UNIVERSITÀ DI PISA

Corso di Laurea Magistrale in Data Science and Business Informatics

BUSINESS PROCESS MODELING

Progetto 34: Software

LUCIA FABBRI

Anno Accademico 2023-2024

Indice

1 Introduzione

Questo progetto si propone di analizzare e progettare i processi relativi a uno scenario di collaborazione tra un project manager e due programmatori per lo sviluppo di un'applicazione software. Inizialmente, nel capitolo 2, verrà affrontata la creazione dei diagrammi di progetto utilizzando la notazione BPMN (Business Process Modeling Notation). Successivamente, nel capitolo 3, l'attenzione si sposterà sull'analisi e la trasformazione di tali diagrammi in reti di Petri. Infine, nel capitolo 4, verranno esaminate brevemente le modifiche al diagramma BPMN e alle reti di Petri, introdotte a seguito di una variante dello scenario iniziale.

2 Modellazione tramite BPMN

Lo scenario di riferimento è stato modellato utilizzando la notazione grafica BPMN con l'editor online *Signavio*. Gli attori principali in questo scenario sono il Project Manager, che ha il compito di creare, gestire e valutare il lavoro svolto dai due Programmatori. Questi ultimi eseguono vari task su un repository condiviso e attendono il feedback del Project Manager, il quale verifica e testa la correttezza del loro lavoro. Per ciascuno di questi attori sono state modellate delle pool distinte: una per il **Project Manager**, una per il **Programmatore1** e un'altra per il **Programmatore2**, mentre la comunicazione tra loro avviene attraverso i *message flows*. Lo schema è illustrato nella figura 1. Concettualmente, si è scelto di implementare la parte relativa ai programmatori utilizzando due pool separate, invece di due lane all'interno della stessa pool, poiché si assume che i programmatori facciano parte di organizzazioni differenti, trattandosi di una collaborazione distribuita. Tuttavia, per ragioni di semplicità e chiarezza, ci concentreremo esclusivamente su una delle due pool di programmatori, dato che sono identiche. Nel diagramma, i task in arancione si distinguono da quelli in giallo grazie alla notazione specifica di *Parallel MI Task* (indicata da tre linee verticali), utilizzata in tutti quei casi in cui il Project Manager debba interfacciarsi simultaneamente con i due programmatori.

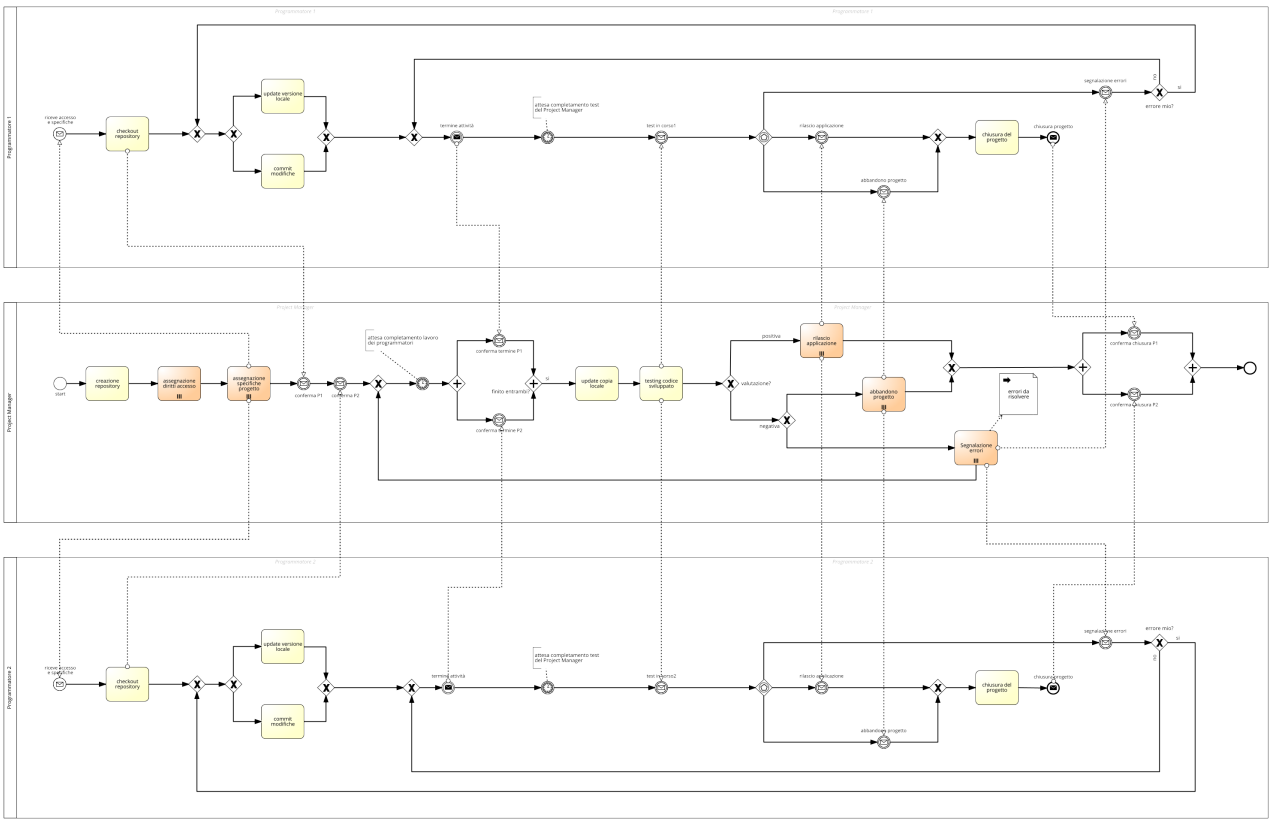


Figura 1: Diagramma BPMN completo dello scenario in esame

2.1 Pool Project Manager

Il processo ha inizio all'interno della pool del Project Manager (d'ora in avanti abbreviato come PM), come illustrato in figura 2. Lo *start event* è seguito da un primo *task* di creazione del repository condiviso su cui si lavorerà. Successivamente, il PM esegue due *multi-instance tasks* (evidenziati in arancione), in cui assegnano i diritti di accesso e le specifiche di progetto ai due programmatori (enti esterni alla pool). Successivamente, il PM riceve conferma di corretto accesso al progetto da parte dei programmatori attraverso due *intermediate message event*. A questo punto (tramite un *XOR-join*) il PM si mette in attesa di ricevere notizie in merito al lavoro dei programmatori, modellato tramite un *intermediate timer event*. Infatti, il PM non può proseguire con le sue attività fintanto che entrambi i programmatori non hanno inviato una notifica di completamento dei propri compiti: questo viene modellato utilizzando un *AND-split* e un *AND-join* che racchiudono al proprio interno due *intermediate message event* di conferma. Il PM (attraverso uno *XOR-split*) può quindi proseguire con il *task* di aggiornamento della copia locale e con il *task* di testing del codice sviluppato. Una volta completato il test e a seconda del risultato ottenuto, il PM presenta diverse casistiche con cui proseguire il flusso (modellate attraverso due porte *XOR-split* consecutive). Il primo split esegue una distinzione tra una valutazione positiva e una negativa: se il codice sviluppato risulta essere corretto (esito positivo), il PM rilascia l'applicazione attraverso un *sequential multi-instance task* (evidenziato in arancione), altrimenti se il codice risulta non essere ancora corretto (esito negativo) si passa attraverso un secondo split che prevede altre alternative. Da un lato, il PM può decidere di abbandonare definitivamente il progetto attraverso un *parallel multi-instance task* mentre dall'altro lato può decidere di segnalare gli errori commessi dai programmatori, sempre attraverso un *parallel multi-instance task* e comunicando gli errori commessi e le nuove specifiche all'interno di un *data object*. La gestione degli errori viene lasciata al singolo programmatore e verrà spiegata nel dettaglio nel paragrafo successivo. Avendo terminato i suoi compiti, il PM si mette nuovamente in attesa del completamento delle attività da parte dei programmatori, utilizzando uno *XOR-join* che lo riporta all'inizio del processo. Il processo raggiunge il termine solamente nei casi in cui il PM decide il rilascio dell'applicazione o l'abbandono del progetto stesso. I due flussi, unificati tramite uno *XOR-join*, devono attendere la conferma di chiusura del progetto tramite due *intermediate message event* distinti da parte di entrambi i programmatori racchiusi tra due porte *AND-split* e *AND-join*. Solo in questo momento il PM può terminare il processo e chiudere definitivamente il progetto (*end event*).

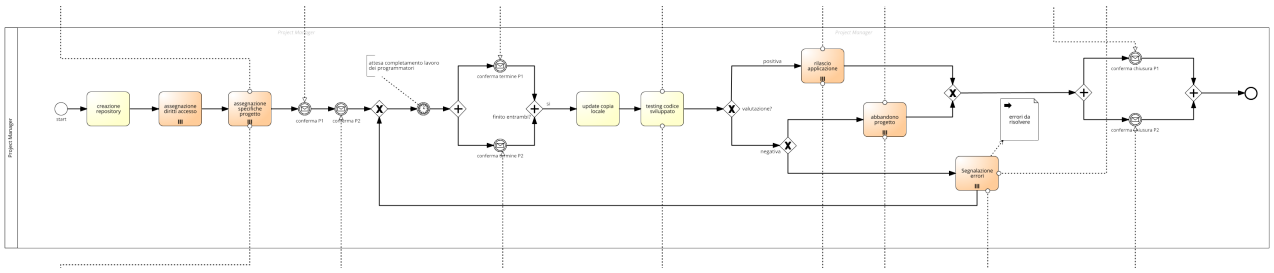


Figura 2: Diagramma BPMN della pool Project Manager

2.2 Pool Programmatore

Come menzionato in precedenza, il processo che i programmatori seguono per interagire con il PM è lo stesso per entrambe le pool. Pertanto, in figura 3 è riportata la pool relativa al Programmatore 1, che risulta identica a quella del Programmatore 2. La pool del Programmatore inizia con un *Start Message Event*, in cui il Programmatore riceve le credenziali di accesso al repository, le specifiche del progetto ed esegue il *task* di checkout repository, in cui recupera una copia locale del repository su cui inizierà a lavorare. Il programmatore esegue quindi i compiti che gli sono stati asseganti e a seconda del caso può scegliere di proseguire (attraverso un *XOR-split*) con il *task* di aggiornamento della versione locale del codice, aggiornando il codice con le ultime modifiche necessarie, oppure con il *task* di commit delle modifiche, salvando definitivamente le modifiche nel repository. Completata una di queste due attività (*XOR-join*), il programmatore invia una notifica al PM (ente esterno alla pool) tramite un *intermediate message event*, segnalando il termine del suo lavoro. In questo punto del processo, rappresentato da un *intermediate timer event*, il programmatore entra in uno stato di attesa per ricevere dal PM la notifica di test in corso tramite un *intermediate message event*. Questa notifica segnala infatti che anche l'altro programmatore ha completato il suo compito e il processo può continuare. Essendo il programmatore in attesa, alla ricezione di questo evento il programmatore non deve interrompere alcuna attività ma al contrario attraverso un *event-based gateway* è pronto per proseguire con le attività a seconda di cosa il PM gli comunica dopo aver eseguito il test (nella pool del Project Manager). Se il test ha esito positivo, il

programmatore riceve dal PM un *intermediate message event* di rilascio applicazione, confermando il successo del progetto dapprima attraverso il *task* di chiusura del progetto e successivamente mandando un *end message event* di chiusura del progetto. In caso di esito negativo, un *intermediate message event* può segnalare la decisione del PM di abbandono del progetto, interrompendo definitivamente il lavoro e portando alla conclusione del processo attraverso uno *XOR-join* che collega il flusso al precedente task di chiusura del progetto. In caso di ricezione dell'*intermediate message event* di segnalazione degli errori, uno *XOR-split* interroga il programmatore riguardo a chi ha commesso l'errore. Se l'errore segnalato non è stato commesso dal programmatore, quest'ultimo utilizza un *XOR-join* per tornare alla fase di attesa, in cui rimane in attesa di ulteriori comunicazioni dal PM riguardanti l'esito del test. In questa fase, il programmatore attende anche che l'altro programmatore completi il suo lavoro. Se invece l'errore è stato commesso dal programmatore stesso, attraverso un altro *XOR-join*, ritorna alla fase precedente per correggere gli errori. Qui può scegliere di eseguire un aggiornamento della versione o un commit delle modifiche e l'intero processo viene reiterato. La gestione degli errori è progettata per essere gestita dai singoli programmatori, in modo che ricevano sempre aggiornamenti sul progetto attraverso notifiche di errore. Se l'errore non è imputabile al programmatore, non è necessario eseguire ulteriori attività e il programmatore rimane in fase di attesa. In questo caso, il PM non inserirà nuove specifiche per il programmatore all'interno del data object.

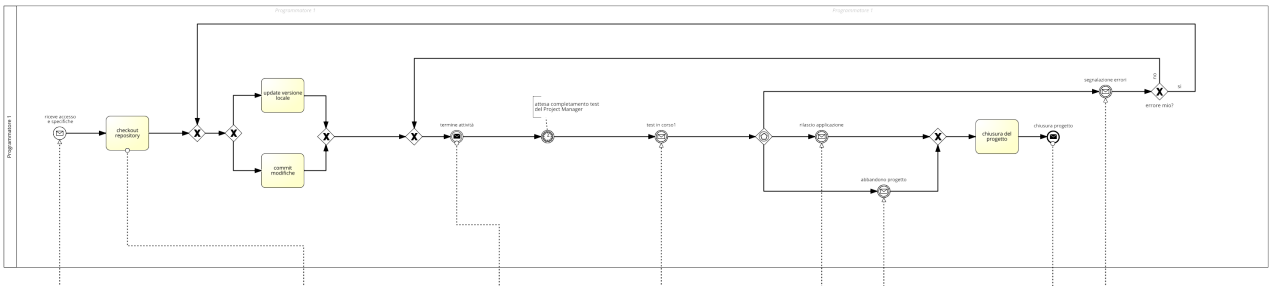


Figura 3: Diagramma BPMN della pool Programmatore

3 Analisi delle Workflow nets

Dopo aver modellato il processo con un diagramma BPMN, questo è stato trasformato in una workflow net, una particolare categoria di reti di Petri, utilizzando il software *WoPeD*, per ottenere una rappresentazione più formale che permetta analisi più approfondite. Nelle sezioni successive, si procederà prima con l'analisi separata delle due reti: quella del Project Manager e quella del Programmatore. Successivamente, sarà esaminata la rete completa, includendo per ciascuna rete il relativo Coverability Graph. Per facilitare la lettura e mantenere la chiarezza, le immagini dei grafici sono state collocate alla fine della relazione. Prima di procedere si elencano le regole adottate per la conversione da BPMN a reti di Petri per facilitarne la comprensione:

- Ogni *sequence flow* è stato tradotto con una piazza (*place*);
- Ogni *task* e ogni *evento* sono stati rappresentati da una transizione (*transition*);
- Sono state aggiunte una *piazza iniziale* e una *piazza finale* per indicare rispettivamente l'inizio e la fine del processo;
- Gli *XOR-split* e *XOR-join* sono stati semplificati utilizzando una piazza per rappresentare la scelta e aggiungendo tante transizioni quanti sono i percorsi possibili, tutte collegate a una nuova piazza per il task successivo.
- Gli *Event-based gateway* sono stati rappresentati da una piazza collegata a tante transizioni quanti sono gli eventi che si possono attivare nel processo.

3.1 Workflow net Project Manager

La rete progettata per il Project Manager è composta da 30 piazze, 29 transizioni e 62 archi ed è riportata in figura 4.

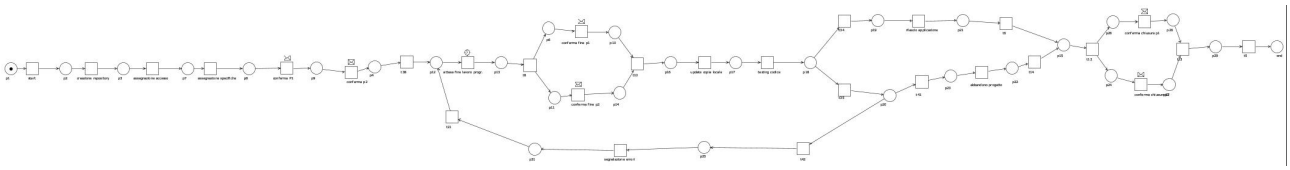


Figura 4: Workflow net Project Manager

La figura 7(a) riporta invece quelli che sono i risultati dell'analisi semantica della rete. Innanzitutto, si può osservare che si tratta di una *workflow net*, poiché presenta una piazza di input i e una piazza di output o, e ogni piazza e transizione si trovano in un cammino che va da i a o. Analizzando ulteriormente la rete, si nota che tutti i *pre-set* e *post-set* di ciascuna transizione sono costituiti da una singola piazza. Questo suggerisce che la rete sia una *S-Net*, poiché ogni transizione ha esattamente una piazza di input e una di output. Di conseguenza, si può affermare che la rete è *safe*, *sound* e *bounded*. Si osserva inoltre come la rete risulti essere una *free-choice net*, poiché per ogni coppia di transizioni i loro pre-set risultano disgiunti o uguali, così come si può dire che la rete sia *well-handled*, in quanto non sono presenti *PT-handles* o *TP-handles*. La proprietà di *soundness* è stata verificata attraverso il *tokengame*, che ha dimostrato l'assenza di dead tasks e ha garantito che il token raggiunga la piazza finale senza rimanere attivo in altre piazze. Poiché la rete è free-choice, bounded e anche *live* in quanto non presenta dead transitions, si può concludere che sia *S-coverable* e presenta 4 S-components. Inoltre, la rete è anche *strongly-connected* in quanto presenta un unico strongly connected component. La Figura 11 mostra il *coverability graph* della rete, ottenuto anch'esso tramite WoPeD. Il grafo è composto da 30 vertici e 33 archi ed trattandosi di una rete bounded possiamo dire che il coverability graph coincide con il reachability Graph.

3.2 Workflow net Programmatore

Anche la rete relativa al Programmatore è stata progettata ed analizzata utilizzando WoPeD, analogamente a quanto fatto per il Project Manager. La figura 5 illustra la rete che è composta da 20 piazze, 23 transizioni e 46 archi.

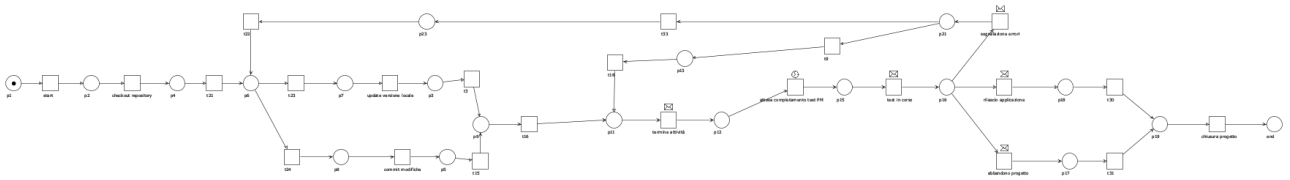


Figura 5: Workflow net Programmatore

Per quanto riguarda l'analisi semantica di questa rete, riportata in figura 7(b), è possibile eseguire le stesse considerazioni fatte in precedenza per il Project Manager. La rete è una *S-net*, quindi è *safe*, *sound* e *bounded*, così come è anche *strongly-connected* e quindi *live*. Essendo una *free-choice net*, live e bounded, risulta essere *S-coverable* con un unico S-component che include tutti i 43 elementi e si può considerare anche *well-structured* per l'assenza di PT-Handles e TP-Handles. Infine, il *coverability graph* della rete mostrato in figura 12 coincide nuovamente con il reachability graph essendo la rete bounded e in questo caso presenta 23 archi e 20 vertici.

3.3 Workflow net completa

Per ottenere una rappresentazione completa dell'intero processo, le tre workflow nets precedentemente analizzate sono state combinate in un unico *workflow module*, come illustrato nella figura 6. Durante l'integrazione dei tre attori, sono state aggiunte piazze per gestire lo scambio di informazioni tra le transizioni coinvolte. La rete risultante è ancora una *workflow net*, con la piazza iniziale e quella finale localizzate nella sottorete del Project Manager, che gestisce l'apertura e la chiusura del progetto e quindi dell'intero processo. L'analisi semantica della rete complessiva riportata in figura 7(c) mostra come questa rete sia costituita da ben 80 piazze, 73 transizioni e 178 archi. L'inserimento delle piazze di interfaccia per modellare il flusso informativo ha portato alla presenza di transizioni con pre-set non disgiunti o non uguali, il che significa che la rete non è più una *Free-choice net* e nemmeno una *S-net*. Inoltre, a differenza delle singole reti, questa rete presenta 56 PT-handles e 55 TP-handles, risultando quindi non *well-structured*. Nonostante ciò la rete conserva le proprietà di *soundness*, *boundedness*, *liveness* e *strong connectivity*. È inoltre *S-coverable* e presenta diversi S-components. Anche in

questo caso, il *coverability graph* coincide con il reachability graph poiché la rete è bounded, ma non viene incluso nel report poiché la sua complessità e dimensione renderebbero la rappresentazione non comprensibile.

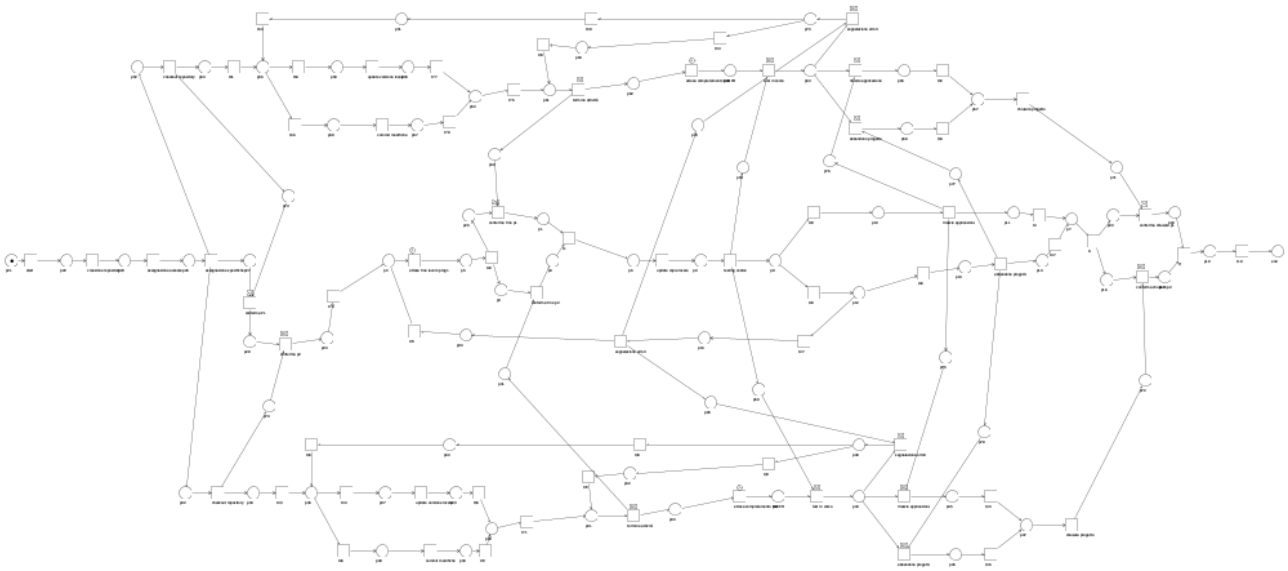
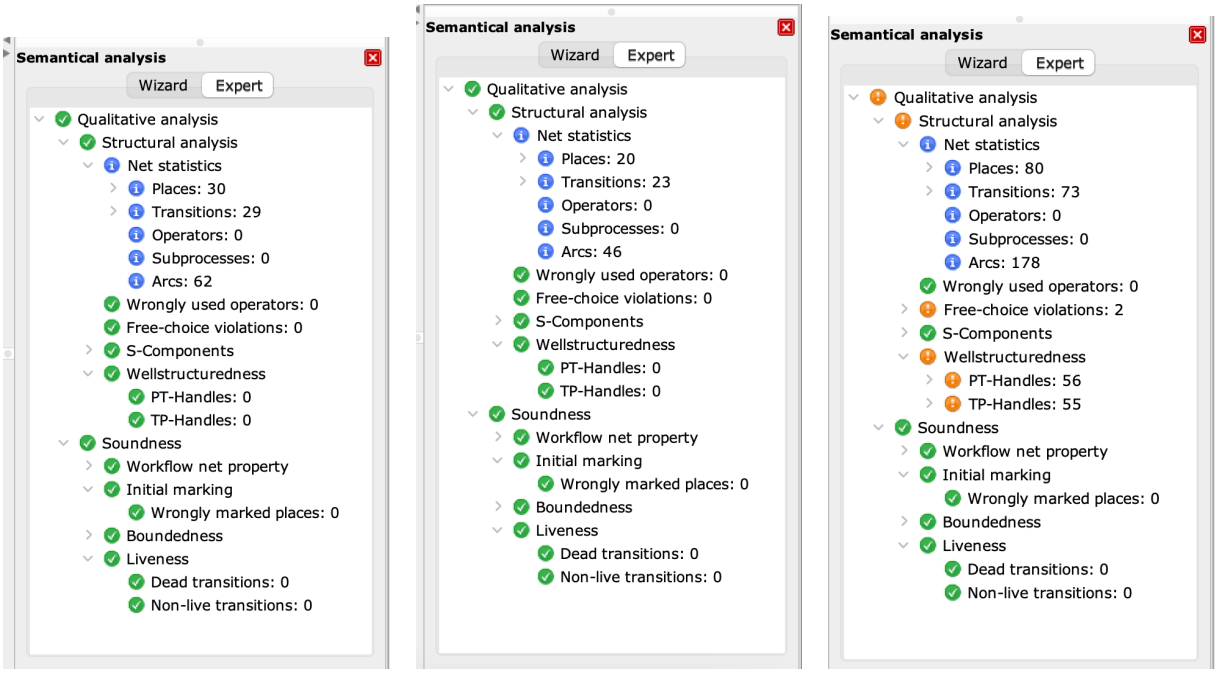


Figura 6: Workflow net completa

Le figure seguenti mostrano visivamente i risultati dell'analisi semantica effettuata su WoPeD per ciascuna delle tre reti descritte nei paragrafi precedenti:



(a) *Project Manager* (b) *Programmatore* (c) *Completa*

Figura 7: Analisi semantica delle workflow nets Project Manager (a), Programmatore (b) e quella completa (c)

4 Variante del processo

Lo scenario descritto finora è stato aggiornato con una variante: dopo il rilascio dell'applicazione, il Project Manager può decidere se avviare lo sviluppo di una nuova applicazione o concludere l'intero processo. Questa

opzione aggiuntiva ha reso necessario aggiornare sia il diagramma BPMN che le reti di Petri per riflettere questa nuova possibilità.

4.1 Diagramma BPMN della variante

Il diagramma BPMN della variante rimane pressoché invariato se non per l'aggiunta di qualche piazze e transizioni. La decisione di sviluppare una nuova applicazione oppure concludere il progetto viene modellata tramite uno *XOR-split* nella pool del PM, posizionato subito dopo il task di rilascio dell'applicazione. Nel caso in cui si scelga di sviluppare una nuova applicazione, il PM invia un *intermediate message event* per comunicare la decisione di nuovo progetto e, attraverso un *XOR-join*, si collega nuovamente all'inizio del processo, ripartendo dal task di creazione del repository. Così facendo, il PM ripete tutti i passaggi descritti in dettaglio nella sezione 2.1. Se invece non è previsto lo sviluppo di un nuovo progetto, il PM comunica questa decisione ad entrambi i programmatori tramite due *intermediate message event* e attraverso un primo *XOR-join* e due successivi *AND-split* e *AND-join* si mette in attesa di ricevere un *intermediate message event* di conferma di chiusura del progetto da parte di entrambi i programmatori. A questo punto, il PM può chiudere definitivamente il progetto e così facendo, anche il processo. Per modellare questa variante, anche la pool del Programmatore subisce alcune modifiche. Dopo aver ricevuto l'*intermediate message event* di rilascio dell'applicazione, il programmatore si pone in attesa di due possibili comunicazioni, modellate attraverso un *event-based gateway*. Le comunicazioni dal PM (ente esterno alla pool) possono indicare l'inizio di un nuovo progetto oppure la fine del lavoro (assenza di nuovi progetti), entrambe modellate attraverso *intermediate message events*. Nel caso in cui non siano previsti nuovi progetti, il programmatore esegue il *task* di chiusura del progetto e invia una notifica di chiusura al PM tramite un *end message event*, concludendo così il suo processo. Al contrario, se viene comunicato l'avvio di un nuovo progetto, il programmatore è pronto a riprendere il processo dall'inizio, seguendo nuovamente tutti i passaggi descritti in precedenza nella sezione 2.2.

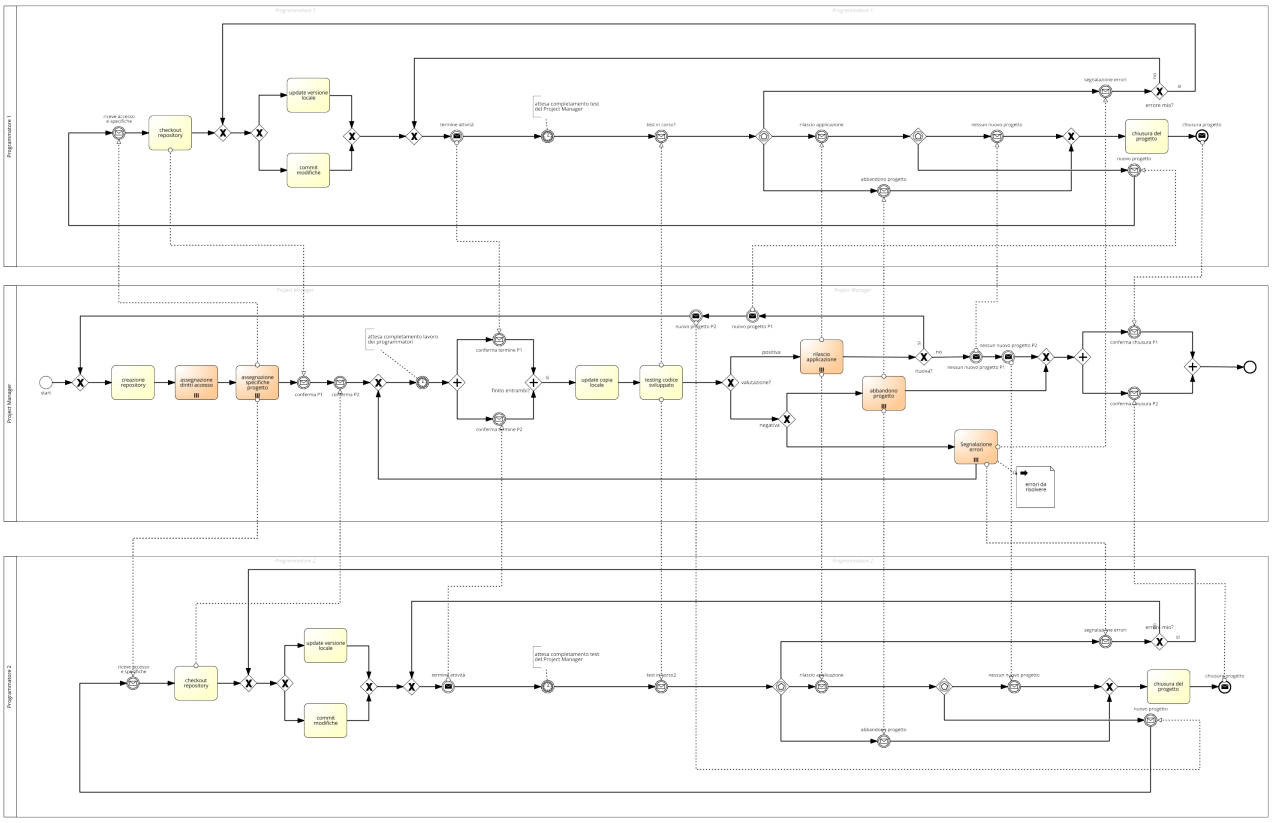


Figura 8: Diagramma BPMN della variante

4.2 Workflow net della variante

Il grafico BPMN sopra riportato è stato tradotto in rete di Petri utilizzando le stesse tecniche e convenzioni descritte in precedenza, sempre utilizzando lo strumento WoPeD. La sua analisi è stata invece condotta utilizzando un nuovo strumento Woflan poichè su WoPeD risultava impossibile, probabilmente data la grandezza

e la complessità della rete. La rete complessiva compresa di variante è mostrata in figura 9 mentre l'analisi semantica è riportata in figura 10. A differenza delle reti precedenti, il caso con variante ci porta ad osservare un errore all'interno della rete che blocca quindi la complessiva analisi. In particolare, si nota la presenza di una serie di *threads of control* che fanno si che la rete sembri non essere collegata. Tuttavia, sono stati svolti numerosi controlli facendo interagire entrambi i software WoPeD e Woflan al fine di risolvere l'errore, ma senza successo. INfatti, nonostante la rete sia collegata correttamente, non è stato possibile risolvere l'errore.

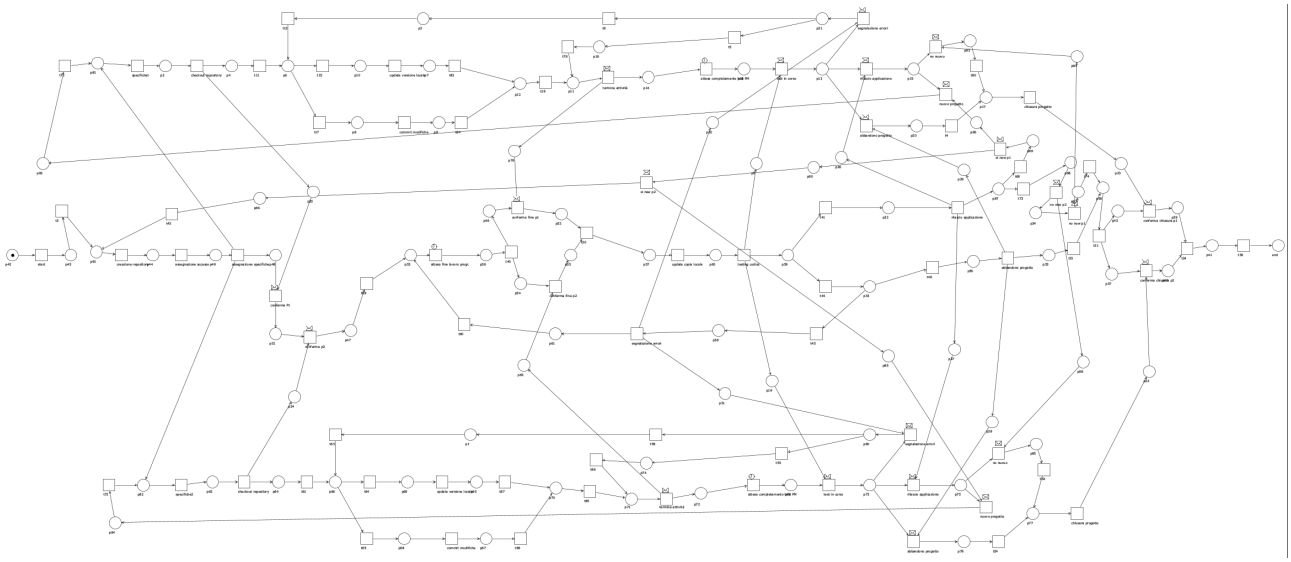


Figura 9: Workflow net della variante

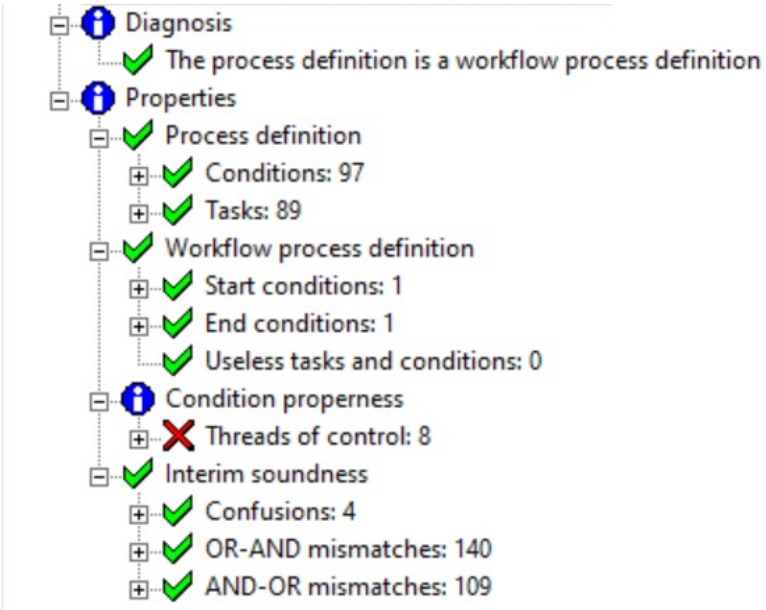


Figura 10: Analisi semantica della workflow net con la variante

5 Coverability graphs

Le immagini seguenti mostrano i *coverability graphs* del Project Manager e del Programmatore. I grafi relativi alla rete complessiva e alla variante non sono inclusi, poiché la loro dimensione e complessità renderebbero la rappresentazione non fruibile.

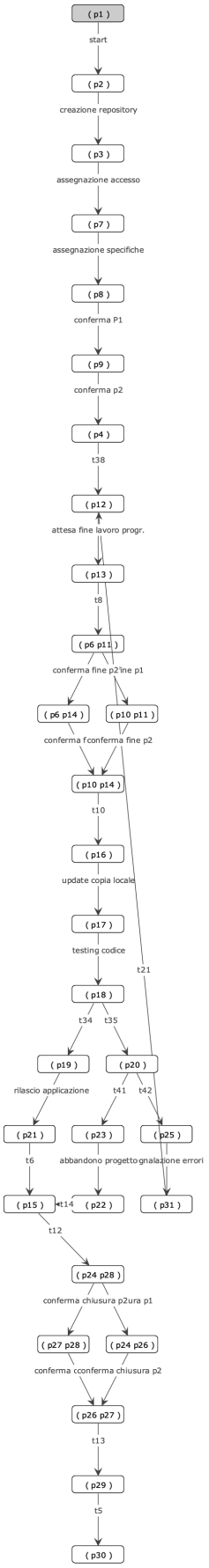


Figura 11: Coverability graph della net Project Manager

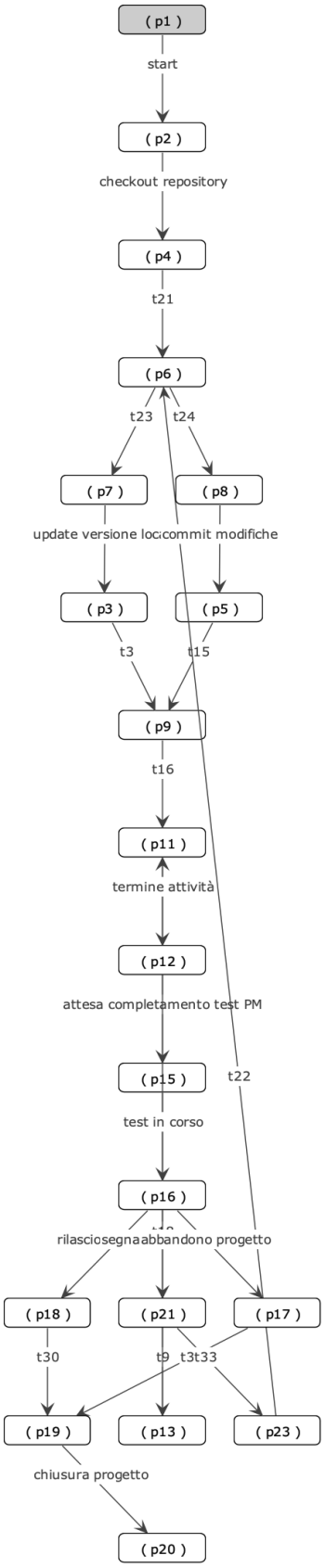


Figura 12: Coverability graph della net Programmatore