione, il plugin recupera per ogni traccia i valori degli attributi riportati in quella traccia, andando così a formare un record del data set a cui viene aggiunto un valore boolenao relativo

alla conformance della traccia medesima. Tale record è rappresentato da una struttura dati di tipo `Instance` che alla fine dell'analisi della traccia viene aggiunto a `Wekainstances`.

Occorre osservare che questo plugin evita di fare preprocessing dei dati; considera validi tutti gli attributi riscontrati e fornisce un data set i cui attributi possono essere preprocessati con l'ausilio degli strumenti messi a disposizione da Weka (sezione 4.2.1), ad esempio tramite i filtri oppure in modo interattivo usando l'arff viewer.

Per dare maggiore concretezza alla parte logica del metodo `generateWekaInstances`, di seguito è presentato l'algoritmo implementato in pseudocodice. Questo è suddiviso logicamente in tre parti: raccolta degli attributi, la loro trasformazione in attributi di weka, ed infine la creazione del data set.

Algorithm 1 generateWekaInstances: raccolta degli attributi

```
1:  procedure GENERATEWEKAINSTANCES(log, conformance)
2:      for all Trace t in log do
3:          countAttributes ← newMap⟨String, int⟩          ▷ per indicizzare
4:          for all Event ev in t do
5:              Attributes ← ev.getAttributes()                ▷ è una mappa
6:              for all (K, v) in Attribues do          ▷ (K, v) = (chiave, valore)
7:                  countAttributes.add(K, countAttributes.get(K) + 1)
8:                  if v instanceof nominalAttribute then
9:                      litAttributes.add(indicizza(K, countAttributes.get(K)))
10:                     litAttributesValues.addValue(K, Att)
11:                 end if
12:                 if v instanceof numericAttribute then
13:                     floatAttributes.add(indicizza(K, countAttributes.get(K)))
14:                 end if
15:                 if v instanceof booleanAttribute then
16:                     boolAttributes.add(indicizza(K, countAttributes.get(K))
17:                 end if
18:             end for
19:         end for
20:     end for
```

Algorithm 2 generateWekaInstances: Trasformazione in attributi di Weka

21: $wAttrs \leftarrow newArray\langle WekaAttributes\rangle$ ▷ contiene attributi trasformati
22: **for all** Attribute a **in** $litAttributes$ **do**
23: $wa \leftarrow CreaLittWekaAttribute(a, litAttributesValues.get(a))$
24: $wAttrs.add(wa)$
25: **end for**
26: **for all** Attribute a **in** $floatAttributes$ **do**
27: $wa \leftarrow CreaNumeWekaAttribute(a)$
28: $wAttrs.add(wa)$
29: **end for**
30: **for all** Attribute a **in** $boolAttributes$ **do**
31: $wa \leftarrow CreaBoolWekaAttribute(a)$
32: $wAttrs.add(wa)$
33: **end for**
34: $wAttrs.add(createConformanceWekaAttribute())$

Algorithm 3 generateWekaInstances: Creazione data set

35: $Wekainstances \leftarrow newInstances(wAttrs)$ ▷ data set da costruire
36: **for all** Trace t **in** log **do**
37: $i \leftarrow newInstance(wAttrs)$ ▷ istanza relativa alla traccia
38: $trAttributes \leftarrow newMap\langle String, String\rangle$ ▷ dati della traccia
39: $countAttributes \leftarrow newMap\langle String, int\rangle$ ▷ per indicizzare
40: **for all** Event ev **in** t **do**
41: $Attributes \leftarrow ev.getAttributes()$
42: **for all** (K, v) **in** $Attributes$ **do**
43: $countAttributes.add(K, countAttributes.get(K) + 1)$
44: $trAttributes.add(indicizza(K, countAttributes.get(K)), v.toString())$
45: **end for**
46: **end for**

Algorithm 4 generateWekaInstances: Creazione data set (segue)

```
47:         for all Attribute a in LittAttributes do
48:             i.add(a, (trAttributes.get(a) or '?'))
49:         end for
50:         for all Attribute a in floatAttributes do
51:             i.add(a, (trAttributes.get(a) or '?'))
52:         end for
53:         for all Attribute a in boolAttributes do
54:             i.add(a, (trAttributes.get(a) or '?'))
55:         end for
56:         trConformance ← Conformance.getConformance(t)
57:         i.add(conformance, trConformance)
58:         Wekainstances.add(i)
59:     end for
60:     return Wekainstances
61: end procedure
```

Estensioni dell'algoritmo

E' opportuno osservare, che la parte dell'algoritmo relativa alla raccolta degli attributi, riflette la scelta effettuata qualora venissero incontrati attributi con lo stesso nome ma riferiti da eventi differenti. Questo si traduce infatti con l'aggiunta di una nuova struttura dati, `countAttributes`, che mantiene l'associazione tra ogni attributo e la quantità di volte che viene riferito. Se siamo invece nello scenario in cui l'attributo denota la stessa entità per tutte le attività, l'algoritmo subisce una leggera modifica: la struttura `countAttributes` e la funzione `indicizza` perdono di utilità e semplicemente l'attributo viene aggiunto alla struttura dati appropriata.

Con riferimento alla estensione discussa nella sezione 5.3 per l'analisi di performance, vale la pena osservare che, per fornire una realizzazione di quanto descritto, bisogna introdurre delle modifiche a questo algoritmo. Ad esempio, occorre prevedere una fase che si occupa di individuare quali sono le piazze della rete di Petri del processo in analisi i cui tempi di sincronizzazione sono rilevanti per la classificazione, oltre ad una fase che si occupa invece della discrettizzazione dei valori temporali.

6.2.2 Plugin: Generate Weka Classifier

In sostanza questo plugin ha l'obiettivo di generare un classificatore a partire da un data set. Il modello di classificazione scelto è un albero di decisione come è stato descritto nel capitolo 5. Il frammento riportato nel lising 6.4 mostra l'intestazione di `Generate Weka Classifier`.

```
public class GenerateClassifier {

    @Plugin(name = "Generate Weka Classifier",
        parameterLabels = { "Istances" },
        returnLabels = { "Weka Classifier" },
        returnTypes = { Classifier.class }  )
    @UITopiaVariant(
        uiLabel="Generate Weka Classifier ",
        affiliation = "University of Pisa",
        author = "Hind",
        email = "chfouka@di.unipi.it")

    public Object generateclassifier(PluginContext contex, Instances data ){

    /* Parte logica del plugin */

    return decisiontree;
    }
}
```

Listing 6.4: Intestazione Generate Weka Classifier

Il plugin prende in input, oltre al `context`, il parametro:

- `instances`: è di tipo `Instances` e rappresenta il data set su cui avviene l'addestramento del classificatore.

l'output invece è dato dal parametro d'uscita:

- `decisiontree`: è di tipo `Classifier` come viene specificato dall'annotazione `@Plugin` ed ovviamente rappresenta il modello di classificazione costruito.

Il codice che implementa questo plugin utilizza in modo massiccio l'API fornita da Weka (sezione 4.2.1). In pratica, dopo aver istanziato un classificatore di tipo `J48` e specificato qual'è l'attributo del data set da considerare come attributo target, viene effettuata l'addestramento dell'albero `decisiontree` mediante una chiamata del tipo:
`decisiontree.buildClassifier(instances);`

Partendo da un event log (rappresentato da un oggetto di tipo `XLog`) e dai risultati di conformance (un oggetto di tipo `TotalConformaceResult`),

è possibile eseguire il plugin `Weka Instances with Conformance` ottenendo in questo modo in output un oggetto di tipo `Instances`. Tale risultato può costituire l'input del plugin `Generate Weka Classifier` per ottenere un modello di classificazione attinente al data set estratto dall'event log di partenza. Ciò è permesso perché all'interno del framework ProM 6, come è descritto in sezione 4.1.1, questi oggetti vengono considerati delle risorse, possono quindi essere passate in input a qualsiasi altro plugin purche i tipi siano compatibili.

6.2.3 Plugin: Generate Instances to classify

Questo plugin ha lo scopo di generare un insieme di istanze che si vuole classificare in base alla conformance. Il data set è rappresentato da una struttura dati di tipo `Instances` contenente delle istanze cui valore dell'attributo target è sconoscituo. Il frammento 6.5 mostra l'intestazione di `Generate Instances to classify`.

```
public class GenerateInstancesToClassify {
  @Plugin(name = "Generate Instances to classify",
      parameterLabels = { "Log" },
      returnLabels = { "Weka Instances To classify" },
      returnTypes = { Instances.class }  )
  @UITopiaVariant(
      uiLabel="Weka Instances to classify",
      affiliation = "University of Pisa",
      author = "Hind",
      email = "chfouka@di.unipi.it")

  public Object generateWekaInstancesToclassify(PluginContext context, XLog
      log) {

  /* Parte logica */

  return ist_toclassify;
  }
}
```

Listing 6.5: Intestazione Generate Instances to classify

Come mostra il listing, il plugin prende in input il seguente paramentro:

- `log`: è di tipo `XLog`, rappresenta l'event log dal quale devono essere estratte le istanze che si vogliono classificare.

In output invece viene restituito un oggetto rappresentato dal seguente paramentro di uscita:

- `ist_toclassify`: è di tipo `Instances` e specifica il data set di cui si vuole predire la classe di appartenenza.

Questo plugin adotta sostanzialmente la stessa logica impiegata da `Weka Instances with Conformance` descritto in sezione 6.2.1. Questo perché, anche in questo caso, si vuole costruire un data set a partire da un event log. Vengono impiegate quindi strutture dati simili, sia nella fase in cui si vuole capire quali attributi caratterizzano il log in questione, sia nella fase della creazione del data set vero e proprio (cioè dell'oggetto `ist_toclassify`). L'unica sostanziale differenza risiede nel fatto che, in questo caso, non si ha nessuna informazione sull'esito di conformance delle tracce contenute nel log (il valore dell'attributo di conformance vale `'?'` per tutte le tracce). L'obiettivo infatti è quello di preparare un insieme di record adatti ad essere classificati con l'ausilio di un modello di classificazione.

E' utile osservare che in questo plugin è stato di grande aiuto il fatto che Weka è in grado di gestire i casi in cui un valore, relativo ad un certo attributo, possa essere sconosciuto per un record.

6.2.4 Plugin: Weka classify Instances

Dato un insieme di istanze, questo plugin ha lo scopo di predire la classe di appartenza di tale istanze. L'intestazione presentata nel listing 6.6 mostra quali sono i paramentri in ingresso ed uscita.

```
public class DMclassifyPlugin {
  @Plugin(name = "Weka classify Instances",
      parameterLabels = { "Istances", "Classifier" },
      returnLabels = { "Weka Instances classified" },
      returnTypes = { Instances.class }  )
  @UITopiaVariant(
      uiLabel="Weka classify Instances",
      affiliation = "University of Pisa",
      author = "Hind",
      email = "chfouka@di.unipi.it")

  public Instances classifyIstances(PluginContext context, Instances data,
      Classifier classifier){

      /*Parte logica*/


      return ist_classified;
  }
}
```

Listing 6.6: Intestazione Weka classify Instances

I parametri in ingresso sono:

- `data`: di tipo `Instances`, rappresenta l'insieme delle istanze che si vogliono classificare.
- `classifier`: di tipo `Classifier`, specifica il modello di classificazione che si vuole impiegare per la predizione. Nel nostro caso si tratta di un albero di decisione.

Il plugin restituisce in output:

- `ist_classified`: di tipo `Instances`, specifica l'insieme delle istanze che sono state classificate per mezzo del modello di classificazione ricevute in ingresso.

Brevemente, questo plugin provvede alla creazione dell'oggetto `ist_classified` a partire dalle istanze avute in ingresso (`data`) ed a specificare l'attributo target. In seguito, sfruttando l'API messa a disposizione dal package "Weka.classifiers", viene impiegato il classificatore avuto in input per classificare le istanze in `ist_classified`.

Dato un event log E_1 contenente delle tracce di cui l'esito di conformance è noto, è possibile costruire un modello di classificazione C impiegando i plugin `Weka Instances with Conformance` e `Generate Weka Classifier`. Successivamente, se si hanno delle tracce in un event log E_2 di cui è sconosciuta la classe di appartenenza, si può costruire un insieme di istanze mediante il plugin `Generate Instances to classify` e provvedere alla loro classificazione impiegando il classificatore C attraverso il plugin `Weka classify Instances`. Occorre osservare che, affinché tutto ciò abbia senso, E_1 ed E_2 devono fare riferimento allo stesso modello di processo di business.

6.2.5 Plugin di gestione delle risorse

In questa sezione sono descritti una serie di plugin utili per permettere la gestione delle risorse in ProM 6 (importazione ed esportazione di file e visualizzazione di risorse).

Export Instances to Arff file

Come si deduce dal titolo, questo plugin è stato sviluppato per poter esportare un oggetto (risorsa) di tipo `Instances` su disco. In particolare viene esportata sotto forma di un file Arff descritto in 4.2.1.

L'intestazione relativa ai plugin di esportazione è leggermente diversa rispetto agli altri. Vengono usate ulteriori annotazioni per informare il framework della natura del plugin come mostra Il listing 6.7.

```
@Plugin(name = "Export Instances to Arff File", parameterLabels = { "
    instances", "file" }, returnLabels = {}, returnTypes = {},
    userAccessible = true)
@UIExportPlugin(description = "Arff files", extension = "arff")
public class ExportWekaArff {

  @UITopiaVariant(affiliation = UITopiaVariant.EHV, author = "Hind", email =
      "chfouka@di.unipi.it")
  @PluginVariant(requiredParameterLabels = { 0, 1 }, variantLabel = "Export
      Instace to Arff File")
  public void export(PluginContext context, Instances dataset, File file){

    /*Parte logica*/

  }
}
```

Listing 6.7: Intestazione Export Instances

L'annotazione `@UIExportPlugin` dichiara che si tratta di un plugin di esportazione e specifica:

- `description`: permette di fornire una descrizione di quanto viene esportato.
- `extension`: specifica l'estensione del file che viene esportato.

Invece l'annotazione `@PluginVariant` descrive una variante del plugin che prende in input solo i parametri nella posizione 0 ed 1 nella lista descritta da `parameterLabels`, ovvero prende in input `instances` e `file`.

La parte logica del plugin, implementata dal metodo `export`, utilizza dei metodi messi a disposizione dal package "Weka.core" per la serializzazione di oggetti di tipo `Instances`, oltre che ai due package Java "java.io.File" e "java.io.FileOutputStream" per poter gestione del file di esportazione.

Import Instances from file Arff

Questo plugin è praticamente analogo a quello descritto per l'esportazione in file Arff, con l'obiettivo però di rendere possibile l'importazione di un data set da un file Arff sotto forma di una risorsa di tipo `Instances` in ProM.

```
@Plugin(name = "Import Instances from file Arff", parameterLabels = { "
    filename" }, returnLabels = { "Instances" }, returnTypes = { Instances.
    class })
@UIImportPlugin(description = "Weka instances file Arff", extensions = { "
    arff" })
public class ImportWekaInstances  extends AbstractImportPlugin {

  @PluginVariant(requiredParameterLabels = { 0 }, variantLabel = "Import
      Weka instances file Arff")
  public Instances importFromStream(PluginContext context, InputStream input
      , String filename,
      long fileSizeInBytes) throws Exception {

    /*Parte logica plugin*/
  }
}
```

Listing 6.8: Intestazione Import Instances

La classe che contiene il plugin deve estendere, come tutti i plugin di importazione, la classe astratta `AbstractImportPlugin` messa a disposizione da ProM. L'annotazione `@UIImportPlugin` informa il framework che si tratta di un plugin di importazione specificando la `description` e la `extension` così come nell'annotazione `@UIExportPlugin`. L'annotazione `@PluginVariant` descrive una variante del plugin che richiede solo il parametro in posizione 0, ovvero il `filename`.

La parte logica del plugin utilizza la classe `DataSource` contenuta nel package "Weka.core". Questa mette a disposizione una serie di metodi per potere costuire un data set a partire da un file di input.

Export Classifier to file Model

Una volta costruito un classificatore, è utile poterlo salvare su disco in modo da permetterne l'utilizzo in altri momenti, questo è reso possibile tramite il plugin `Export Classifier to Model` che ha lo scopo di esportare un classificatore sotto forma di un file di estensione `model` come specifica Weka.

```
@Plugin(name = "Export Classifier to file Model", parameterLabels = { "
    Classifier", "File" }, returnLabels = {}, returnTypes = {},
    userAccessible = true)
@UIExportPlugin(description = "Model files", extension = "model")
public class ExportClassifierToModel {
```

```
  @UITopiaVariant(affiliation = UITopiaVariant.EHV, author = "Hind", email =
      "chfouka@di.unipi.it")
  @PluginVariant(requiredParameterLabels = { 0, 1 }, variantLabel = "Export
      Classifier to Model File")

  public void exportModel(PluginContext context, Classifier classifier, File
      file){
    /* Parte logica */
  }
}
```

Listing 6.9: Intestazione Export Classifier

Essendo un plugin di esportazione, l'intestazione in listing 6.9 è analoga a quella di `Export Instances to Arff file`. La parte logica invece provvede ad una serializzazione del classificatore passato in ingresso al plugin con l'ausilio dei metodi messi a disposizione dal package "Weka.core".

Import Classifier from file Model

E' un plugin molto simile a quello per l'esportazione di un classificatore, con la differenza che questo ha il compito contrario, ovvero l'importazione di un classificatore a partire da un file di esentesion `model` che lo rappresenta (deserializzazione).

```
@Plugin(name = "Import Weka Classifier model file", parameterLabels = { "
    Filename" }, returnLabels = { "Classifier" }, returnTypes = { Classifier
    .class })
@UIImportPlugin(description = "Weka Classifier model file", extensions = { "
    model" })

public class ImportWekaModel  extends AbstractImportPlugin {

public Classifier importFromStream(PluginContext context, InputStream input,
     String filename, long fileSizeInBytes) {

  /*Parte logica del plugin*/

    return classifier;
  }
}
```

Listing 6.10: Intestazione Import Classifier

Essendo di importazione, la classe che realizza il plugin estende la classe `AbstractImportPlugin` e provvede a fornire un'implementazione del metodo `importFromStream`. Le annotazioni riportate nell'intestazione nel listing 6.10 sono simili a quelle descritte per il plugin `Import Instances from file Arff`. Vale anche in questo caso che i metodi utilizzati nella parte logica per la deserializzazione sono forniti dai package "Weka.core".

Plugin: Visualize data set

Nei plugin di gestione delle risorse in ProM 6, rientrano anche quelli che permettono alla Vista Workspace esplorata in 4.1.1 di dare una visualizzazione grafica alle risorse. Il plugin `Visualize data set` permette di visualizzare una risorsa di tipo `Instances`.

```
@Visualizer
@Plugin(name = "Visualize data set", parameterLabels = "Visualizze Instances
    Weka", returnLabels = "Instances Weka Visualized", returnTypes =
    JComponent.class)

public class VisualizzeInstances {

  @PluginVariant(requiredParameterLabels = { 0 },variantLabel="Visualizer
      Instances Weka")
  @UITopiaVariant(affiliation = UITopiaVariant.EHV, author = "Hind", email =
      "chfouka@di.unipi.it")
  public JComponent visualize(UIPluginContext context, Instances instace) {

    /*Parte logica*/

    return panel;
  }
```

Listing 6.11: Intestazione Visualize data set

Anche per i plugin di visualizzazione, esite l'annotazione `@Visualiser` in grado di informare il framework che si tratta appunto di un plugin di visualizzazione come mostra il listing 6.11. Va detto che ProM 6 si basa sulla libreria `Swing` per l'implementazione dell'interfaccia grafica. Il metodo `visualize`, che implementa la parte logica del plugin, provvede dunque a costruire un oggetto di tipo `JComponet` definito nel package "javax.swing.JComponent". In particolare viene costruito un `JPanel` in grado di visualizzare un oggetto di tipo `Instances`.

Plugin: Visualize decision tree

Come da titolo, il plugin permette alla Vista Workspace di ProM 6 di visualizzare un albero di decisione che mostra le regole individuate nei dati.

```
@Visualizer
@Plugin(name = "Visualizer Weka Desicion Tree", parameterLabels = "
    Visualizze decision tree Weka", returnLabels = "Decision Tree Weka
    Visualized", returnTypes = JComponent.class)

public class VisualizeDecisionTree {
  @PluginVariant(requiredParameterLabels = { 0 },variantLabel="Visualizer
      Instances Weka")
```

```
  @UITopiaVariant(affiliation = UITopiaVariant.EHV, author = "Hind", email =
      "chfouka@di.unipi.it")
  public JPanel visualize(UIPluginContext context, Classifier classifier)
      throws Exception {

    /*Parte logica*/

    return panel;

  }
}
```

Listing 6.12: Intestazione Visualize decision tree

Il metodo `visualize` anche in questo caso restituisce un oggetto `JPanel` in grado di visualizzare un albero di decisione. Per fare questo si appoggia alle classi fornite dal package "weka.gui.treevisualizer" usato dal software Weka appositamente per la visualizzazione degli alberi di decisione.

Plugin	Abbreviazione	Input	Output	Desrizione
Weka Instances with Conformance	WIWC	Event log, Conformance	Data Set	Costruisce una data set a partire dalle tracce dell'event log e dal risultato di conformance
Generate Weka Classifier	GWC	Data set	Classificatore	Costruisce un albero di decisione a partire dal data set in input.
Generate Instances to Classify	GITC	Event log	Data set	Costruisce un data set da classificare a partire dalle tracce dell'event log preso in input.
Weka classify Instances	WCI	Data set, Classificatore	Data set	Classifica le istanze nel data set di input impiegando il classificatore in ingresso.
Export Instances to Arff file	Export Arff	Data set, File		Esporta il data set sotto forma di un file di estensione Arff.
Import Instances from file Arff	Import Arff	File	Data set	Importa le istanze contenute nel file di input sotto forma di un data set
Export Classifier to file Model	Export Model	Classificatore, File		Esporta il classificatore su un file di estensione Model (serializzazione).
Import Classifier from file Model	Import Model	InputStream, File		Importa il classificatore dal file preso in ingresso (deserializzazione).
Visualize data set	Visualize	data set	Componente grafico	Permette la visualizzazione di un data set all'interno di ProM.
Visualize decision tree	Visualize	Classificatore	Componente grafico	Permette la visualizzazione di un albero di decisione all'interno di ProM.

Tabella 6.1: Plugin sviluppati

Capitolo 7

Conclusioni e sviluppi futuri

L'obiettivo del tirocinio era quello di estendere le tecniche di analisi dei processi di business, rendendo possibile sfruttare le ingenti quantità dei dati registrati negli event log. Tale obiettivo è stato raggiunto tramite la sperimentazione di un nuovo approccio basato sia sui risultati apportati dalle tecniche già esistenti, sia su tecniche di data mining. L'integrazione di questi due strumenti ha permesso di evidenziare l'influenza che i dati possono avere sul comportamento reale di un processo. La tecnica di analisi dei dati impiegata, ovvero la classificazione, permette inoltre di fornire indicazioni utili per potere attuare delle correzioni, qualora il comportamento del processo si discosti molto dal modello.

Il risultato del lavoro di tirocinio può essere riassunto da un processo di analisi i cui passi sono:

1. A partire da un modello di processo di business espresso in BPMN viene effettuata la trasformazione del modello in una rete di Petri adatta per un'analisi formale.

2. Viene eseguito l'algoritmo di analisi log replay sulla rete di Petri ottenuta e sull'event log relativo al processo in questione.

3. A partire dai risultati di conformance (ottenuti con il log replay) e dai dati nell'event log, viene formulato un problema di classificazione.

4. Impiegando gli algoritmi di data mining viene costruito un classificatore in grado di descrivere le regole nei dati in corrispondenza delle quali si verificano errori di conformance nelle istanze.

5. Il classificatore ottenuto al passo precedente è utilizzabile in senso predittivo per scoprire la conformance delle nuove istanze di processo che si presentano.

Uno degli sviluppi futuri di questo lavoro consiste nella realizzazione dell'estensione dell'approccio basato su classificazione prendendo in considerazione le metriche di performance come è discusso in 5.3. Un'alta tappa invece consiste nella speriamentazione di quanto realizzato prendendo in esame processi ed event log che caratterizzano contesti organizzativi reali. Ciò comporterà sicuramente un'analisi preliminare dei dati contenuti negli event log per renderli utilizzabili dall'approccio sperimentato.

Il tirocinio ha permesso l'esplorazione di nuovi campi dell'informatica che erano sconosciuti precedentemente alla tirocinante, quali le discipline del data mining e del process mining. Inoltre, il tirocinio è stato molto formativo anche dal punto di vista tecnico; ha permesso di acquisire un'ulteriore esperienza di programmazione in linguaggio Java usando Eclipse come ambiente di sviluppo, di sperimentare come viene ampliato un framework esistente mediante plugins, di conoscere ed imparare ad usare tool mai sperimentati in precedenza sia relativi al process mining che al data mining. Infine, molto interessante è stato il lavoro di tirocinio visto come un'esperienza svolta all'interno del Dipartimento di Informatica, l'integrazione e la collaborazione all'interno di un gruppo di ricerca hanno permesso infatti di prendere atto di come vengono affrontate nuove problematiche in un ambiente di ricerca.

Capitolo 8

Tutorial

Flusso operativo del framework di analisi

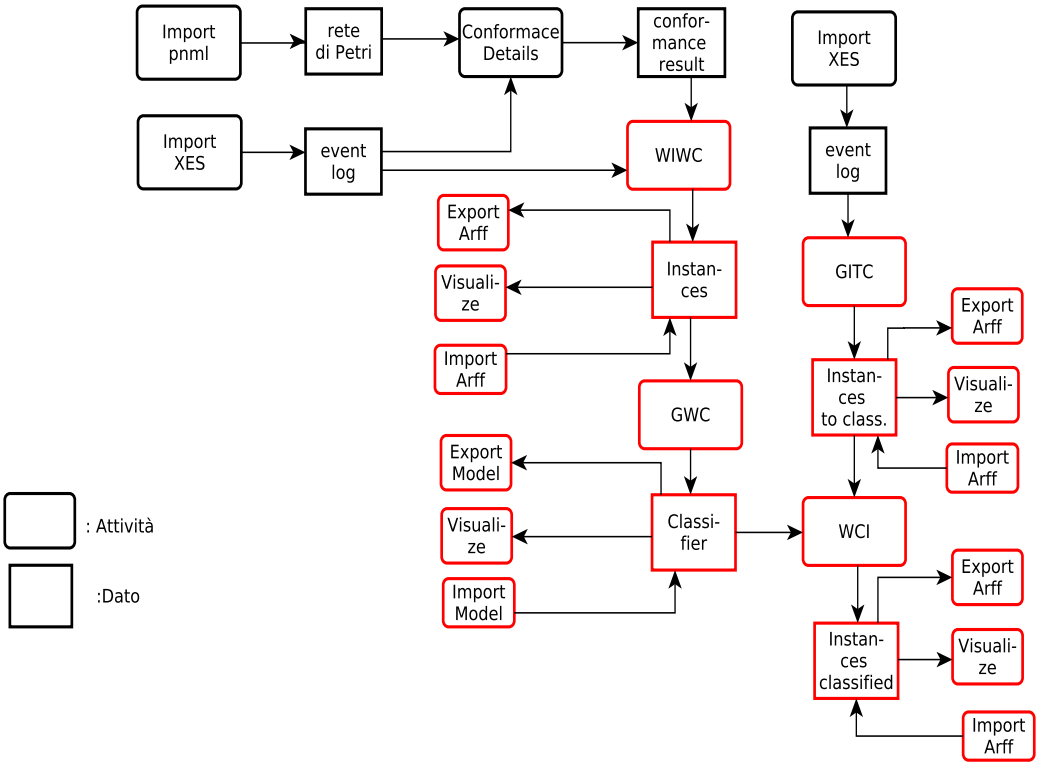


Figura 8.1: Flusso operativo

Le ultime pagine di questo elaborato sono dedicate a mostrare l'utilizzo del framework di ananlisi realizzato. Questo viene fatto illustrando come si possono usare i plugin sviluppati sul prototipo di processo descritto nel capitolo 5 nella sezione 5.1. La figura 8.1 mostra il flusso operativo che descrive l'ordine delle attività da effettuare. Il colore nero è usato per le componenti software già esistenti, mentre il colore rosso denota le componenti aggiunte durante il lavoro di tirocinio. Sono impiegate inoltre delle abbreviazioni per indicare i nomi dei plugin riassunti in tabella 6.1.

Siano:

- `BugFix.pnml`: il file contenente la descrizione della rete di Petri rappresentante il processo di Business "FixBug".
- `logBugFix.xes`: il file contenente l'event log relativo al processo.

Per potere disporre dei file `BugFix.pnml` e `logBugFix.xes` come risorsa, occorre importarli in ProM, e questo avviene dalla vista workspace (esiste un apposito bottone per fare ciò). La figura 8.2(a) mostra il risultato di questa operazione.

Dopo l'operazione di importazione, per ricavare informazioni sulla conformance delle tracce in `logBugFix.xes`, è possibile eseguire il plugin `ConformanceDetails` che realizza l'analisi di conformance implementando l'algoritmo log replay. Tale plugin si presenta come un'azione nella vista action di ProM (figura 8.2(b)) e richiede come input una risorsa `Event Log` ed una `Petrinet`, mentre restituisce in output una risorsa di tipo `TotalConformance-Resutl`. Il risultato dell'esecuzione di questo plugin sui file `logBugFix.xes` e `logBugFix.xes` è mostrato nella vista view come riporta la figura 8.2(c).

A questo punto, si hanno a disposizione tutte le risorse necessarie per l'esecuzione del plugin `Weka Instances with Conformance` che si presenta come un'azione nella vista action (figura 8.2(d)). Le risorse da passare in input sono quindi: l'event log importato ed il risultato di conformance ottenuto al passo precedente. Il plugin rilascia in output una risorsa di tipo `Instances` visualizzata nella figura 8.2(e) grazie al plugin `Visualize data set`.

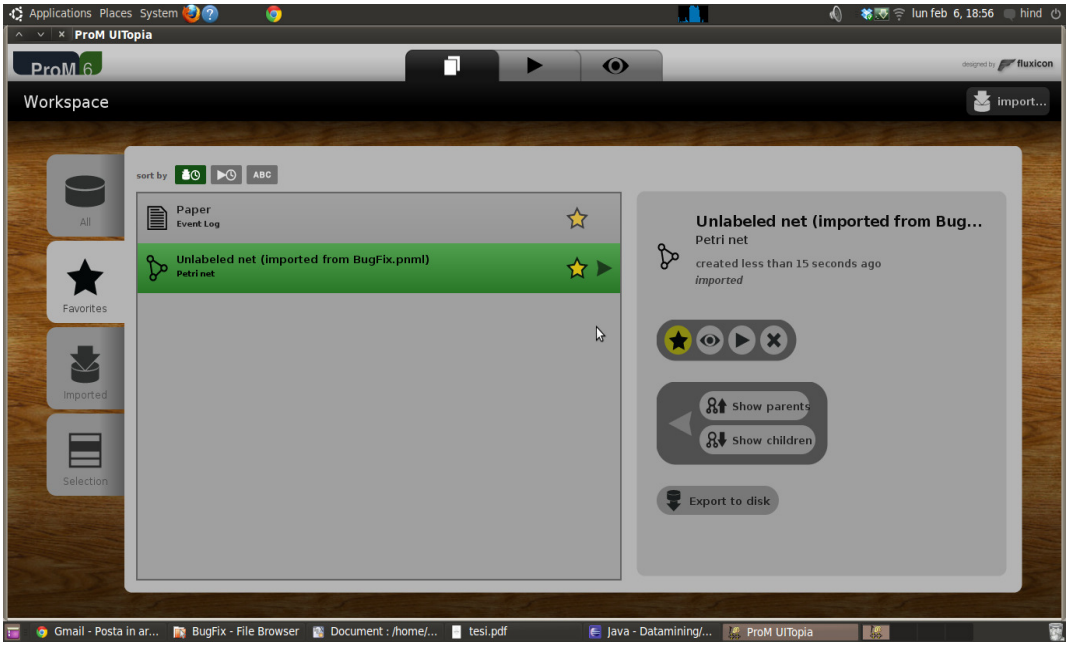
La risorsa rilasciata al passo precedente permette l'esecuzione del plugin `Generate Weka Classifier`. Tale plugin, come è mostrato dalla vista action in figura 8.2(f) richiede una risorsa di tipo `Instances` e rilascia in output una

risorsa di tipo `Classifier`. L'output è visualizzato nella vista view grazie al plugin `Visualize decision tree` ed è mostrato nella figura 8.2(g). La figura 8.2(h) mostra invece l'insieme delle risorse presenti nel workspace di ProM.

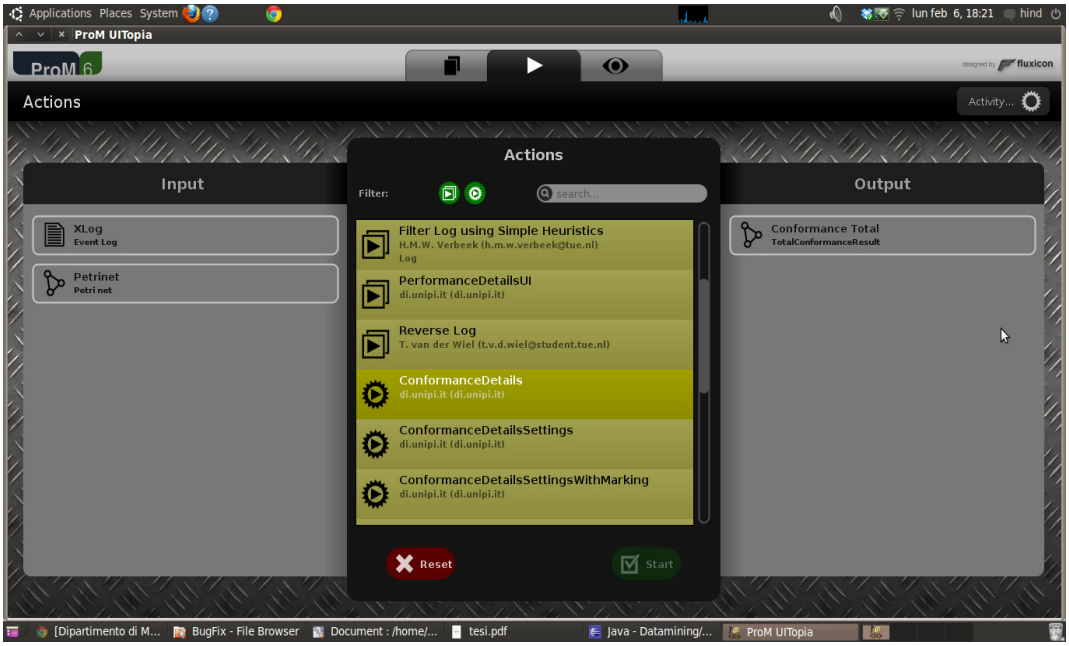
Supponniamo a questo punto di avere:

- `logBugFix_tocheck`: un file contenente un event log con delle tracce delle quali si vuole predire la conformità al modello di processo.

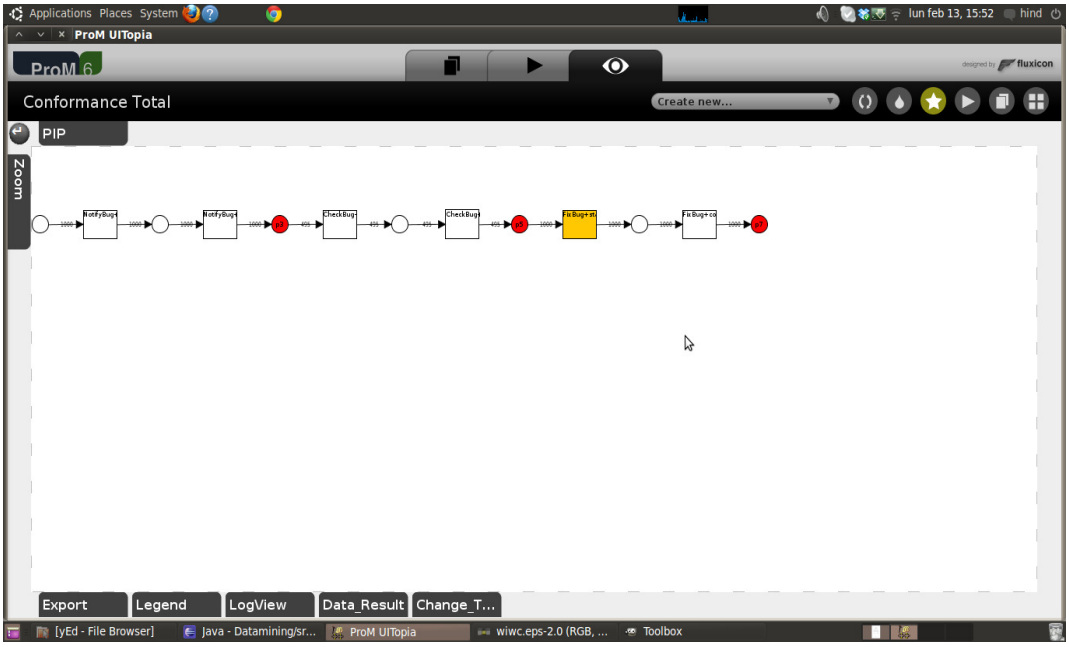
Dopo l'operazione di importazione dell'event log all'interno di PorM, è possibile eseguire il plugin `Generate Instances to classify` per l'estrazione delle istanze in un data set di tipo `Instances` (risultato mostrato in figura 8.2(j)). A questo punto, per potere classificare queste istanze, è sufficiente chiamare il plugin `Weka classify Instances` passando come risorse in input le istanze stesse e la risorsa `Classifier` prodotta precedentemente. In figura 8.2(i) è mostrato il plugin `Weka classify Instances` nella vista action, mentre nella figura 8.2(k) è mostrato il data set classificato.



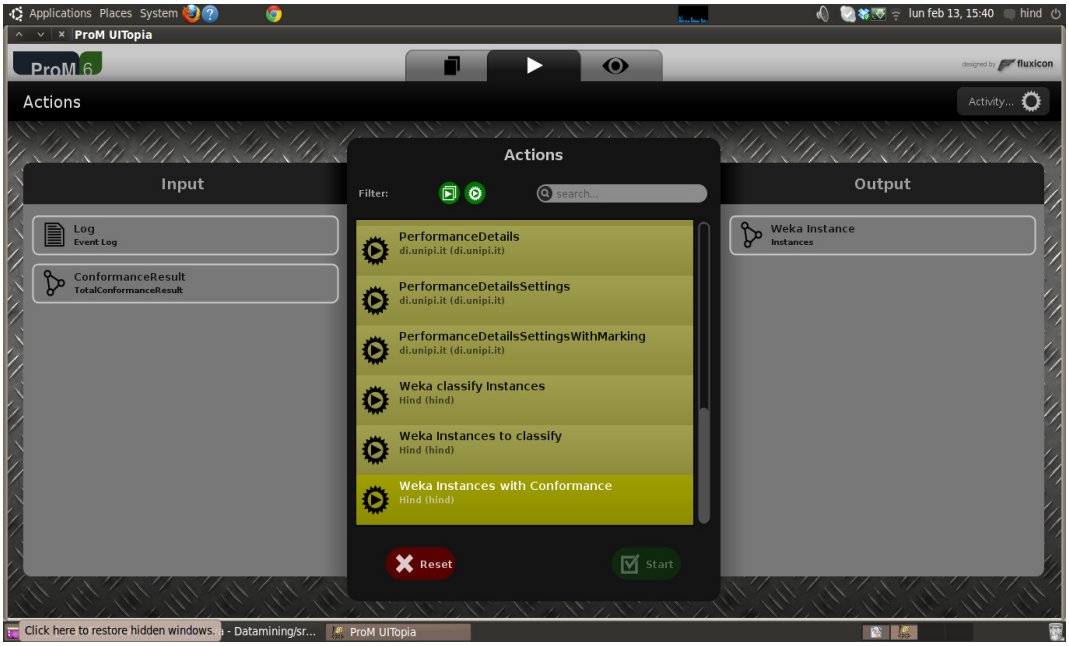
(a) Vista workspace: risorse importate



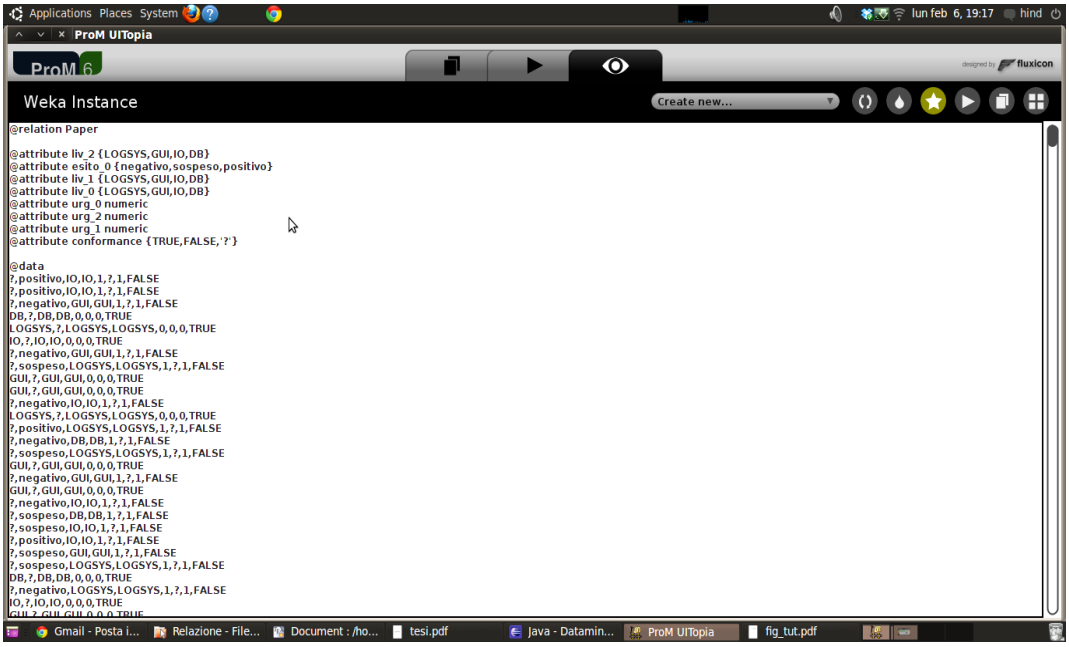
(b) Vista action: `ConformanceDetails`



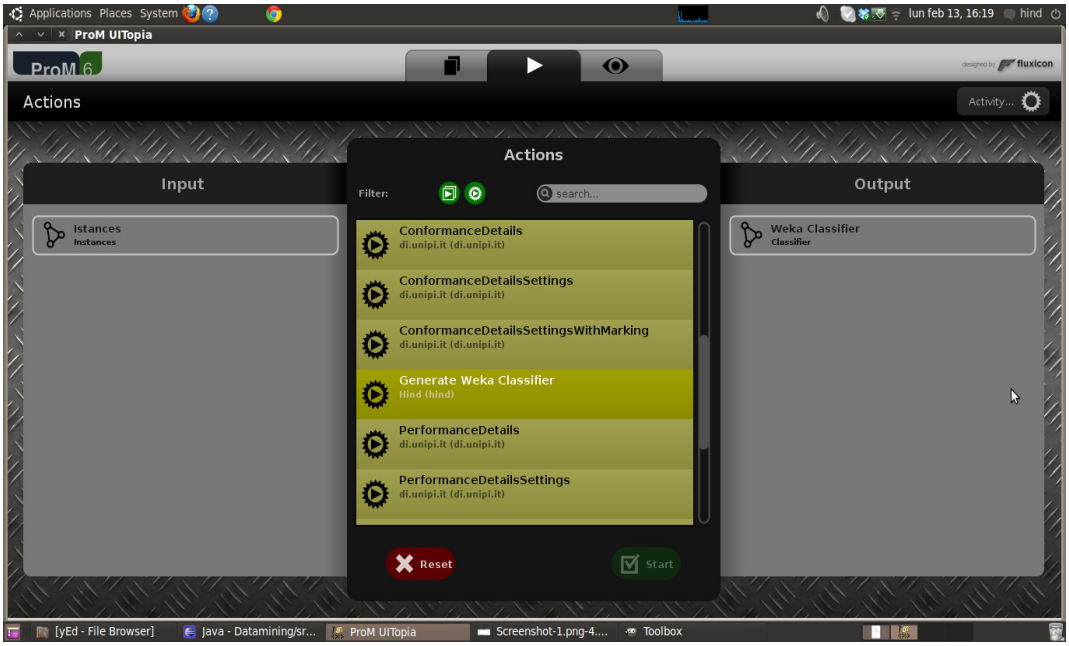
(c) Vista view: `TotalConformanceRisult`



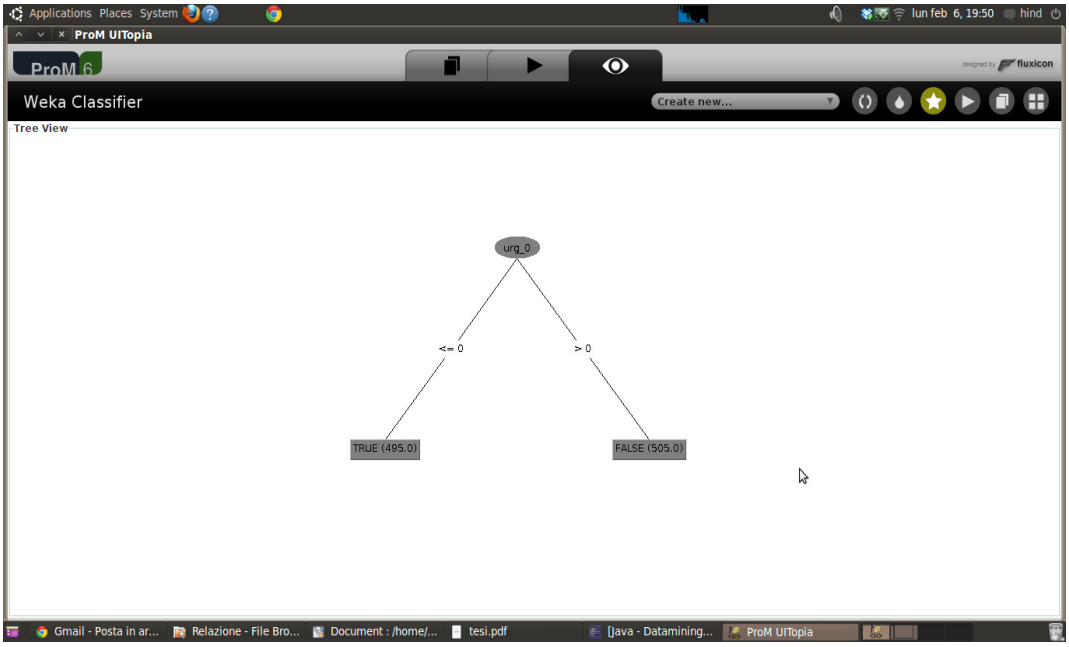
(d) Vista Action: plugin `Weka Instances with Conformance`



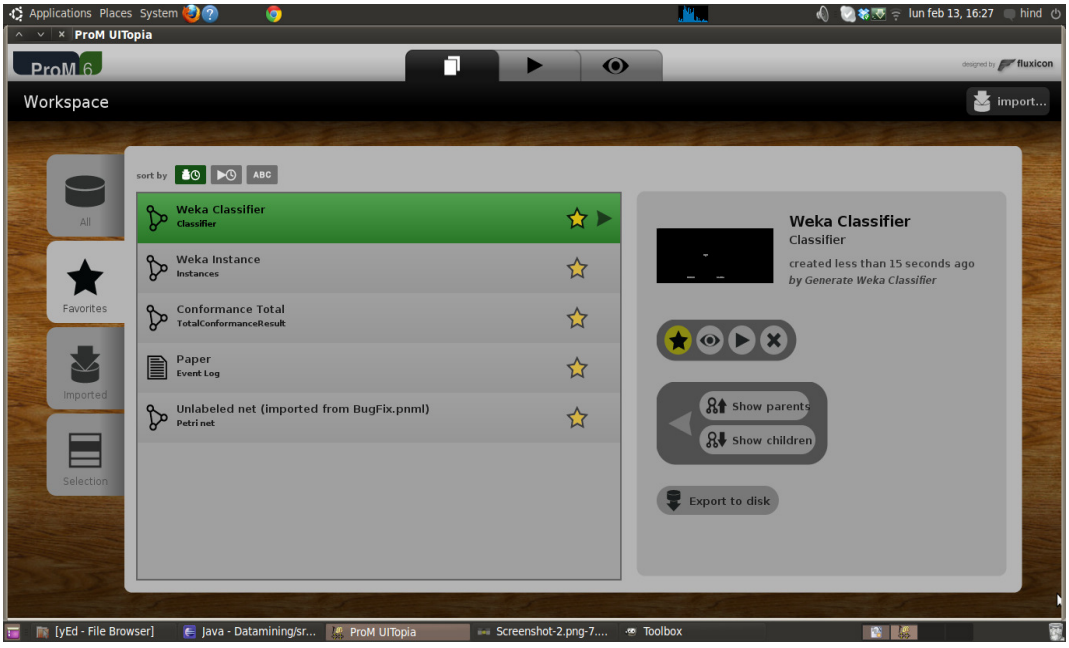
(e) Vista view: data set



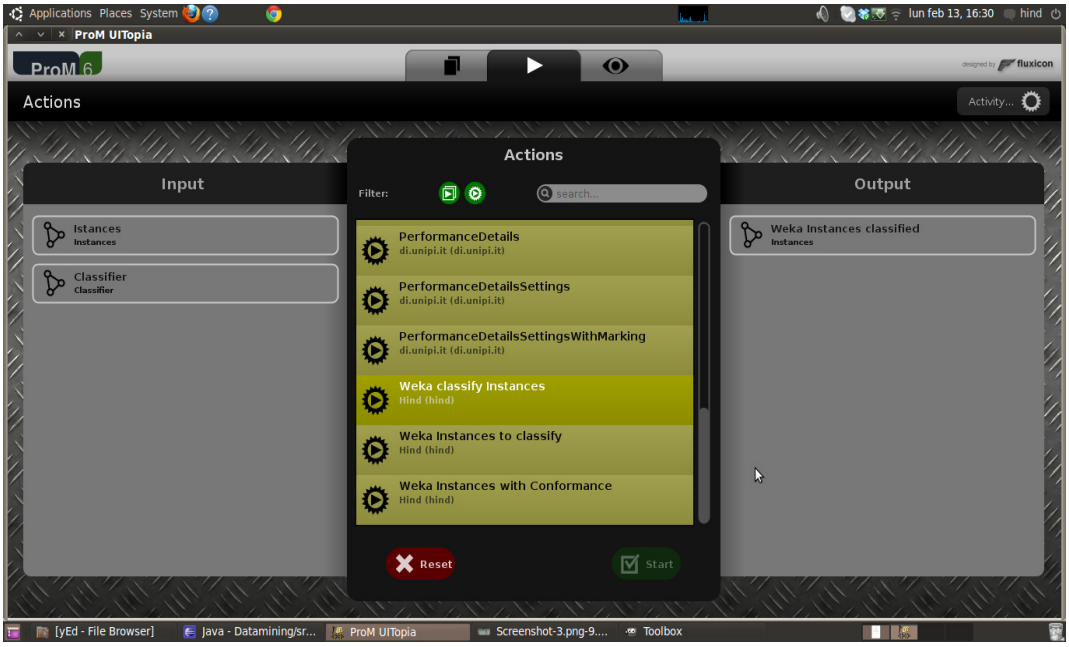
(f) Vista action: plugin `Generate Weka Classifier`



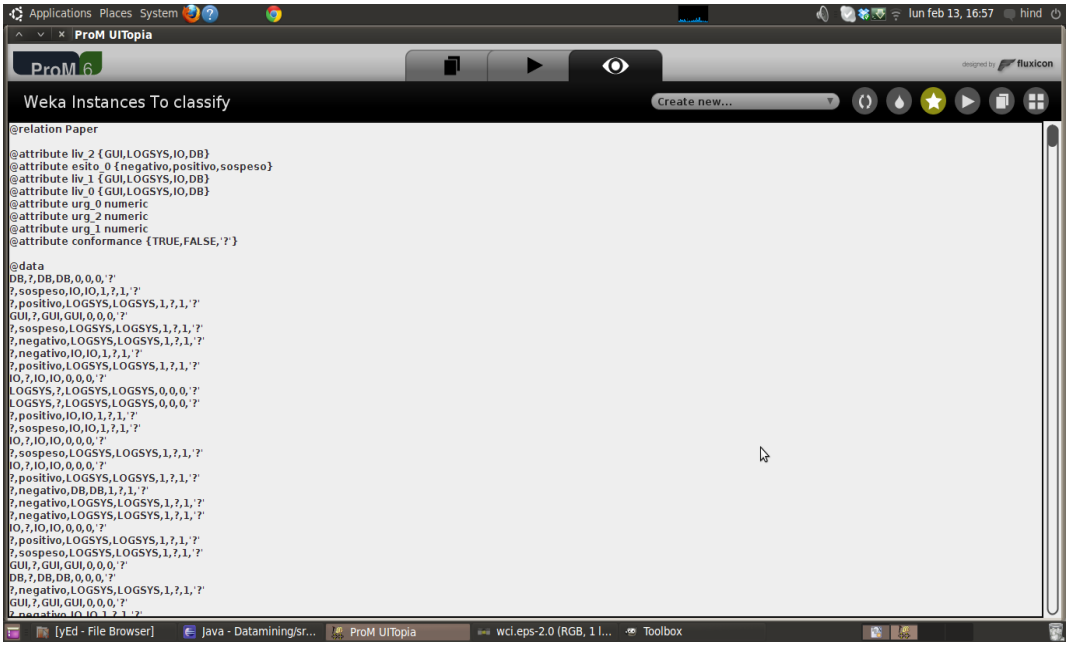
(g) Vista view: decision tree



(h) Vista workspace: insieme delle risorse



(i) Vista action: plugin `Weka Classify Instances`



(j) Vista view: istanze da classificare

(k) Vista view: istanze classificate

Figura 8.2: Figure illustrative.

Bibliografia

[1] http://www.pnml.org/, 2011.

[2] G. 360. http://www.sketchpadbpmn.sourceforge.net/. available on sourceforge.

[3] R. A, V. D. Aalst, and W. M. P. Decision mining in prom. *Business Process Management*, 2006.

[4] W. Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[5] M. L. G. at University of Waikato. http://www.cs.waikato.ac.nz/ml/weka/, 2011.

[6] R. Bruni, A. Corradini, G. Ferrari, T. Flagella, R. Guanciale, and G. O. Spagnolo. Misurare processi di business. *Congresso AICA*, 2011.

[7] C. Bussler and A. Haller, editors. *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, volume 3812, 2006.

[8] F. P. L. Christian W. Günther. http://www.xes-standard.org/_media/openxes/developerguide7.pdf, November 2009. This is an electronic document.

[9] F. P. L. Christian W. Günther. http://www.xes-standard.org/_media/xes/xes_standard_proposal.pdf, November 2009. This is an electronic document.

[10] W. M. Coalition. http://www.wfmc.org/xpdl.html.

[11] J. Muñoz-Gama and J. Carmona. A fresh look at precision in process conformance. In R. Hull, J. Mendling, and S. Tai, editors, *BPM*, volume

6336 of *Lecture Notes in Computer Science*, pages 211–226. Springer, 2010.

[12] OMG. `http://www.bpmn.org/`, 2011.

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

[14] E. T. U. Process Mining Group. `http://www.processmining.org/prom/start`, 2010.

[15] E. T. U. Process Mining Group. `http://www.processmining.org/prom/start`, 04 2011.

[16] J. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann Publishers, 1993.

[17] P. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson Addison Wesley, 2006.

[18] A. E. Thomas Freytag. `http://www.woped.org/`, 2011.

[19] W. van der Aalst, A. Weijter, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16:2004, 2003.

[20] Wikipedia. `http://it.wikipedia.org/wiki/Service-oriented_architecture`, 2011.