Università degli Studi di Firenze

Scuola di Ingegneria - Dipartimento di Ingegneria dell'Informazione

Tesi di Laurea Triennale in Ingegneria Informatica

Progettazione e sviluppo di un componente software per la derivazione di modelli di Petri preemptive da specifiche timeline

Candidato
Kevin Maggi

Relatore
Prof. Enrico Vicario

Correlatori
Prof. Laura Carnevali
Prof. Fulvio Patara
Dott. Leonardo Scommegna

Anno Accademico 2019/2020

alla mia famiglia

e a Elisa

Ringraziamenti

Ringrazio il mio relatore, professor Enrico Vicario, per avermi dato l'opportunità di svolgere questa tesi e i miei correlatori, prof. Laura Carnevali, prof. Fulvio Patara e dott. Leonardo Scommegna per l'aiuto offertomi nel comprendere la teoria su cui si basa questo lavoro.

Ringrazio anche Francesca, Leonardo, Marco e Simone con cui ho condiviso questo percorso di studi e che sono stati di supporto durante tutto questo tempo.

Sommario

Nello sviluppo di *sistemi real-time* la correttezza dipende non solo dalla validità del risultato, ma anche dal tempo al quale il risultato è prodotto.

L'analisi di schedulabilità può essere affrontata con tecniche analitiche fissando delle ipotesi sulla struttura del taskset: ad esempio i test di schedulabilità per Rate Monotonic Scheduling e Earliest Deadline First Scheduling assumono che il taskset sia composto da tasks periodici con deadline uguale al periodo, tempo di computazione deterministico e non sincronizzati in alcun modo tra loro.

Un'altra possibilità è offerta da tecniche basate sull'analisi dello spazio degli stati che permettono di rilassare le menzionate assunzioni al costo di una maggiore complessità di analisi. In letteratura sono stati proposti e usati svariati formalismi a questo fine. Uno tra i più espressivi è quello delle *Preemptive Time Petri Nets* (PTPN), un'estensione delle *Time Petri Nets* (PTN), che consente di modellare tasksets complessi con processi periodici, sporadici e jittering, con offset, priorità statica su più processori, sincronizzazione tramite uso esclusivo di risorse, vincoli di precedenza e tempi di computazione non deterministici.

Le PTPN che modellano i tasksets rappresentano utili strumenti di supporto nella progettazione di sistemi real-time, in particolare in fase di design, implementazione e validazione.

Tuttavia la modellazione dei tasksets tramite metodi formali, come quello delle PTPN, non eccelle in intuitività e non viene adottato nella pratica industriale dello sviluppo di software real-time, dove la specifica semi-formale delle *timeline* offre maggiore interoperabilità. A questo scopo si rende necessario tradurre specifiche timeline in modelli PTPN.

In questa tesi vengono proposti un semplice ma efficace formato per la rappresentazione di taskset in forma di timeline e un algoritmo per la loro traduzione in modelli PTPN; sulla base di questi due strumenti viene progettato e sviluppato un componente software che implementa la traduzione. A margine viene definito un formato per l'esportazione di questi modelli e integrato un componente software che implementa l'esportazione.

Indice

Capitolo 1

Introduzione

Lo sviluppo di sistemi real-time (RT) si basa sulla verifica della correttezza temporale degli eventi, intesa anche come istante temporale nel quale il risultato della computazione viene prodotto. Per fare questo è possibile avvalersi di tecniche di analisi analitiche tra le più svariate. Un esempio degno di nota è la teoria Rate Monotonic, con cui è possibile trattare processi periodici indipendenti con tempi di esecuzione deterministici: in questo caso la *schedulabilità* è verificata semplicemente confrontando il tempo totale di utilizzo con una soglia che dipende dal numero di processi. Questo può essere esteso a modelli comprendenti sincronizzazioni mediante uso esclusivo di risorse [9] e a processi sporadici, con opportune trasformazioni [10]; rimangono però esclusi vincoli di precedenza dettati da Inter-Process Communication (IPC), tempi di esecuzioni non deterministici compresi tra un massimo e un minimo e sistemi con più processori.
Discorso analogo anche per l'altrettanto diffuso Earliest Deadline First Scheduling.

Possono essere rilassate queste ipotesi preliminari sulla struttura del taskset ricorrendo a tecniche basate sull'analisi dello spazio degli stati. In

letteratura si trovano numerosi formalismi molto espressivi che estendono il parco dei meccanismi che possono essere inclusi nei tasksets, sebbene pochi siano in grado di trattare tutti gli aspetti appena elencati, come evidenziato in [2].

In [2] viene proposto il formalismo delle *Preemptive Time Petri Nets* (PTPN) come strumento di analisi di taskset complessi che includono tutti questi aspetti e non solo. Grazie a queste infatti è possibile modellare anche l'*accettazione dinamica* di un task, che così possa essere obbligatorio o opzionale, l'*implementazione polimorfica* di una computazione, dove la scelta della versione avviene a runtime, e la priorità dipendente dalla marcatura; tutti questi aspetti sono introdotti in [1], ma non verranno trattati in questo progetto.

La versatilità delle PTPN che modellano dei tasksets fa sì che queste siano particolarmente utili in più fasi del ciclo di vita dei sistemi RT [3] [4] [5]. In fase di *design* può essere condotta la simulazione o l'analisi dello spazio degli stati della PTPN per verificare la schedulabilità e validare il rispettivo taskset; in fase di *implementazione* può essere automatizzata la generazione del codice grazie ad algoritmi di traduzione delle PTPN in codice RT; infine può essere usata come oracolo in fase di *testing*.

Tuttavia i tasksets non vengono, solitamente, modellati con PTPN (o altri formalismi), ma attraverso *timelines*: si tratta di una specifica semi-formale che colma il gap tra la pratica industriale dello sviluppo dei software real-time e l'uso di metodi formali, risultando molto più intuitiva di questi ultimi.

Risulta quindi particolarmente utile uno strumento sia di traduzione dei tasksets in PTPN, sia di analisi di quest'ultime. *ORIS 2.0* [7] [8] è un tool sviluppato da STLAB per la rappresentazione e l'analisi di Time Petri Nets

(PTN) e Stochastic Time Petri Nets (STPN). Il tool include anche la *libreria Sirio* [6], scritta in Java, che estende queste funzionalità e le porta a livello programmatico. In particolare include l'analisi delle PTPN presentata e implementata in [18].

Questo progetto si integra nel tool: dota Sirio di un componente per la traduzione di tasksets in forma di timeline in PTPN e di un componente per l'esportazione su file delle Petri Nets (PN) per la successiva importazione nel tool grafico ORIS.
Dal momento che in Sirio la tipologia di PN non è codificata staticamente ma è determinata dinamicamente dalle *feature*, l'esportazione prende in considerazione non solo le PTPN ma le generiche PN, includendo tutte le tipologie rappresentabili nel dominio di Sirio, sebbene non tutte siano supportate da ORIS (incluse le PTPN).

1.1 Preemptive Time Petri Nets

Le PTPN sono un'estensione delle TPN che ai vincoli temporali di quest'ultime aggiungono un meccanismo di assegnazione delle risorse, che condiziona l'avanzamento dei timer delle transizioni *enabled*.

1.1.1 Sintassi

Una PTPN è una tupla

$$PTPN = \langle P; T; A^{+}; A^{-}; M; FI^{s}; Res; Req; Prio \rangle$$

dove:

- i primi 5 termini rappresentano una PN:
  - P è un insieme di *posti*;

  - T è un insieme di *transizioni*. P e T sono disgiunti;
  - $A^+ \subseteq T \times P$ è un insieme di *postcondizioni*;
  - $A^- \subseteq P \times T$ è un insieme di *precondizioni* e
  - $M : P \to \mathbb{N} \cup \{0\}$ è il *marking iniziale*, ovvero il numero di *token* inizialmente assegnati a ciascun posto;

- FI^s associa ciascuna transitione t a un *Firing Interval* delimitato da un *Earliest Firing Time* $EFT^s(t)$ e un *Latest Firing Time* $LFT^s(t)$, con $EFT \leq LFT$;
- Res è un insieme di *risorse*, disgiunto da P e T;
- $Req : T \to 2^{Res}$ associa ciascuna transizione a un sottoinsieme di Res e
- $Prio : T \to \mathbb{N}$ assegna a ciascuna transizione una *priorità*.

1.1.2 Semantica

La semantica di una PTPN, disaminata formalmente in [2] e [18], non è un aspetto rilevante nella presentazione di questo progetto, per cui la seguente trattazione non è e non vuole essere esaustiva.

Lo stato di una PTPN è una coppia $\langle M, \tau \rangle$, dove M è il marking e τ associa a ciascuna transizione con un possibile *time to fire* ($\tau : T \to \mathbb{R}^+ \cup \{\infty\}$). Lo stato evolve secondo il *firing* di una transizione.

Una transizione t_0 è *enabled* se ogni suo posto di input contiene almeno un token. Una transizione enabled si dice *progressing* se e solo se ogni risorsa che richiede non è richiesta da un'altra transizione *enabled* con una priorità maggiore, altrimenti viene detta *suspended*. Una transizione t_0 è *firable* se

è *progressing* e il suo *time to fire* $\tau(t_0)$ non è maggiore del *time to fire* di qualsiasi altra transizione.

In seguito al *firing* di una transizione t_0 lo stato $s = \langle M, \tau \rangle$ è sostituito da un nuovo stato $s' = \langle M', \tau' \rangle$, dove il marking M' è ottenuto da M in due step: 1) rimuovendo un token da ogni posto di input di t_0; 2) aggiungendone uno in ogni posto di output di t_0.
Una transizione che dopo il primo step rimane *enabled* è detta *persistent*, mentre una transizione che non è *enabled* dopo il primo step ma lo è dopo il secondo è detta *newly enabled*.
τ' è calcolato in maniera diversa per le transizione *newly enabled*, per le quali assume un valore non deterministico in $[EFT, LFT]$, quelle *persistent-progressing*, per le quali assume il valore che aveva in s diminuito di $\tau(t_0)$, e quelle *persistent-suspended*, per le quali rimane invariato.

1.1.3 Rappresentazione grafica

Una PTPN può essere rappresentata graficamente come un grafo bipartito, dove i posti, rappresentati come cerchi vuoti, e le transizioni, rappresentate come barre verticali, sono i nodi e gli archi sono rappresentati dalle precondizioni e postcondizioni; i tokens sono rappresentati da punti all'interno dei posti e a ogni transizione vengono affiancati l'intervallo di firing scritto nella forma $[EFT, LFT]$ e delle coppie risorsa-priorità nella forma $\{res\} : prio$, come in figura 1.1.

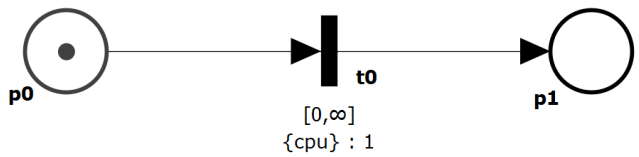

Figura 1.1: Esempio di rappresentazione grafica di PTPN

Nell'esempio in figura 1.1 p_0 e p_1 sono posti, rispettivamente, di input e di output per t_0; la transizione $t0$ ha $EFT = 0$ e $LFT = \infty$ e ha assegnata la risorsa *cpu* con priorità 1; p_0 contiene un token.

1.2 Timeline

La rappresentazione dei taskset in forma di timeline è un formalismo semplice e intuitivo che offre interoperabilità rispetto alle PTPN, di cui restringe l'espressività (per esempio non prevede un'esplicita rappresentazione del protocollo di priority ceiling, di cui parleremo più avanti), pur comprendendo tutti i concetti chiave della teoria dei sistemi RT.

1.2.1 Struttura di un taskset

Prendiamo in considerazione tasksets composti da *tasks* con la seguente struttura (raffigurata in figura 1.2):

- un task rilascia *jobs* in maniera ricorsiva con una fra tre politiche:
  - *periodico*: il tempo di interarrivo tra due istanze consecutive dei jobs è deterministico;
  - *sporadico*: il tempo di interarrivo tra due istanze consecutive dei jobs è limitato inferiormente, ma non superiormente;
  - *jittering*: il tempo di interarrivo tra due istance consecutive dei jobs è compreso tra un minimo e un massimo.

  La *deadline* del task, usualmente, si assume coincidere con il tempo minimo di interarrivo. Considereremo un'estensione di questo modello che prevede anche la presenza di *offset*;

- un job è internamente strutturato come sequenza di *chunks*;
- un chunk è caratterizzato da un *execution time* compreso tra un *Best Case Execution Time* (BCET) e un *Worst Case Execution Time* (WCET);
- un chunk può richiedere risorse (tipicamente processori) ognuna con un livello di priorità: in questo caso viene eseguito sotto scheduling preemptive con priorità statica;
- i chunk possono essere sincronizzati tra loro mediante uso esclusivo di semafori binari (*mutex*), caso in cui un chunk ne acquisisce l'uso all'inizio dell'esecuzione e lo rilascia al termine dell'esecuzione; e comunicazione tramite scambio di messaggi (*mailbox*), caso in cui la ricezione deve avvenire all'inizio dell'esecuzione, mentre l'invio al termine dell'esecuzione.

I chunk possono anche essere associati a una funzione *entry-point*; questo aspetto però esula dalla trattazione di questo lavoro.

Nella formulazione del modello finora descritta può verificarsi il fenomeno dell'*inversione di priorità*, quando un task a priorità maggiore che richiede uso esclusivo di una risorsa viene bloccato da un task a priorità minore che ha già acquisito tale risorsa ed è stato sospeso, per via del comportamento preemptive dello scheduling, prima del rilascio da un task a priorità intermedia. Per evitare ciò viene adottato il protocollo di *priority ceiling*, per cui ogni chunk che acquisisce un semaforo subisce un'operazione di *boosting* (e successivo *deboosting* al termine) che ne porta la priorità a quella del chunk con priorità maggiore che usa quel semaforo.

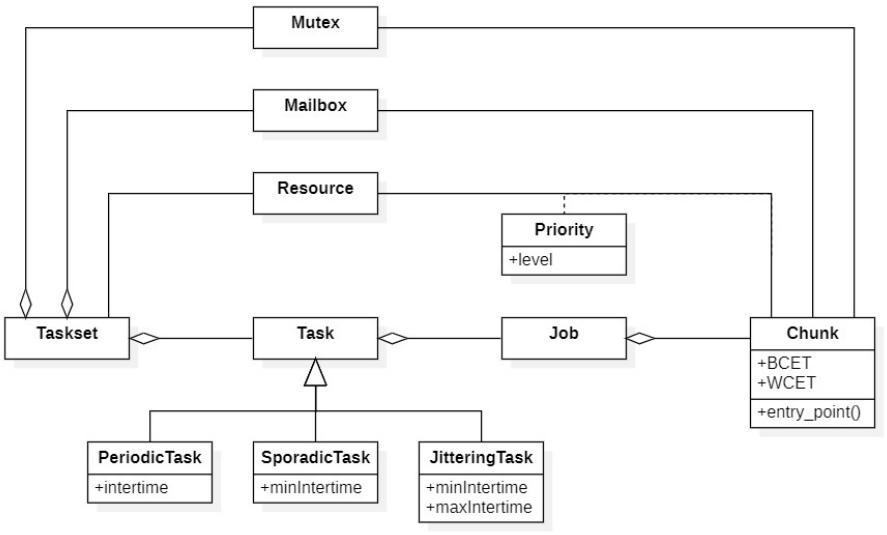

Figura 1.2: Modello concettuale di un taskset

1.2.2 Rappresentazione grafica

La rappresentazione grafica di un taskset in forma di timeline è mostrata in figura 1.3. Per ogni task è rappresentato il periodo minimo e massimo di interarrivo e la deadline (la freccia più spessa); ogni task contiene uno o più chunks, di cui sono riportati BCET e WCET, entry-point, risorse richieste e relative priorità; in caso di uso esclusivo di semaforo, ne viene rappresentata l'acquisizione (*wait*) e il rilascio (*signal*) con cerchi apposti all'inizio e alla fine del chunk, abbinati al nome del semaforo; le operazioni di mailbox vengono rappresentate con i rispettivi simboli all'interno del chunk.

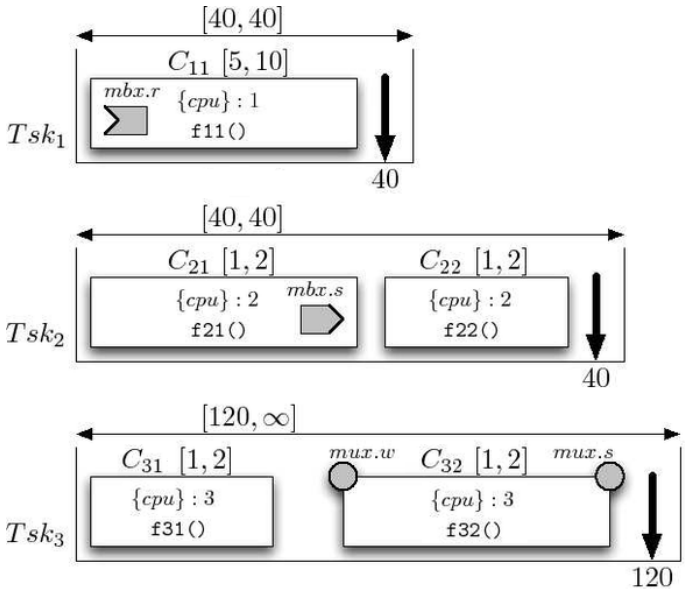

Figura 1.3: Esempio di rappresentazione grafica di timeline

Capitolo 2

Analisi

Il primo requisito che appare fin da subito evidente è la necessità di definire delle timelines. La soluzione più immediata è la creazione di API che permettano di farlo, verosimilmente attraverso l'uso di un pattern *builder*, per via programmatica. Tuttavia questa soluzione non eccelle in usabilità: dotare il sistema di un'interfaccia per la creazione delle timelines è di gran lunga la soluzione più usabile, anche immaginando che l'utilizzatore potrebbe essere un esperto del dominio con capacità di programmazione limitate. Una soluzione intermedia (che comunque traccia la via per la creazione in futuro di un'interfaccia) è l'utilizzo di un formato file esterno per la rappresentazione di timelines che possa poi essere affidato a un parser per l'importazione della timeline in ambiente Java.

Risulterà particolarmente conveniente usare il formato XML, non solo per la sua semplicità e versatilità, ma anche e soprattutto per la sua integrazione in Java grazie alla libreria *JAXB* (in approfondimento 2.1). Questo renderà estremamente semplice l'importazione di una timeline in formato XML in oggetti Java: sarà sufficiente definire uno schema per la rappresentazione in XML delle timelines. Il formato deputato a fare ciò è l'*XSD* (XML Schema

Definition) [17].

Approfondimento 2.1

JAXB

JAXB, acronimo per Java Architecture for XML Binding [13], è una API Java che semplifica l'accesso e la manipolazione di documenti in formato XML in ambiente Java. I quattro servizi fondamentali che offre sono:

- *binding* di uno schema XSD: generazione automatica di un insieme di classi Java che rappresentano uno schema. Per ogni tipo definito nello schema viene generata una classe con attributi, riferimenti ad altri elementi e metodi getter e setter;
- *unmarshalling* di un documento XML: conversione del documento in un "albero di oggetti" Java, istanze delle classi generate;
- *marshalling* in un documento XML: conversione di un "albero di oggetti" del tipo delle classi generate in un documento XML;
- *validazione* di un documento attraverso uno schema: verifica che il documento XML sia ben formato, ovvero che possa analizzato da un *parser*, ma anche che rispetti il dato schema, non solo nella forma, ovvero nei tipi, ma anche nei contenuti, ossia che soddisfi eventuali vincoli definiti nello schema stesso.

Per quanto riguarda invece l'esportazione su file delle PN, il formato dovrà essere quello utilizzato da Oris, ovvero `.xpn` . Il formato XPN è un formato XML per il quale però, non è stato preventivamente definito uno schema.

Sarà utile quindi, per le motivazioni di cui sopra, definire uno schema XSD per il formato XPN, che rispetti l'attuale "schema virtuale" usato dal tool Oris.

L'ultimo requisito, non per importanza e già ampiamente discusso, è la conversione delle timelines in PTPN. Questi requisiti portano alla definizione dei casi d'uso che guideranno la progettazione.

2.1 Use Case

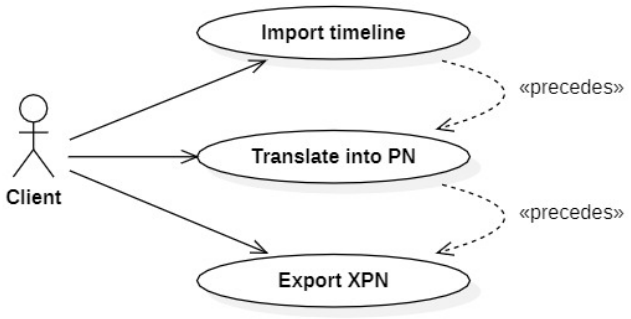

Figura 2.1: Use Case Diagram

Di fatto i casi d'uso sono già stati definiti nel capitolo 1 in fase di presentazione di questo lavoro:

- **importazione di una timeline**: è utile separarlo dal prossimo caso d'uso, per lasciare che l'implementazione di questo riguardi il solo algoritmo di traduzione;
- **traduzione in PN**: è di fatto l'implementazione dell'algoritmo di traduzione;
- **esportazione in XPN**: riguarda il processo di esportazione di una PN in formato XPN.

Data la semplicità, lo Use Case Diagram raffigurato in figura 2.1 è stato reso più espressivo con l'aggiunta delle relazioni `<precedes>`, anche queste molto basilari.

2.2 Algoritmo di traduzione

L'algoritmo di traduzione adottato si basa sul procedimento presentato in [5], di cui segue una breve descrizione:

- **task**: un task, il cui compito è quello di rilasciare i jobs, viene rappresentato da una transizione senza posti di input, quindi sempre *enabled*, senza richiesta di risorse e con

$$[EFT, LFT] = [minIntertime, maxIntertime]$$

come in figura 2.2;

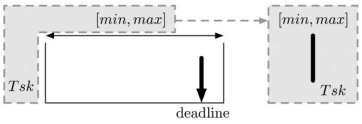

Figura 2.2: Traduzione di un task

- **chunk**: un chunk viene tradotto, come in figura 2.3, in una transizione con un posto di input, $[EFT, LFT] = [BCET, WCET]$ e richiedente le risorse chieste dal chunk. La sequenza dei chunks che formano il task è rappresentata dalla concatenazione delle loro traduzioni a partire dalla transizione del task;

- **mutex**: un semaforo è tradotto in un posto inizializzato con un token. L'operazione di *wait* di un chunk su un mutex è rappresentata con una transizione immediata anteposta al "blocco" del chunk, con le stesse

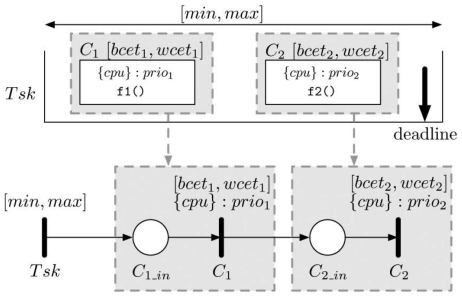

Figura 2.3: Traduzione di un chunk

risorse richieste, con un posto di input e precondizionata dal posto corrispondente al mutex; l'operazione di *signal* invece si traduce con una postcondizione dalla transizione di completamento del chunk al posto corrispondente al mutex.

Qualora l'operazione di *wait* incorpori il *boost*, è necessario anteporre a tutto ciò un'ulteriore transizione immediata con un posto di input. Le risorse richieste da questa transizione sono quelle chieste dal chunk, con le stesse priorità, mentre alle transizioni di *wait* e di completamento dell'esecuzione precedentemente aggiunte vengono aumentate le priorità al valore di *ceiling* del mutex acquisito. Il *deboost* è incorporato col completamento dell'esecuzione del chunk. La figura 2.4 chiarisce maggiormente;

- **mailbox**: una mailbox viene rappresentata con un posto con nessun token iniziale. Come si può vedere in figura 2.5, l'operazione di *send* viene tradotta in una postcondizione dalla transizione di completamento del chunk al posto corrispondente alla mailbox, mentre quella di *receive* viene tradotta, in modo simile al mutex, anteponendo al "blocco" del chunk una transizione immediata con le stesse risorse richieste, un posto di input e precondizionata dal posto corrispondente alla mailbox.

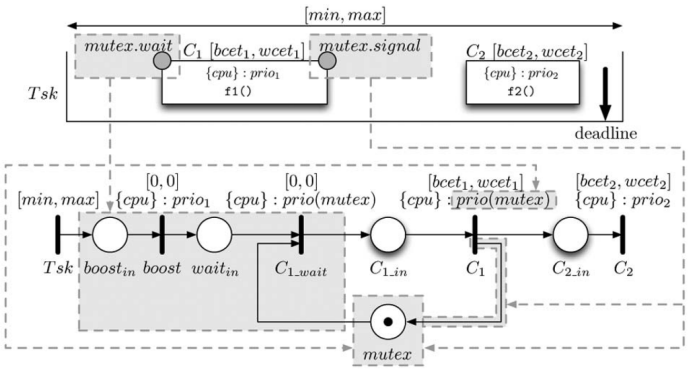

Figura 2.4: Traduzione di un mutex

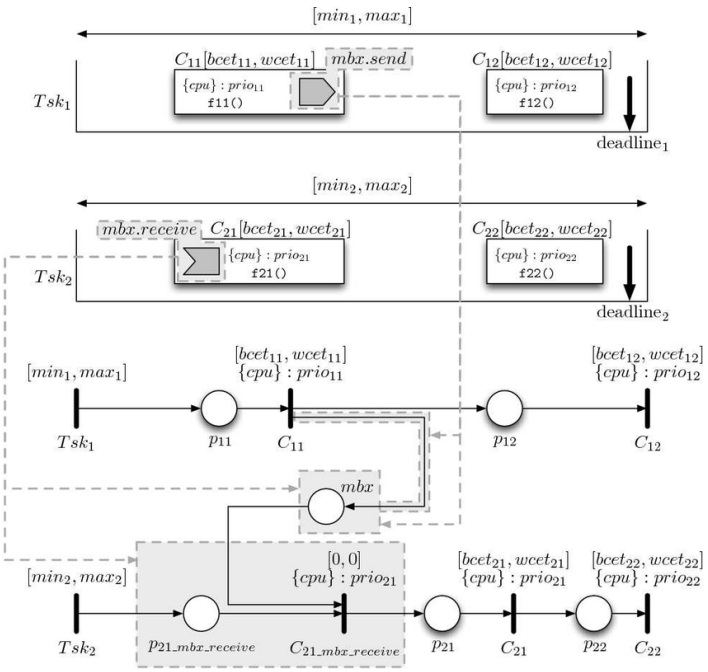

Figura 2.5: Traduzione di una mailbox

Capitolo 3

Progettazione

In fase di progettazione, in primo luogo sono stati definiti i due schemi e l'algoritmo di traduzione, successivamente è stato trattato il design che implementasse questi ultimi e come ultimo aspetto sono stati definiti alcuni algoritmi accessori. Nei paragrafi che seguono si offre una breve, ma quanto più esaustiva, panoramica di queste fasi.

3.1 Schemi

Attualmente JAXB non supporta nativamente la versione 1.1 del formato XSD e con essa le funzionalità che introduce, tipo `xs:assert`. Inoltre non supporta nemmeno la conversione in Java dei vincoli di identità `xs:key`, `xs:keyref` e `xs:unique`. Tutti questi elementi sono molto utili per una definizione precisa e accurata degli schemi da definire.

Possono essere estese le funzionalità supportate optando per l'utilizzo, in luogo di JAXB, della libreria EclipseLink MOXy. Tuttavia in questo progetto si è deciso per l'utilizzo di JAXB e, quindi, la versione 1.0 del formato XSD per definire gli schemi, che quindi specificano solamente la forma dei rispet-

tivi XML, ma non impongono condizioni sui contenuti. D'altra parte questi schemi sono solo di supporto per Sirio e Oris, per cui il controllo sui vincoli dei contenuti, comunque documentati, sarà delegato ad altre componenti del software, i.e. la GUI o gli algoritmi.

Di seguito vengono discussi gli schemi, più o meno dettagliatamente, nei vari aspetti strutturali; in appendice A vengono riportati integralmente i codici dei due XSD.

3.1.1 Timeline

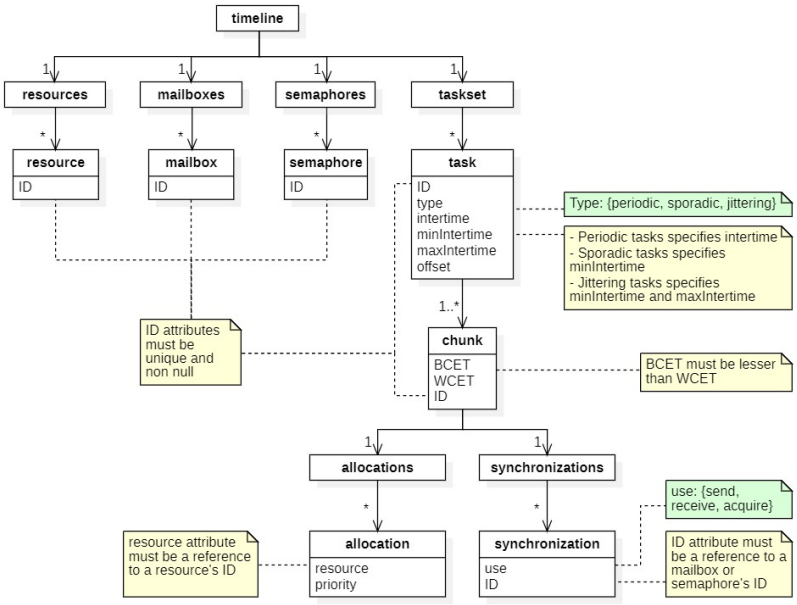

Figura 3.1: Modello concettuale di timeline

La figura 3.1 rappresenta il modello concettuale di una timeline. Le annotazioni descrivono i vincoli che devono rispettare i contenuti, in particolare quelle verdi sono codificate nella struttura, mentre quelle gialle sono quelle che devono essere controllate ad altro livello. A partire da un documento

XML valido secondo questo schema, si considererà valido anche nei contenuti, ovvero si assume che il controllo di questi vincoli sia già stato operato dal componente software che lo ha generato.

XSD

Adesso verrà proposta una panoramica su alcuni punti salienti:

- l'elemento radice è il tag `<timeline>` (in figura 3.2), che deve contenere, nell'ordine, una lista di risorse (`<resources>`), una lista di semafori (`<semaphores>`), una lista di mailboxes (`<mailboxes>`) e un `taskset`, ovvero una lista di `<task>`;

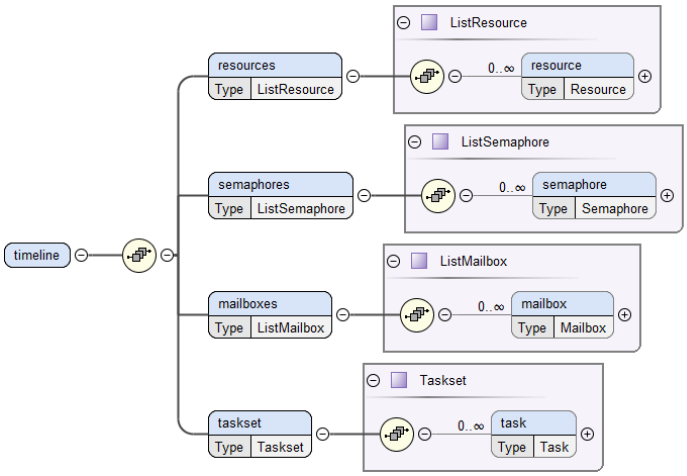

Figura 3.2: Visualizzazione grafica del tag `<timeline>`

- i tag `<resource>`, `<semaphore>` e `<mailbox>` sono tag vuoti, con un solo attributo `ID`;

- il tag `<task>` (figura 3.3) ha gli attributi `ID`, `type`, a scelta fra le 3 opzioni, `intertime`, `minIntertime`, `maxIntertime` e `offset`. I primi due sono obbligatori, i seguenti tre dipendono dal tipo, mentre l'ultimo è opzionale. Infine contiene una sequenza di almeno un `<chunk>`;

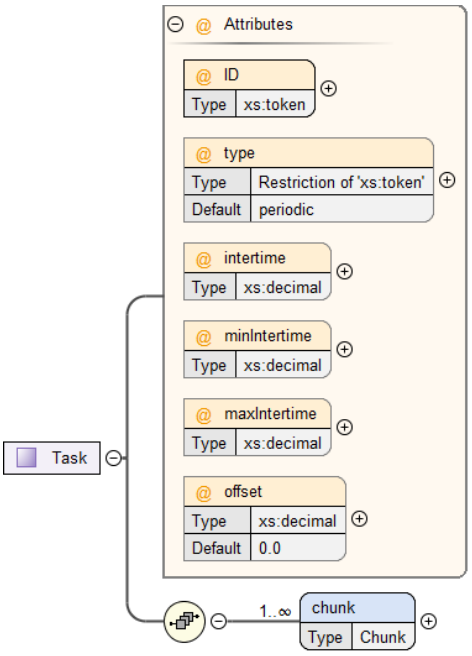

Figura 3.3: Visualizzazione grafica del tag `<task>`

- il tag `<chunk>` (figura 3.4) ha gli attributi obbligatori `BCET`, `WCET` e `ID` e due elementi: una lista di allocazioni `<allocations>` e una lista di sincronizzazioni `<synchronizations>`;

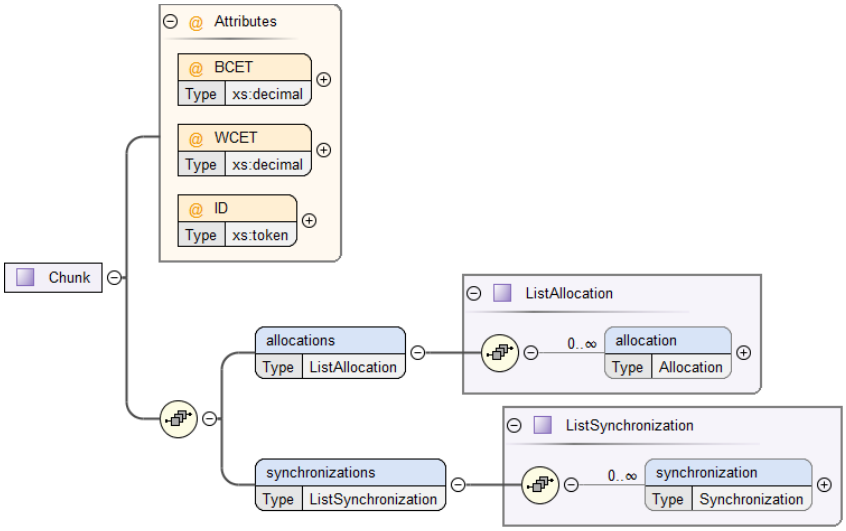

Figura 3.4: Visualizzazione grafica del tag `<chunk>`

- il tag `<allocation>` è vuoto e ha attributi obbligatori `resource` e `priority`;
- il tag `<synchronization>` è vuoto e ha attributi obbligatori `use` e `ID`.

Un aspetto degno di nota è l'attributo `offset` del tag `<task>`: può essere specificato indipendentemente dal tipo di task. Questo non solo a livello di schema ma anche a livello di documentazione allegata. Anche l'algoritmo che verrà presentato dopo accetta che l'offset sia specificato per qualsiasi tipo di task. Nella pratica la presenza dell'offset in un task di tipo sporadico non ha senso, in quanto non c'è un limite superiore, ma solo inferiore, al tempo di interarrivo tra jobs consecutivi. Tuttavia la sua specifica non è concettualmente errata, per cui viene consentita.

Esempio

Nell'esempio che segue si descrive una timeline con una risorsa, un semaforo e un task. Quest'ultimo è costituito da un solo chunk che acquisisce il semaforo.

```
<timeline>
  <resources>
    <resource ID="cpu" />
  </resources>
  <semaphores>
    <semaphore ID="mtx" />
  </semaphores>
  <mailboxes />
  <taskset>
    <task ID="Tsk1" type="periodic" intertime="40.0">
      <chunk BCET="5.0" WCET="10.0" ID="c1">
        <allocations>
          <allocation resource="cpu" priority="1" />
        </allocations>
        <synchronizations>
          <synchronization use="acquire" ID="mtx" />
        </synchronizations>
      </chunk>
    </task>
```

```
20   </taskset>
21 </timeline>
```

Documentazione

A proposito dei vincoli di contenuto inclusi nella documentazione, viene specificato quanto segue:

`timeline/resources/resource[@ID]` (3.1)

`timeline/semaphores/semaphore[@ID]` (3.2)

`timeline/mailboxes/mailbox[@ID]` (3.3)

`timeline/taskset/task[@ID]` (3.4)

`timeline/taskset/task/chunk[@ID]` (3.5)

- gli attributi 3.1, 3.2, 3.3, 3.4, 3.5 devono costituire delle chiavi e non devono contenere "_" o parole riservate[1];
- la coppia di attributi 3.2-3.3 deve essere vincolata da un vincolo di unicità;

`timeline/.../allocations/allocation[@resource]` (3.6)

`timeline/.../synchronizations/synchronization[@ID]` (3.7)

`timeline/.../synchronizations/synchronization[@use]` (3.8)

- l'attributo 3.6 deve essere un riferimento alla chiave costituita da 3.1;

[1]Le keyword riservate sono: `release`, `get`, `wait`, `boost` ed `exec`

- l'attributo 3.7 deve essere un riferimento alla chiave costituita da 3.2 o 3.3, rispettivamente quando l'attributo 3.8 ha valore *acquire* o *send / receive*;

```
timeline/taskset/task/chunk[@BCET]                (3.9)
timeline/taskset/task/chunk[@WCET]                (3.10)
timeline/taskset/task[@type]                      (3.11)
timeline/taskset/task[@intertime]                 (3.12)
timeline/taskset/task[@minIntertime]              (3.13)
timeline/taskset/task[@maxIntertime]              (3.14)
```

- l'attributo 3.10 deve essere maggiore dell'attributo 3.9 ed entrambi devono essere non nulli;
- se l'attributo 3.11 ha valore *periodic*, l'attributo 3.12 è obbligatorio, mentre 3.13 e 3.14 sono vietati;
- se l'attributo 3.11 ha valore *sporadic*, l'attributo 3.13 è obbligatorio, mentre 3.12 e 3.14 sono vietati;
- se l'attributo 3.11 ha valore *jittering*, gli attributi 3.13 e 3.14 sono obbligatori, mentre 3.12 è vietato;

3.1.2 XPN

Il formato XPN, derivato dall'XML, permette la rappresentazione di una PN nella sintassi quanto nella rappresentazione grafica (entrambe esaminate nel capitolo 1), attraverso attributi di posizione sia per posti e transizioni,

sia per gli aspetti più prettamente grafici come i relativi nome e parametri. Per quanto concerne lo schema di un file `.xpn`, c'è la necessità che questo sia compatibile con gli attuali documenti generati dall'editor Oris. In fase di progettazione dello schema, dunque, va tenuto conto dell'attuale conformazione di questi documenti. Vale la pena evidenziare che in Oris, ma anche in Sirio, viene rappresentata una versione estesa delle PN descritta nell'approfondimento 3.1.

Approfondimento 3.1

Estensione di una PN

La definizione di PN può essere estesa aggiungendo i seguenti termini:

- *archi inibitori* - $A^{\cdot} \subseteq P \times T$: una transizione t è *enabled* solo se tutti i suoi posti inibitori non hanno token;
- *enabling function* - $B(t) : M \to \{TRUE, FALSE\}$: restringe la condizione *enabled* di una transizione su vincoli basati sulla marcatura. Il caso di default è una condizione sempre vera, ovvero $b(t)(m) = TRUE$;
- *update function* - $U(t) : M \to M$: è un ulteriore aggiornamento della marcatura dopo il *firing* di una transizione. Il caso di default è l'identità $U(t)(m) = m$;
- *reset set* - $R(t) \subseteq T$: è un insieme di transizioni da considerarsi *newly enabled* dopo il *firing* della transizione. Il caso di default è l'insieme vuoto $R(t) = \varnothing$.

In figura 3.5 è riportata la struttura di base di un documento XPN (non

è completa, in quanto non enumera le possibili *features* e *properties* dei vari elementi che le prevedono). Rispetto alla struttura già in uso è stato necessario aggiungere l'elemento *resource* (ricordiamo che Oris attualmente non supporta le PTPN) e le *feature* e *property* che modellano il comportamento preemptive di una transizione.

L'elemento *joint* rappresenta un punto di interruzione di un arco. Di fatto un arco, qualora non sia una linea retta, ma spezzata, viene rappresentato come un sequenza di archi che, partendo dal punto di origine, si concatenano nei *joint*, per arrivare infine al punto di destinazione.

Circa i vincoli sui contenuti vale quanto detto per lo schema delle timelines: lo schema delinea solo la struttura e ignora l'aspetto dei contenuti.

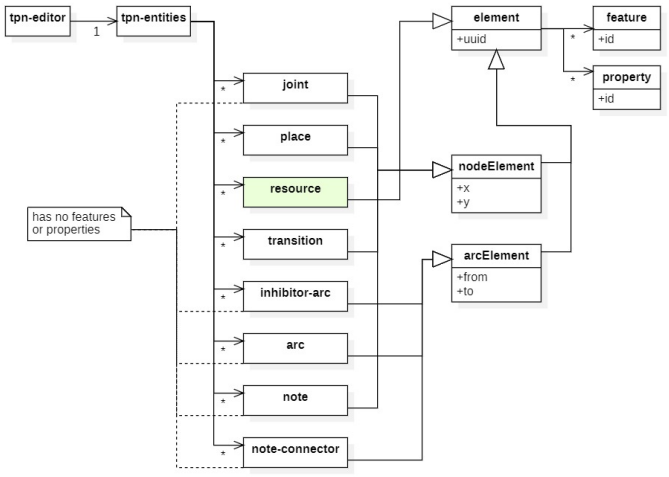

Figura 3.5: Modello concettuale della struttura di un XPN

XSD

Un'analisi dettagliata di questo schema sarebbe oltremodo lunga e, talvolta, fuoriluogo nella trattazione di questo progetto, per cui verranno descritti solamente alcuni punti particolarmente importanti:

- l'elemento radice è il tag `<tpn-editor>`, che, come si vede in figura 3.6, contiene il tag `<tpn-entities>`, il quale contiene tutti gli altri tag che corrispondono ai vari elementi di una PN;

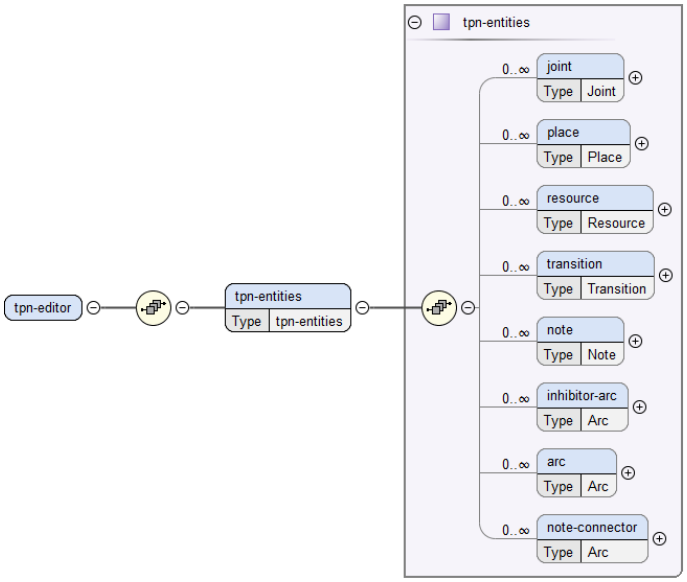

Figura 3.6: Visualizzazione grafica del tag `<tpn-editor>`

- i tag `<joint>`, `<note>`, `<inhibitor-arc>`, `<arc>` e `<note-connector>` contengono dei tag vuoti `features` e `properties`;

- il tag `<place>` contiene un tag vuoto `<features>` e un tag `<properties>` che contiene una sola `<property>` relativa al nome;

- il tag `<transition>` contiene un tag `<features>`, che può contenere dei tag `<feature>` relativi al comportamento *timed*, *stochastic* o *preemptive*, e un tag `<properties>`. Quest'ultimo contiene sempre dei tag `<property>` relativi a nome, *enabling function*, *marking update* e *reset transitions* e altri corrispondenti alle eventuali proprietà temporizzate, stocastiche o preemptive.

I punti appena toccati riguardano gli elementi ai quali lo schema si è adattato. Una trattazione più approfondita invece sarà fatta per gli elementi che sono stati aggiunti per la rappresentazione del comportamento preemptive:

- il tag `<resource>`, in figura 3.7, contiene un tag vuoto `<features>` e un tag `<properties>`. Questo contiene una sola istanza del tag `<property>` che deve avere attributi `id="0.default.name"` e `name`;

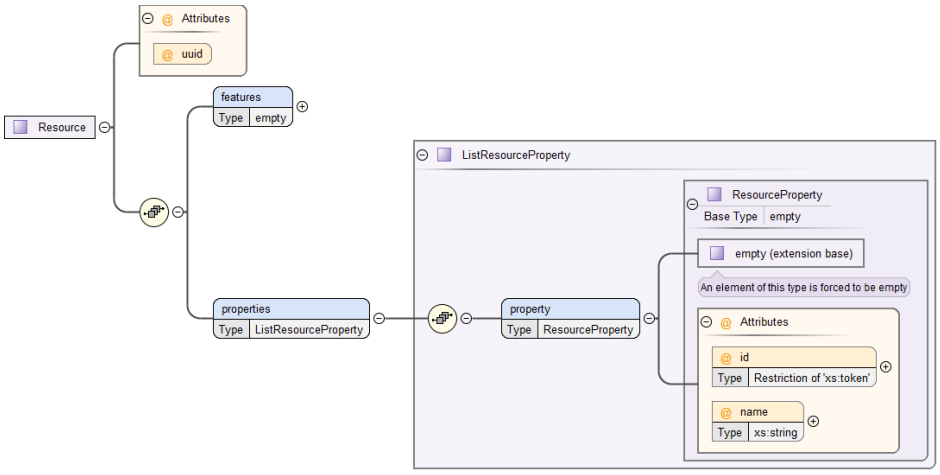

Figura 3.7: Visualizzazione grafica del tag `<resource>`

- una transizione preemptive deve avere:
  - una *feature* preemptive così rappresentata:

```
<feature id="transition.preemptive" />
```

  - una *property* preemptive così formata:

```
<property id="transition.preemptive"
    resources="res1;res2..."
    priorities="prio1;prio2..." />
```

Esempio

Un esempio esaustivo in tutti i possibili aspetti e funzioni sarebbe di dimensioni al limite dell'intrattabile, dunque ne viene proposto uno molto

breve che illustra gli aspetti riguardanti il comportamento preemptive di una PN. In questo esempio abbiamo una sola risorsa e una sola transizione con proprietà di temporizzazione e preemption.

```
<tpn-editor>
  <tpn-entities>
    <resource uuid="0...0">
      <features/>
      <properties>
        <property id="0.default.name" name="res"/>
      </properties>
    </resource>
    <transition x="0" y="0" rotation-angle="0" uuid="1...1">
      <features>
        <feature id="transition.timed"/>
        <feature id="transition.preemptive"/>
      </features>
      <properties>
        <property id="0.default.name" name="t" satellite-x="10"
            satellite-y="10"/>
        <property id="10.default.enablingFunction" enabling-
            function=""/>
        <property id="11.default.markingUpdate"  marking-update
            =""/>
        <property id="12.default.resetTransitions" reset-
            transitions=""/>
        <property id="transition.timed" eft="1" lft="2"/>
        <property id="transition.preemptive" resources="0...0"
            priorities="1"/>
      </properties>
    </transition>
  </tpn-entities>
</tpn-editor>
```

Documentazione

I vincoli di contenuto in questo schema sono molti e riguardano perlopiù gli attributi dei tag `<property>`, in quanto la maggior parte di essi dipendono dai valori assunti dagli altri attributi. Per esempio se una transizione ha un tag `<property id="transition.timed">`, allora questo dovrà avere anche gli attributi `eft` e `lft` e non dovrà contenere nessuno degli altri attributi previsti per questo tag. Questi vincoli sono molti e in generale di scarso

interesse per questi scopi. Gli unici che possono avere una qualche rilevanza sono già stati implicitamente trattati nella presentazione dello schema e sono quelli relativi ai comportamenti preemptive, per i quali sono state riportate le esatte forme che devono avere i tag.

Per una trattazione maggiormente approfondita di alcuni di questi vincoli si rimanda alle appendici A, dove si trovano gli XSD completi di documentazione integrata.

3.2 Algoritmo *timeline2ptpn*

L'algoritmo *timeline2ptpn* si basa, come detto, sul procedimento descritto nel capitolo 2. Le aggiunte sono state soprattutto precisazioni su aspetti non esplicitamente trattati. Un esempio è la presenza in un chunk di più di un meccanismo di sincronizzazione (sia esso l'acquisizione di un semaforo o il ricevimento di un messaggio). In questo caso i blocchi di acquisizione/ricezione devono essere concatenati. Sarebbe possibile anche collassare tutte le acquisizioni/ricezioni in un unico blocco, ma questo impedirebbe eventuali preemption fra l'acquisizione dei semafori; inoltre in una ipotetica traduzione della PTPN in codice real-time, ogni blocco di *wait* potrebbe essere tradotto in rapporto 1:1 con un'istruzione di *wait* senza ulteriore conoscenza sul taskset. Alla luce di questo aspetto quindi diventa importante l'ordine con cui i tag `<synchronization>` vengono specificati nella timeline, perché le rispettive sincronizzazioni rispettano tale ordine.
Inoltre viene aggiunto l'aspetto relativo all'*offset* di un task.

Di seguito viene proposto l'algoritmo. Data la lunghezza, i commenti non sono rimandati tutti in coda, ma viene spezzato l'algoritmo in più parti intervallate dai commenti. L'algoritmo a partire dalla timeline di input restituisce

la PTPN che ne rappresenta la sua traduzione.

Algoritmo 1 *timeline2ptpn*

Input: timeline da tradurre
Output: PTPN derivata dalla timeline

```
1: function TIMELINE2PTPN(timeline)
2:   ptpn ← new PTPN()
3:   for each r in timeline.resources do                ▷ adding resources
4:     resource ← new Resource(name = r.id)
5:     Add resource to ptpn
6:   for each s in timeline.semaphores do               ▷ adding semaphores
7:     semaphore ← new Place(name = s.id)
8:     semaphore.mark = 1
9:     Add semaphore to ptpn
10:  for each m in timeline.mailboxes do               ▷ adding mailboxes
11:    mailbox ← new Place(name = m.id)
12:    Add mailbox to ptpn
```

Risorse, semafori e mailbox hanno un equivalente elementare nel modello PTPN, quindi per ognuno di questi viene aggiunto alla PTPN i rispettivi risorsa e posti mantenendone inoltre il nome inalterato. Questa prima parte, molto semplice consta quindi di soli tre cicli `for each`.

```
13:  for each task in timeline.taskset do
14:    t_release ← new TimedTransition(name = task.id)   ▷ release transition
15:    switch on task.type
16:      case "periodic"
17:        t_release.firingTimes ← (task.intertime, task.intertime)
18:      case "sporadic"
19:        t_release.firingTimes ← (task.minIntertime, infinite)
20:      case "jittering"
21:        t_release.firingTimes ← (task.minIntertime, task.maxIntertime)
22:    Add t_release to ptpn
23:    if task.offset ≠ 0
24:      p_offset ← new Place()                          ▷ offset place
25:      p_offset.mark = 1
26:      Add p_offset to ptpn
```

27: $t_{offset} \leftarrow$ **new** $TimedTransition()$ ▷ offset transition
28: $t_{offset}.firingTimes \leftarrow (task.offset,\ task.offset)$
29: Add t_{offset} to $ptpn$
30: Add $p_{offset} \rightarrow t_{offset}$ arc to $ptpn$
31: $p_{release} \leftarrow$ **new** $Place()$ ▷ release place
32: Add $p_{release}$ to $ptpn$
33: Add $t_{offset} \rightarrow p_{release}$ arc to $ptpn$
34: Add $p_{release} \rightarrow t_{release}$ arc to $ptpn$
35: Add $t_{release} \rightarrow p_{release}$ arc to $ptpn$
36: $t_{last} \leftarrow t_{release}$ ▷ last chained transition

Il blocco del task è costituito da una sola transizione di rilascio (con proprietà di temporizzazione diverse a seconda del tipo di task) qualora il task non abbia offset. In caso contrario è prevista l'aggiunta di un'altra transizione che modella l'attesa dell'offset, un posto tra le due transizioni e un posto iniziale con un token. Questo all'inizio abiliterà, per un'unica volta, la transizione di offset, al cui *firing* viene abilitata la transizione di rilascio che autonomamente si riabilita al suo *firing*. In figura 3.8 una rappresentazione grafica di questo blocco.

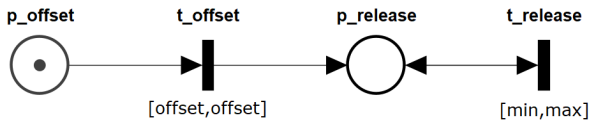

Figura 3.8: Traduzione di un task con offset

Alla linea 36, nella variabile t_{last} viene tenuto il riferimento all'ultima transizione del task (a questo punto quella di rilascio) per permettere la concatenazione dei blocchi successivi.

37: **for each** *chunk* **in** $task.chunks$ **do**
38: $p_{exec} \leftarrow$ **new** $Place()$ ▷ chunk place
39: Add p_{exec} to $ptpn$
40: $t_{exec} \leftarrow$ **new** $PreemptiveTransition()$ ▷ chunk transition

41: $t_{exec}.resources \leftarrow (chunk.resources,\ chunk.priorities)$
42: $t_{exec}.firingTimes \leftarrow (chunk.bcet,\ chunk.wcet)$
43: Add t_{exec} to $ptpn$
44: Add $p_{exec} \rightarrow t_{exec}$ arc to $ptpn$

La traduzione di un chunk rimane invariata, questo corrisponderà ad un posto e una transizione che rappresenta la sua esecuzione. Non viene ancora concatenato al resto del task perché eventuali sincronizzazioni sono da anteporre a questo.

45: **for each** *element* **in** $chunk.synchronizations$ **do**
46: **switch on** *element*'s type
47: **case** *semaphore*
48: $p_{wait} \leftarrow$ **new** $Place()$ ▷ waiting place
49: Add p_{wait} to $ptpn$
50: $t_{wait} \leftarrow$ **new** $PreemptiveTransition()$ ▷ waiting transition
51: $t_{wait}.resources \leftarrow (chunk.resources,\ chunk.priorities)$
52: $t_{wait}.firingTimes \leftarrow (0,\ 0)$
53: Add t_{wait} to $ptpn$
54: Add $p_{wait} \rightarrow t_{wait}$ arc to $ptpn$
55: Add $p_{semaphore.id} \rightarrow t_{wait}$ arc to $ptpn$
56: Add $t_{exec} \rightarrow p_{semaphore.id}$ arc to $ptpn$
57: **if** $chunk.priorities < semaphore.ceilings$ for some resource
58: $p_{boost} \leftarrow$ **new** $Place()$ ▷ boosting place
59: Add p_{boost} to $ptpn$
60: $t_{boost} \leftarrow$ **new** $PreemptiveTransition()$ ▷ boosting transition
61: $t_{boost}.resources \leftarrow (chunk.resources,\ chunk.priorities)$
62: $t_{boost}.firingTimes \leftarrow (0,\ 0)$
63: Add t_{boost} to $ptpn$
64: Add $p_{boost} \rightarrow t_{boost}$ arc to $ptpn$
65: $t_{wait}.priorities \leftarrow max\{t_{wait}.priorities,\ semaphore.ceilings\}$ for every resource
66: $t_{exec}.priorities \leftarrow max\{t_{exec}.priorities,\ semaphore.ceilings\}$ for every resource
67: Add $t_{boost} \rightarrow p_{wait}$ arc to $ptpn$
68: Add $t_{last} \rightarrow p_{boost}$ arc to $ptpn$
69: **if** $task.offset \neq 0$ **and** $t_{last} == t_{release}$
70: Add $t_{offset} \rightarrow p_{boost}$ arc to $ptpn$

71: **else**
72: Add $t_{last} \rightarrow p_{wait}$ arc to *ptpn*
73: **if** $task.offset \neq 0$ **and** $t_{last} == t_{release}$
74: Add $t_{offset} \rightarrow p_{wait}$ arc to *ptpn*
75: $t_{last} \leftarrow t_{wait}$ ▷ update last chained transition

A questo punto parte l'iterazione sugli elementi di sincronizzazione. Il primo caso trattato, che è anche quello più complesso, è il caso di un semaforo. Dopo la creazione del posto e transizione di *wait*, sempre presenti, viene controllato (linea 57) se è richiesto il *boost* delle priorità. Qualora sia richiesto, vengono aggiunti il posto e la transizione di *boost* (che viene collegata al posto di *wait*). In questa fase è importante anche provvedere all'effettivo *boost* delle priorità delle altre transizioni (linee 65-66). A questo punto il posto di *boost* viene concatenato all'ultima transizione del task e, se questa è quella di rilascio (ovvero l'acquisizione del semaforo è la prima operazione del primo chunk), anche all'eventuale transizione di offset.
Se invece non è richiesto il *boost* (linea 71), sarà la transizione di *wait* ad essere collegata all'ultima transizione del task ed eventualmente a quella di offset.

In ogni caso, alle linee 56-57, vengono aggiunte le precondizioni e postcondizioni da e verso il posto che rappresenta il semaforo.

Alla fine di tutto viene aggiornato il riferimento dell'ultima transizione del task che dovrà puntare alla transizione di *wait* appena creata.

76: **case** *incoming message*
77: $p_{receiving} \leftarrow$ **new** $Place()$ ▷ receiving place
78: Add $p_{receiving}$ to *ptpn*
79: $t_{receiving} \leftarrow$ **new** $PreemptiveTransition()$ ▷ receiving transition
80: $t_{receiving}.resources \leftarrow (chunk.resources,\ chunk.priorities)$
81: $t_{receiving}.firingTimes \leftarrow (0,\ 0)$
82: Add $t_{receiving}$ to *ptpn*

83: Add $p_{receiving} \rightarrow t_{receiving}$ arc to *ptpn*
84: Add $p_{mailbox.id} \rightarrow t_{receiving}$ arc to *ptpn*
85: Add $t_{last} \rightarrow p_{receiving}$ arc to *ptpn*
86: **if** $task.offset \neq 0$ **and** $t_{last} == t_{release}$
87: Add $t_{offset} \rightarrow p_{receiving}$ arc to *ptpn*
88: $t_{last} \leftarrow t_{receiving}$ ▷ update last chained transition
89: **case** *outgoing message*
90: Add $t_{exec} \rightarrow p_{mailbox.id}$ arc to *ptpn*

Per lo scambio di messaggi, il caso più complesso è quello di ricezione. Dopo la creazione della transizione e del posto di ricezione, quest'ultimo è collegato all'ultima transizione del task e, se questa è quella di rilascio ed è impostato un offset, anche a quella di offset.

Caso banale invece quello di invio che si traduce in un solo arco dalla transizione del chunk al posto che rappresenta la mailbox.

91: Add $t_{last} \rightarrow p_{exec}$ arc to *ptpn*
92: **if** $task.offset \neq 0$ **and** $t_{last} == t_{release}$
93: Add $t_{offset} \rightarrow p_{exec}$ arc to *ptpn*
94: $t_{last} \leftarrow t_{exec}$ ▷ update last chained transition
95: **return** *ptpn*

L'ultimo passo, prima del chunk o task successivo, è quello del collegamento del blocco del chunk con l'ultima transizione inserita. Anche in questo caso, analogamente a quanto descritto precedentemente, dovrà essere inserita anche un'eventuale postcondizione dalla transizione di offset. Infine viene ancora aggiornato il riferimento all'ultima transizione del task.

3.3 Design

La fase di design ha prodotto il class diagram in figura 3.9 e che verrà discusso. Il package *petrinet*, in grigio, era già implementato in Sirio.

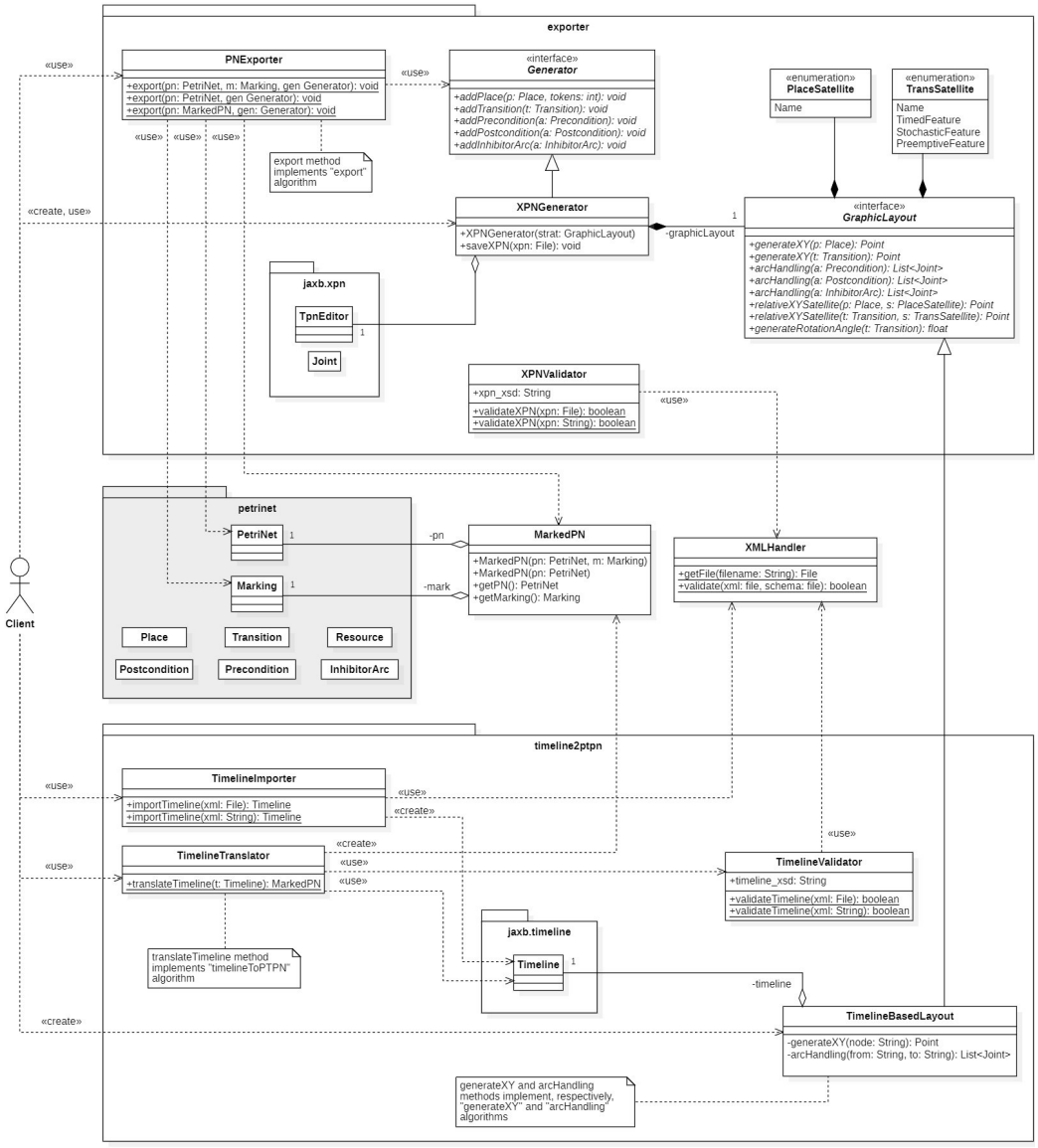

Figura 3.9: Class Diagram

Innanzi tutto sono state definite le classi:

- **MarkedPN**: è una *helper class*, la sua unica responsabilità è quella di aggregare un oggetto **PetriNet** e un oggetto **Marking**, per trattarli come un unico oggetto, che rappresenta una PN con marcatura;

- **XMLHandler** è una *utility class*, fornisce due metodi per la gestione dei documenti XML: ***getFile()*** restituisce, qualora esista, un file con un determinato nome e ***validate()*** valida un XML in base a un XSD.

Package exporter

In questo package sono contenute le classi che operano l'esportazione di una PN. L'esportazione è un'operazione generica, non dovrebbe essere vincolata a un formato particolare; perlomeno a livello teorico dovrebbe essere possibile definire un altro formato di output e voler esportare una PN in tale formato. L'output del processo di esportazione è un'altra rappresentazione della PN e sicuramente ci sarà una tappa nel corso del processo in cui questa è un oggetto complesso (anche qualora il prodotto finale sia un documento XML, prima del marshalling questo assume la forma di un oggetto complesso Java). Sembra quindi opportuno e conveniente che strutturalmente il processo di esportazione sia effettuato da un pattern *builder*.

Circa, poi, l'esportazione in formato XPN, questo prevede delle coordinate grafiche. L'assegnazione delle coordinate ai vari elementi in generale potrebbe avvenire in seguito ad una visita della PN e un algoritmo di sbroglio, tuttavia questa soluzione potrebbe essere notevolmente complessa. Nel caso particolare in cui si ha conoscenza della struttura della timeline, come in questo caso, questo compito potrebbe essere semplificato proprio a partire da questo aspetto. La soluzione più logica sembra quindi essere l'adozione di un pattern *strategy*, che consenta di scegliere dinamicamente una strategia per la disposizione grafica della PN.

Vediamo le classi:

- **PNExporter**: gioca il ruolo di *director* nel pattern *builder*; la sua responsabilità è quella di implementare un algoritmo per l'esportazione

dei vari elementi costituenti la PN, di cui poi incaricherà il *builder*. Fornisce tre varianti in overloading del metodo ***export()*** rispettivamente per PN senza marking, PN con marking e *MarkedPN*;

- **Generator**: è un'interfaccia e gioca il ruolo di *builder* nell'omonimo pattern; dichiara i metodi necessari ad un *builder* concreto per l'esportazione di una PN;

- **XPNGenerator**: costituisce il *builder* concreto relativo al formato XPN e il *context* per il il pattern *strategy*; tiene il riferimento ad un oggetto complesso ***xpn*** che potrà, alla fine del processo di esportazione, essere salvato su file tramite il metodo ***saveXPN()***. I metodi ***addPlace()*** e ***addTransition()*** attribuiscono come coordinate il risultato dei metodi ***generateXY()*** della strategia chiesta come parametro dal costruttore. I metodi ***addPrecondition()***, ***addPostcondition()*** e ***addInhibitorArc()*** invece sfruttano i metodi ***arcHandling()***;

- **GraphicLayout**: è l'interfaccia *strategy*; dichiara i metodi necessari ad attribuire tutti i parametri grafici ad un elemento della PN:

  - ***generateXY()*** nelle due versioni overloading per posti e transizioni genera le coordinate di questi elementi;

  - ***arcHandling()*** nelle tre versioni overloading per precondizioni, postcondizioni e archi inibitori genera una lista di *joints* (completi di coordinate) dai quali questi archi dovranno passare;

  - ***generateRotationAngle()*** genera l'angolo di rotazione di una transizione e

  - ***relativeXYSatellite()*** nelle due versioni overloading per posti e transizioni genera le coordinate per le informazioni satellite (nome, parametri temporali, stocastici e preemptive).

Il package contiene anche una *utility class* **XPNValidator** che contiene i metodi per la validazione di un documento XPN.

Package timeline2ptpn

Questo package contiene tre *utility class* rispettivamente per l'esportazione, la traduzione e la validazione di timeline:

- **TimelineImporter**: opera l'*unmarshalling* di una timeline;
- **TimelineTranslator**: il metodo ***translateTimeline()*** implementa l'algoritmo *timeline2ptpn* e restituisce una **MarkedPN**;
- **TimelineValidator**: contiene i metodi per la validazione di una timeline;

Infine contiene una classe **TimelineBasedLayout** che implementa l'interfaccia **GraphicLayout**: i suoi metodi implementano degli algoritmi per la generazione dei parametri grafici di una PN derivata da timeline a partire dalla conoscenza della timeline da cui è stata generata. Questa classe fornisce quindi l'ultimo strumento necessario per il processo di traduzione ed esportazione di una timeline in PTPN. Gli algoritmi implementati saranno discussi nelle prossime sezioni.

3.3.1 Realizzazione degli Use Case

Di seguito viene illustrato grazie a dei Sequence Diagram come il Class Diagram progettato permetta la realizzazione degli Use Case individuati in fase di analisi.

Importazione timeline e traduzione in PN

Il Sequence Diagram in figura 3.10 è molto semplice e non ha bisogno di ulteriori commenti. Viene trattato il caso in cui viene coinvolto anche il metodo ***getFile()***.

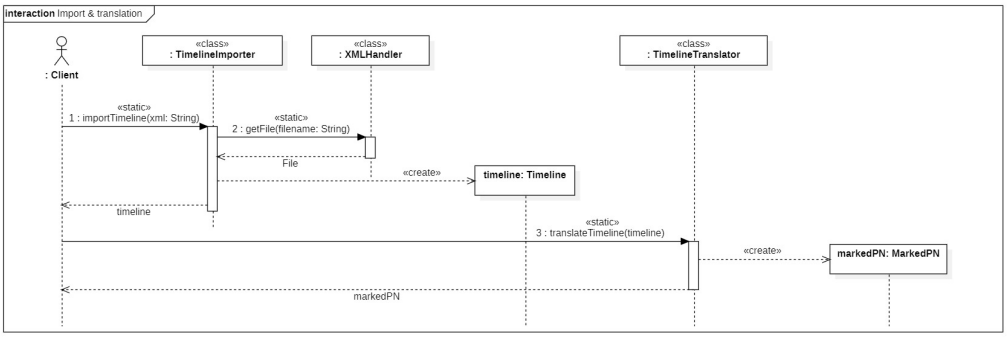

Figura 3.10: Sequence Diagram di importazione e traduzione di una timeline

Esportazione in XPN

A partire dal Sequence Diagram che riporta un'esportazione generica di una PN (figura 3.11), sono riportati nelle figure 3.12 e 3.13 i Sequence Diagrams che illustrano, rispettivamente, l'esportazione di una PN in formato XPN e l'esportazione in formato XPN di una PTPN derivata da una timeline, quindi con l'utilizzo della strategia **TimelineBasedLayout**.

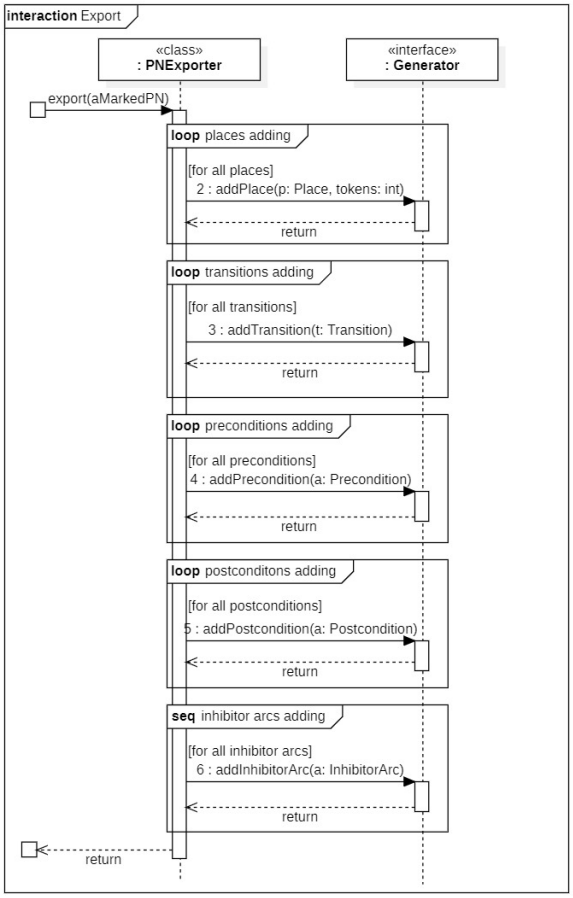

Figura 3.11: Sequence Diagram di esportazione di una PN

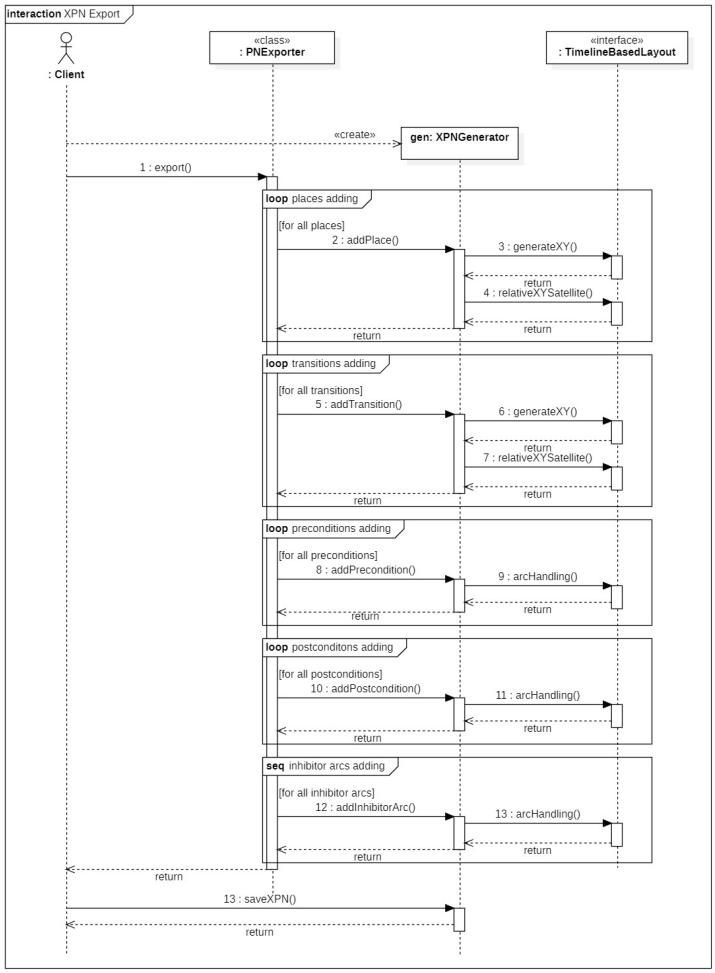

Figura 3.12: Sequence Diagram di esportazione in XPN di una PN

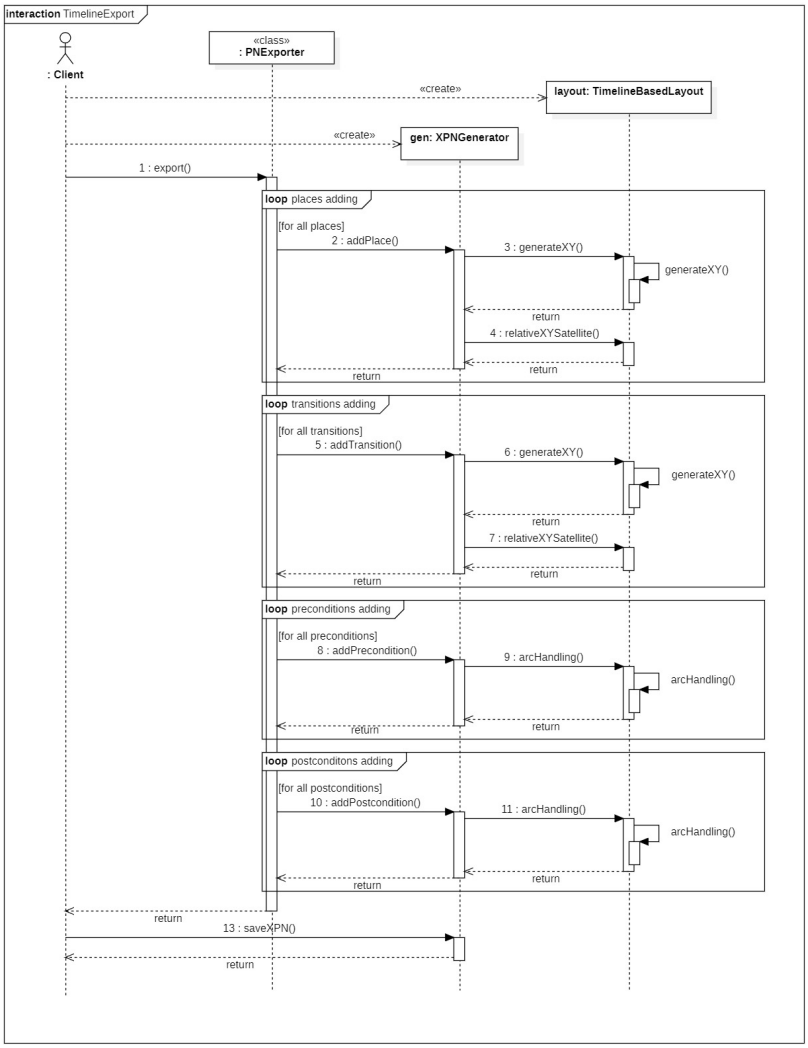

Figura 3.13: Sequence Diagram di esportazione in XPN di una PTPN derivata da timeline

3.4 Algoritmi per l'esportazione

3.4.1 export

L'algoritmo di esportazione implementato dalla classe **PNExport** è molto semplice: tutto ciò in cui consiste è l'iterazione sui vari componenti base della PN e per ognuno di essi la chiamata ad un metodo del builder che lo aggiunga al prodotto complesso.

Algoritmo 2 export

Input: PN da esportare, builder da usare per l'esportazione

```
1: function EXPORT(pn, builder)
2:     for each res in pn.resources do
3:         builder.addResource(res)
4:     for each p in pn.places do
5:         builder.addPlace(p)
6:     for each t in pn.transitions do
7:         builder.addTransition(t)
8:     for each pre in pn.preconditions do
9:         builder.addPrecondition(pre)
10:    for each post in pn.postconditions do
11:        builder.addPostconditions(post)
12:    for each ia in pn.inhibitorArcs do
13:        builder.addInhibitorArc(ia)
```

3.4.2 generateXY & arcHandling

Per la generazione delle coordinate l'idea di base è lo sviluppo dei task in linea: un task è rappresentato dal blocco di rilascio e dai vari blocchi dei chunks concatenati, per cui tutti questi elementi possono essere allineati. Ogni task si svilupperà su una linea e i task saranno incolonnati e allineati a sinistra. Rimangono da sistemare i posti relativi ai semafori e alle mailbox. Essendo i task allineati a sinistra e non essendoci un limite superiore alla

lunghezza di un task, viene naturale incolonnare questi posti sulla sinistra, prima dell'inizio del task. Il risultato si può vedere nell'esempio in figura 3.14, in cui non sono stati riportati gli archi da o verso semafori e mailboxes perché verranno discussi più avanti.

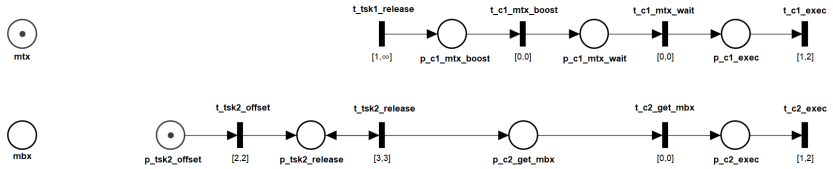

Figura 3.14: Esempio di assegnazione delle coordinate

Questa configurazione grafica risulta semplice e intuitiva, facilitando la comprensione della rete.

Prima di presentare l'algoritmo fissiamo delle costanti.

Algoritmo 3 definizione delle costanti

1: static $taskVerticalDistance = 150$
2: static $baseNodeDistance = 120$
3: static $distanceFromTask = 45$
4: static $arcDistance = 10$
5: static $deviation = 25$
6: static $synchBlockWidth = 4 * baseNodeDistance$
7: static $chunkBlockWidth = 2 * baseNodeDistance$
8: static $longNodeDistance = 2 * baseNodeDistance$
9: static $shortNodeDistance = baseNodeDistance$

Algoritmo 4 generateXY

Input: nome del nodo per cui generare le coordinate, timeline dalla quale è stata derivatala PTPN
Output: coordinate cartesiane generate

1: **function** GENERATEXY(node, timeline)
2: *point*
3: **if** *node* has $\_offset$ or $\_release$ suffix
4: $task = node$ without affixes

```
5:      switch on node's affixes
6:        case 'p_' and '_initial'                          ▷ offset place
7:          point.x = −3 ∗ shortNodeDistance
8:        case 't_' and '_offset'                           ▷ offset transition
9:          point.x = −2 ∗ shortNodeDistance
10:       case 'p_' and '_release'                          ▷ release place
11:         point.x = −shortNodeDistance
12:       case 'p_' and '_release'                          ▷ release transition
13:         point.x = 0
14:     point.y = timeline.getTasks().indexOf(task) ∗ taskVerticalDistance
```

La transizione di rilascio giace sull'ascissa 0, eventuali altri elementi di offset e posto di rilascio prenderanno valori negativi dell'ascissa. La distanza orizzontale è sempre calcolata tenendo presente il valore elementare $shortNodeDistance$, mentre la distanza verticale dei task facendo riferimento a $taskVerticalDistance$.

```
15:   else if node has _boost, _wait, or _get_ affix then
16:     chunk = node (without affix) first part
17:     task = timeline.taskWith(chunk)
18:     synch = node (without affix) last part
19:     previousSynchs =number of synch blocks before synch in the task
20:     chunkRank = task.getChunks.indexOf(chunk)
21:     synchBlockX = previousSynchs ∗ synchBlockWidth+
                                          chunkRank ∗ chunkBlockWidth
22:     switch on node's affixes
23:       case 'p_' and '_boost'                            ▷ boosting place
24:         point.x = synchBlockX + shortNodeDistance
25:       case 't_' and '_boost'                            ▷ boosting transition
26:         point.x = synchBlockX + 2 ∗ shortNodeDistance
27:       case 'p_' and '_wait'                             ▷ waiting place
28:         point.x = synchBlockX + 3 ∗ shortNodeDistance
29:       case 't_' and '_wait'                             ▷ waiting transition
30:         point.x = synchBlockX + 4 ∗ shortNodeDistance
31:       case 'p_' and '_get_'                             ▷ receiving place
32:         point.x = synchBlockX + longNodeDistance
```

```
33:        case 't_' and '_get_'                          ▷ receiving transition
34:          point.x = synchBlockX + 2 * longNodeDistance
35:      point.y = timeline.getTasks().indexOf(task) * taskVerticalDistance
```

Per i nodi relativi a *boost* e *wait* il valore dell'ascissa è un multiplo di *shortNodeDistance*, in particolare si ottiene calcolando il numero degli elementi precedenti a quello sulla linea del task. Per il blocco di ricezione si tiene conto del valore *longNodeDistance* (doppio rispetto a *shortNodeDistance*), per rendere uguale la lunghezza di un blocco ricezione con quella di un blocco di acquisizione (che in caso di boost prevede più elementi).

```
36:    else if node has _exec suffix then
37:      chunk = node without affixes
38:      task = timeline.taskWith(chunk)
39:      previousSynchs = number of synch blocks of task's chunks until
                                                          chunk (included)
40:      chunkRank = task.getChunks.indexOf(chunk)
41:      chunkBlockX = previousSynchs * synchBlockWidth+
                                          chunkRank * chunkBlockWidth
42:      switch on node
43:        case 'p_' and '_exec'                        ▷ it's a chunk place
44:          point.x = chunkBlockX + shortNodeDistance
45:        case 't_' and '_exec'                        ▷ it's a chunk transition
46:          point.x = chunkBlockX + 2 * shortNodeDistance
47:      point.y = timeline.getTasks().indexOf(task) * taskVerticalDistance
```

Ragionamento analogo al precedente anche per posto e transizione di esecuzione del chunk. Da notare che fino a questo punto il ruolo dei posti e delle transizioni si evince analizzandone il nome. Il nome degli elementi è fortemente standardizzato proprio per permettere che ciò avvenga. Questo è anche il motivo per il quale, come illustrato nella documentazione degli schemi, i nomi non possono contenere le keyword riservate. Gli elementi che non ne contengono sono per esclusione posti che rappresentano un semaforo

o una mailbox.

48: **else** ▷ mutex or mailbox
49: $height = (timeline.numberOfTask() - 1) * taskVerticalDistance$
50: $numOfSynchs = timeline.getNumberOfMutexAndMailboxes()$
51: $point.x = -5 * shortNodeDistance$
52: $point.y = (timeline.getMutexAndMailboxes().indexOf(node) + 1) * height/(numOfPlace + 1)$
53: **return** *point*

In questo caso l'ascissa è fissata. L'ordinata viene calcolata distribuendo gli elementi in maniera equidistante su tutta l'altezza disponibile.

Rimane da definire il percorso degli archi, ovvero i loro punti di interruzione. Gli archi verranno fatti passare parallelamente alla linea del task, sopra o sotto di essa. Gli archi che hanno origine in un semaforo o mailbox e sono diretti verso la linea del task passeranno sotto a questa, mentre quelli "uscenti", ovvero diretti verso semaforo o mailbox, passeranno sopra. Ogni arco di questo tipo avrà bisogno quindi di due *joints*, uno in prossimità del semaforo o mailbox e uno in prossimità della transizione del chunk: l'ascissa può quindi essere calcolata in base a questi punti di riferimento. L'ordinata sarà incrementata/decrementata a partire da un valore di base ad ogni arco successivo dello stesso task, per evitarne la sovrapposizione.

Un altro caso da trattare sono gli archi che dalla transizione di offset sono diretti verso il primo posto del primo chunk. Questi devono obbligatoriamente avere dei punti di spezzamento, per non rimanere sovrapposti a tutti gli altri. Questi passeranno sopra la linea del task a un'ordinata dedicata.

Tutti gli altri archi saranno diretti e formeranno (visivamente) la "linea" del task.

In figura 3.15 un esempio. L'algoritmo 5 non ha bisogno di ulteriori commenti ed è riportato di seguito.

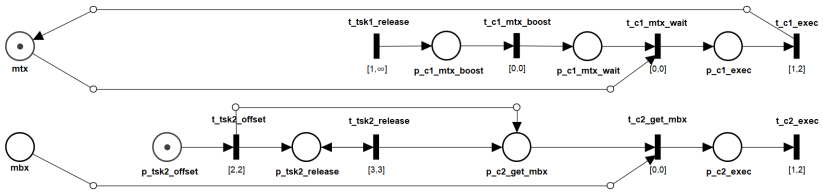

Figura 3.15: Esempio di gestione degli archi

Algoritmo 5 arcHandling

Input: nome del nodo di partenza, nome del nodo di destinazione, timeline da cui è stata derivata la PTPN

Output: lista di joint da cui l'arco deve passare

1: **function** ARCHANDLING(from, to, timeline)
2: $joints$ = list of Joint
3: **if** to is a mailbox or a mutex
4: $chunk = from$' prefix
5: $sync = to$
6: $ranking = timeline.taskWith(chunk).getSynchronizations$
$.indexOf(sync)$
7: $firstJoint =$ **new** $Joint()$
8: $firstJoint.x = GenerateXY(from).x - 25$
9: $firstJoint.y = GenerateXY(from).y - distanceFromTask -$
$arcDistance * (ranking + 1)$
10: $joints.add(firstJoint)$
11: $secondJoint =$ **new** $Joint()$
12: $secondJoint.x = -4 * shortNodeDistance$
13: $secondJoint.y = GenerateXY(from).y - distanceFromTask -$
$arcDistance * (ranking + 1)$
14: $joints.add(secondJoint)$
15: **else if** $from$ is a mailbox or a mutex **then**
16: $chunk = to$'s prefix
17: $sync = from$
18: $ranking = timeline.getChunk(chunk).getSynchronizations()$
$.indexOf(sync)$
19: $firstJoint =$ **new** $Joint()$
20: $firstJoint.x = -4 * shortNodeDistance$
21: $firstJoint.y = GenerateXY(to).y + distanceFromTask +$
$arcDistance * (ranking)$

22: $joints.add(firstJoint)$
23: $secondJoint =$ **new** $Joint()$
24: $secondJoint.x = GenerateXY(to).x - deviation$
25: $secondJoint.y = GenerateXY(to).y + distanceFromTask +$
$arcDistance * (ranking)$
26: $joints.add(secondJoint)$
27: **else if** $from$ is an offset transition and to is the first place of a chunk **then**
28: $firstJoint =$ **new** $Joint()$
29: $firstJoint.x = GenerateXY(from).x$
30: $firstJoint.y = GenerateXY(from).y -$
$distanceFromTask$
31: $joints.add(firstJoint)$
32: $secondJoint =$ **new** $Joint()$
33: $secondJoint.x = GenerateXY(to).x$
34: $secondJoint.y = GenerateXY(to).y -$
$distanceFromTask$
35: $joints.add(secondJoint)$
36: **return** $joints$

Capitolo 4

Implementazione e testing

4.1 Implementazione

In questo capitolo verranno commentati brevemente alcuni aspetti implementativi di rilievo alla luce di quanto definito in fase di progettazione dagli algoritmi e dal Class Diagram.

4.1.1 TimelineTranslator

In questa classe, troviamo gli aspetti implementativi probabilmente più interessanti da trattare. In particolare nell'algoritmo *timeline2ptpn* troviamo alcuni particolari che non possono essere trasformati in codice immediatamente.
In particolare è necessario determinare se un chunk che acquisisce un semaforo necessita di boosting (linea 57 dell'algoritmo) e, in caso affermativo, sottoporre le transizioni di *wait* e di completamento del chunk all'incremento della priorità. Sicuramente per fare ciò è utile anzitutto memorizzare i *ceilings* dei semafori, ovvero la priorità più alta con cui quel semaforo può essere richiesto. In questo caso ogni semaforo ha un *ceiling* per ogni risorsa.

Questo è stato tradotto in Java in una mappa annidata inizializzata da un metodo privato che altro non fa che iterare su ogni task, chunk, acquisizione di semaforo e allocazione per determinare le priorità di *ceiling*:

```
HashMap<String,HashMap<Resource,ResourcePriority>> ceilings =
    determineCeilings(timeline, resources);
```

dove `resources` a sua volta è una mappa `HashMap<String, Resource>` usata per memorizzare le risorse, perché in Sirio le PTPN non memorizzano le risorse, se non nelle *features* delle transizioni preemptive. Questa mappa, usata come registro delle risorse, ne semplifica la gestione durante tutto il processo di traduzione.

Il controllo per determinare se un chunk ha necessità di subire il *boost* viene implementato in un altro *helper method*:

```
boolean needBoost(
    Chunk chunk,
    HashMap<String,Resource> resources,
    HashMap<Resource,ResourcePriority> semaphoreCeilings
)
```

che si limita all'iterazione sulle risorse allocate al chunk per verificare se la priorità sia uguale al *ceiling*, in caso contrario è necessario aumentarla.

Nella traduzione in PTPN questo significa aumentare le priorità di alcune transizioni. Questo è il compito dell'ultimo *helper method*, che iterando sulle risorse della transizione ne aumenta la priorità (a quella di *ceiling*) solo per quelle che lo richiedono.

```
void boostTransitions(
    HashMap<Resource,ResourcePriority> semaphoreCeilings,
    Transition... transitions
)
```

4.1.2 GraphicLayout

L'interfaccia **GraphicLayout** è stata dotata di implementazioni di default per tutti i metodi, in particolare: i metodi che devono ritornare delle coordinate (***generateXY()***, ***relativeXYSatellite()***) restituiscono il punto $(0, 0)$, mentre quelli che ritornano una lista di joints (***arcHandling()***) restituiscono una lista vuota.

4.1.3 XPNGenerator

Durante l'aggiunta di un arco, ma non solo, è necessario, a partire dal nome di un elemento della PTPN precedentemente inserito, risalire all'identificativo che gli era stato assegnato in fase di inserimento. Implementare una visita sulla PTPN per la ricerca sarebbe oneroso e soprattutto non necessario: infatti è sufficiente tenere un registro degli elementi inseriti con relativo identificativo da aggiornare ogniqualvolta ne venga inserito uno. Questo viene fatto con l'uso della mappa:

```
1 HashMap<String,String> uuidDict
```

In fase di esportazione di una transizione vengono operati tutti gli accorgimenti necessari per la corretta traduzione delle *features* che in Oris devono essere rappresentate in modi diversi da quanto accade in Sirio (un approfondimento su ciò si trova in appendice B). Per le componenti che invece non sono supportate viene lanciata un'eccezione.

4.1.4 TimelineBasedLayout

In questa classe, come si evince dagli algoritmi che implementa presentati nel capitolo 3, sono necessari degli *helper methods* che implementano una visita sulla timeline affinché, a partire dal nome di un posto o transizione, venga

ritrovato il chunk, il task o l'elemento di sincronizzazione a cui corrisponde. Sono inoltre necessari dei metodi che a partire da un posto o transizione restituiscano il numero di blocchi di sincronizzazione che lo precedono. Questo per calcolare in modo corretto l'ascissa dell'elemento. Questi compiti sono assolti dai seguenti metodi, che come è facile intuire consistono in iterazioni annidate su task, chunk ed elementi di sincronizzazione.

```
1 Chunk findChunk(String nodeName)
2 Task findTask(String nodeName)
3 Synchronization findSynchElem(String nodeName)
4 int previousSynchBlock(String nodeName)
5 int previousSynchElem(String nodeName)
```

4.2 Testing

Tutto il codice è stato sottoposto a *unit test*. Il framework usato per il testing è *jUnit* [14] nella sua ultima versione *jUnit 5*. Tramite il plugin *JaCoCo*, poi, è stato valutato e analizzato il *branch coverage*.

Complessivamente è stato ottenuto un *branch coverage* del 93%. Sebbene questa percentuale possa sembrare lontana dalla copertura completa, può essere ritenuta sufficiente dopo l'analisi dei rami mancanti. Infatti si scopre che i rami mancanti sono in realtà inesistenti: sono frutti di controlli eccessivi, duplicati, che però constribuiscono a dare al codice maggiore leggibilità. Molti di questi si trovano in **timelineBasedLayout**, che non a caso è la classe con più blocchi condizionali, spesso annidati, e quindi con più rami (con *branch coverage* dell'89%). Per questi motivi è stato ritenuto non necessario il *refactor*. Ragionamento analogo, anche se su minor scala data la copertura del 94%, per la classe **XPNGenerator**.

Altri rami, e istruzioni, non coperti sono quelli relativi alle eccezioni che vengono lanciate in caso di mancanza degli schemi XSD. Al netto di questi

casi, il package *timeline2ptpn* ha una copertura totale.

In una prima fase di testing sono stati testati gli *helper methods*, introdotti nel capitolo 4. Il test di questi metodi privati è stato reso possibile ricorrendo alla *Reflection API* di Java che permette esaminare e modificare il comportamento di metodi e classi a runtime. In questo caso è stato abilitato l'accesso a questi metodi per eseguire i test.
Solo dopo aver testato questi metodi di appoggio, sono stati posti sotto test gli altri metodi. Per questo scopo sono state create per via programmatica delle semplici timeline o delle PN che mettessero in risalto di volta in volta l'aspetto di interesse.

Per il testing delle classi e dei metodi relativi alla traduzione sono stati messi a punto numerosi test che coprissero ogni possibile componente (task, chunk, acquisizione di semaforo, invio e ricezione di un messaggio) in ogni combinazione (presenza o meno di offset, presenza singola o multipla di sincronizzazioni, etc.). Per quanto riguarda l'aspetto dell'esportazione invece lo sforzo maggiore è stato quello di determinare tutti i possibili modi con cui una transizione può essere settata, sia quelle temporizzate e preemptive, ma soprattutto quelle stocastiche, che hanno un ampio parco di possibilità.

Capitolo 5

Esempi

Verrà mostrato adesso il funzionamento in pratica del sistema. Come timeline di esempio da cui partire viene usato un taskset con cinque tasks sincronizzati con due mutex e una mailbox, proposto in [5] e riportato in figura 5.1. Al fine di mostrare anche un risultato relativo alla presenza di offset viene presa in considerazione anche una sua variante che differisce unicamente per la presenza di offset nel task *Tsk3*.

Le timeline sono state espresse in XML secondo il relativo schema, tradotte in PTPN e successivamente esportate in XPN; i risultati possono essere visti in figure 5.2 e 5.3. Da notare la mancanza nelle figure di risorse e priorità, in quanto non supportando Oris le transizioni preemptive, in fase di esportazione, i modelli PTPN sono stati privati delle caratteristiche preemptive.

Da questi esempi si evince che in caso di timeline con un grado di complessità maggiore, il posizionamento grafico dei posti di semafori e mailboxes potrebbe non essere soddisfacente, ovvero non abbastanza chiaro e intuitivo per via della presenza di numerosi archi. D'altronde è ragionevole pensare che all'aumentare della complessità, diminuisca l'importanza dell'esportazio-

ne nel tool grafico: una PN particolarmente estesa non offre più, da un punto di vista grafico, intuitività e maneggevolezza. In questo caso l'unico aspetto di reale interesse è la possibilità di analizzare la PN programmaticamente nel dominio Sirio.

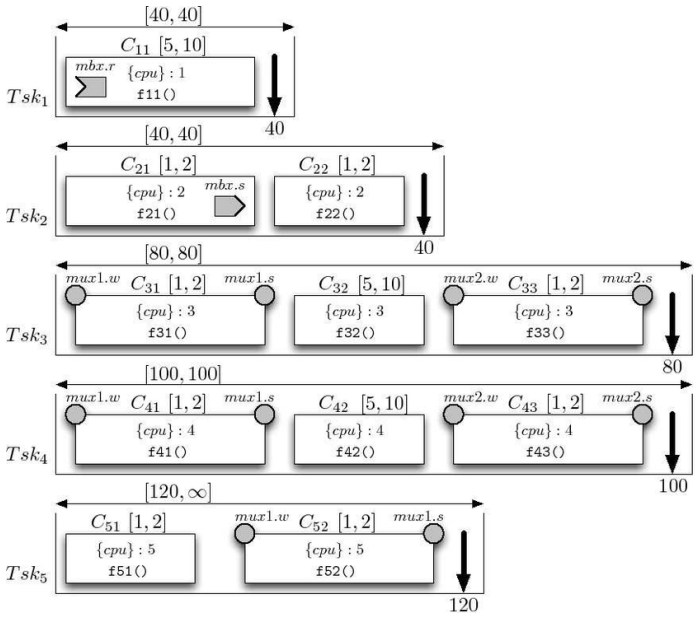

Figura 5.1: Esempio di timeline

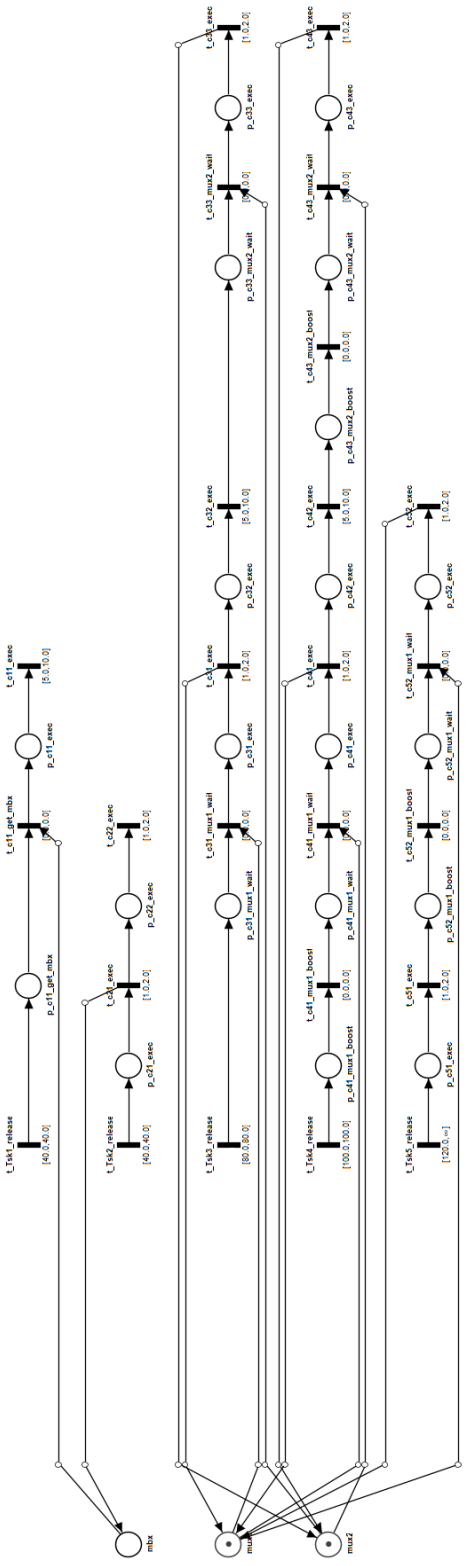

Figura 5.2: Traduzione in PN della timeline in figura 5.1

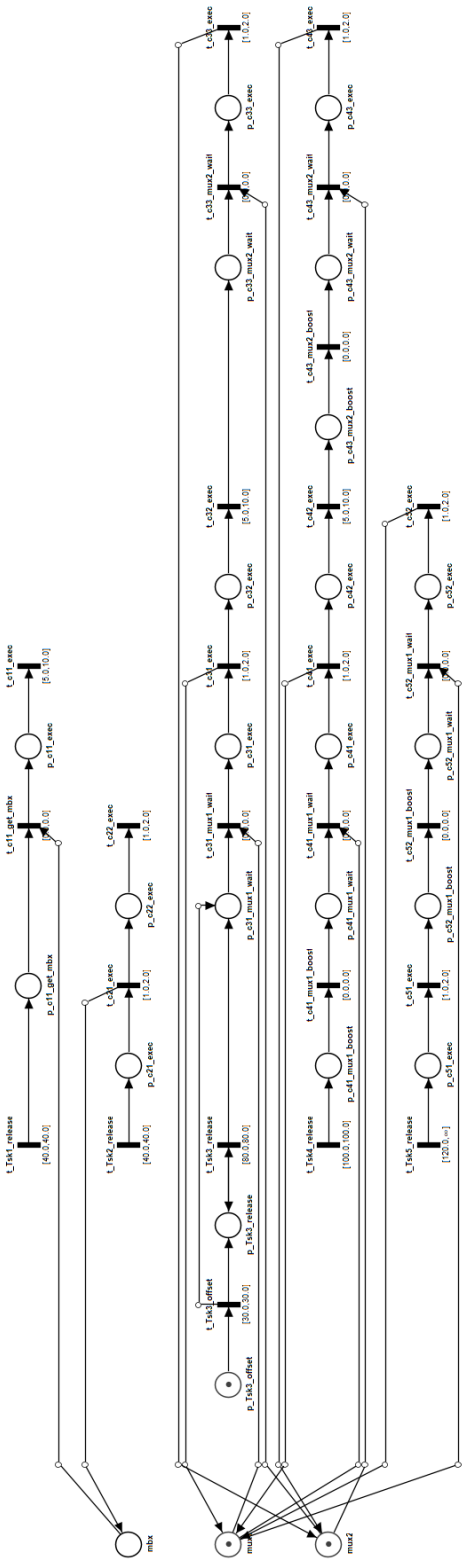

Figura 5.3: Traduzione in PN della variante con offset della timeline in figura 5.1

Capitolo 6

Conclusioni

Il formalismo delle PTPN estende le TPN con un meccanismo di assegnazione delle risorse. Questo permette la modellazione di tasksets, risultando di sostegno in varie fasi dello sviluppo di sistemi RT, dal design fino al testing.

L'algoritmo presentato in questo lavoro automatizza il processo di traduzione dei tasksets in forma di timeline che comprendono task periodici, sporadici e jittering, con o senza offset, formati da chunk con tempi di esecuzione non deterministici, sincronizzati con semafori e mailboxes e che adottano il protocollo di *priority ceiling*. Dopo la definizione di uno schema per la rappresentazione dei tasksets in forma di timeline, l'algoritmo è stato implementato in un componente software che si integra nella libreria Sirio e che opera la traduzione a partire proprio da una timeline. In questo dominio è possibile procedere all'analisi del modello.

A sostegno di questo componente, ma non solo, ne è stato sviluppato un secondo che permette l'esportazione dei modelli PN dal dominio Sirio al tool Oris, così da consentire la manipolazione grafica degli stessi. Per la sistemazione grafica degli elementi è stato possibile avvalersi della conoscenza sulla struttura dei tasksets.

Questo progetto apre gli scenari alla progettazione e sviluppo di altri componenti software come, per esempio, l'automatizzazione del processo di scrittura del codice RT a partire dai modelli PTPN così ottenuti o lo sbroglio di una PN per l'esportazione grafica.

Appendice A

Schemi XSD

JAXB permette la *personalizzazione del binding*, attraverso la specifica nell'XSD di un comportamento diverso da quello di default nel processo di conversione. Un esempio è la possibilità di forzare un tipo primitivo dell'XSD ad essere tradotto nelle classi Java in un tipo diverso da quello previsto di default (i.e. convertire un `xsd:short` in un `int` anziché in uno `short`).

In questo progetto è stato utile ricorrere a questa soluzione in due casi:

- la classe del tipo `Resource` generata dall'XSD relativo alla timeline, produceva ambiguità con la classe `Resource` di Sirio. Non essendoci i motivi per cambiare nome del tipo nell'XSD, è stato specificato che la classe relativa a quel tipo dovesse chiamarsi `TimelineResource`, risolvendo così l'ambiguità;

- una enumerazione dell'XSD non viene tipicamente tradotta in un Java `enum`. Grazie alla personalizzazione del binding questo è possibile ed è risultato particolarmente utile in questo progetto (entrambi gli schemi presentano diverse enumerazioni: tipo di task, proprietà di un

posto, proprietà di una transizione, proprietà di una risorsa, tipo di distribuzione stocastica).

Adesso viene proposta la versione integrale dei due schemi (mentre un loro esempio è già stato proposto nel capitolo 3).

A.1 timeline.xsd

```
<xs:schema xmlns="http://www.oris-tool.org" elementFormDefault="
    qualified" targetNamespace="http://www.oris-tool.org" xmlns:xs="http
    ://www.w3.org/2001/XMLSchema" xmlns:jaxb="http://java.sun.com/xml/ns
    /jaxb" jaxb:version="2.1">

  <xs:annotation>
    <xs:appinfo>
      <jaxb:schemaBindings>
        <jaxb:package name="org.oristool.timeline" />
      </jaxb:schemaBindings>
      <jaxb:bindings node="//xs:complexType[@name='Resource']">
        <jaxb:class name="TimelineResource" />
        <jaxb:property name="TimelineResource" />
      </jaxb:bindings>
    </xs:appinfo>
  </xs:annotation>

  <xs:element name="timeline">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="resources" type="ListResource" minOccurs="1"
            maxOccurs="1" />
        <xs:element name="semaphores" type="ListSemaphore"
            minOccurs="1" maxOccurs="1" />
        <xs:element name="mailboxes" type="ListMailbox" minOccurs="1"
            maxOccurs="1" />
        <xs:element name="taskset" type="Taskset" minOccurs="1"
            maxOccurs="1" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="empty">
    <xs:annotation>
      <xs:documentation xml:lang="EN">An element of this type is forced
          to be empty</xs:documentation>
    </xs:annotation>
  </xs:complexType>

```

```
32  <xs:complexType name="ListResource">
33    <xs:sequence>
34      <xs:element name="resource" type="Resource" minOccurs="0"
            maxOccurs="unbounded" />
35    </xs:sequence>
36  </xs:complexType>
37
38  <xs:complexType name="Resource">
39    <xs:complexContent>
40      <xs:extension base="empty">
41        <xs:attribute name="ID" type="xs:token" use="required" />
42      </xs:extension>
43    </xs:complexContent>
44  </xs:complexType>
45
46  <xs:complexType name="ListSemaphore">
47    <xs:sequence>
48      <xs:element name="semaphore" type="Semaphore" minOccurs="0"
            maxOccurs="unbounded" />
49    </xs:sequence>
50  </xs:complexType>
51
52  <xs:complexType name="Semaphore">
53    <xs:complexContent>
54      <xs:extension base="empty">
55        <xs:attribute name="ID" type="xs:token" use="required" />
56      </xs:extension>
57    </xs:complexContent>
58  </xs:complexType>
59
60  <xs:complexType name="ListMailbox">
61    <xs:sequence>
62      <xs:element name="mailbox" type="Mailbox" minOccurs="0" maxOccurs
            ="unbounded" />
63    </xs:sequence>
64  </xs:complexType>
65
66  <xs:complexType name="Mailbox">
67    <xs:complexContent>
68      <xs:extension base="empty">
69        <xs:attribute name="ID" type="xs:token" use="required" />
70      </xs:extension>
71    </xs:complexContent>
72  </xs:complexType>
73
74  <xs:complexType name="Taskset">
75    <xs:sequence>
76      <xs:element name="task" type="Task" minOccurs="0" maxOccurs="
            unbounded" />
77    </xs:sequence>
78  </xs:complexType>
79
80  <xs:complexType name="Task">
```

```
81      <xs:annotation>
82        <xs:documentation xml:lang="EN">A periodic task must specify
              intertime value, a sporadic task must specify minIntertime
              value and a jittering task must specify minIntertime and
              maxIntertime values</xs:documentation>
83        <xs:appinfo>
84          <jaxb:bindings node="//xs:attribute[@name='type']/xs:simpleType
                ">
85            <jaxb:typesafeEnumClass name="TaskType" />
86          </jaxb:bindings>
87        </xs:appinfo>
88      </xs:annotation>
89      <xs:sequence>
90        <xs:element name="chunk" type="Chunk" maxOccurs="unbounded" />
91      </xs:sequence>
92      <xs:attribute name="ID" type="xs:token" use="required" />
93      <xs:attribute name="type" default="periodic">
94        <xs:simpleType>
95          <xs:restriction base="xs:token">
96            <xs:enumeration value="periodic" />
97            <xs:enumeration value="sporadic" />
98            <xs:enumeration value="jittering" />
99          </xs:restriction>
100       </xs:simpleType>
101     </xs:attribute>
102     <xs:attribute name="intertime" type="xs:decimal" use="optional" />
103     <xs:attribute name="minIntertime" type="xs:decimal" use="optional"
          />
104     <xs:attribute name="maxIntertime" type="xs:decimal" use="optional"
          />
105     <xs:attribute name="offset" default="0.0" type="xs:decimal" use="
          optional" />
106   </xs:complexType>
107
108   <xs:complexType name="Chunk">
109     <xs:sequence>
110       <xs:element name="allocations" type="ListAllocation" minOccurs="1"
            maxOccurs="1" />
111       <xs:element name="synchronizations" type="ListSynchronization"
            minOccurs="1" maxOccurs="1" />
112     </xs:sequence>
113     <xs:attribute name="BCET" type="xs:decimal" use="required" />
114     <xs:attribute name="WCET" type="xs:decimal" use="required" />
115     <xs:attribute name="ID" type="xs:token" use="required" />
116   </xs:complexType>
117
118   <xs:complexType name="ListAllocation">
119     <xs:sequence>
120       <xs:element name="allocation" type="Allocation" minOccurs="0"
            maxOccurs="unbounded" />
121     </xs:sequence>
122   </xs:complexType>
123
```

```
124   <xs:complexType name="Allocation">
125     <xs:complexContent>
126       <xs:extension base="empty">
127         <xs:attribute name="resource" type="xs:token" use="required" />
128         <xs:attribute name="priority" type="xs:unsignedShort" use="
              required" />
129       </xs:extension>
130     </xs:complexContent>
131   </xs:complexType>
132
133   <xs:complexType name="ListSynchronization">
134     <xs:sequence>
135       <xs:element name="synchronization" type="Synchronization"
            minOccurs="0" maxOccurs="unbounded" />
136     </xs:sequence>
137   </xs:complexType>
138
139   <xs:complexType name="Synchronization">
140     <xs:annotation>
141       <xs:appinfo>
142         <jaxb:bindings node="//xs:attribute[@name='use']/xs:simpleType">
143           <jaxb:typesafeEnumClass name="SynchronizationType" />
144         </jaxb:bindings>
145       </xs:appinfo>
146     </xs:annotation>
147     <xs:complexContent>
148       <xs:extension base="empty">
149         <xs:attribute name="ID" type="xs:token" use="required" />
150         <xs:attribute name="use" use="required">
151           <xs:simpleType>
152             <xs:restriction base="xs:token">
153               <xs:enumeration value="acquire" />
154               <xs:enumeration value="send" />
155               <xs:enumeration value="receive" />
156             </xs:restriction>
157           </xs:simpleType>
158         </xs:attribute>
159       </xs:extension>
160     </xs:complexContent>
161   </xs:complexType>
162 </xs:schema>
```

A.2 XPN.xsd

```
1 <xs:schema xmlns="http://www.oris-tool.org" elementFormDefault="
    qualified" targetNamespace="http://www.oris-tool.org" xmlns:xs="http
    ://www.w3.org/2001/XMLSchema" xmlns:jaxb="http://java.sun.com/xml/ns
    /jaxb" jaxb:version="2.1">
2
```

```
3   <xs:annotation>
4     <xs:appinfo>
5       <jaxb:schemaBindings>
6         <jaxb:package name="org.oristool.xpn" />
7       </jaxb:schemaBindings>
8       <jaxb:bindings node="//xs:complexType[@name='ResourceProperty']/xs
            :complexContent/xs:extension/xs:attribute[@name='id']/xs:
            simpleType">
9         <jaxb:typesafeEnumClass name="ResourcePropertyType">
10          <jaxb:typesafeEnumMember value="0.default.name" name="NAME" />
11        </jaxb:typesafeEnumClass>
12      </jaxb:bindings>
13      <jaxb:bindings node="//xs:complexType[@name='PlaceProperty']/xs:
            complexContent/xs:extension/xs:attribute[@name='id']/xs:
            simpleType">
14        <jaxb:typesafeEnumClass name="PlacePropertyType">
15          <jaxb:typesafeEnumMember value="0.default.name" name="NAME" />
16                  <jaxb:typesafeEnumMember value="default.marking" name
                        ="MARKING" />
17                </jaxb:typesafeEnumClass>
18          </jaxb:bindings>
19          <jaxb:bindings node="//xs:complexType[@name='TransitionFeature
                ']/xs:complexContent/xs:extension/xs:attribute[@name='id
                ']/xs:simpleType">
20                <jaxb:typesafeEnumClass name="TransitionFeatureType">
21                  <jaxb:typesafeEnumMember value="transition.timed" name
                        ="TIMED" />
22                  <jaxb:typesafeEnumMember value="transition.stochastic"
                         name="STOCHASTIC" />
23                  <jaxb:typesafeEnumMember value="transition.preemptive"
                         name="PREEMPTIVE" />
24            </jaxb:typesafeEnumClass>
25          </jaxb:bindings>
26          <jaxb:bindings node="//xs:complexType[@name='
                TransitionProperty']/xs:complexContent/xs:extension/xs:
                attribute[@name='id']/xs:simpleType">
27                <jaxb:typesafeEnumClass name="TransitionPropertyType">
28                  <jaxb:typesafeEnumMember value="0.default.name" name="
                        NAME" />
29                  <jaxb:typesafeEnumMember value="10.default.
                        enablingFunction" name="ENABLING_FUNCTION" />
30                  <jaxb:typesafeEnumMember value="11.default.
                        markingUpdate" name="MARKING_UPDATE" />
31                  <jaxb:typesafeEnumMember value="12.default.
                        resetTransitions" name="RESET_TRANSITIONS" />
32                  <jaxb:typesafeEnumMember value="transition.timed" name
                        ="TIMED" />
33                  <jaxb:typesafeEnumMember value="transition.stochastic"
                         name="STOCHASTIC" />
34                  <jaxb:typesafeEnumMember value="transition.preemptive"
                         name="PREEMPTIVE" />
35                </jaxb:typesafeEnumClass>
36          </jaxb:bindings>
```

```
37          <jaxb:bindings node="//xs:complexType[@name='
                TransitionProperty']/xs:complexContent/xs:extension/xs:
                attribute[@name='property-data-type']/xs:simpleType">
38              <jaxb:typesafeEnumClass name="StochasticDistribution">
39                <jaxb:typesafeEnumMember value="0.type.immediate" name
                      ="IMMEDIATE" />
40                <jaxb:typesafeEnumMember value="1.type.uniform" name="
                      UNIFORM" />
41                <jaxb:typesafeEnumMember value="2.type.deterministic"
                      name="DETERMINISTIC" />
42                <jaxb:typesafeEnumMember value="3.type.exponential"
                      name="EXPONENTIAL" />
43                <jaxb:typesafeEnumMember value="4.type.erlang" name="
                      ERLANG" />
44                <jaxb:typesafeEnumMember value="5.type.expolynomial"
                      name="EXPOLYNOMIAL" />
45              </jaxb:typesafeEnumClass>
46          </jaxb:bindings>
47        </xs:appinfo>
48  </xs:annotation>
49
50
51  <xs:element name="tpn-editor">
52    <xs:complexType>
53          <xs:sequence>
54            <xs:element name="tpn-entities" type="tpn-entities"
                  maxOccurs="1" minOccurs="1" />
55          </xs:sequence>
56        </xs:complexType>
57  </xs:element>
58
59  <xs:complexType name="tpn-entities">
60        <xs:sequence>
61          <xs:element name="joint" type="Joint" minOccurs="0" maxOccurs
                ="unbounded" />
62          <xs:element name="place" type="Place" minOccurs="0" maxOccurs
                ="unbounded" />
63          <xs:element name="resource" type="Resource" minOccurs="0"
                maxOccurs="unbounded" />
64          <xs:element name="transition" type="Transition" minOccurs="0"
                maxOccurs="unbounded" />
65          <xs:element name="note" type="Note" minOccurs="0" maxOccurs="
                unbounded" />
66          <xs:element name="inhibitor-arc" type="Arc" minOccurs="0"
                maxOccurs="unbounded" />
67          <xs:element name="arc" type="Arc" minOccurs="0" maxOccurs="
                unbounded" />
68          <xs:element name="note-connector" type="Arc" minOccurs="0"
                maxOccurs="unbounded" />
69        </xs:sequence>
70  </xs:complexType>
71
72  <xs:complexType name="empty">
```

```
73        <xs:annotation>
74          <xs:documentation xml:lang="EN">An element of this type is
                forced to be empty</xs:documentation>
75        </xs:annotation>
76    </xs:complexType>
77
78    <xs:complexType name="Resource">
79          <xs:sequence>
80        <xs:element name="features" type="empty" minOccurs="1" maxOccurs
              ="1" />
81          <xs:element name="properties" type="ListResourceProperty"
                minOccurs="1" maxOccurs="1" />
82          </xs:sequence>
83          <xs:attribute name="uuid" />
84    </xs:complexType>
85
86    <xs:complexType name="ListResourceProperty">
87          <xs:sequence>
88        <xs:element name="property" type="ResourceProperty" minOccurs="1"
              maxOccurs="1" />
89          </xs:sequence>
90    </xs:complexType>
91
92    <xs:complexType name="ResourceProperty">
93          <xs:complexContent>
94            <xs:extension base="empty">
95                  <xs:attribute name="id" use="optional">
96                    <xs:simpleType>
97                          <xs:restriction base="xs:token">
98                            <xs:enumeration value="0.default.name" />
99                          </xs:restriction>
100                   </xs:simpleType>
101                 </xs:attribute>
102                 <xs:attribute name="name" type="xs:string" use="optional
                        " />
103           </xs:extension>
104         </xs:complexContent>
105   </xs:complexType>
106
107   <xs:complexType name="NodeType">
108         <xs:attribute name="uuid" type="xs:string" use="required" />
109         <xs:attribute name="y" type="xs:int" use="required" />
110         <xs:attribute name="x" type="xs:int" use="required" />
111   </xs:complexType>
112
113   <xs:complexType name="Joint">
114         <xs:complexContent>
115           <xs:extension base="NodeType">
116             <xs:sequence>
117                   <xs:element name="features" type="empty" minOccurs="1"
                          maxOccurs="1" />
118                   <xs:element name="properties" type="empty" minOccurs
                        ="1" maxOccurs="1" />
```

```
119                 </xs:sequence>
120               </xs:extension>
121             </xs:complexContent>
122   </xs:complexType>
123
124   <xs:complexType name="Note">
125     <xs:complexContent>
126           <xs:extension base="NodeType">
127                 <xs:sequence>
128                   <xs:element name="features" type="empty" minOccurs="1"
                          ="1" />
129                   <xs:element name="properties" type="empty" minOccurs
                         ="1" maxOccurs="1" />
130                 </xs:sequence>
131                 <xs:attribute name="width" type="xs:int" use="required"
                        />
132                 <xs:attribute name="height" type="xs:int" use="required"
                         />
133           </xs:extension>
134         </xs:complexContent>
135   </xs:complexType>
136
137   <xs:complexType name="Place">
138         <xs:complexContent>
139           <xs:extension base="NodeType">
140             <xs:sequence>
141                   <xs:element name="features" type="empty" minOccurs="1"
                          maxOccurs="1" />
142                   <xs:element name="properties" type="ListPlaceProperty"
                          minOccurs="1" maxOccurs="1" />
143                 </xs:sequence>
144           </xs:extension>
145         </xs:complexContent>
146   </xs:complexType>
147
148   <xs:complexType name="ListPlaceProperty">
149         <xs:annotation>
150           <xs:documentation xml:lang="EN">Place element have to contains
                name and marking properties</xs:documentation>
151         </xs:annotation>
152         <xs:sequence>
153           <xs:element name="property" type="PlaceProperty" minOccurs="2"
                maxOccurs="2" />
154         </xs:sequence>
155   </xs:complexType>
156
157   <xs:complexType name="PlaceProperty">
158         <xs:complexContent>
159           <xs:extension base="empty">
160                 <xs:attribute name="id" use="optional">
161                   <xs:simpleType>
162                         <xs:restriction base="xs:token">
163                           <xs:enumeration value="0.default.name" />
```

```
164                         <xs:enumeration value="default.marking" />
165                       </xs:restriction>
166                   </xs:simpleType>
167                 </xs:attribute>
168                 <xs:attribute name="name" type="xs:string" ="optional"
                        />
169                 <xs:attribute name="marking" type="xs:unsignedShort" use
                        ="optional" />
170                 <xs:attribute name="satellite-x" type="xs:int" use="
                        optional" />
171                 <xs:attribute name="satellite-y" type="xs:int" use="
                        optional" />
172           </xs:extension>
173         </xs:complexContent>
174   </xs:complexType>
175
176   <xs:complexType name="Transition">
177         <xs:complexContent>
178           <xs:extension base="NodeType">
179                 <xs:sequence>
180                   <xs:element name="features" type="
                          ListTransitionFeatures" maxOccurs="1" minOccurs
                          ="1" />
181                   <xs:element name="properties" type="
                          ListTransitionProperty" maxOccurs="1" minOccurs
                          ="1" />
182         </xs:sequence>
183                 <xs:attribute name="rotation-angle" type="xs:float" use
                        ="required" />
184           </xs:extension>
185         </xs:complexContent>
186   </xs:complexType>
187
188   <xs:complexType name="ListTransitionFeatures">
189         <xs:sequence>
190           <xs:element name="feature" type="TransitionFeature" minOccurs
                ="0" maxOccurs="unbounded" />
191         </xs:sequence>
192   </xs:complexType>
193
194   <xs:complexType name="TransitionFeature">
195         <xs:complexContent>
196           <xs:extension base="empty">
197                 <xs:attribute name="id" use="required">
198                   <xs:simpleType>
199                         <xs:restriction base="xs:token">
200                           <xs:enumeration value="transition.stochastic"
                                  />
201                           <xs:enumeration value="transition.timed" />
202                           <xs:enumeration value="transition.preemptive"
                                  />
203                         </xs:restriction>
204                   </xs:simpleType>
```

```
205                 </xs:attribute>
206               </xs:extension>
207             </xs:complexContent>
208       </xs:complexType>
209
210       <xs:complexType name="ListTransitionProperty">
211             <xs:annotation>
212               <xs:documentation xml:lang="EN">Every transition element have
                      to contains name, enabling function, marking update and
                      reset transition properties. Stochastic, timed and
                      preemptive transition element have to contains (
                      respectively) stochastic, timed and preemptive properties
                      </xs:documentation>
213             </xs:annotation>
214             <xs:sequence>
215               <xs:element name="property" type="TransitionProperty"
                      minOccurs="4" maxOccurs="unbounded" />
216             </xs:sequence>
217       </xs:complexType>
218
219       <xs:complexType name="TransitionProperty">
220             <xs:annotation>
221               <xs:documentation xml:lang="EN">Every transition element have
                      to contains name, enabling function, marking update and
                      reset transition properties. Stochastic, timed and
                      preemptive transition element have to contains (
                      respectively) stochastic, timed and preemptive properties
                      </xs:documentation>
222             </xs:annotation>
223             <xs:complexContent>
224               <xs:extension base="empty">
225                     <xs:attribute name="id" use="required">
226                       <xs:simpleType>
227                             <xs:restriction base="xs:string">
228                               <xs:enumeration value="0.default.name" />
229                               <xs:enumeration value="10.default.
                                      enablingFunction" />
230                               <xs:enumeration value="11.default.
                                      markingUpdate" />
231                               <xs:enumeration value="12.default.
                                      resetTransitions" />
232                               <xs:enumeration value="transition.stochastic"
                                      />
233                               <xs:enumeration value="transition.timed" />
234                               <xs:enumeration value="transition.preemptive"
                                      />
235                             </xs:restriction>
236                       </xs:simpleType>
237                     </xs:attribute>
238                     <xs:attribute name="name" type="xs:string" use="optional
                          ">
239                       <xs:annotation>
```

```
240                         <xs:documentation xml:lang="EN">For name
                                property</xs:documentation>
241                   </xs:annotation>
242                 </xs:attribute>
243                 <xs:attribute name="satellite-x" type="xs:int" use="
                      optional">
244                   <xs:annotation>
245                         <xs:documentation xml:lang="EN">For name,
                                stochastic, timed and preemptive properties
                                </xs:documentation>
246                   </xs:annotation>
247                 </xs:attribute>
248                 <xs:attribute name="satellite-y" type="xs:int" use="
                      optional">
249                   <xs:annotation>
250                         <xs:documentation xml:lang="EN">For name,
                                stochastic, timed and preemptive properties
                                </xs:documentation>
251                   </xs:annotation>
252                 </xs:attribute>
253                 <xs:attribute name="enabling-function" type="xs:string"
                      use="optional">
254                   <xs:annotation>
255                         <xs:documentation xml:lang="EN">For enabling
                                function property</xs:documentation>
256                   </xs:annotation>
257                 </xs:attribute>
258                 <xs:attribute name="marking-update" type="xs:string" use
                      ="optional">
259                   <xs:annotation>
260                         <xs:documentation xml:lang="EN">For marking
                                update property</xs:documentation>
261                   </xs:annotation>
262                 </xs:attribute>
263                 <xs:attribute name="reset-transitions" type="xs:string"
                      use="optional">
264                   <xs:annotation>
265                         <xs:documentation xml:lang="EN">For reset
                                transition property</xs:documentation>
266                   </xs:annotation>
267                 </xs:attribute>
268                 <xs:attribute name="property-data-type" use="optional">
269                   <xs:annotation>
270                     <xs:documentation xml:lang="EN">For stochastic
                            property</xs:documentation>
271                   </xs:annotation>
272                   <xs:simpleType>
273                         <xs:restriction base="xs:string">
274                           <xs:enumeration value="0.type.immediate" />
275                           <xs:enumeration value="1.type.uniform" />
276                           <xs:enumeration value="2.type.deterministic"
                                />
277                           <xs:enumeration value="3.type.exponential" />
```

```
278                             <xs:enumeration value="4.type.erlang" />
279                             <xs:enumeration value="5.type.expolynomial" />
280                           </xs:restriction>
281                       </xs:simpleType>
282                     </xs:attribute>
283                     <xs:attribute name="priority" type="xs:int" use="
                            optional">
284                   <xs:annotation>
285                           <xs:documentation xml:lang="EN">For stochastic (
                                immediate and deterministic) property </xs:
                                documentation>
286                       </xs:annotation>
287                     </xs:attribute>
288                     <xs:attribute name="weight" type="xs:string" use="
                            optional">
289                       <xs:annotation>
290                           <xs:documentation xml:lang="EN">For stochastic (
                                immediate and deterministic) property</xs:
                                documentation>
291                       </xs:annotation>
292                     </xs:attribute>
293                     <xs:attribute name="eft" type="xs:decimal" use="optional
                            ">
294                       <xs:annotation>
295                           <xs:documentation xml:lang="EN">For stochastic(
                                uniform), timed and preemptive property</xs:
                                documentation>
296                       </xs:annotation>
297                     </xs:attribute>
298                     <xs:attribute name="lft" type="xs:string" use="optional
                            ">
299                       <xs:annotation>
300                           <xs:documentation xml:lang="EN">For stochastic (
                                uniform), timed and preemptive property</xs:
                                documentation>
301                       </xs:annotation>
302                     </xs:attribute>
303                     <xs:attribute name="value" type="xs:decimal" use="
                            optional">
304                       <xs:annotation>
305                           <xs:documentation xml:lang="EN">For stochastic (
                                deterministic) property</xs:documentation>
306                       </xs:annotation>
307                     </xs:attribute>
308                     <xs:attribute name="lambda" type="xs:decimal" use="
                            optional">
309                       <xs:annotation>
310                           <xs:documentation xml:lang="EN">For stochastic (
                                exponential and Erlang) property</xs:
                                documentation>
311                       </xs:annotation>
312                     </xs:attribute>
313                     <xs:attribute name="k" type="xs:int" use="optional">
```

```
314                   <xs:annotation>
315                         <xs:documentation xml:lang="EN">For stochastic (
                              Erlang) property</xs:documentation>
316                   </xs:annotation>
317                 </xs:attribute>
318                 <xs:attribute name="efts" type="xs:string" use="optional
                      ">
319                   <xs:annotation>
320                         <xs:documentation xml:lang="EN">For stochastic (
                              expolynomial) property</xs:documentation>
321                   </xs:annotation>
322                 </xs:attribute>
323                 <xs:attribute name="lfts" type="xs:string" use="optional
                      ">
324                   <xs:annotation>
325                         <xs:documentation xml:lang="EN">For stochastic (
                              expolynomial) property</xs:documentation>
326                   </xs:annotation>
327                 </xs:attribute>
328                 <xs:attribute name="expressions" type="xs:string" use="
                      optional">
329                   <xs:annotation>
330                         <xs:documentation xml:lang="EN">For stochastic (
                              expolynomial) property</xs:documentation>
331                   </xs:annotation>
332                 </xs:attribute>
333                 <xs:attribute name="normalizationFactor" type="xs:
                      decimal" use="optional">
334                   <xs:annotation>
335                         <xs:documentation xml:lang="EN">For stochastic (
                              expolynomial) property</xs:documentation>
336                   </xs:annotation>
337                 </xs:attribute>
338                 <xs:attribute name="resources" type="xs:string" use="
                      optional">
339                   <xs:annotation>
340                         <xs:documentation xml:lang="EN">For preemptive
                              property</xs:documentation>
341                   </xs:annotation>
342                 </xs:attribute>
343                 <xs:attribute name="priorities" type="xs:string" use="
                      optional">
344                   <xs:annotation>
345                         <xs:documentation xml:lang="EN">For preemptive
                              property</xs:documentation>
346                   </xs:annotation>
347                 </xs:attribute>
348           </xs:extension>
349         </xs:complexContent>
350   </xs:complexType>
351
352   <xs:complexType name="Arc">
353         <xs:sequence>
```

```
354         <xs:element name="features" type="empty" minOccurs="1" maxOccurs
                ="1" />
355             <xs:element name="properties" type="empty" minOccurs="1"
                    maxOccurs="1" />
356           </xs:sequence>
357           <xs:attribute name="from" type="xs:string" use="required" />
358           <xs:attribute name="to" type="xs:string" use="required" />
359           <xs:attribute name="uuid" type="xs:string" use="required" />
360     </xs:complexType>
361 </xs:schema>
```

Appendice B

Oris e Sirio a confronto

Oris e Sirio modellano le PN su domini diversi. Mentre il dominio di Oris è stato già descritto nel capitolo 3 dallo schema XPN, quello di Sirio è raffigurato in figura B.1.

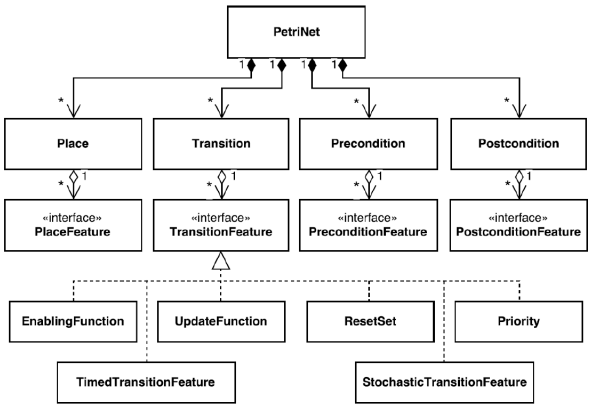

Figura B.1: Modello di dominio di Sirio

Questo concorre a portare i due domini a presentare alcune differenze. In particolare alcune funzioni hanno rappresentazioni diverse nei due domini o, talvolta, addirittura non esiste una controparte, perché non necessaria o perché emulabile da altre funzionalità. Di seguito viene offerta una panoramica sulle differenze più rilevanti:

- in Sirio le risorse, i posti e le transizioni hanno un attributo relativo al nome. In Oris invece questi elementi hanno solo l'attributo dell'identificativo univoco; tutto il resto, nome compreso, viene trattato come una proprietà. Questo vuol dire che tutti questi elementi quando salvati su file `.xpn` devono contenere un tag:

```
<property id="0.default.name" name="..." />
```

- in Sirio la marcatura di una PN è un concetto separato dalla PN stessa, mentre in Oris il numero di token di un posto è strettamente legato ad esso. Questo si traduce, nel documento XPN, in un tag del posto:

```
<property id="default.marking" marking="..." />
```

- in Sirio le precondizioni, postcondizioni e gli archi inibitori hanno una molteplicità a differenza di Oris. Queste però possono essere emulate da *enabling functions* e *update functions*:

  - precondizione da p a t con molteplicità m: la transizione sarà *enabled* solo se il posto contiene almeno m token. Questo si traduce quindi nella condizione $p \geq m$, che dovrà essere messa in `&&` con quelle già presenti nella *enabling function*;
  - postcondizione da t a p con molteplicità m: in seguito al *firing* della transizione, nel posto saranno aggiunti m token. Si traduce nell'espressione $p \mathrel{+}= m$, da concatenarsi alla *update function* in testa ad essa perché dovrà essere eseguita prima dell'ulteriore aggiornamento della marcatura;
  - arco inibitore da p a t con molteplicità m: la transizione sarà *enabled* solo se il posto contiene al più $m - 1$ token. Si traduce

nella condizione $p < m$, che dovrà essere messa in `&&` con quelle già presenti nella *enabling function*;

- in Oris *enabling functions*, *update functions* e *reset sets* sono sempre presenti, eventualmente vuoti, a differenza di Sirio che sono rappresentati da *features* da aggiungere dinamicamente alle transizioni. In un documento XPN, quindi, una transizione ha sempre i tag:

```
<property id="10.default.enablingFunction" enabling-function=""/>

   <property id="11.default.markingUpdate" marking-update=""/>

<property id="12.default.resetTransitions" reset-transitions=""/>
```

B.1 Transizioni stocastiche

Infine, aspetto molto importante, Sirio non associa alle transizioni stocastiche una distribuzione di probabilità come Oris, ma una densità di probabilità del *time to fire*. In Oris, nel documento XPN, il tag `<property id="transition.stochastic" ... />`, a seconda della distribuzione si completa uno dei seguenti gruppi di attributi:

```
property-data-type="0.type.immediate" priority="..."
    weight="..."
property-data-type="1.type.uniform" eft="..." lft"..."
property-data-type="2.type.deterministic" value="..."
    priority="..." weight="..."
property-data-type="3.type.exponential" lambda="..."
property-data-type="4.type.erlang" lambda="..." k="..."
property-data-type="5.type.expolynomial" efts="..." lfts="..."
```

```
expressions="..." normalizationFactor="..."
```

Inoltre in Sirio è possibile associare ad ogni transizione stocastica:

- *weight*: un'espressione della marcatura che determina la probabilità di *firing* in caso di *time to fire* identici, che però possono presentarsi solo in transizioni immediate e deterministiche, infatti Oris lo associa solo a questi tipi;
- *clock rate* non unitario: espressione della marcatura che determina il rate di decremento dei timer. In Oris non è presente questa impostazione.

Anche la *feature Priority* ha senso solamente in transizioni immediate o deterministiche perché serve a determinare la precedenza di *fire* in caso di *time to fire* identici. Un settaggio di questi parametri non conforme al modello di Oris non può essere esportato in XPN.

In fase di esportazione, dunque, si deve riuscire ad identificare il tipo di distribuzione ed estrapolarne i parametri a partire dalla *PDF*. Inoltre devono essere controllati gli altri parametri per verificare che l'esportazione in XPN possa essere effettuata, ovvero non abbia parametri non supportati da Oris.

In Sirio le densità di probabilità possono essere di quattro tipi:

- *EXP*: rappresenta un'esponenziale;
- *Erlang*: rappresenta una Erlang;
- *GEN*: rappresenta una PDF multimensionale non definita a tratti e
- *PartitionedGEN*: rappresenta ancora una PDF multidimensionale, ma anche eventualmente definita a tratti.

I primi due casi permettono di individuare immediatamente il tipo di distribuzione, come l'ultimo che non può che rappresentare una *expolinomiale*. Il caso più complesso è *GEN*, che può rappresentare tutti gli altri tipi (*immediata*, *deterministica*, *uniforme* e anche expolinomiale). Di seguito è riportato l'algoritmo per l'identificazione del tipo di distribuzione e l'estrapolazione dei parametri.

Algoritmo 6 StochasticDistribution

Input: PDF

```
1: function StochasticDistribution(PDF)
2:   if PDF instance of EXP
3:     Exponential distribution with λ = EXP.λ
4:   else if PDF instance of Erlang then
5:     Erlang distribution with λ = Erlang.λ, k = Erlang.k
6:   else
7:     if PDF is not multidimensional
8:       density ← expolynomial form of PDF
9:       eft ← lowerbound of PDF's domain
10:      lft ← upperbound of PDF's domain
11:      if density has only 1 term and it is constant
12:        if eft == lft
13:          if density.constantTerm == 1
14:            if eft = lft == 0
15:              Immediate distribution
16:            else
17:              Deterministic distribution with value = eft = lft
18:          else
19:            Defective density
20:        else
21:          if density.constantTerm == 1/(lft−eft)
22:            Uniform distribution in [eft, lft]
23:          else
24:            Defective density
25:      else
26:        Expolynomial distribution
27:    else
```

28: Expolynomial distribution

Nel caso di distribuzione expolinomiale i parametri sono dati dagli eft e lft dei domini, le espressioni dalle espressioni expolinomiali delle distribuzioni e il fattore di normalizzazione dall'inverso dell'integrale sul dominio.

Bibliografia

Articoli

[1] G. Bucci, A. Fedeli, L. Sassoli, and E. Vicario, "Modeling Flexible Real Time System with Preemptive Time Petri Nets," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS03)*, Luglio 2003.

[2] G. Bucci, A. Fedeli, and E. Vicario, "Timed State Space Analysis of Real-Time Preemptive System," *IEEE Transaction on Software Engineering*, vol. 30, pp. 97–111, Febbraio 2004.

[3] L. Carnevali, L. Sassoli, and E. Vicario, "Casting Preemptive Time Petri Nets in the Development Life Cycle of Real-Time Software," in *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS07)*, Luglio 2007.

[4] L. Carnevali, L. Grassi, and E. Vicario, "A tailored V-Model exploiting the theory of preemptive Time Petri Nets," in *Proceedings of the Ada Europe: 13th int.conference on Reliable SW Technologies*, Giugno 2008.

[5] L. Carnevali, L. Ridi, and E. Vicario, "Putting Preemptive Time Petri Nets to Work in a V-Model SW Life Cycle," *IEEE Transaction on Software Engineering*, vol. 37, no. 6, pp. 826–844, 2011.

[6] L. Carnevali, L. Ridi, and E. Vicario, "Sirio: A Framework for Simulation and Symbolic State Space Analysis of non-Markovian Models," in *Proceedings of the 8th International Conference on the Quantitative Evaluation of Systems (QEST11)*, Settembre 2011.

[7] M. Biagi, L. Carnevali, E. Vicario, and M. Paolieri, "An introduction to the ORIS tool," in *Proceedings of the 11th EAI International Conference on Performance Evaluation Methodologies and Tools (ValueTools17)*, 2017.

[8] M. Paolieri, M. Biagi, L. Carnevali, and E. Vicario, "The ORIS Tool: Quantitative Evaluation of Non-Markovian System," *IEEE Transaction on Software Engineering*, 2019. Early access, DOI 10.1109/TSE.2019.2917202.

[9] L. Sha, R. Rajkumar, and S. Sathayeo, "Generalized Rate Monotonic Scheduling Theory: A Framework for Developing Real Time Systems," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 68–82, 1994.

[10] J. Xu and D. Parnas, "On Satisfying Timing Constraints in Hard Real Time Systems," *IEEE Transaction on Software Engineering*, vol. 19, no. 1, pp. 70–84, 1993.

Testi

[11] E. Gamma, J. Vlissides, R. Helm, and R. Johnson. *Design Patterns: Element of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[12] J. Wang. *Timed Petri Nets: Theory and Application.* Springer, 1998.

Risorse online

[13] E. Ort and B. Mehta. Java Architecture for XML Binding (JAXB), Marzo 2003. `https://www.oracle.com/technical-resources/articles/javase/jaxb.html`.

[14] JUnit 5. `https://junit.org/junit5`.

[15] ORIS tool. `http://www.oris-tool.org`.

[16] Sirio library. `https://github.com/oris-tool/sirio`.

[17] XML Schema. `https://www.w3.org/2001/XMLSchema`.

Tesi di Laurea

[18] L. Macchiarini. Design and development of a software component for the analysis of preemptive timed models. Tesi di Laurea Triennale, Università degli Studi di Firenze, 2020.

Elenco delle figure