Títol: Process Mining: descobrint models formals a partir de logs d'un sistema

Volum: 1
Alumne: Jorge Muñoz Gama

Director/Ponent: Josep Carmona Vargas
Departament: Llenguatges i Sistemes Informàtics
Data: Juny 2009

Dades del projecte

Títol del projecte: Process Mining: descobrint models formals a partir de logs d'un sistema

Nom de l'estudiant: Jorge Muñoz Gama

Titulació: Enginyeria en Informàtica

Crèdits: 37.5

Director/Ponent: Josep Carmona Vargas

Departament: Llenguatges i Sistemes Informàtics

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Jordi Cortadella Fortuny

Vocal: Tomàs Aluja Banet

Secretari: Josep Carmona Vargas

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

A la meva família

Índex

Índex de figures

Índex de taules

Índex de Codis

Índex d'Algoritmes

1 Introducció

1.1 Context

La tendència des de fa temps és que tota la informació relacionada amb els diferents processos que tenen lloc en un determinat sistema quedi enregistrada en els anomenats *logs* o dietaris del sistema. Quan hom pensa en sistemes que facin això, els primers sistemes que venen al cap són tots aquells dissenyats per a la gestió i organització d'empreses. Però la veritat és que la quantitat de sistemes que enregistren informació sobre els seus processos és enorme i molt diversa, anant des de fotocopiadores d'última generació fins a hospitals on queda constància dels passos seguits en el diagnòstic i tractament d'un pacient.

Malgrat la riquesa de tota aquesta informació, en la majoria dels casos només s'usa per a resoldre qüestions relativament senzilles i sempre sota la premissa que es coneix amb exactitud el funcionament del procés que s'està analitzant. Però què passaria si la idea que es té del procés no estigués clara del tot o fos errònia? Seria interessant poder usar tota aquesta informació dels logs per a obtenir una idea exacta de com és *realment* aquest procés, i així poder comprovar si es correspont amb la teoria, o bé analitzar-lo amb la finalitat de millorar-lo i fer-lo més eficient. Aquesta és precisament la funció del *Process Mining*.

1.2 Què és el *Process Mining*?

Process Mining [4] [12] és el mecanisme que ens permet obtenir models formals a partir de la informació dels processos enregistrada en els logs d'un sistema. Però per a entendre això, primer cal entendre que és un *procés*.

Un *procés* es defineix com el conjunt de passos o accions executats en un determinat ordre per aconseguir una fita. Per poder entendre millor que és un procés i com funciona el Process Mining, farem servir com exemple un procés senzill que realitzen milers de persones en el seu dia a dia: anar al gimnàs.

> Al arribar al gimnàs el primer que es fa són els *estiraments*. Un cop estirats, el següent pas és fer una mica de *bicicleta*. Després, si és diumenge, es fa classe de *ioga*. En canvi, si és un altre dia de la setmana, es fa *piscina*. I l'ultim pas, tan si s'ha fet piscina o ioga, és *dutxar-se* abans de marxar.

Aquest exemple mostra el procés *anar al gimnàs*. Les accions, passos o tasques necessàries per a realitzar aquest procés són: *estirar*, *bicicleta*, *ioga*, *piscina* i *dutxa*. Si es realitzen aquestes accions en l'ordre determinat s'està realitzant el procés.

Cada cop que es produeix una d'aquestes accions, el sistema ho enregistra. Cadascuna d'aquestes anotacions, s'anomena un *event*. Cada event correspont a una acció del procés, i pot contenir altra informació com: qui ha executat aquesta acció o quan. Tots aquests events queden emmagatzemats en els anomenats *logs d'events*. Seguint amb el nostre exemple, en la Taula 1.1 es poden veure 3 seqüències (o *traces*) d'events enregistrades pel sistema.

Dimarts	Dijous	Diumenge
estirar	estirar	estirar
bicicleta	bicicleta	bicicleta
piscina	piscina	ioga
dutxa	dutxa	dutxa

Taula 1.1: Traces de l'exemple del gimnàs

Com es pot veure en la taula, els events enregistrats el dimarts i el dijous coincideixen. En canvi, el diumenge l'acció "ioga" es realitzada en el lloc de "piscina".

L'objectiu del Process Mining és el camí oposat al que acabem de realitzar: a partir dels logs d'events d'un sistema, ser capaços d'obtenir una descripció del procés que es defineix. En el cas de les traces de la Taula 1.1, una possible descripció del procés podria ser la mostrada en la Figura 1.1.

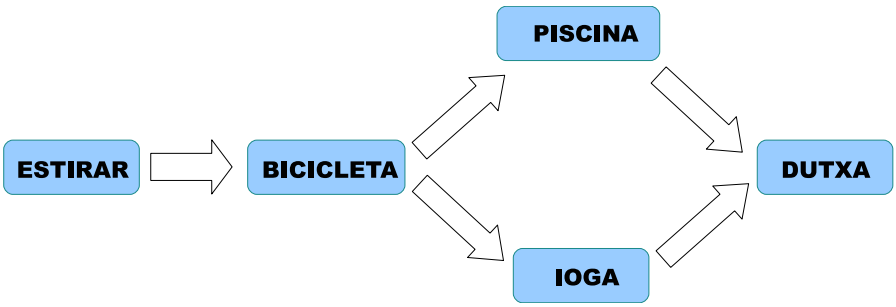


Figura 1.1: Procés "anar al gimnàs"

Tal com es pot veure en la figura, el procés "anar al gimnàs" comença sempre amb l'acció "estirar". Aquesta acció es seguida sempre per l'acció "bicicleta". Després, si és diumenge, l'acció executada és "ioga". En canvi, si és un altre dia de la setmana, l'acció que es realitza és "piscina". Al acabar, sempre es produeix l'acció "dutxa".

Aquest exemple ens ha permés mostrar en que consisteix el Process Mining. Però és fàcil veure que els processos que hom es troba a la vida real son molt més complexos. La informació sobre els processos executats es enregistrada en infinitat de llocs. Per exemple, la majoria de sistemes per a la organització i gestió d'empreses, com per exemple els ERP[1], ja incorporen l'enregistrament d'aquesta informació. Però també ho fan altres sistemes pensats per a altres coses, com poden ser els "Web Servers" o els sistemes de control de versions. Últimament estan sortint al mercat aparells mèdics o robots per a la producció, sovint de nova generació i especialment costosos, que també incorporen aquesta informació. D'aquesta manera és pot monitoritzar i realitzar un manteniment preventiu, estalviant així molts diners.

[1]Enterprise Resource Planning

Un procés és un element molt complexe, i que contempla molts aspectes. Gràcies al Process Mining es pot obtenir informació sobre molts dels aspectes (també anomenats *perspectives*) d'un procés determinat a partir dels logs d'events. Alguns dels aspectes que ens poden interessar són:

Control de flux En un procés ens pot interessar saber l'ordre que segueixen els diferents events, com es sincronitzen aquests events, o quins son necessaris per que s'executin d'altres. Un exemple d'això és el vist anteriorment amb el procés "anar al gimnàs".

Xarxa social Donat un procés pot ser interessant veure com interactuen les diferents persones que participen en la seva realització, com treballen juntes, com comparteixen informació unes amb les altres o si dues persones estan realitzant la mateixa tasca. A més, tota aquesta informació pot ser usada de cara a la creació de rols o grups de persones, és a dir, conjunts de persones amb unes característiques comunes i uns privilegis determinats.

Característiques del funcionament Analitzar l'execució d'un procés ens pot permetre obtenir informació molt valuosa sobre el seu funcionament. Aquesta informació pot ser usada per comprovar que el sistema funcioni de manera òptima i no tingui cap *coll d'ampolla*. Un exemple d'això podria ser el fet de que quan treballen juntes dues persones determinades, el temps d'execució del procés és dispara. El Process Mining permetria detectar aquestes situacions i corregir-les, estalviant així temps i diners.

Auditoria i seguretat L'ús del Process Mining com a mecanisme d'auditoria i seguretat permetria comprovar que a l'executar un procés no és violi cap restricció de seguretat, i prenent les accions pertinents donat el cas. Imaginem un servei web que permet comprar productes a usuaris registrats. Després de triar els productes, l'usuari introdueix el password i la comanda es enviada a casa seva. Què passaria si el sistema estigués mal dissenyat i permetés fer comandes sense introduir el password? És aquí on entraria el Process Mining, que seria capaç d'analitzar el procés a partir dels logs del sistema i detectar el problema de seguretat.

Regles de decisió Davant d'un procés, com el d'"anar al gimnàs", Process Mining et permet esbrinar que només es va a "ioga" si es diumenge. És a dir, que la raó per decidir anar a "ioga" o a la "piscina" és si és diumenge o no.

Fins ara hem vist què és el Process Mining i què ens permet fer. Però a més, segons el propòsit pel qual s'usa, el Process Mining es pot classificar en 3 tipus [18]:

Descobriment S'usa en casos on no existeix un model previ del procés. El Process Mining es usat per a construir un model a partir dels logs d'events del procés.

Confirmació En aquest cas sí que existeix un model previ del procés, i el que es pretén és usar Process Mining per a confirmar si la realitat (representada en els logs) correspont amb aquest model, i viceversa.

Extensió Donat un model previ, l'objectiu del Process Mining no és tan comprovar aquest model, com estendre'l, enriquint-lo amb nous aspectes que puguin sorgir.

Un bon exemple de les aplicacions del Process Mining en el món real es pot veure en [14], on els autors exposen l'ús del Process Mining el el procés d'oncologia ginecològica d'un hospital holandès, centrant-se en els aspectes del control del flux, la xarxa social i les característiques del funcionament.

1.3 Objectius del projecte

L'objectiu d'aquest projecte és el desenvolupament d'una eina que faciliti certs aspectes del Process Mining. En concret, es pretén crear una aplicació que complementi el funcionament de l'eina *Genet*.

Genet [9] és una eina pensada per al Process Mining i desenvolupada per en Josep Carmona de la Universitat Politècnica de Catalunya. En concret, aquesta eina permet transformar Sistemes de Transicions en Xarxes de Petri[2], cosa que pot ser interessant a l'hora d'analitzar processos. Aquesta aplicació està pensada per ser executada en mode consola, i no compta amb cap tipus de part gràfica.

L'objectiu del projecte és crear una aplicació anomenada *GenetGUI*, basant-se en *Genet*, que estengui el funcionament d'aquesta. En concret es pretén que inclogui:

- Visualització gràfica dels elements: l'aplicació ha de permetre visualitzar els diferents models amb els que es treballa. Aquesta visualització ha de poder-se manipular, per així facilitat la seva comprensió.
- Interfície gràfica: el programa ha de comptar amb una capa de presentació gràfica que permeti manipular i interactuar amb les opcions que ofereix Genet. A més, ha de facilitar la realització de les tasques més carregoses.
- Ampliació dels models d'entrada: s'han d'ampliar el nombre de models diferents acceptats per a poder ser processats.
- Estendre el funcionament: completar el procés lògic de creació i anàlisis de Xarxes de Petri, afegint la possibilitat de composar diverses d'aquestes xarxes.

1.4 Estructura de la memòria

L'estructura de la memòria és la següent:

Capítol 2: ***Teoria Bàsica*** Els conceptes i definicions necessàries per a entendre el projecte.

Capítol 3: ***Especificació*** Especificació formal de l'aplicació.

Capítol 4: ***Disseny i implementació*** Els aspectes sobre el disseny i la implementació de l'eina, així com les diverses tecnologies empleades en la seva creació.

Capítol 5: ***Algoritmes*** Descripció i explicació dels diferents algoritmes utilitzats per l'aplicació.

Capítol 6: ***Exemple d'ús*** Descripció pas per pas d'un exemple d'ús de l'aplicació.

[2]Tots aquests conceptes estan definits en el capítol 2

Capítol 7: ***Planificació i anàlisi econòmic*** On es detallen els aspectes econòmics i logístics del desenvolupament de l'aplicació.

Capítol 8: ***Conclusions*** Conclusions a las que s'ha arribat després de realitzar el projecte.

Apèndix A: ***Manual d'Usuari*** Manual en anglès sobre el funcionament de l'aplicació.

Apèndix B: ***Tutorial de Prefuse*** Tutorial en anglès sobre la creació de visualitzacions de Xarxes de Petri fent ús de la llibreria Prefuse.

2
Teoria bàsica

En aquest capítol es detallen els conceptes i definicions necessàries per a entendre el projecte.

2.1 Activitats, Casos i Events

El Process Mining, objectiu final d'aquest projecte, son una sèrie de tècniques pensades per a extraure informació sobre un procés a partir dels events succeïts i emmagatzemats en un sistema. És per això que cal definir què s'entén per *event*. Però per fer-ho primer cal definir els conceptes de *activitats* i *casos*.

Activitat Una activitat (*activity*) és un pas completament definit d'un procés.

Cas Un cas (*case*) és una instància d'un procés.

Event Un event es l'anotació de que un esdeveniment ha succeït en el sistema. En concret:

- un event fa referència a una activitat.
- un event fa refèrencia a un cas.
- els events estan ordenats segons l'ordre en que han anat succeint.

A més, l'event pot contenir altra informació com qui l'ha originat o en quin moment, entre altres.

2.2 Traces, Logs d'Events i Sistemes de Transicions

Un cop definit que es un event, cal descriure els elements per emmagatzemar aquests events.

Cal indicar que, tot i que s'accepta que els events continguin altra informació, l'unica informació que es tindrà en compte en aquest projecte és l'activitat a la que fan referència. Això fa que quan es parli de "traces" o "logs d'events", en veritat s'estigui parlant de "traces simples" i "logs d'events simples". Per un abús de notació ometrem la paraula "simple".

Si està perfectement definit el context del *cas* al que es pertany, és anàlog referir-se a un event com a una activitat i viceversa.

Donat un cas, aquest pot ser representat com una seqüència d'events. Això és el que es coneix com *traça*.

Definició 2.1 (Traça) *Sigui* c *un cas, i* E *el conjunt d'events, es defineix la traça σ que representa* c *com $\sigma = e_1 e_2 \dots e_n$ on $e_i \in E$.*

L'ordre en el que apareixen els events en la traça ve determinat per l'ordre en que aquests events han succeït. Exemples de traces es poden veure en la Taula 1.1 dins del procés "anar al gimnàs" vist en el capítol 1. Per exemple, la traça enregistrada el dijous és (estirar - bicicleta - piscina - dutxa) on estirar, bicicleta, piscina i dutxa són els events que la composen.

Un cop definit el que és una traça, toca definir què són els conjunts de Traces i els Logs d'Events. Al llarg del projecte es fa referència al "conjunt de traces" (o simplement Traces) i als "logs d'events" com conceptes diferents. Però de fet, aquesta diferència només existeix a nivell de format de codificació, i a la possibilitat de que els events dels logs d'events continguin altra informació a part de l'activitat. Però des del punt de vista conceptual, i considerant que en aquest projecte només es té en compte l'activitat a la que fa referencia un event, els dos conceptes es poden definir d'igual forma.

Les *Traces* o els *Logs d'Events* es descriuen com a un conjunt de traces.

Definició 2.2 (Traces / Logs d'Events) *Sent E el conjunt d'events, i sabent que $\mathcal{P}$ defineix el* conjunt de les parts[1]*, llavors $L \in \mathcal{P}(E^*)$ és un Log d'Events/Traces.*

Un cop definits els Logs d'Events cal definir els Sistemes de Transicions. Els Sistemes de Transitions són els models formals triats per a representar els Logs d'Events, i amb els quals treballa Genet. Precisament passar dels Logs d'Events a Sistemes de Transicions és un dels objectius d'aquest projecte, i el procés està explicat en capítols posteriors.

Definició 2.3 (Sistema de Transicions) *Un sistema de transicions és una tuple (S,E,A,s_{in}), on:*

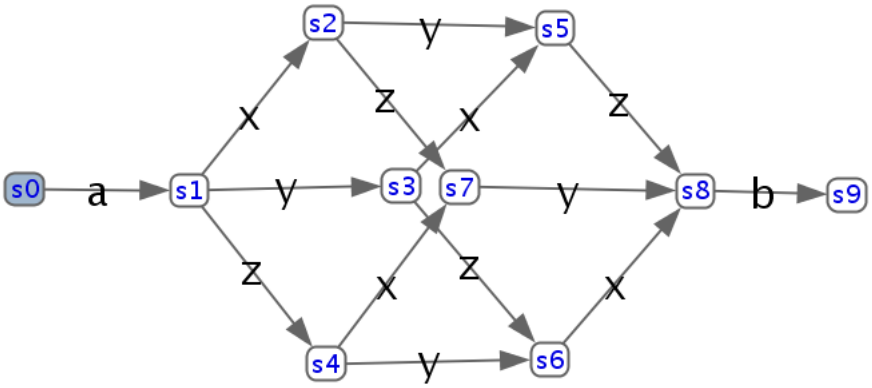
- *S és un conjunt d'estats*
- *E és un alfabet d'accions, tal que $S \cap E = \emptyset$*
- *A és un conjunt de transicions (etiquetades) tal que $A \subseteq S \times E \times S$*
- *s_{in} és l'estat inicial*

En la Figura 2.1(a) es pot veure un exemple d'un sistema de transicions. En concret correspont a un sistema amb 5 accions (a, b, x, y, z), tres de les quals son concurrents (x, y, z) i van precedides per (a) i seguides per (b). L'estat inicial del sistema és *s0*, indicat amb un color diferent. A traves de l'acció *a*, es va a un estat *s1* a partir del qual les tres accions *x*, *y* i *z* poden ser executades. Segons quina sigui la triada es va a parar a un estat o un altre, a partir del qual es podran executar alguna de les dues altres accions restants. Així s'arriba a algun dels estats *s5*, *s6*, *s7* on es pot llançar l'última acció que resta, anant així al estat *s8* on només queda accionar *b*.

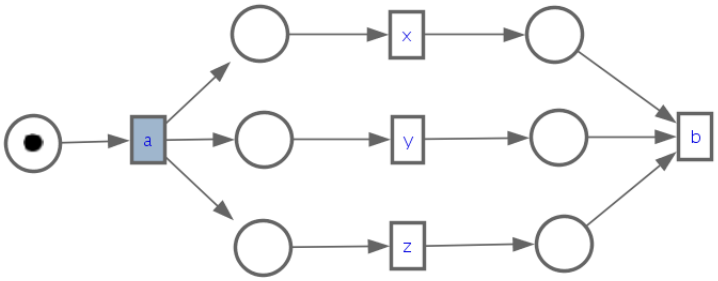
2.3 Xarxes de Petri

Tot i que els Sistemes de Transicions són models formals perfectament vàlids, pateixen d'un problema que els fa poc tractables: el problema de l'*explosió d'estats*. A mesura que el procés analitzat creix, també creix el nombre d'estats del Sistema de Transicions que el representa, i ho fa de forma exponencial. Això fa que en poc temps el nombre d'estats sigui tan gran i

[1] Conjunt de les parts (o powerset) $\mathcal{P}(B) = \{X | X \subseteq B\}$



(a) Sistema de Transicions



(b) Xarxa de Petri

Figura 2.1: Sistema de Transicions *vs* Xarxa de Petri (I)

que existeixin tantes transicions entre els estats, que aquesta representació perdi tota la seva manejabilitat. En l'exemple anterior, si el nombre d'accions concurrents hagués estat 6 en comptes de 3 com era el cas, el nombre d'estats hagués crescut exponencialment.

És per això que cal buscar un altre model formal basat en events que permeti expressar el comportament d'un procés i que sigui més manejable. En aquest cas el model triat són les anomenades *Xarxes de Petri*. El pas d'un Sistema de Transicions a una Xarxa de Petri és precisament l'objectiu de l'eina *Genet*. La Figura 2.1 mostra la diferencia entre un Sistema de Transicions i una Xarxa de Petri.

Definició 2.4 (Xarxa de Petri) *Una Xarxa de Petri marcada (o simplement Xarxa de Petri) és una tuple (P,T,f,s) on:*

- *P és un conjunt finit de* places
- *T és un conjunt finint de* transicions *tal que* $P \cap T = \emptyset$
- *f és una funció anomenada* relacions de flux $f\colon (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ *on el nombre indica el pes de la relació (o arc) i 0 indica que no hi ha cap relació.*
- *s denota el marcatge de la xarxa. És una funció* $s\colon P \rightarrow \mathbb{N}$*, on el número indica el nombre de* tokens *que hi ha en un place.*

En la Figura 2.1(b) es pot veure un exemple de representació gràfica de Xarxa de Petri. Els cercles representen els Places, mentre que els rectangles representen les Transicions. Els punts negres corresponen als tokens que hi ha en un determinat Place. I finalment, les fletxes determinen les relacions de flux que hi ha entre els places i les transitions. Si aquesta relació tingués un pes diferent a 1, aquest apareixeria indicat al costat de la fletxa.

Aquesta Xarxa de Petri correspont exactament al mateix sistema que l'usat per explicar el funcionament dels Sistemes de Transicions en la figura 2.1(a). Ràpidament salta a la vista la diferència que hi ha entre els dos models en quan a la simplicitat i manejabilitat. I seria més evident si en comptes de 3 accions concurrents (x,y,z) fossin moltes més. Mentre que en el Sistema de Transicions es dispararien els estats, en la Xarxa de Petri només caldria afegir dos Places i una Transició més.

Definició 2.5 (Nodes, Nodes d'entrada, Nodes de sortida) *Els elements de* $P \cup T$ *reben el nom de* nodes*. El node* x *és un* node d'entrada *d'un altre node* y*, si i solament si existeix un arc des de* x *a* y*, és a dir,* $(x, y) \in F$*. El node* x *és un* node de sortida *d'el node* y *si i solament si existeix un arc de* y *a* x*, és a dir* $(y, x) \in F$*. El conjunt de tots els nodes d'entrada d'un node* x *es representa com* $\bullet x$*. El conjunt de tots els nodes de sortida d'un node* x *es representa com* $x\bullet$*.*

La Figura 2.2 mostra amb colors diferents quins son els nodes d'entrada i els nodes de sortida de la transició *a*.

Una transició està activada si i solament si hi ha un token en tots els seus nodes d'entrada (en el cas d'una Transició els nodes d'entrada corresponen a Places). En cas de ser un arc amb pes, el nombre de tokens mínim necessari ve determinat per aquest pes.

Definició 2.6 (Transició activada) *Donada una Xarxa de Petri* $N = (P, T, f, s)$*, x és una transició activada* $\iff \{x \in T | \forall p \in \bullet x \Rightarrow s(p) \geq f(p, x)\}$

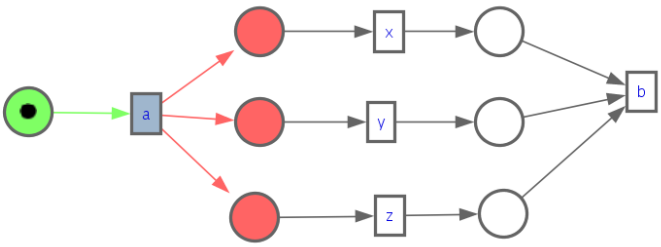


Figura 2.2: Nodes d'entrada i Nodes de sortida

En els exemples d'aquest capítol les Transicions activades estan indicades amb un color diferent. Les Transicions activades poden ser *disparades*.

Una Xarxa de Petri evoluciona en el temps, a mesura que es disparen algunes de les Transicions activades. La evolució de la Xarxa de Petri ve determinada per la *regla de disparament*.

Al disparar una transició activada, un token es eliminat de tots els nodes d'entrada de la transició, i s'afegeix un token a tots els nodes de sortida. En cas d'arcs amb pesos, el nombre de tokens eliminats i afegits ve determinat per aquest pes.

Definició 2.7 (Regla de disparament) *Donada una Xarxa de Petri* $N = (P, T, f, s)$*, i una transició activada* x *llavors* $fire(N, x) \Rightarrow N' = (P, T, f, s')$ *on* $\forall p_{in} \in \bullet x \Rightarrow s'(p_{in}) = s(p_{in}) - f(p_{in}, x) \wedge \forall p_{out} \in x\bullet \Rightarrow s'(p_{out}) = s(p_{out}) + f(x, p_{out})$

La Figura 2.3 mostra una possible evolució de la Xarxa de Petri. Al principi es disparada la única transició activada que hi ha inicialment, a. Això fa que es tregui el token del seu place d'entrada i apareguin tres tokens, un a cadascun dels places de sortida de la transició a. Arribats a aquest punt, tan x com y com z poden ser disparades (ja que tenen tokens suficients en el seus places d'entrada). En aquest exemple, les accions son disparades en el següent ordre: y, z i x. Finalment, i un cop les tres han estat disparades, només resta disparar b, que ara ja està activada.

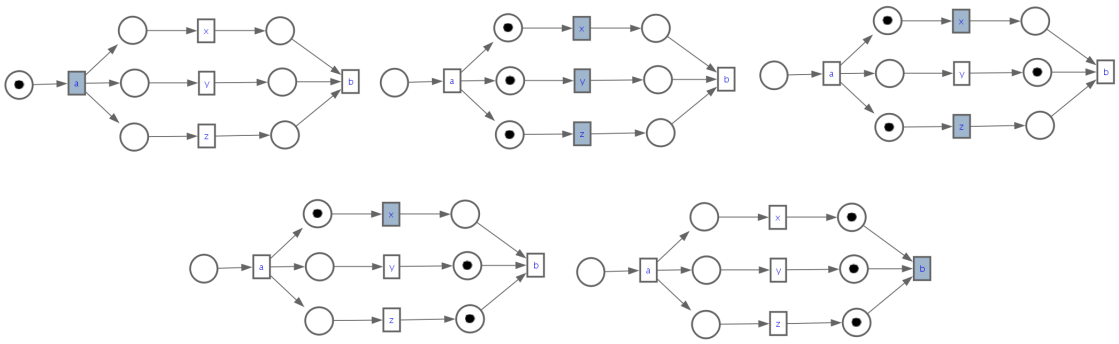


Figura 2.3: Evolució d'una Xarxa de Petri

En la Figura 2.4 encara queda més palesa la diferència entre que hi ha entre Sistemes de Transicions i Xarxes de Petri en processos més complexos.

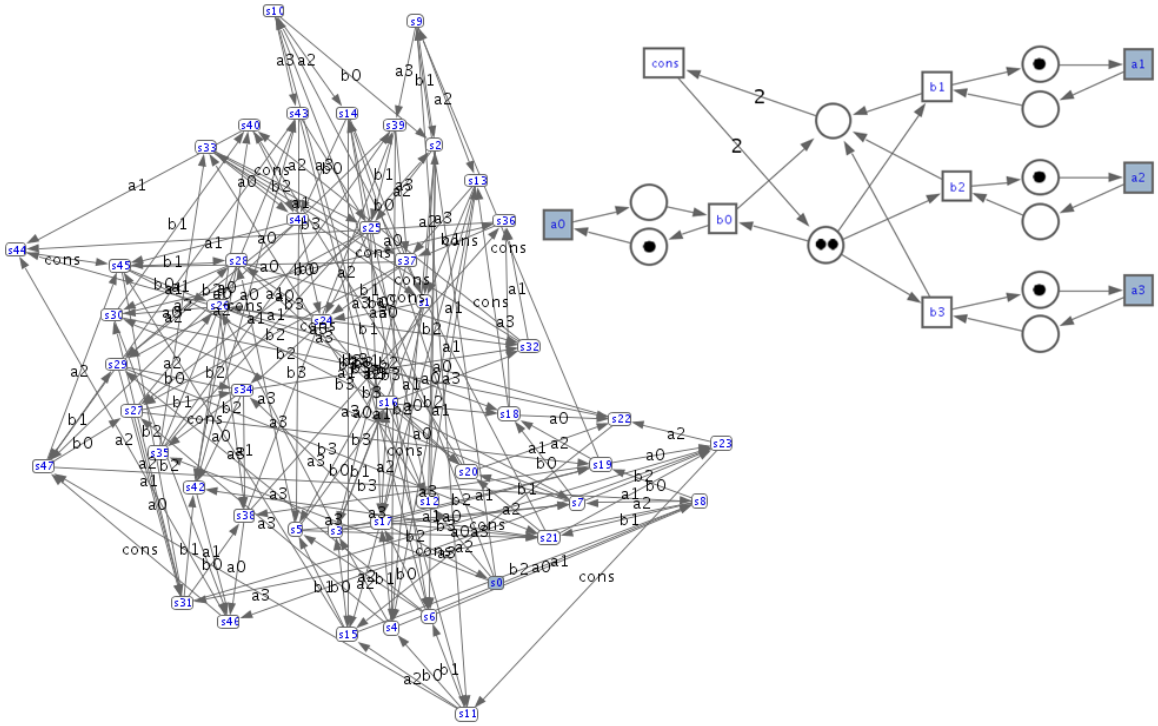


Figura 2.4: A l'esquerra un Sistema de Transicions. A la dreta la Xarxa de Petri corresponent.

3
Especificació

Aquest captíol conté l'especificació [10] del programa *GenetGUI*. La funció d'aquesta especificació és la de definir d'una manera formal què és el que farà el programa.

Aquesta especificació es divideix en quatre parts:

- Anàlisi de requeriments. (sec 3.1)
- Model de casos d'ús. (sec 3.2)
- Model conceptual de dades. (sec 3.3)
- Model de comportament del sistema. (sec 3.4)

Per dur a terme aquesta especificació (i posteriorment també el disseny) s'ha decidit usar un llenguatge àmpliament utilitzat per especificar, dissenyar i documentar sistemes software: l'UML[1].

El primer pas abans de començar amb l'especificació és definir quin serà el *cicle de vida* utilitzat a l'hora de desenvolupar l'aplicació. El cicle de vida triat determinarà com es coordinaran les diferents etapes del procés de creació del software.

Les característiques del desenvolupament d'una aplicació com *GenetGUI* en el marc d'un Projecte Final de Carrera fan que sigui necessària la creació de diverses versions provisionals del programa, que seran examinades pel director del projecte, i posteriorment refinades d'acord a les seves observacions. Això fa que ràpidament descartem la elecció d'un cicle de vida clàssic (o en cascada). I son precisament aquestes raons les que ens fan triar un cicle de vida evolutiu, en concret, el *model iteratiu incremental*.

Tal i com es pot veure en la Figura 3.1, aquesta opció de cicle de vida consta de quatre etapes:

Requisits En aquesta etapa es defineixen i planifiquen els requisits i funcionalitats que s'implementaran en aquesta iteració.

Anàlisi i Disseny Aquí són especificats i dissenyats tots els requisits definits en l'etapa anterior.

Implementació S'implementen els requisits seguint el disseny realitzat en l'etapa anterior.

Evaluació S'evaluen els canvis realitzats i es deixa constància d'altres possibles canvis i noves funcionalitats a afegir.

[1] Unified Modeling Language

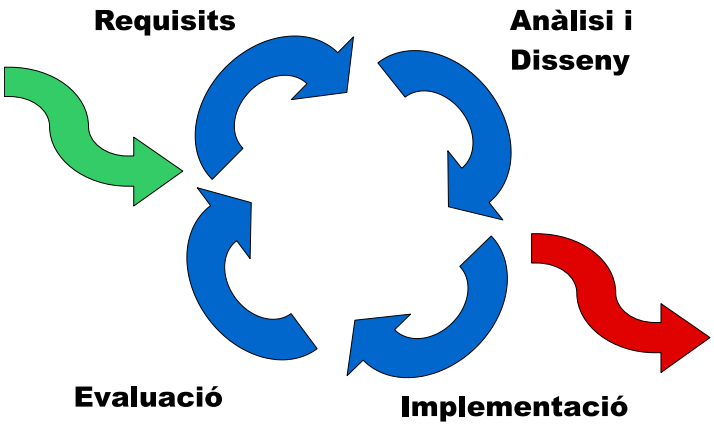


Figura 3.1: Cicle de vida *iteratiu i incremental*

Al començar el cicle de vida, la primera cosa que es realitza és la primera definició dels requisits. A partir de llavors, a cada iteració, les funcionalitats del programa van creixent fins que finalment s'acaba el seu desenvolupament. Aquest cicle de vida encaixa molt bé amb les necessitats de *GenetGUI*, i permet reunir-se amb el director al final de cada iteració i mostrar-li el progrés del projecte.

Evidentment, la informació sobre especificació i disseny que es documenta en aquesta memòria corresponen als estats finals del desenvolupament d'aquest cicle de vida, ja que dividir-ho en iteracions no hagués proporcionat informació útil, sinó més aviat tot el contrari, un excés d'informació totalment innecessari.

3.1 Anàlisi de requeriments

El primer pas abans de començar a especificar qualsevol de les altres parts és fer un anàlisi de requeriments. L'anàlisi de requeriments no és més que una especificació a molt alt nivell de la descripció i abast del projecte, així com les funcionalitats que ha de proporcionar.

El programa està pensat per a poder generar models formals de sortida, Xarxes de Petri en el nostre cas, a partir d'una entrada. A més, ha de ser capaç de mostrar gràficament aquests models, tan els d'entrada com els de sortida, i proporcionar elements que facilitin la comprensió i manipulació d'aquestes representacions gràfiques. Finalment, el sistema ha de permetre exportar els models formals, tan els finals, com els intermedis que s'hagin generat durant el procés.

Tenint en compte tot això, podem dividir el programa en les següents parts:

Format d'entrada Els diferents formats acceptats pel programa, i a partir dels quals, generarà les sortides.

Opcions Les diferents opcions que es tindran en compte a la hora de generar les Xarxes de Petri.

Projeccions Les projeccions són les particions que fem de l'entrada. El programa generarà les sortides a partir de la informació de cada partició, i sense tenir en compte la informació

de la resta de particions.

Xarxes de Petri És el model formal triat per a representar la sortida. El sistema generarà una Xarxa de Petri per a cada projecció feta sobre l'entrada. Aquestes Xarxes de Petri podran ser visualitzades gràficament així com manipulades.

Composició de Xarxes de Petri Poder composar les Xarxes de Petri corresponents a varies projeccions formant una única Xarxa de Petri, que també podrà ser visualitzada gràficament.

Guardar models El sistema proporcionarà la possibilitat de guardar les representacions en mode text dels diferents models utilitzats durant el procés.

3.1.1 Format d'entrada

S'ha decidit que les representacions formals a partir de les quals el sistema generarà les sortides seran els Sistemes de Transicions. Tot i així, el sistema ha de ser capaç d'obrir altres representacions i convertir-les a Sistemes de Transicions, per així, poder procesar-les. Aquestes altres representacions són Logs d'Events en format MXML i llistats de Traces.

A més, el sistema permetrà visualitzar de forma gràfica el Sistema de Transicions, ja sigui l'entrat directament o el generat a partir d'una entrada definida en un altre format. Aquesta representació gràfica anirà acompanyada d'elements que facilitin la visualització, com poden ser, l'arrossegar i desplaçar nodes, destacar el estat inicial del Sistema de Transicions i fer Zoom o Pan, entre altres.

3.1.2 Opcions

L'usuari ha de poder especificar les opcions desitjades a la hora de crear les Xarxes de Petri. Moltes d'aquestes opcions venen determinades per l'eina *Genet*.

3.1.3 Projeccions

El sistema ha de proporcionar els elements necessaris per poder fer diverses projeccions sobre el Sistema de Transicions. En concret, el sistema ha de permetre:

Assignar Assignar un conjunt d'events del Sistema de Transicions a una projecció. Un event pot tenir assignada més d'una projecció.

Treure Treure un event d'una projecció.

A més de permetre la creació de projeccions de forma manual, el sistema ha de proporcionar una serie d'algoritmes de creació de projeccions de forma automàtica i heurística. Aquestes projeccions creades automàticament hauran de poder ser modificades per l'usuari. És per això que aquests algoritmes reben el nom de *Algoritmes de Pre-Projecció*.

Tot aquest procés de creació de projeccions ha de poder ser realitzat de forma intuïtiva, ràpida i còmoda.

3.1.4 Xarxes de Petri

El sistema ha de permetre generar les Xarxes de Petri tenint en compte les opcions especificades per l'usuari i les projeccions fetes. Cada projecció ha de derivar en la seva pròpia Xarxa de Petri. Al igual que passava amb els Sistemes de Transicions, el sistema ha de mostrar de forma gràfica i entenedora les diverses Xarxes de Petri, així com proporcionar els elements per a fer intuïtiva i comprensible aquesta visualització. En el cas de les Xarxes de Petri, a part del Zoom, el Pan i el poder moure i arrossegar els elements, el sistema també ha de proporcionar:

Transicions activades El programa ha de facilitar la identificació de les Transicions Activades de la Xarxa de Petri, que són aquelles que poden ser disparades.

Disparar Transicions El sistema ha de permetre disparar una Transició activada de la Xarxa de Petri, i les conseqüències s'han de reflectir en la representació gràfica. D'aquesta manera l'usuari pot experimentar amb l'evolució que pot tenir una determinada Xarxa de Petri d'una manera ràpida i intuïtiva.

3.1.5 Composició de Xarxes de Petri

Un cop obtingudes diverses Xarxes de Petri de les diverses projeccions, el sistema ha de permetre composar varies d'aquestes projeccions obtenint una única Xarxa de Petri resultant. El subconjunt a composar ha de poder ser triat per l'usuari.

Al igual que les Xarxes de Petri generades a partir de les projeccions, aquesta Xarxa de Petri composada també ha de poder ser mostrada de forma gràfica i comptar amb els mateixos elements, com la possibilitat d'identificar i disparar Transicions activades.

3.1.6 Guardar models

El programa ha de permetre a l'usuari la possibilitat de guardar la representació en mode text dels models utilitzats. En concret, el Sistema de Transicions, les Xarxes de Petri intermèdies i la Xarxa de Petri composada.

3.1.7 Estructura del Programa

La Figura 3.2 mostra com seria l'estructura interna de l'aplicació tenint en compte els requeriments que s'acaben de definir.

3.2 Model de casos d'ús

El model de casos d'ús del sistema ens proporciona dues informacions bàsiques:

- Quines funcionalitats proporciona el nostre sistema a l'exterior.
- Quines entitats externes, anomenades *actors*, interactuen amb el nostre sistema.

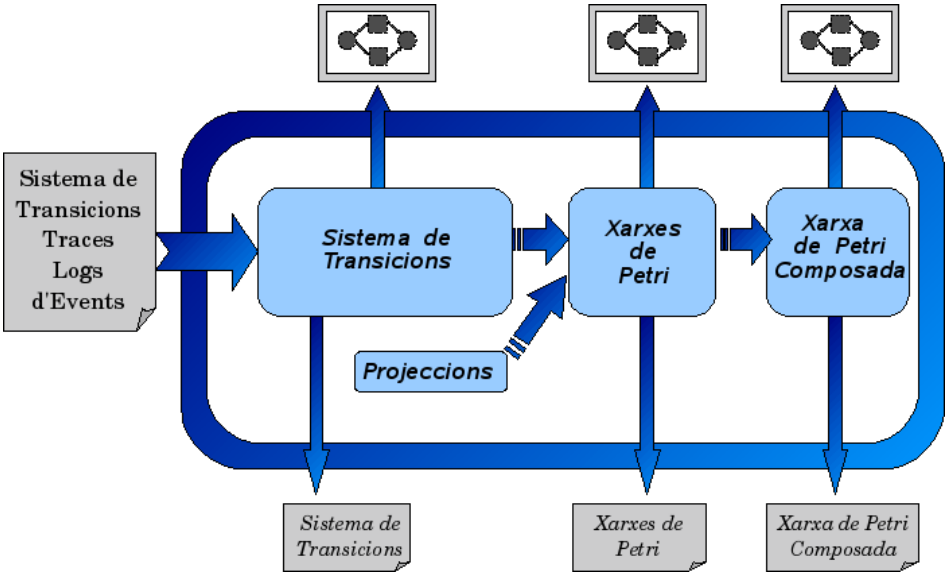


Figura 3.2: Estructura interna de l'aplicació

3.2.1 Actors

En primer lloc cal identificar quins són els actors que intervenen en el nostre programa.

Estem davant d'un programa monousuari, on el sistema només interactua amb un usuari. Assignarem doncs a aquest actor el nom d'*User*. Però a més també apareixen dos nous actors: *Genet* i *PComp* . Aquests actors fan referència als dos programes externs que seran usats per l'apliació per a dur a terme la seva tasca.

En la figura 3.3 es pot veure la representació gràfica dels actors del sistema.

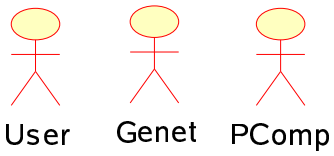


Figura 3.3: Actors del sistema

3.2.2 Diagrama de casos d'ús

Els diagrames de casos d'ús ens permeten veure quines són les funcionalitats que el nostre sistema software ofereix a l'exterior. En altres paraules: què ens permet fer. També mostren quins són els actors que inicien aquests casos d'ús o funcionalitats del sistema.

S'ha separat els diagrama en diversos subdiagrames per a facilitar-ne la comprensió. Així doncs, tenint en compte aquesta divisió, els casos d'ús quedarien de la següent manera:

- Casos d'ús relacionats amb la gestió de les Projeccions (Fig 3.4)
- Casos d'ús relacionats amb la gestió de les Xarxes de Petri (Fig 3.5)

- Casos d'ús relacionats amb l'obrir i exportar models (Fig 3.6)
- Casos d'ús relacionats amb la visualització dels models (Fig 3.7)

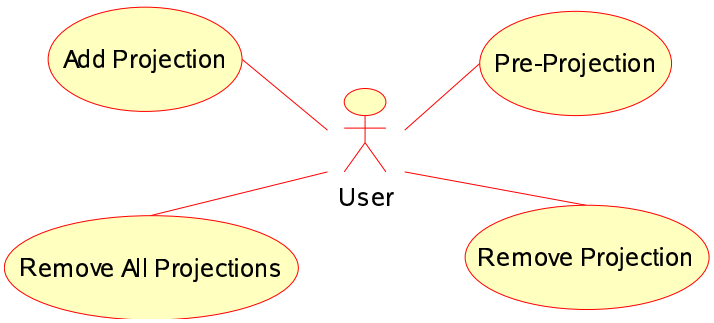


Figura 3.4: Diagrama de casos d'ús de les Projeccions

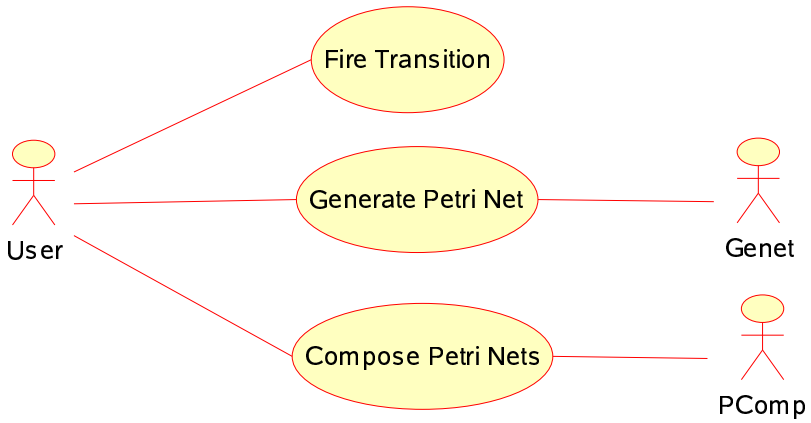


Figura 3.5: Diagrama de casos d'ús de les Xarxes de Petri

3.2.3 Especificació dels casos d'ús

A continuació mostrem l'especificació formal dels casos d'ús vistos a l'apartat 3.2.2.

3.2.3.1 Casos d'ús de les Projeccions

Cas d'ús: *Add Projection*
Context: L'usuari vol crear una nova projecció.
Actors Usuari
Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions d'èxit:

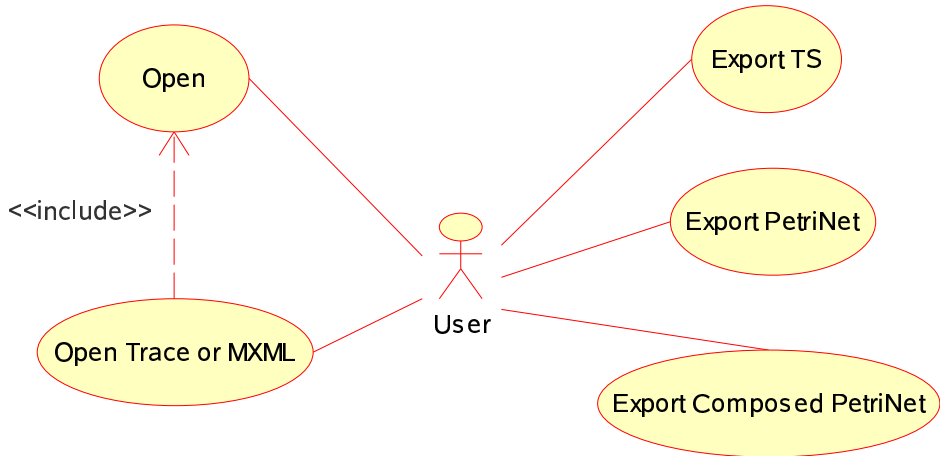


Figura 3.6: Diagrama de casos d'ús d'obrir i exportar models

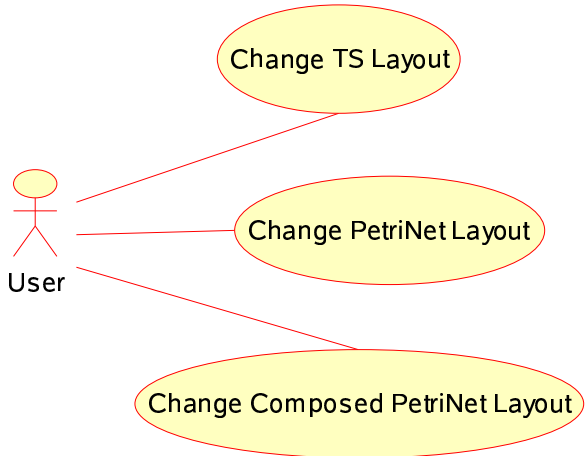


Figura 3.7: Diagrama de casos d'ús de les visualitzacions dels models

- El sistema afegeix una nova projecció al sistema amb els events triats per l'usuari.
- Si la projecció ja existeix, afegeix els events que no hi siguin als ja existents en aquesta projecció.

Escenari d'èxit principal:

1. L'usuari indica al sistema els events i el nom de la projecció a crear.
2. El sistema crea la projecció amb els events indicats. Si la projecció ja existeix, afegeix els events que no hi siguin als ja existents en aquesta projecció.

Escenaris alternatius:

- Nom de Projecció no vàlid: el sistema indica que el nom no és vàlid i torna al pas 1.

Cas d'ús: *Remove Projection*
Context: L'usuari vol treure alguns events d'una projecció.
Actors Usuari

Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions d'èxit:
- El sistema elimina de la projecció els events triats per l'usuari.
- Si la projecció es queda sense events, el sistema la elimina.

Escenari d'èxit principal:
1. L'usuari indica al sistema el nom de la projecció i els events a treure.
2. El sistema elimina els events indicats de la projecció. Si la projecció es queda sense events, els sistema la elimina.

Escenaris alternatius:
- Si el nom de Projecció no és vàlid, el sistema ho indica i torna al pas 1.

Cas d'ús: *Remove All Projections*
Context: L'usuari vol eliminar totes les projeccions del sistema.
Actors Usuari
Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions d'èxit:
- El sistema elimina totes les projeccions.

Escenari d'èxit principal:
1. L'usuari indica al sistema que vol eliminar totes les projeccions.
2. El sistema elimina totes les projeccions del sistema.

Escenaris alternatius:

Cas d'ús: *Pre-Projection*
Context: L'usuari vol usar un algortime de pre-projecció per a crear projeccions.
Actors Usuari
Precondicions:
- El sistema té un model d'entrada obert.
- El sistema té la informació necessària per poder usar els algortimes de pre-projecció.

Postcondicions d'èxit:
- El sistema crea les projeccions determinades per l'algoritme executat.

Escenari d'èxit principal:
1. L'usuari indica quin algortime vol utilitzar.
2. El sistema crea les projeccions determinades per l'algortime executat.

Escenaris alternatius:

3.2.3.2 Casos d'ús de les Xarxes de Petri

Cas d'ús: *Generate PetriNet*
Context: L'usuari vol generar les Xarxes de Petri a partir de l'entrada.
Actors Usuari, Genet
Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions d'èxit:
- El sistema genera i mostra les Xarxes de Petri.

Escenari d'èxit principal:
1. L'usuari indica al sistema que vol generar les Xarxes de Petri, amb les opcions que vol.
2. L'usuari transmet les opcions, les projeccions i el model d'entrada a *Genet*.
3. Genet genera les Xarxes de Petri tenint en compte les opcions i les projeccions.
4. El sistema mostra les Xarxes de Petri generades.

Escenaris alternatius:

Cas d'ús: *Compose PetriNets*
Context: L'usuari vol composar diverses Xarxes de Petri intermèdies.
Actors Usuari, PComp
Precondicions:
- El sistema té almenys una Xarxa de Petri intermèdia.

Postcondicions d'èxit:
- El sistema genera i mostra la Xarxes de Petri composada.

Escenari d'èxit principal:
1. L'usuari indica al sistema que vol composar les Xarxes de Petri, indicant quines xarxes vol composar.
2. El sistema transmet quines xarxes es volen composar i *PComp* les composa.
3. El sistema mostra la xarxa composada.

Escenaris alternatius:

Cas d'ús: *Fire Transition*
Context: L'usuari vol disparar una Transició d'una Xarxa de Petri.
Actors Usuari
Precondicions:
- La Transició a disparar és una Transició activa.

Postcondicions d'èxit:
- El sistema mostra la Xarxa de Petri després d'haver disparat la Transició.

Escenari d'èxit principal:
1. L'usuari indica la Transició a disparar.
2. El sistema canvia el marcatge de la Xarxa de Petri segons la regla de llançament de Transicions.
3. El sistema mostra la Xarxa de Petri amb el nou marcatge.

Escenaris alternatius:

3.2.3.3 Casos d'ús d'obrir i exportar models

Cas d'ús: *Open*
Context: L'usuari vol obrir un fitxer per a treballar amb ell.
Actors Usuari
Precondicions:
Postcondicions d'èxit:

- El sistema carrega el Sistema de Transicions.

Escenari d'èxit principal:

1. L'usuari indica que vol obrir un fitxer.
2. L'usuari indica la ruta al fitxer a obrir.
3. El sistema carrega i mostra el Sistema de Transicions. Si el fitxer a obrir no es un Sistema de Transicions, la entrada es convertida a Sistema de Transicions, abans de carregar-lo i mostrar-lo

Escenaris alternatius:

- Si el fitxer no és un fitxer vàlid, el sistema mostra un error i no modifica el seu estat intern.

Cas d'ús: *Open Trace or MXML*
Context: L'usuari vol obrir un fitxer de Traces o MXML especificant les opcions per convertir-lo en Sistema de Transicions.
Actors Usuari
Precondicions:
Postcondicions d'èxit:

- El sistema transforma l'entrada en un Sistema de Transicions i el carrega.

Escenari d'èxit principal:

1. L'usuari indica que vol obrir un fitxer.
2. L'usuari indica les opcions per a transformar l'entrada en un Sistema de Transicions.
3. L'usuari indica la ruta al fitxer a obrir.
4. La entrada es convertida a Sistema de Transicions segons les opcions triades per l'usuari, abans de carregar-lo i mostrar-lo.

Escenaris alternatius:

- Si el fitxer no és un fitxer vàlid, el sistema. mostra un error i no modifica el seu estat intern.

Cas d'ús: *Export TS*
Context: L'usuari vol exportar el Sistema de Transicions amb el que treballa.
Actors Usuari
Precondicions:

- Hi ha un Sistema de Transicions carregat.

Postcondicions d'èxit:

- El sistema guarda el Sistema de Transicions en el format text correponent.

Escenari d'èxit principal:

1. L'usuari indica que vol exportar el Sistema de Transicions.
2. L'usuari indica la ruta on desar aquest fitxer.
3. El sistema crea el fitxer en la ruta indicada amb la especificació del Sistema de Transicions en el format adecuat.

Escenaris alternatius:

- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Cas d'ús: *Export PetriNet*

Context: L'usuari vol exportar una de les Xarxes de Petri intermitges amb les que treballa.
Actors Usuari
Precondicions:
- Hi ha alguna Xarxa de Petri generada.

Postcondicions d'èxit:
- El sistema guarda la Xarxa de Petri en el format text correponent.

Escenari d'èxit principal:
1. L'usuari indica quina Xarxa de Petri vol exportar.
2. L'usuari indica la ruta on desar aquest fitxer.
3. El sistema crea el fitxer en la ruta indicada amb la especificació de la Xarxa de Petri a exportar en el format adecuat.

Escenaris alternatius:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Cas d'ús: *Export Composed PetriNet*
Context: L'usuari vol exportar la Xarxes de Petri composada amb les que treballa.
Actors Usuari
Precondicions:
- Hi ha alguna Xarxa de Petri composada.

Postcondicions d'èxit:
- El sistema guarda Xarxa de Petri en el format text correponent.

Escenari d'èxit principal:
1. L'usuari indica que vol exportar la Xarxa de Petri composada.
2. L'usuari indica la ruta on desar aquest fitxer.
3. El sistema crea el fitxer en la ruta indicada amb la especificació de la Xarxa de Petri a exportar en el format adecuat.

Escenaris alternatius:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

3.2.3.4 Casos d'ús de les visualitzacions dels models

Cas d'ús: *Change TS Layout*
Context: L'usuari vol canviar el layout dels elements del Sistema de Transicions.
Actors Usuari
Precondicions:
- Hi ha un Sistema de Transicions carregat.

Postcondicions d'èxit:
- El sistema canvia el layout dels elements del Sistema de Transicions.

Escenari d'èxit principal:
1. L'usuari indica el nou layout que vol utilitzar per a mostrar el Sistema de Transicions.
2. El sistema mostra el Sistema de Transicions amb el nou layout.

Escenaris alternatius:

Cas d'ús: *Change PetriNet Layout*
Context: L'usuari vol canviar el layout de les Xarxes de Petri intermèdies.
Actors Usuari
Precondicions:
- Hi ha alguna Xarxa de Petri generada.

Postcondicions d'èxit:
- El sistema canvia el layout de les Xarxes de Petri intermèdia.

Escenari d'èxit principal:
1. L'usuari indica el nou layout que vol utilitzar per a mostrar les Xarxes de Petri.
2. El sistema mostra les Xarxes de Petri amb el nou layout.

Escenaris alternatius:

Cas d'ús: *Change Composed PetriNet Layout*
Context: L'usuari vol canviar el layout de la Xarxa de Petri composada.
Actors Usuari
Precondicions:
- Hi ha una Xarxa de Petri composada.

Postcondicions d'èxit:
- El sistema canvia el layout de la Xarxa de Petri composada.

Escenari d'èxit principal:
1. L'usuari indica el nou layout que vol utilitzar per a mostrar la Xarxa de Petri.
2. El sistema mostra la Xarxa de Petri composada amb el nou layout.

Escenaris alternatius:

3.3 Model conceptual de dades

El model conceptual de dades ens permetrà veure quines són les entitats principals del domini del nostre projecte, així com les relacions entre elles.

Cal parar atenció en que el que estem fent ara és el domini, es a dir, les entitats del projecte. Es deixen de banda aquí classes corresponents a la posterior etapa de disseny, com poden ser els controladors o les vistes.

La figura 3.8 mostra les diferents classes del domini i les seves relacions.

El primer que es pot veure al observar el diagrama del model conceptual de les dades és la creació de dos tipus de dades especials: *TS* i *PN*. Aquest tipus de dades fan referència als models formals dels Sistemes de Transicions (*Transition System* en anglès) i les Xarxes de Petri (*Petri Net* en anglès). Els elements de cadascun d'aquests models així com les restriccions i funcions que aquests posseeixen estan definides en el capítol d'introducció d'aquesta memòria.

La segona cosa que es pot veure en el diagrama és que el sistema només accepta que hi hagi un Sistema de Transicions i una Xarxa de Petri composada en tot moment. En canvi, sí permet crear tantes Xarxes de Petri intermèdies com es vulguin.

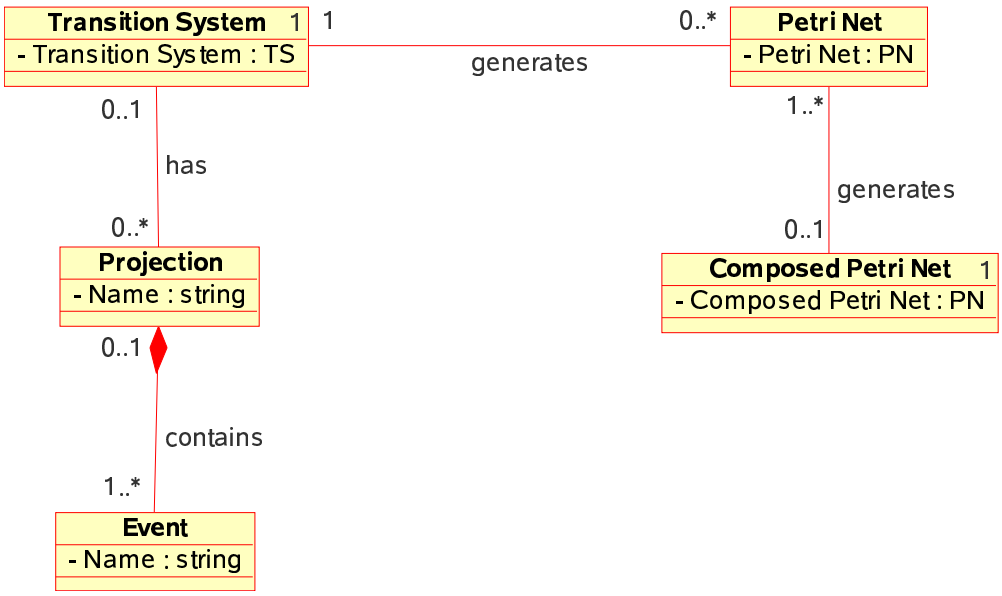


Figura 3.8: Diagrama de classes del domini

El diagrama també il·lustra la creació de Projeccions. Aquestes projeccions estan composades d'un o diversos events. L'única restricció és que els events que composen les projeccions han de formar part del conjunt d'events definits pel Sistema de Transicions amb el que es treballa.

3.4 Model del comportament del sistema

El model del comportament del sistema va una mica més enllà i ens proporciona informació sobre quin és el comportament del sistema en cada un dels casos d'ús especificats anteriorment.

En aquesta primera etapa, però, considerarem el sistema com a única entitat a la qual podem invocar mètodes. Serà més endavant, en l'etapa de disseny, quan es definiran les interaccions entre les diferents classes del sistema. Això és així perquè en aquests moments encara no sabem amb exactitud quina informació necessitarem en el transcurs del cas d'ús, i per tant és encara d'hora per designar les responsabilitats i expertesa de cada una de les classes.

Per a fer el model del comportament del sistema primer definirem els diagrames de seqüència i després els contractes de les operacions.

3.4.1 Diagrames de seqüència

Els diagrames de seqüència ens permetran veure quina és la seqüència d'esdeveniments que es duen a terme entre l'usuari i el sistema en tots i cadascun dels casos d'ús especificats a l'apartat 3.2.2. Aquests diagrames ens permetran saber quines operacions ha de dur a terme el sistema per satisfer els casos d'ús, la qual cosa ens facilitarà enormement la labor de disseny,

que es durà a terme posteriorment, i ens ajudarà a assegurar-nos que no passem res per alt que ens pogués dificultar la implementació del projecte en un futur estat de desenvolupament.

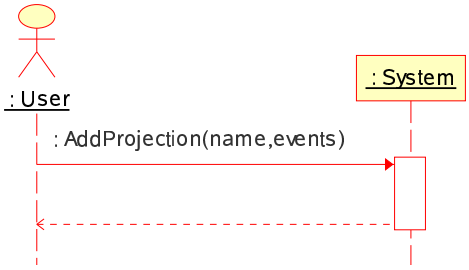


Figura 3.9: Diagrama de seqüència “Add Projection”

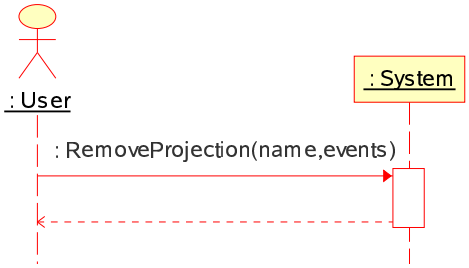


Figura 3.10: Diagrama de seqüència “Remove Projection”

Figura 3.11: Diagrama de seqüència “Remove All”

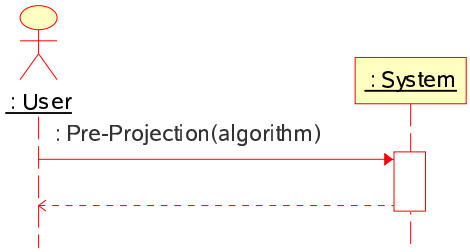


Figura 3.12: Diagrama de seqüència “Pre-Projection”

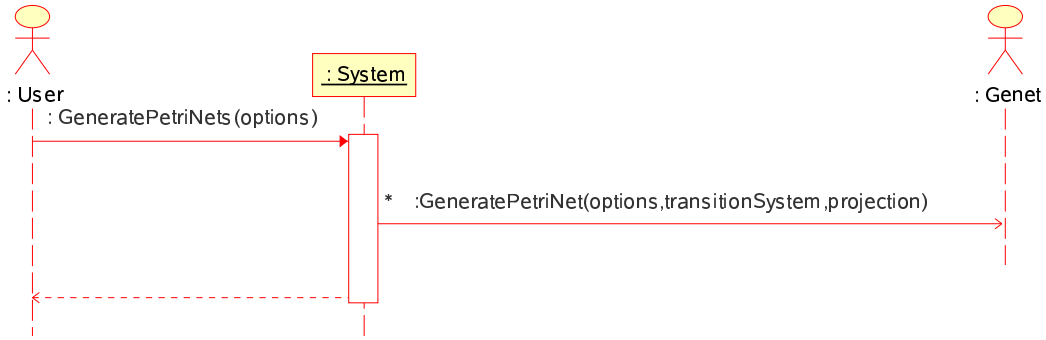


Figura 3.13: Diagrama de seqüència “Generate Petri Nets”

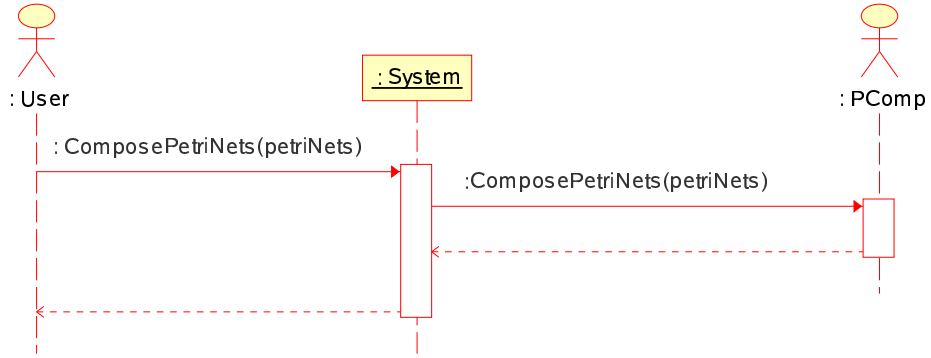


Figura 3.14: Diagrama de seqüència “Compose Petri Nets”

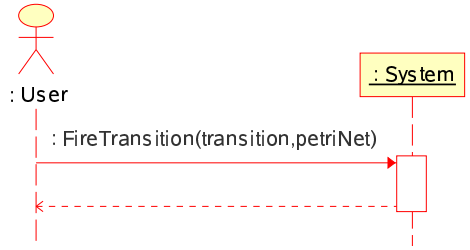


Figura 3.15: Diagrama de seqüència “Fire Transition”

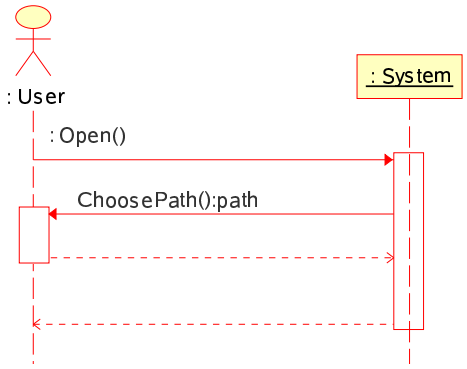


Figura 3.16: Diagrama de seqüència “Open”

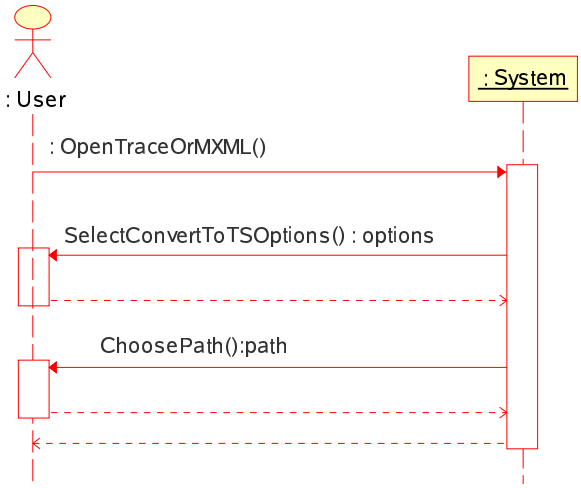


Figura 3.17: Diagrama de seqüència “Open Trace or MXML”

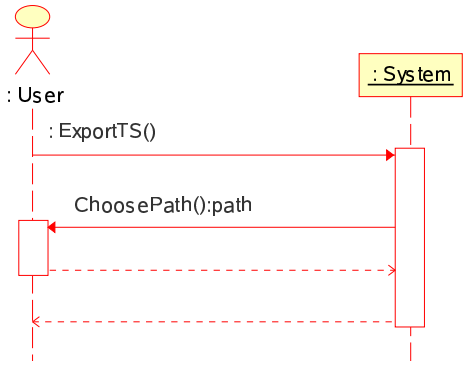


Figura 3.18: Diagrama de seqüència “Export TS”

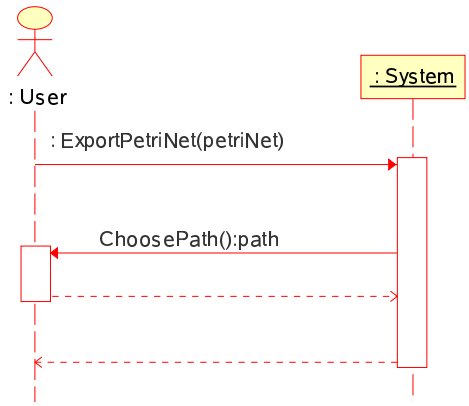


Figura 3.19: Diagrama de seqüència “Export Petri Net”

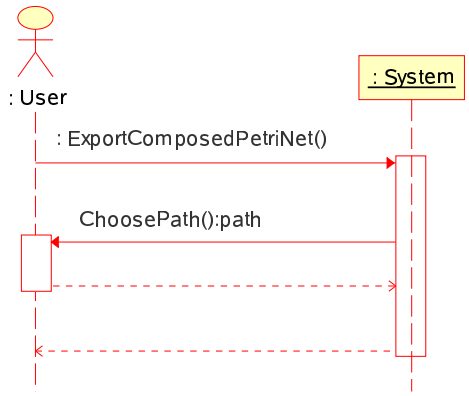


Figura 3.20: Diagrama de seqüència “Export Composed Petri Net”

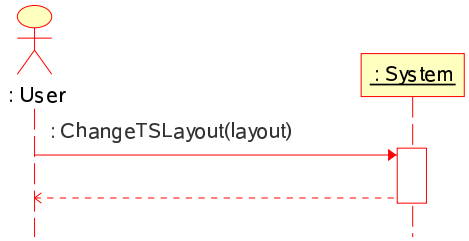


Figura 3.21: Diagrama de seqüència “Change TS Layout”

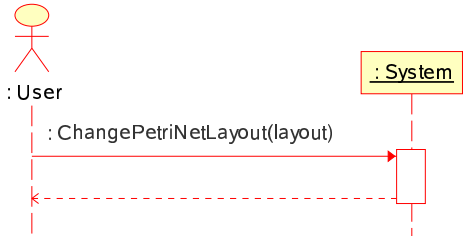


Figura 3.22: Diagrama de seqüència “Change Petri Net Layout”

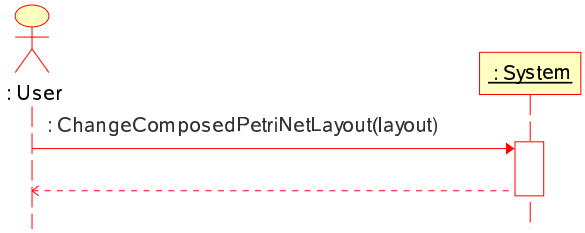


Figura 3.23: Diagrama de seqüència “Change Composed Petri Net Layout”

3.4.2 Contractes de les operacions

Un cop fets els diagrames de seqüència és el moment de dur a terme les definicions dels contractes de totes les operacions que hi han aparegut.

3.4.2.1 Contractes de les Projeccions

Nom: *AddProjection(name,events)*
Responsabilitats: Crear una nova Projecció al sistema amb el nom i els events especificats com a parametres.
Excepcions:
- Si el nom de la projecció no és vàlid (només espais) el sistema retorna un error.

Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- El sistema afegeix una nova projecció al sistema amb els events triats per l'usuari.
- Si la projecció ja existeix, afegeix els events que no hi siguin als ja existents en aquesta projecció.

Sortida:

Nom: *RemoveProjection(name,events)*
Responsabilitats: Elimina els events indicats de la projecció especificada com a paràmetre.
Excepcions:
- Si el nom de la projecció no és vàlid (només espais) el sistema retorna un error.

Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- El sistema elimina de la projecció els events triats per l'usuari.
- Si la projecció es queda sense events, el sistema la elimina.

Sortida:

Nom: *RemoveAll()*
Responsabilitats: Elimina totes les projeccions del sistema.
Excepcions:
Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- El sistema elimina totes les projeccions.

Sortida:

Nom: *Pre-Projection(algorithm)*
Responsabilitats: S'usa un algortime de Pre-Projecció per a crear noves projeccions en el sistema.
Excepcions:
Precondicions:

- El sistema té un model d'entrada obert.
- El sistema té la informació necessària per a poder usar els algortimes de pre-projecció.

Postcondicions:

- El sistema crea les projeccions determinades per l'algoritme executat.

Sortida:

3.4.2.2 Contractes de les Xarxes de Petri

Nom: *GeneratePetriNets(options)*
Responsabilitats: Demanar al sistema la generació de les Xarxes de Petri tenint en compte les opcions especificades.
Excepcions:
Precondicions:

- El sistema té un model d'entrada obert.

Postcondicions:

- El sistema genera i mostra les Xarxes de Petri.

Sortida:

Nom: *GeneratePetriNet(options,transitionSystem,projections)*
Responsabilitats: Demanar a *Genet* que generi una Xarxa de Petri a partir del Sistema de Transicions, la projecció i les opcions especificades.
Excepcions:
Precondicions:
Postcondicions:

- *Genet* crea un arxiu amb la Xarxa de Petri resultant.

Sortida:

Nom: *ComposePetriNets(petriNets)*
Responsabilitats: Demanar al sistema la composició de les Xarxes de Petri indicades.
Excepcions:
Precondicions:

- El sistema té almenys una Xarxa de Petri intermèdia.

Postcondicions:

- El sistema genera la Xarxes de Petri composada.

Sortida:

Nom: *ComposePetriNets(petriNets)*
Responsabilitats: Demanar a *PComp* la composició de les Xarxes de Petri indicades.
Excepcions:
Precondicions:

- Existeix almenys una Xarxa de Petri a composar.

Postcondicions:

- *PComp* crea un arxiu amb la Xarxes de Petri composada.

Sortida:

Nom: *FireTransition(transition, petriNet)*
Responsabilitats: Demanar al sistema que dispari la transició indicada.
Excepcions:
Precondicions:
- La Transició a disparar és una Transició activa.

Postcondicions:
- El sistema cambia l'estat de la Xarxa de Petri seguint les regles de disparament d'una transició.

Sortida:

3.4.2.3 Contractes d'obrir i exportar models

Nom: *Open()*
Responsabilitats: Demanar al sistema que es vol obrir un model d'entrada. El sistema haurà de demanar on és el fitxer a obrir.
Excepcions:
- Si l'usuari cancel·la l'operació no es modificarà l'estat intern del sistema.
- Si el fitxer no és un fitxer vàlid, el sistema mostra un error i no modifica el seu estat intern.

Precondicions:
Postcondicions:
- El sistema carrega el Sistema de Transicions.

Sortida:

Nom: *ChoosePath()*
Responsabilitats: Demanar a l'usuari que indiqui un ruta.
Excepcions:
- Si l'usuari cancel·la, no es retornara cap ruta.

Precondicions:
Postcondicions:
Sortida: Ruta seleccionada per l'usuari.

Nom: *OpenTraceOrMXML()*
Responsabilitats: Demanar al sistema que es vol obrir un fitxer MXML o de Traces. El sistema haurà de demanar les opcions per a transformar-lo en Sistema de Transicions, així com l'ubicació del fitxer.
Excepcions:
- Si l'usuari cancel·la l'operació no es modificarà l'estat intern del sistema.
- Si el fitxer no és un fitxer vàlid, el sistema mostra un error i no modifica el seu estat intern.

Precondicions:
Postcondicions:
- El sistema transforma l'entrada en un Sistema de Transicions i el carrega.

Sortida:

Nom: *SelectConvertToTSOptions()*
Responsabilitats: Demanar a l'usuari que especifiqui les opcions per a convertir un arxiu en Sistema de Transicions
Excepcions:
- Si l'usuari cancel·la, no es retornara cap opció.

Precondicions:
Postcondicions:
Sortida: Opcions a tenir en compte per a convertir un model a Sistema de Transicions.

Nom: *ExportTS()*
Responsabilitats: Demanar al sistema l'exportació del Sistema de Transicions. El sistema demanarà on es vol guardar aquest fitxer.
Excepcions:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Precondicions:
- Hi ha un Sistema de Transicions carregat.

Postcondicions:
- El sistema guarda el Sistema de Transicions en el format text correponent.

Sortida:

Nom: *ExportPetriNet(petriNet)*
Responsabilitats: Demanar al sistema l'exportació de la Xarxa de Petri indicada. El sistema demanarà on es vol guardar aquest fitxer.
Excepcions:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Precondicions:
- Hi ha alguna Xarxa de Petri generada.

Postcondicions:
- El sistema guarda la Xarxa de Petri en el format text correponent.

Sortida:

Nom: *ExportComposedPetriNet()*
Responsabilitats: Demanar al sistema l'exportació de la Xarxa de Petri composada. El sistema demanarà on es vol guardar aquest fitxer.
Excepcions:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Precondicions:
- Hi ha alguna Xarxa de Petri composada.

Postcondicions:
- El sistema guarda Xarxa de Petri en el format text correponent.

Sortida:

3.4.2.4 Contractes de les visualitzacions dels models

Nom: *ChangeTSLayout(layout)*
Responsabilitats: Demanar al sistema cambiar el layout de la visualització del Sistema de Transicions pel layout indicat.
Excepcions:
Precondicions:
- Hi ha un Sistema de Transicions carregat.

Postcondicions:
- El sistema canvia el layout dels elements del Sistema de Transicions.

Sortida:

Nom: *ChangePetriNetLayout(layout)*
Responsabilitats: Demanar al sistema cambiar el layout de la visualització de les Xarxes de Petri pel layout indicat.
Excepcions:
Precondicions:
- Hi ha alguna Xarxa de Petri generada.

Postcondicions:
- El sistema canvia el layout de les Xarxes de Petri intermèdies.

Sortida:

Nom: *ChangeComposedPetriNet(layout)*
Responsabilitats: Demanar al sistema cambiar el layout de la visualització de la Xarxes de Petri Composada pel layout indicat.
Excepcions:
Precondicions:
- Hi ha una Xarxa de Petri composada.

Postcondicions:
- El sistema canvia el layout de la Xarxa de Petri composada.

Sortida:

4
Disseny i implementació

Un cop acabada l'etapa d'especificació del projecte, ja disposem amb detall quines són les funcionalitats que el nostre sistema ha d'oferir.

Ara és el moment de començar l'etapa del disseny [13]. En aquesta etapa fem el pas del *què* al *com*.

Una de les primeres qüestions que haurem de resoldre és quina és la tecnologia que utilitzarem per implementar el nostre projecte (llenguatge o llenguatges de programació i totes aquelles altres biblioteques o elements importants que podem necessitar).

L'etapa de disseny ha de definir en detall el nostre sistema, perquè això ens pugui dur amb més facilitat a la implementació final del mateix. Així doncs, definirem primer quin o quins patrons arquitectònics volem utilitzar per al nostre projecte. Haurem de normalitzar els diagrames de classes especificats amb anterioritat, així com els diferents contractes que han aparegut.

4.1 Patró arquitectònic

Arribats a aquest punt cal definir quin patró arquitectònic serà el triat per al desenvolupament del programa. L'escollit ha estat el *Patró en capes*, un dels patrons més populars i usats i que s'adapta a la perfecció a les característiques de la nostra aplicació. En concret s'ha optat per una de les implementacions més conegudes del patró: el *Patró en 3 capes*. Com el seu nom indica, el sistema queda dividit en 3 capes. Aquestes capes i les seves funcions són:

Capa de presentació És la que s'encarrega de la comunicació amb l'usuari.

Capa de domini S'encarrega d'executar les accions encomanades, canviant l'estat del domini.

Capa de persistència de dades S'ocupa de gestionar la persistència de les dades.

La divisió en aquestes capes, juntament amb el fet de que la comunicació entre capes estigui limitada, tal i com indica la Figura 4.1 fa que es mantingui un alt nivell d'abstracció i una baixa cohesió entre els elements. Producte d'això, el codi generat serà un codi fàcil de mantenir, canviable, reutilitzable i portable a altres plataformes.

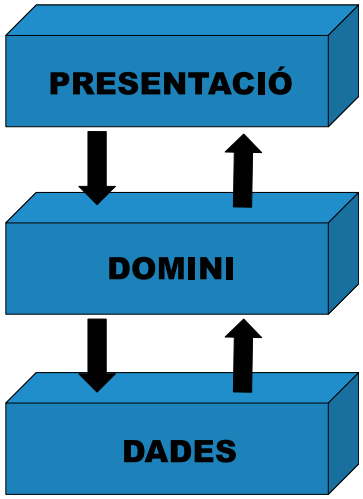


Figura 4.1: Patró arquitectònic de tres capes

4.2 Tecnologia utilitzada

En aquesta secció es detallen les diferents tecnologies triades a l'hora de dur a terme la implementació del nostre sistema, i les justificacions per aquestes tries. Al referirnos a tecnologia ens referim al llenguatge de programació i a les diferents llibreries que usarem, un dels punts més importants en el cas d'un programa com *GenetGUI*.

4.2.1 Llenguatge de programació

A l'hora de triar un llenguatge era important que aquest implementés el paradigma d'Orientació a Objectes, donada la seva naturalesa i les enormes i contrastades bondats d'aquest paradigma de programació.

Finalment el llenguatge triat ha estat *Java*. Java és un llenguatge intrínsecament multiplataforma, ja que es basa en fabricar codi per una màquina virtual, que està implementada per multitud de plataformes i sistemes operatius. És a més un llenguatge purament orientat a objectes, molt popular i estès, i té a la seva disposició tota una sèrie de llibreries que faciliten molts processos i tasques. En un segon pla també hi havia el fet de que és el llenguatge amb el que més he treballat, que més domino i amb el que em sento més còmode.

4.2.2 Tecnologia per a la capa de presentació

El nostre programa és un sistema on el visualitzar la informació és una part funamental. És per aquesta raó que tan important com triar el llenguatge de programació, és triar la tecnologia per a la presentació.

La llibreria escollida ha estat *Prefuse*. Prefuse es un *toolkit* basat en Java, pensat per construir aplicacions per la visualització interactiva d'informació. Prefuse proporciona una gran quantitat d'elements pel modelatge d'informació, la visualització i la interacció, així com donar suport per l'animació, les consultes dinàmiques i la connectivitat amb bases de dades. Prefuse ha estat escrit en Java, i pot ser integrat fàcilment en una aplicació *Java Swing* o un

Web Applet. A més, la seva llicència és *BSD*, i pot ser usat lliurement per a propòsits comercials o no comercials.

Aquestes han estat les raons que ens han portat a l'ús de Prefuse per a *GenetGUI*. L'únic inconvenient d'utilitzar Prefuse ha estat la manca de documentació oficial d'aquesta llibreria. De fet, l'única documentació que hi ha és la *API* de la llibreria i un fòrum per resoldre dubtes. Per intentar pal·liar aquesta mancança, es va decidir realitzar un petit tutorial de Prefuse, com a tasca complementària al desenvolupament del projecte. La finalitat d'aquest tutorial és la d'explicar com implementar una visualització amb Prefuse de principi a fi, usant la creació de la representació gràfica d'una Xarxa de Petri com a exemple. Una copia d'aquest tutorial es pot trobar a l'apartat d'apèndixs d'aquesta memòria. Remarcar que el tutorial ha estat escrit en anglès per facilitar el seu accés a persones de tot el món. De fet, ja ha servit com a guia per al projecte d'un alumne de doctorat de la FIB.

Per a la resta de la interfície gràfica del programa, s'ha triat *Swing*, tancant així el triangle Java-Prefuse-Swing i aprofitant tots els avantatges de compatibilitat que això comporta. A més, al triar Swing, s'ha pogut fer ús de l'editor d'interfícies gràfiques que proporciona *Netbeans*, l'entorn de desenvolupament d'aplicacions triat per al projecte.

4.2.3 Tecnologia per a la capa de domini

A part de les classes propies del domini que proporciona Prefuse, s'han usat dues llibreries més: *JGraphT* i *Jung*. Les dues son llibreries per Java que faciliten el manipular i emmagatzemar grafs. Han estat especialment útils per a la implementació dels algoritmes de Pre-Projecció.

Però els elements més importants en la capa de domini són, sense dubte, les dues aplicacions externes usades pel programa: *Genet* i *PComp*.

Genet [1] [9] és una eina que permet fer *mining* i sintetitzar Xarxes de Petri a partir de Sistemes de Transicions. L'aplicació ha estat desenvolupada per Josep Carmona de la Universitat Politècnica de Catalunya.

PComp [3] és una aplicació per la composició en paral·lel de Xarxes de Petri. L'eina ha estat desenvolupada per Victor Khomenko de la Universitat de Newcastle.

4.2.4 Tecnologia per a la capa de dades

Una de les funcionalitats que proporciona *GenetGUI* és la de poder obrir fitxers codificats en diferents formats. I és aquí on entra l'ús de les llibreries *Antlr* i *Xerces*.

Antlr[15] és un generador de parsejadors: un programa per a generar codi capaç de traduir un determinat llenguatge d'entrada a diverses estructures de dades. Java és entre els molts llenguatges de programació per als quals es pot generar el codi del parsejador. Això afegit al fet de la seva facilitat i a que va ser la llibreria triada per a l'assignatura de Compiladors, fan d'Antlr una bona opció. A més, s'ha fet servir *AntlrWorks*[7] com a entorn de desenvolupament per a Antlr.

Mentre que per llegir fitxers que codifiquin Sistemes de Transicions, Traces o Xarxes de Petri, Antlr és l'ideal, no és així per als fitxer de Logs d'Events, codificats en MXML. MXML és una variant del XML pensada per especificar Logs d'Events. I es per aquesta raó, que es pot fer servir alguna de les moltes eines altament optimitzades pel tractament d'arxius XML. En concret, la llibreria triada per aquest projecte ha estat *Xerces*, una de les més famoses per Java.

4.2.5 Arquitectura de les tecnologies

La Figura 4.2 mostra la arquitectura de les diferents tecnologies usades, i com es relacionen entre elles.

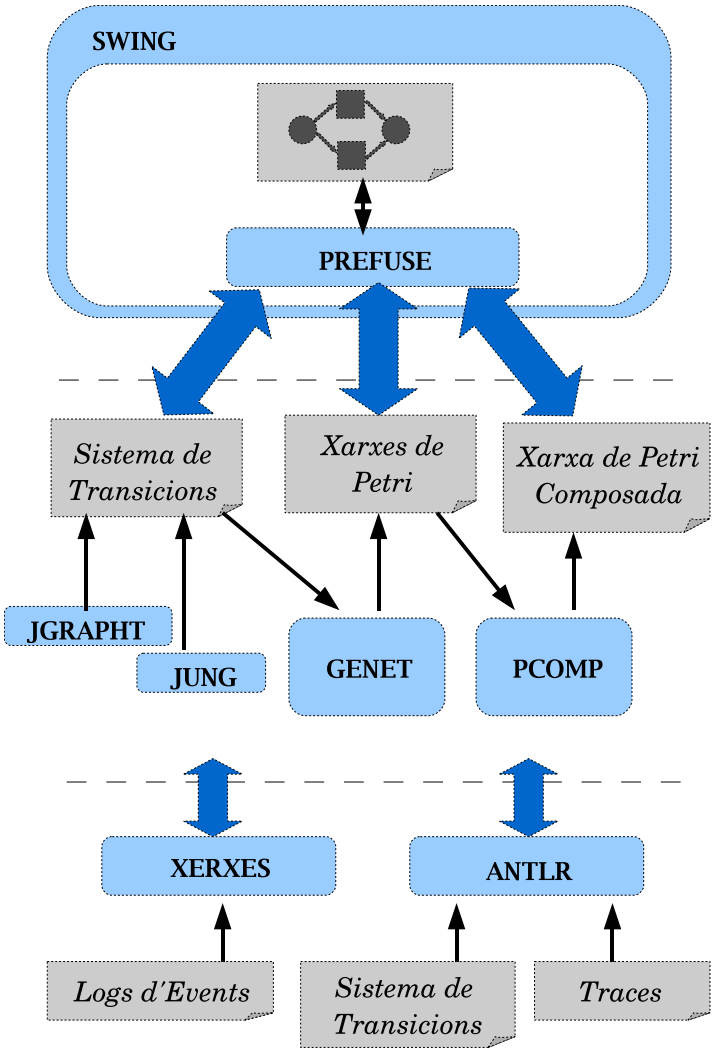


Figura 4.2: Arquitectura de les tecnologies

4.3 Domini

Arribats a aquest punt el que cal és dissenyar la capa del domini de l'aplicació. Això passa per diverses etapes:

- Normalització del model de dades.
- Normalització dels contractes de les operacions.

4.3.1 Normalització del model de dades

El primer que hem de fer és dur a terme una normalització del diagrama de classes. No tenim operacions ternàries, i per tant, la part de la normalització referent a l'eliminació d'aquestes associacions no serà necessària. Tampoc tenim atributs derivats. Tot i això, cal començar a pensar quins patrons i quines estructures de dades voldrem aplicar.

El primer patró que s'utilitzarà serà el *Patró Façana*. Fixem-nos que tenim diferents classes a gestionar: projeccions, events i xarxes de petri. Com haurem de manegar col·leccions d'aquestes classes, s'ha decidit crear una sèrie de controladors per a cada una d'aquestes entitats, que s'encarregaran de gestionar aquests objectes, així com de proporcionar una sèrie de mètodes per al seu maneig.

Així doncs, aplicarem el patró façana creant tres controladors (o *Managers*) que seran la façana que ens proporcionarà els mètodes d'accés a aquestes classes:

- Controlador de projeccions (Projection Manager).
- Controlador de events (Event Manager).
- Controlador de xarxes de Petri (PetriNet Manager).

A més, per dur a terme la comunicació entre controladors, tindrem un quart controlador general que anomenarem *Controlador de domini (Domain Controller)*. Aquest controlador gestionarà el domini i comunicarà la resta de controladors entre sí quan sigui necessari, tot i que com veurem després, alguns dels controladors es podran comunicar entre ells directament per facilitar les crides.

Pel que fa a les gestió de les col·leccions de projeccions,events i xarxes de petri, s'usarà el *Patró Diccionari*. Aquest patró ens permetrà accedir a un element de la col·lecció a través d'una clau única. Per les projeccions i els events usarem com a clau un nom, i per les xarxes de petri un índex. De cara a la implementació, l'accés per índex el durem a terme usant *vectors*. En canvi, per a un accés per nom utilitzarem la classe *Map* que ens proporciona les *Collections* de *Java*.

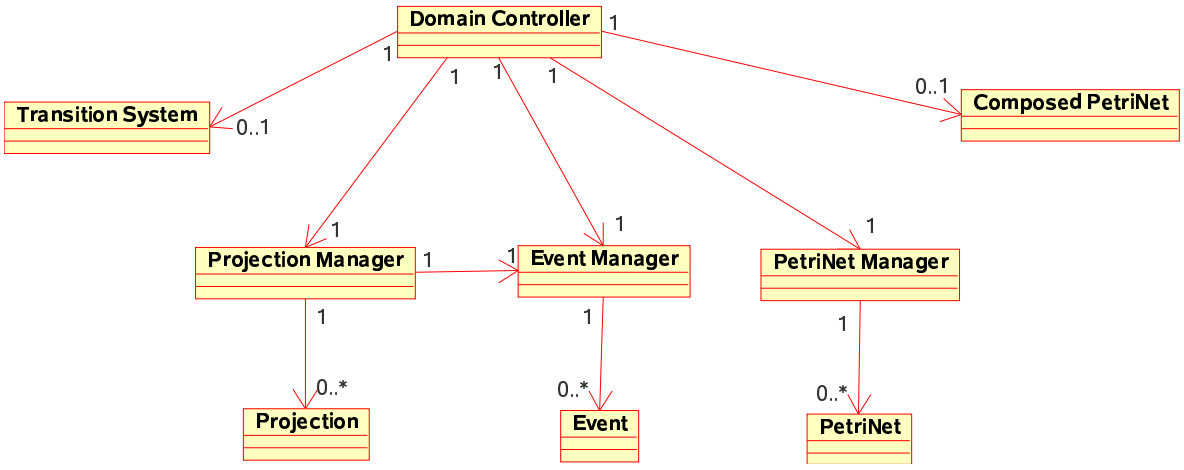


Figura 4.3: Diagrama de classes dels controladors

Per simplificar el dibuix, en la Figura 4.3 es pot veure el diagrama de classes que relaciona els controladors amb els diferents elements del domini. S'han omès expressament els atributs i les altres relacions per no embrutar excessivament el dibuix amb informació que ja s'ha explicat textualment.

4.3.2 Normalització dels contractes de les operacions

El següent pas és el de la normalització dels contractes de les operacions especificats a l'apartat 3.4.1. Serà en aquesta normalització on podrem veure amb més detall quines són les operacions que s'han de dur a terme i quines són les classes que disposen de la informació necessària per endur-se l'assignació d'aquestes operacions.

Manca dir que, serà la pròpia capa de presentació l'encarregada de garantir que es compleixen les precondicions especificades en el contracte. Això és gràcies a la forma de dissenyar la interfície gràfica del programa, que no permet executar les operacions si no es compleixen les precondicions. La capa de presentació està explicada en l'apartat 3.3.

4.3.2.1 Contractes de les Projeccions

Nom: *AddProjection(name,events)*
Responsabilitats: Crear una nova Projecció al sistema amb el nom i els events especificats com a parametres.
Excepcions:
- Si el nom de la projecció no és vàlid (només espais) el sistema retorna un error.

Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- S'obté la projecció que correspon amb el nom donat (Projection Manager).
- En cas de no existir, es crea(Projection Manager).
- S'obtenen els events corresponents als noms d'events donats i s'afegeixen els events a la projecció (Event Manager, Projection).

Sortida:

Nom: *RemoveProjection(name,events)*
Responsabilitats: Elimina els events indicats de la projecció especificada com a parametre.
Excepcions:
- Si el nom de la projecció no és vàlid (només espais) el sistema retorna un error.

Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- S'obté la projecció que correspon amb el nom donat (Projection Manager).
- En cas d'existir, s'obtenen els events corresponents als noms d'events donats i s'eliminen els que formin part de la projecció (Event Manager, Projection).
- S'obté el nombre d'events restants de la projecció. Si es 0 s'elimina la projecció. (Projection Manager).

Sortida:

Nom: *RemoveAll()*
Responsabilitats: Elimina totes les projeccions del sistema.
Excepcions:
Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- S'obté el llistat de totes les instàncies de les projeccions, i s'eliminen (Projection Manager).

Sortida:

Nom: *Pre-Projection(algorithm)*
Responsabilitats: S'usa un algortime de Pre-Projecció per a crear noves projeccions en el sistema.
Excepcions:
Precondicions:
- El sistema té un model d'entrada obert.
- El sistema té la informació necessària per a poder usar els algortimes de pre-projecció.

Postcondicions:
- S'executa el algortime determinat, creant així grups d'events que correspondran a noves projeccions (Domain Controller).
- Per cada grup, es crea una nova projecció, s'obtenen els events que composaran aquella projecció, i s'hi inclueixen (Projection Manager, Event Manager, Projection).

Sortida:

4.3.2.2 Contractes de les Xarxes de Petri

Nom: *GeneratePetriNets(options)*
Responsabilitats: Demanar al sistema la generació de les Xarxes de Petri tenint en compte les opcions especificades.
Excepcions:
Precondicions:
- El sistema té un model d'entrada obert.

Postcondicions:
- El sistema obté les projeccions (Projection Manager).
- El sistema obté la ruta del Sistema de Transicions amb el que està treballant (TransitionSystem).
- Per cada projecció el sistema executa la operació GeneratePetriNet, amb les opcions especificades, la ruta del Sistema de Transicions, i els events que la composen (Domain Controller).

Sortida:

Nom: *GeneratePetriNet(options,transitionSystem,projections)*
Responsabilitats: Demanar a *Genet* que generi una Xarxa de Petri a partir del Sistema de Transicions, la projecció i les opcions especificades.
Excepcions:
Precondicions:
Postcondicions:
- El sistema demana a *Genet* que crei un arxiu amb la Xarxa de Petri a partir del Sistema de Transicions, la projecció i les opcions especificades (Domain Controller).

Sortida:

Nom: *ComposePetriNets(petriNets)*
Responsabilitats: Demanar al sistema la composició de les Xarxes de Petri indicades.
Excepcions:
Precondicions:
- El sistema té almenys una Xarxa de Petri intermèdia.

Postcondicions:
- El sistema executa l'operació ComposePetriNets amb les Xarxes de Petri indicades (Domain Controller).

Sortida:

Nom: *ComposePetriNets(petriNets)*
Responsabilitats: Demanar a *PComp* la composició de les Xarxes de Petri indicades.
Excepcions:
Precondicions:
- Existeix almenys una Xarxa de Petri a composar.

Postcondicions:
- El sistema demana a *PComp* que crei un arxiu amb la Xarxes de Petri composada (Domain Controller).

Sortida:

Nom: *FireTransition(transition, petriNet)*
Responsabilitats: Demanar al sistema que dispari la transició indicada.
Excepcions:
Precondicions:
- La Transició a disparar és una Transició activa.

Postcondicions:
- El sistema canvia l'estat de la Xarxa de Petri seguint les regles de disparament d'una transició (PetriNet).

Sortida:

4.3.2.3 Contractes d'obrir i exportar models

Nom: *Open()*
Responsabilitats: Demanar al sistema que es vol obrir un model d'entrada. El sistema haurà de demanar on és el fitxer a obrir.
Excepcions:
- Si l'usuari cancel·la l'operació no es modificarà l'estat intern del sistema.
- Si el fitxer no és un fitxer vàlid, el sistema mostra un error i no modifica el seu estat intern.

Precondicions:
Postcondicions:
- El sistema demana el fitxer a obrir (Presentació).
- El sistema carrega el model especificat en el fitxer (Dades).

Sortida:

Nom: *ChoosePath()*
Responsabilitats: Demanar a l'usuari que indiqui un ruta.
Excepcions:
- Si l'usuari cancel·la, no es retornarà cap ruta.

Precondicions:
Postcondicions:
Sortida: Ruta seleccionada per l'usuari (Presentació)

Nom: *OpenTraceOrMXML()*
Responsabilitats: Demanar al sistema que es vol obrir un fitxer MXML o de Traces. El sistema haurà de demanar les opcions per a transformar-lo en Sistema de Transicions, així com la ubicació del fitxer.
Excepcions:
- Si l'usuari cancel·la l'operació no es modificarà l'estat intern del sistema.
- Si el fitxer no és un fitxer vàlid, el sistema mostra un error i no modifica el seu estat intern.

Precondicions:
Postcondicions:
- El sistema demana l'ubiació del fitxer i les opcions per a transformar-lo en Sistema de Transicions (Presentació).
- El sistema transforma l'entrada en un Sistema de Transicions i el carrega (Dades).

Sortida:

Nom: *SelectConvertToTSOptions()*
Responsabilitats: Demanar a l'usuari que especifiqui les opcions per a convertir un arxiu en Sistema de Transicions.
Excepcions:
- Si l'usuari cancel·la, no es retornarà cap opció.

Precondicions:
Postcondicions:
Sortida: Opcions a tenir en compte per a convertir un model a Sistema de Transicions (Presentació).

Nom: *ExportTS()*
Responsabilitats: Demanar al sistema l'exportació del Sistema de Transicions. El sistema demanarà on es vol guardar aquest fitxer.
Excepcions:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Precondicions:
- Hi ha un Sistema de Transicions carregat.

Postcondicions:
- El sistema demana la ubicació per a guardar el fitxer (Prestació).
- El sistema guarda el Sistema de Transicions en el format text correponent (Dades).

Sortida:

Nom: *ExportPetriNet(petriNet)*
Responsabilitats: Demanar al sistema l'exportació de la Xarxa de Petri indicada. El sistema demanarà on es vol guardar aquest fitxer.
Excepcions:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Precondicions:
- Hi ha alguna Xarxa de Petri generada.

Postcondicions:
- El sistema demana la ubicació per guardar el fitxer (Prestació).
- El sistema guarda la Xarxa de Petri en el format text correponent (Dades).

Sortida:

Nom: *ExportComposedPetriNet()*
Responsabilitats: Demanar al sistema l'exportació de la Xarxa de Petri composada. El sistema demanarà on es vol guardar aquest fitxer.
Excepcions:
- Si el fitxer no es pot escriure o bé no hi ha espai al disc, el sistema ho indica i no desa l'arxiu.

Precondicions:
- Hi ha alguna Xarxa de Petri composada.

Postcondicions:
- El sistema demana la ubicació per guardar el fitxer (Prestació).
- El sistema guarda Xarxa de Petri en el format text correponent (Dades).

Sortida:

4.3.2.4 Contractes de les visualitzacions dels models

Nom: *ChangeTSLayout(layout)*
Responsabilitats: Demanar al sistema canviar el layout de la visualització del Sistema de Transicions pel layout indicat.
Excepcions:
Precondicions:
- Hi ha un Sistema de Transicions carregat.

Postcondicions:
- El sistema canvia el layout dels elements del Sistema de Transicions (Presentació).

Sortida:

Nom: *ChangePetriNetLayout(layout)*
Responsabilitats: Demanar al sistema canviar el layout de la visualització de les Xarxes de Petri pel layout indicat.
Excepcions:
Precondicions:
- Hi ha alguna Xarxa de Petri generada.

Postcondicions:
- El sistema canvia el layout de les Xarxes de Petri intermitges (Presentació).

Sortida:

Nom: *ChangeComposedPetriNet(layout)*
Responsabilitats: Demanar al sistema canviar el layout de la visualització de la Xarxes de Petri Composada pel layout indicat.
Excepcions:
Precondicions:
- Hi ha una Xarxa de Petri composada.

Postcondicions:
- El sistema canvia el layout de la Xarxa de Petri composada (Presentació).

Sortida:

4.4 Persistència de dades

En aquest projecte, tot el que fa referència a la persistència de les dades està fortament lligat a l'ús dels programes *Genet* i *PComp*. Tan *Genet* com *PComp* són aplicacions pensades per ser usades de forma aïllada. Conseqüència d'això és el fet de que per especificar una entrada s'hagi d'especificar la ruta a un fitxer que contingui l'esmentada entrada. Aquesta és la raó per la qual *GenetGUI* ha d'emmagatzemar els diversos models en fitxers abans d'utilitzar alguna d'aquestes aplicacions com part del seu procés.

Aquests dos programes també imposen certes limitacions a l'hora de determinar el format dels diferents fitxers. De fet, no són només aquests dos programas qui limiten, sinó que també ho fan els formats usats en el camp del *Process Mining*.

A continuació es detallen els diferents formats usats per *GenetGUI*.

4.4.1 Format de les Xarxes de Petri

El format en que es codifiquen les Xarxes de Petri ve determinat pel format de sortida de *Genet* i pel format requerit per a l'entrada de *PComp*. De fet, aquesta codificació és la mateixa en ambdós casos, i una de les mes usades a l'hora d'especificar Xarxes de Petri. En el Codi4.1 es pot veure un exemple il·lustratiu d'aquest format.

Codi 4.1: Format d'una Xarxa de Petri

```
#Example of a Petri Net
.outputs a b x y z
.graph
p4 a
p1 b
p5 b(2)
p6 b
p2 x(2)
p3 y
p0 z
a p0
a p2
a p3
x p5
y p1
z p6
.capacity p0=3 p1=3 p2=4 p3=3 p4=3 p5=3 p6=3
.marking { p4 p2=3 }
.end
```

El significat dels diferents camps usats en la codificació de Xarxes de Petri és el següent:

.outputs Especifica els events del sistema, és a dir, les *Transitions* de la Xarxa de Petri. A l'hora de fer servir *PComp* per composar Xarxes de Petri, un mateix event no pot aparèixer com a *output* en dues xarxes. Per solucionar això pot fer-se ús del camp *.input* que si que ho permet.

.graph Tipus de model. En el cas d'una Xarxa de Petri és *.graph*.

.marking Determina el marcatge inicial de la xarxa, és a dir, els tokens inicials. Si no s'especifica cap valor vol dir que el nombre de tokens és un.

.capacity Indica el nombre de tokens màxim que pot contenir un determinat *Place*.

.end Determina el final de l'especificació de la Xarxa de Petri.

x y La resta de línies indiquen un arc de la Xarxa de Petri. Aquest arc pot ser de la forma *Transitions-Place* o viceversa. Aquest arc pot tenir un pes, cosa que serà indicada entre parèntesi.

La Figura 4.4 mostra la representació gràfica de la Xarxa de Petri de l'exemple.

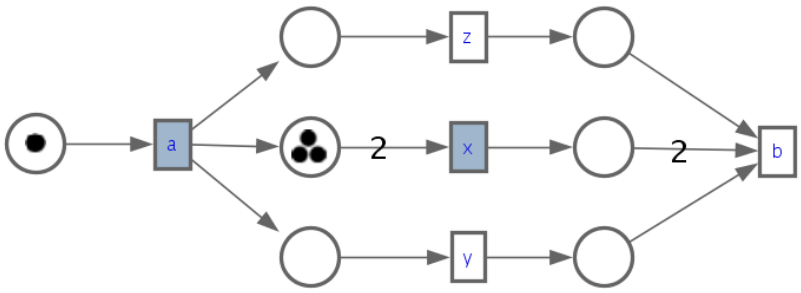


Figura 4.4: Xarxa de Petri del codi

4.4.2 Format dels Sistemes de Transicions

El format del Sistemes de Transicions ve determinat pel format que accepta *Genet* com a entrada. Un exemple d'aquesta codificació es pot veure en el Codi4.2

Codi 4.2: Format d'un Sistema de Transicions

```
# Example of a Transition System
.model example
.outputs a b c d e
.state graph
s0 a s1
s1 x s2
s1 y s3
s1 z s4
s2 y s5
s2 z s7
s3 x s5
s3 z s6
s4 y s6
s4 x s7
s5 z s8
s6 x s8
s7 y s8
s8 b s9
.marking {s0}
.end
```

El significat dels diferents camps usats en la codificació de Sistemes de Transicions és el següent:

.model Nom del Sistema de Transicions. El camp *.model* és opcional.

.outputs Especifica els events.

.state Tipus de model. En el cas d'un Sistema de Transicions és *graph*.

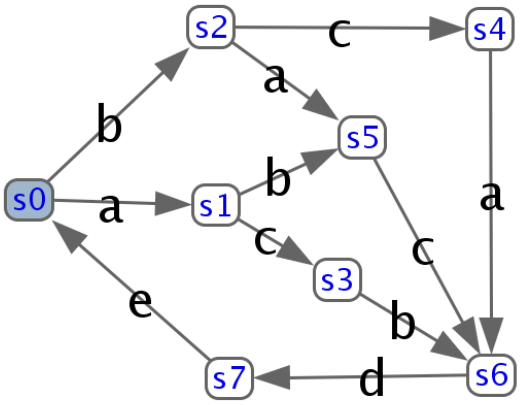


Figura 4.5: Sistema de Transicions del codi

.marking Indica l'estat inicial.

.end Determina el final de l'especificació del Sistema de Transicions.

x y z La resta de línies indiquen un arc del Sistema de Transicions. El primer element és l'estat origen de l'arc, seguit de l'event i finalment de l'estat destí.

La Figura 4.5 mostra la representació gràfica del Sistema de Transicions de l'exemple.

4.4.3 Format de les Traces

A diferència del format de les Xarxes de Petri i dels Sistemes de Transicions, el format de les Traces no ve determinat per l'ús de cap programa. Tot i així, a l'hora de definir aquest format s'ha procurat fet servir una codificació fàcil i intuïtiva, i que recordés a la usada normalment per a definir aquests tipus de models. En el Codi4.3 és pot veure un exemple il·lustratiu d'aquest format.

Codi 4.3: Format d'un conjunt de Traces

```
# Example of a set of Traces
x v
b c j k
v w x y z
z x b k
```

En aquest format cada línia del fitxer representa una Traça diferent. Cada un dels events de la Traça està separat per un espai. Events amb un mateix nom, fan referència al mateix event.

4.4.4 Format dels Logs d'Events

A l'hora de definir un format per als Logs d'Events, es va decidir triar un format ja existent, per tal de maximitzar la usabilitat del programa. El format elegit ha estat *Mining XML*. MXML és

un dels formats més utilitzats i és el format usat per *ProM*[20], l'eina més potent actualment en el camp del *Process Mining*. A tot això cal sumar-li el fet de ser una extensió del *XML*, amb totes els avantatges que això comporta, i també el fet que existeix una eina capaç de convertir un format de log determinat en MXML, d'una forma ràpida i fàcil: *ProM Import*[6].

En el Codi4.4 és pot veure un exemple il·lustratiu de l'estructura dels camps més important d'aquest format.

Codi 4.4: Format d'un Log d'Events

```
<WorkflowLog>
  <Process id="0" description="a42f0n00_5.trc">
    <ProcessInstance id="0" description="">
      <AuditTrailEntry>
        <WorkflowModelElement>S</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a1</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a31</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="1" description="">
      <AuditTrailEntry>
        <WorkflowModelElement>S</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a1</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a34</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
    </ProcessInstance>
  </Process>
</WorkflowLog>
```

Les etiquetes més importants són:

Process Defineix un nou procés.

ProcessInstance Defineix una nova instància del procés.

AuditTrailEntry Defineix un nou event dins de l'instància del procés. L'ordre dels events ve determinat per l'ordre en el que apareixen en l'arxiu.

WorkflowModelElement Nom de l'event.

EventType Tipus de l'event. Els tipus més importants són *complete*, *start* i *schedule*.

A [2] es pot trobar l'especificació completa del format MXML.

4.5 Presentació

Per la capa de presentació s'ha decidit fer ús d'un altre patró arquitectònic molt usat en aquestes situacions: el *Patró Model-Vista-Controlador* (MVC).

Aquest patró consisteix en dividir una aplicació, o una part d'ella, en tres capes diferents:

Vista Mostrar la informació a l'usuari.

Model Representació de la informació amb la que opera l'aplicació.

Controlador Rep els events de l'usuari i invoca els canvis en el model.

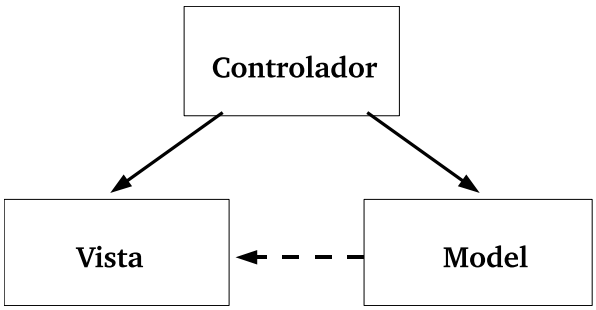


Figura 4.6: Patró arquitectònic "Model-Vista-Controlador" (MVC)

En el nostre cas, el model ve definit pel domini de l'aplicació. Només serà necessària la creació d'un controlador que s'encarregui de gestionar les accions que vol fer l'usuari i les transmeti al domini, i de les vistes per mostrar la informació per pantalla. Fent ús d'aquest patró aconseguirem un nivell més d'abstracció, fent el codi més fàcil de mantenir.

4.5.1 Disseny extern

Un cop explicat el disseny intern de la presentació, explicarem lleugerament el disseny extern de les vistes.

No cal perdre de vista que estem fabricant una eina visual, i que per tant ha de ser intuïtiva, agradable a la vista, i fàcil d'usar.

El disseny extern de l'aplicació pot ser descomposat en 5 parts clarament diferenciades, on cadascuna d'elles correspon a una etapa del procés de descobrir models formals a partir de logs d'un sistema.

En la figura 4.7 es pot veure la pantalla usada per l'aplicació per especificar les diferents opcions que seran usades per *Genet* per generar les Xarxes de Petri. Una d'aquestes opcions és

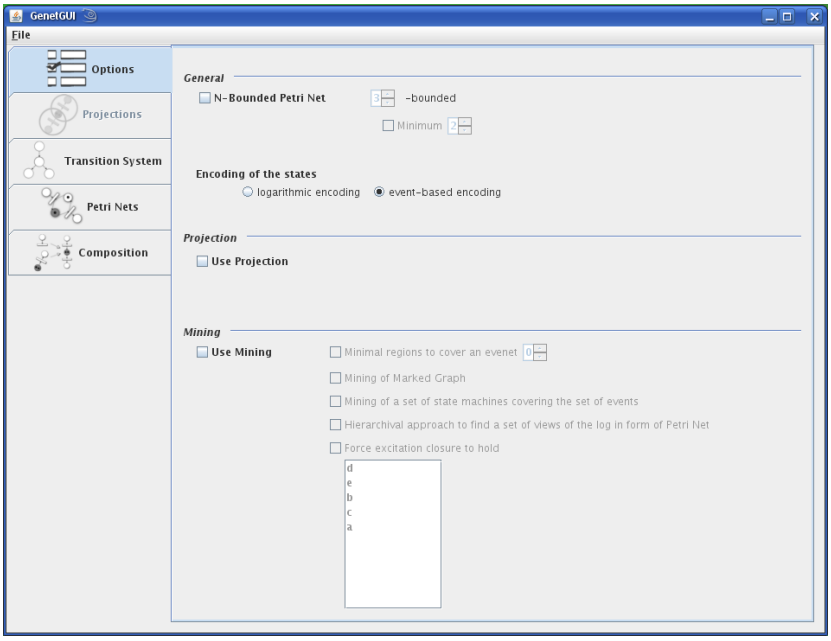


Figura 4.7: Captura de la pantalla d'Opcions

la possibilitat d'usar Projeccions. A l'activar aquesta opció, el sistema et dona accés a l'apartat encarregat de gestionar les Projeccions.

A la figura 4.8 es pot veure una captura de la secció de les Projeccions. Allà es poden crear o eliminar Projeccions, gestionar els events que les composen, i fer-ne ús dels algoritmes de Pre-Projecció.

El sistema també permet mostrar de forma gràfica el Sistema de Transicions amb el qual s'està treballant en aquell moment. La visualització està feta usant *Prefuse*, i permet moure els elements usant el ratolí i fer *Zoom* o *Pan*. La figura 4.9 mostra aquesta visualització.

Juntament amb la visualització dels Sistemes de Transicions, el sistema també permet visualitzar les Xarxes de Petri. La visualització també està feta usant *Prefuse*. A més del *Zoom*, el *Pan*, i de poder moure els elements, el sistema permet disparar Transicions activades i usa diferents colors per ajudar a la comprensió de les xarxes. El color de les Transicions activades es diferent de les que no ho són, i els *Places* d'entrada i sortida de la Transicions sobre la que està el cursor són verds i vermells respectivament. Un exemple de visualització pot veure's en la figura 4.10

L'última secció del programa és la que permet composar diferents Xarxes de Petri. Tal i com es pot veure en la figura 4.11, la visualització de la xarxa composada és molt semblant a les xarxes intermèdies.

Una descripció més detallada sobre el funcionament de l'eina pot trobar-se en el manual d'usuari de l'aplicació que apareix en la secció apèndix d'aquesta memòria.

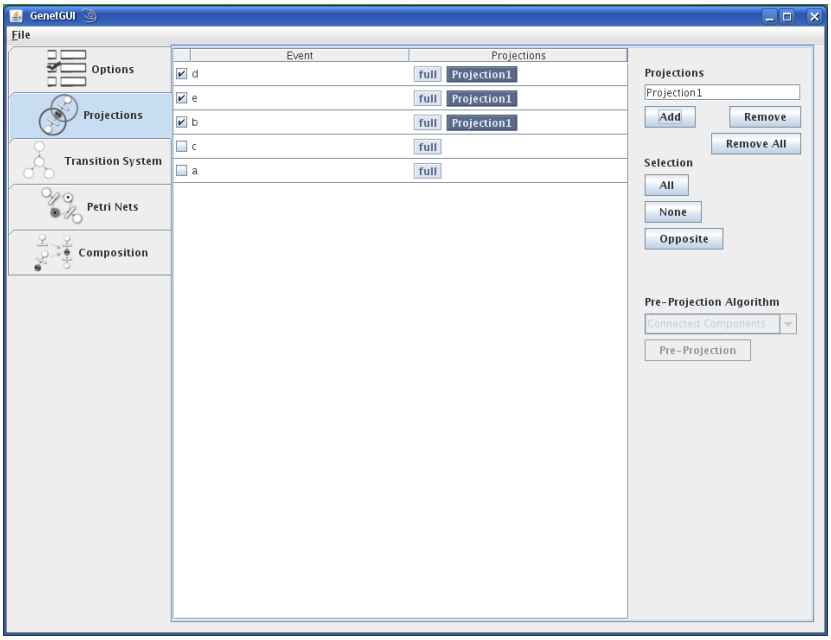


Figura 4.8: Captura de la pantalla de Projeccions

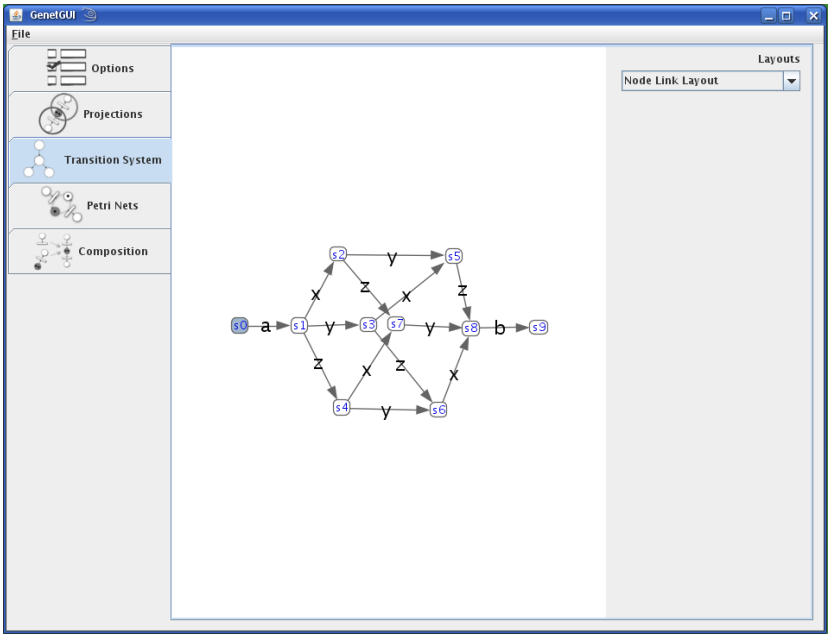


Figura 4.9: Captura de la pantalla del Sistema de Transicions

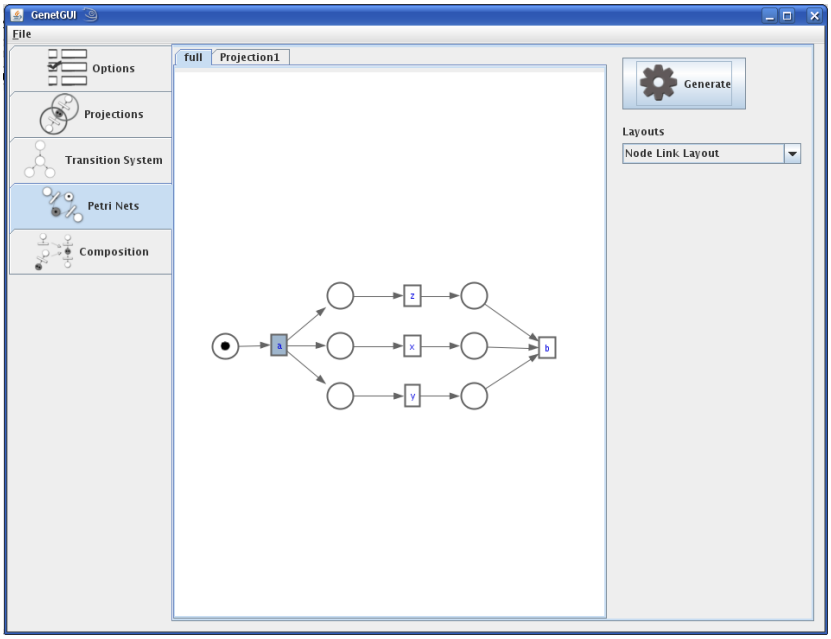


Figura 4.10: Captura de la pantalla de les Xarxes de Petri

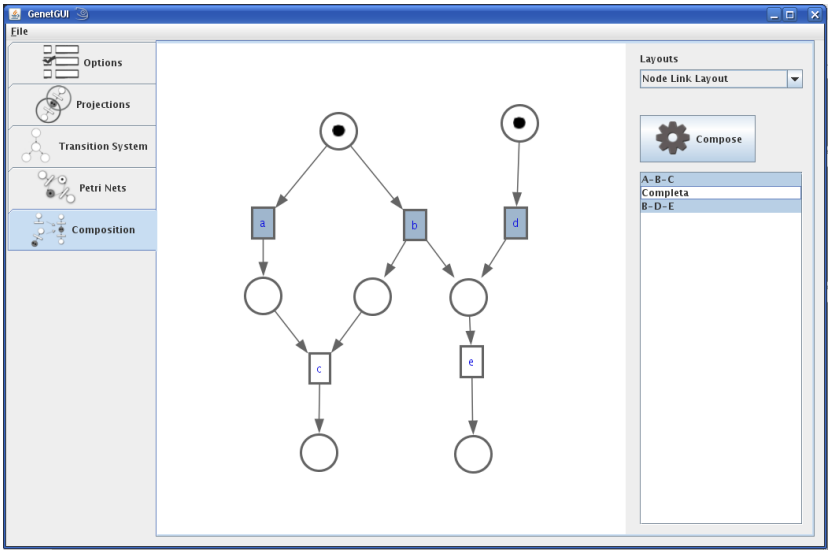


Figura 4.11: Captura de la pantalla de la Xarxa de Petri Composada

5 Algoritmes

En aquesta secció es detallen els principals algoritmes utilitzats en el projecte. Per poder veure quins tipus d'algoritmes hi ha, quines són les seves funcions i quin lloc ocupen en tot el procés, farem ús d'un exemple.

> Una determinada empresa ha decidit fer ús de les tècniques del Process Mining per analitzar els seus processos. Uns pocs d'aquests processos estan representats usant Sistemes de Transicions (el model acceptat com entrada de Genet). Però en canvi, la resta està modelada usant Traces o Logs d'Events, com a resultat dels programes que utilitza l'empresa per a gestionar-los.

Per solucionar això, el GenetGUI incorpora els anomenats *Conversors*, parts del programa encarregades de transformar una determinada entrada en un Sistema de Transicions. Els Conversors es detallen en la secció 5.1.

> Una vegada ja han aconseguit introduir dins el sistema la informació, el responsable de l'empresa ha decidit analitzar un dels processos. En concret ha decidit analitzar un procés força complex, que involucra molts events. És per això que ha decidit projectar grups d'events per analitzar-los de forma independent. Però degut a la complexitat i la mida del procés, aquesta esdevé una tasca costosa, pesada i llarga.

Facilitar la realització de les projeccions és l'objectiu dels anomenats *Algoritmes de Pre-Projecció*, una sèrie d'algoritmes pensats per fer les projeccions de forma automàtica i heurística. Els Algoritmes de Pre-Projecció es detallen en la secció 5.2.

5.1 Conversors

L'eina *Genet* ha estat dissenyada per poder generar Xarxes de Petri a partir només de Sistemes de Transicions. Però hi ha moments en els que pot resultar útil generar aquestes xarxes a partir d'altres fonts, com poden ser determinades traces d'events o logs de sistemes codificats en diferents formats. És aquí on es manifesta la importància dels anomenats Conversors.

Conversor Un Conversor és una aplicació encarregada de transformar la entrada, modelitzada d'una determinada manera, en un altre model.

L'objectiu final és acabar transformant l'entrada en un Sistema de Transicions, i poder ser així processat per *Genet*.

A continuació es detallen els diferents conversors dels que consta *GenetGUI*, així com els algoritmes usats per portar a terme aquestes transformacions. Cal a dir que el sistema està pensat per poder afegir nous conversors posteriorment d'una manera fàcil i ràpida.

5.1.1 Conversor de Logs d'Events a Traces

El primer conversor inclòs en *GenetGUI* és el que permet transformar Logs d'Events, codificats en MXML, en conjunts de Traces. Com es pot veure a simple vista, aquest conversor no transforma l'entrada en un Sistema de Transicions. Però precisament la missió d'un dels altres conversors de *GenetGUI* (que es veurà més endavant) serà la de transformar aquestes Traces en Sistemes de Transicions.

Tan el format dels Logs d'Events com el de les Traces es poden veure en els apartats 4.4.4 i 4.4.3 respectivament.

Per a dur a terme aquesta conversió, s'ha decidit no fer servir tota la potència del format MXML i optar per fer una abstracció d'alguns dels seus camps. En concret s'ha decidit només tenir en compte els events un cop finalitzats, és a dir, que el seu tipus sigui *complete*. A més no s'han tingut en compte altres camps opcionals del MXML. Cal a dir que en un futur es poden desenvolupar altres conversors que tinguin en compte el tipus de l'event, i el tractin d'una altra forma per a dur a terme la transformació.

Tenint en compte aquesta simplificació, i la definició de Logs d'Events i Traces explicada al apartat 2.2, la conversió d'un a l'altre és bastant directa. L'Algoritme 5.1 mostra com dur a terme la conversió.

Algoritme 5.1 Convertir Logs d'Events en Traces

```
repeat
  tag := readTag()
  if (tag = "ProcessInstance") then
    trace := newTrace()
  else if (tag = "AuditTrailEntry") then
    if (readEventType = "complete") then
      nom := readWorkflowModelElement()
      event := newEvent(nom)
      addEvent(trace,event)
    end if
  end if
until endLog()
```

Com es pot veure en l'algoritme, es crea una nova traça per cada *ProcessInstance* trobat. Després, s'afegeix un nou event a aquesta traça per cada *AuditTrailEntry*, sempre i quan aquest event ja s'hagi completat.

5.1.2 Conversor de Traces a Sistemes de Transicions

En el conversor anterior s'ha vist com transformar un Log d'Events en Traces. Ara es veurà com transformar les Traces en Sistemes de Transicions[18]. Aquestes Traces poden ser resul-

tat del conversor anterior o bé ser especificades directament per l'usuari.

Tan el format de les Traces com el dels Sistemes de Transicions es poden veure en els apartats 4.4.3 i 4.4.2 respectivament.

Per explicar aquest procés de conversió, primer cal explicar i definir una sèrie de conceptes, per després poder descriure l'algoritme utilitzat.

Quan es treballa amb Traces, és molt fàcil obtenir la llista d'events que succeeixen al executar-se un determinat procés. En canvi, deduir en quin estat esta el procés a partir d'aquestes Traces no és tan trivial. Hi ha casos en els que cada event que apareix en les Traces conté informació necessària per definir l'estat. Però normalment això no es així. És per això que cal definir l'estat. Per fer això hi ha dos propostes:

Passat L'estat es construeix tenint en compte els events succeïts abans del punt actual en la Traça. També anomenat història o prefix de la Traça.

Futur L'estat es construeix tenint en compte els events que manquen per succeir des del punt actual. També anomenat postfix.

La Figura 5.1 mostra el diagrama amb les diferents propostes per deduir l'estat a partir de la Traça.

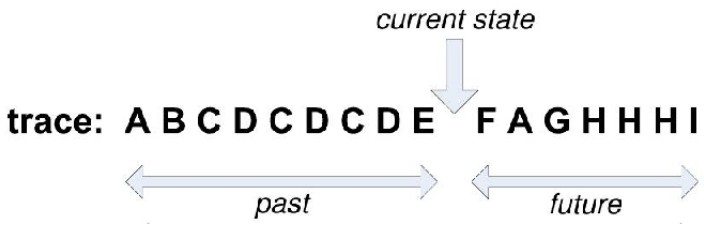


Figura 5.1: Propostes per a construir els estats

A més d'usar el passat o el futur, hi ha una sèrie de *extensions* que es poden aplicar a l'hora de generar els estats. En concret, aquestes *extensions* són:

Horitzó Màxim En comptes de considerar tots els events del passat (o del futur,segons sigui el cas), només es consideren els *h* events més pròxims a la posició actual.

Paràmetre Q El paràmetre Q defineix la forma de representar la informació del passat (o del futur, segons sigui el cas). En concret, hi ha dues formes de represtar aquesta informació:

- *Seqüència*: L'ordre dels events es té en compte a l'hora de crear l'estat.
- *Conjunt*: Només es té en compte si els events han aparegut.

Es pot usar una d'aquestes *extensions*, o l'altra, o fins i tot combinar les dues per generar els estats. Si no s'especifica res, es suposa que l'horitzó és *infinit*, i que l'estructura utilitzada és una *seqüència*.

Un cop exposats aquests conceptes només manca definir què és un *Sistema de Transicions* en el àmbit d'aquest conversor. Per poder fer això, primer cal definir una sèrie de funcions:

state(σ,*k*) Donada una traça σ i un número k que indica el nombre d'events de σ que ja han succeït, produeix una representació r d'un estat. Aquesta representació pot ser una seqüència o un conjunt, segons s'hagi especificat, i els events que la composen venen determinats per les opcions triades (l'horitzó màxim, i si es té en compte el passat o el futur).

transition(s,e,t) Donat un estat origen s, un estat destí t i un event, retorna la transició amb origen s, destí t i etiqueta e.

σ(*k*) Donada una traça σ i un número k, retorna l'event que ocupa la posició k en la traça σ.

Definició 5.1 (Sistema de Transicions) *Sigui A el conjunt d'events que apareixen a les traces, i L el conjunt de traces. Definim un Sistema de Transicions com la tuple (S,E,T,S^{start},S^{end}) on:*

- Espai d'estats*:* $S = \{state(\sigma, k) | \sigma \in L \wedge 0 \leq k \leq |\sigma|\}$
- Conjunt d'events*:* $E = A$
- Relació de transició*:*$T \subseteq S \times E \times S$ *on* $T = \{(state(\sigma, k), \sigma(k+1), state(\sigma, k+1)) | \sigma \in L \wedge 0 \leq k < |\sigma|\}$
- Conjunt d'estats inicials*:* $S^{start} = \{state(\sigma, 0) | \sigma \in L\}$
- Conjunt d'estats finals*:* $S^{end} = \{state(\sigma, |\sigma|) | \sigma \in L\}$

Un cop definits tot els elements, ja es pot mostrar l'algoritme usat per crear un Sistema de Transicions a partir d'un conjunt de Traces. L'Algoritme 5.2 mostra com dur a terme la conversió.

En la primera part de l'algoritme es creen tots els estats i els events que hi haurà en l'espai d'estats i l'espai d'events respectivament. El segon pas es crear les transicions entre els estats. Després es fa la tria de quins seran els estats inicials i quins els estats finals. L'últim pas de l'algoritme ve determinat pel fet que *Genet* només accepta un únic estat inicial. Per solucionar això es crea un estat fantasma nou fora de l'espai d'estats, i es creen transicions entre aquest estat i els estats inicials. Finalment s'afegeix aquest estat conjunt d'estats inicials i s'eliminen la resta.

La Figura 5.2 mostra un exemple del funcionament d'aquest conversor. En aquest exemple es mostra el funcionament segons si es tria utilitzar el *passat* o el *futur* per generar els estats. A més, l'horitzó màxim considerat és *infinit* i s'usen *seqüències* per als estats.

Val a dir que aquest algoritme pot ser modificat amb diverses *extensions* addicionals. Algunes d'aquests afegits són: considerar el passat i el futur al mateix temps, usar altres estructures per representar els estats com podrien ser *bosses* i tenir en compte altra informació que puguin contenir les Traces (el tipus d'event o qui l'ha executat,...).

5.2 Algoritmes de Pre-Projecció

En aquesta secció es detallen els diferents algoritmes de Pre-Projecció dels que consta *Genet-GUI*. Per a fer això el primer pas és definir què és un algoritme de Pre-Projecció.

Algoritme 5.2 Convertir Traces en un Sistema de Transicions

```
{Create the States and Events}
for all trace σ : Traces do
  for 0 ≤ k ≤ |σ| do
    if not exist(state(σ,k)) then
      createState(state(σ,k))
    end if
    if not exist(σ(k)) then
      createEvent(σ(k))
    end if
  end for
end for
{Create the Transitions}
for all trace σ : Traces do
  for 0 ≤ k ≤ |σ| do
    if not exist( transition( state(σ,k-1),σ(k),state(σ,k-1))) then
      createTransition( state(σ,k-1),σ(k),state(σ,k-1))
    end if
  end for
end for
{Create Initial and Final States}
for all trace σ : Traces do
  addInitialState(state(σ,0))
  addFinalState(state(σ,|σ|))
end for
{Set just one Initial State}
if size(initialStates) > 1 then
  ghostState := newState()
  ghostEvent := newEvent()
  for all state s : InitialStates do
    createTransition(ghostState,ghostEvent,s)
  end for
  removeAllInitialStates()
  addInitialState(ghostState)
end if
```

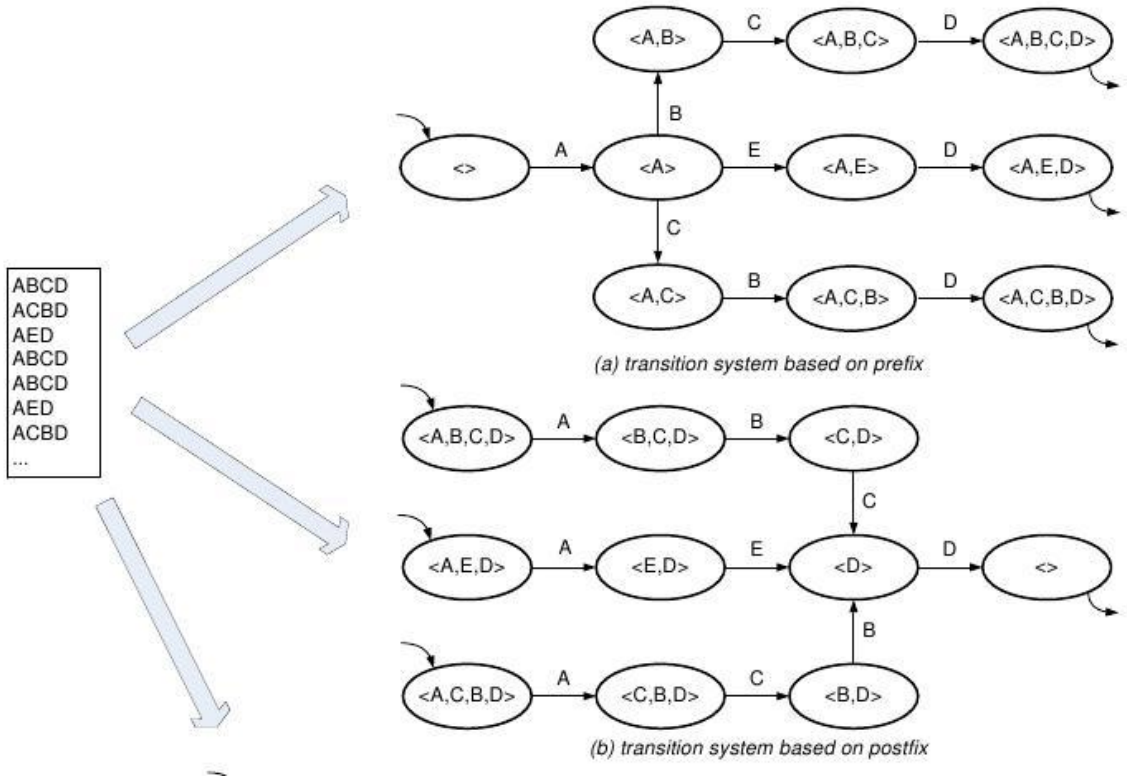


Figura 5.2: Exemple de construcció d'un Sistema de Transicions

Algoritme de Pre-Projecció Un algoritme de Pre-Projecció és un procés encarregat de realitzar una o varies Projeccions sobre els events d'un Sistema de Transicions d'una manera *heurística* i *automàtica*. Les Projeccions resultants poden ser moficiades posteriorment de forma manual per l'usuari. És per això que es coneixen com *Pre*-Projeccions.

La utilitat d'aquest tipus d'algoritmes salta a la vista. Donats Sistemes de Transicions que donen lloc a Xarxes de Petri grans i gairebé intractables, es fa necessària una partició en trossos més petits i manejables. Però aquesta partició dista molt de ser trivial de fer, afegint-hi el fet de ser farragosa i lenta. I és aquí on entren en joc els Algoritmes de Pre-Projecció, que s'encarreguen de fer les particions, les quals després poden ser refinades per l'usuari de forma ràpida i fàcil.

El sistema està pensat per poder afegir nous algoritmes de Pre-Projecció amb facilitat. Això dóna la llibertat crear nous algoritmes que utilitzin la informació que creguin necessària en cada cas per realitzar les Projeccions. Així es poden implementar algoritmes específics per a casos més concrets, o d'altres més generals vàlids per a tot tipus de casos.

Tot i això *GenetGUI* ja proporciona 3 algoritmes de Pre-Projecció. Aquests algoritmes es detallen a continuació, així com la informació necessària que fan servir per generar les projeccions: el *Graf de Causalitats*.

5.2.1 Graf de Causalitats

Els 3 algoritmes de Pre-Projecció creats inicialment per a *GenetGUI* fan ús de la mateixa informació per crear les Projeccions. Es tracta del *Graf de Causalitats*. En aquest apartat es preten explicar que es el Graf de Causalitat, la seva utilitat i com generar-lo. Cal indicar que el Graf de Causalitats és un concepte lligat a les Traces/Logs d'Events. És per això que la seva generació i l'ús dels algoritmes de pre-projecció que l'usen està limitat a entradas d'aquests tipus. Per poder definir el Graf de Causalitats primer s'han de definir una sèrie de conceptes:

Definició 5.2 (Relacions d'ordenació en logs) *Siguin E el conjunt d'events, L un log tal L* $\in \mathcal{P}(E^*)$ [1]*, i a,b* $\in$ *E llavors:*

- $a >_L b$ *si i només si hi ha una traça* $\sigma = t_1 t_2 t_3 \dots t_{n-1} i i \in \{1, \dots, n-2\}$ *tal que* $\sigma \in L$ *i* $t_i = a$ *i* $t_{i+1} = b$*. Aquesta relació es coneix com Relació Directe.*
- $a \rightarrow_L b$ *si i només si* $a >_L b$ *i* $b \not>_L a$*. Aquesta relació es coneix com Relació de Causalitat.*

Tal com s'ha definit just a dalt, hi ha una relació de causalitat entre dos events si aquests dos events apareixen un darrera de l'altre en alguna de les traces, però no apareixen en l'ordre oposat en cap traça del log. Un cop definida aquesta relació, el Graf de Causalitats es defineix com el graf format pels events i les relacions de causalitat entre ells.

Graf de Causalitats El Graf de Causalitats és un graf no dirigit, on els vertexs són els events, i existeix una aresta entre dos events si hi ha una relació de causalitat entre aquests dos events, sense importar el sentit d'aquesta relació.

Un cop definits els conceptes, la raó de l'ús del Graf de Causalitats per realitzar les Projeccions no esdevé molt difícil de veure. És lògic pensar que events que estan fortament relacionats de forma causal entre ells han d'aparèixer a la mateixa Projecció, per així poder veure el funcionament conjunt. En canvi, events que no estan relacionats entre si, no interactuaran entre ells de cap manera, i per tant, poden ser separats i observats independentment.

L'Algoritme 5.3 mostra el procés empleat per generar el Graf de Causalitats[19].

Algoritme 5.3 Generar Graf de Causalitats

```
for all Traces do
  for 0 ≤ k < numEvents() do
    addDirectRelation(event(k),event(k+1))
  end for
end for
for all directRelation di : DirectRelations do
  op := opposed(di)
  if not isDirectRelation(op) then
    addCasualRelation(di)
  end if
end for
```

El primer pas de l'algoritme és establir totes les relacions directes. Un cop establertes, es comprova si donada una relació (x,y) del conjunt de relacions directes, el seu oposat (y,x)

[1] Conjunt de les parts (o powerset) $\mathcal{P}(B) = \{X | X \subseteq B\}$

està també en el conjunt. Si no hi és, vol dir que hi ha una relació causal entre aquests dos events. Aquest procés es repeteix per a totes les relacions del conjunt de relacions directes.

La Figura 5.3 mostra el Graf de Causalitats de l'exemple de Traces vist en l'apartat 4.4.3.

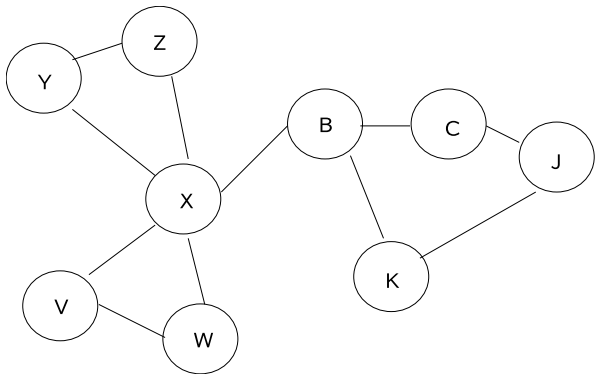


Figura 5.3: Graf de Causalitats

5.2.2 Algoritme de components connexos

El primer algoritme a explicar és l'*Algoritme de components connexos*. La idea d'aquest algoritme és que els grups d'events que no estan relacionats amb altres grups poden ser analitzats de forma separada. Això es tradueix en separar el Graf de Causalitat en els diferents components connexos que el composen, i crear una projecció per a cada component.

El problema de descomposar un graf en els seus components connexos és un problema clàssic de la Teoria de Grafs [17], i fàcil de resoldre amb cost lineal. L'Algoritme 5.4 mostra la solució emprada per *GenetGUI*.

Algoritme 5.4 Algoritme de components connexos

```
for all v: Vertexs do
  if not isInComponent(v) then
    comp := newComponent();
    vertexSet := BreadthFirstSearch(v)
    addVertexs(comp,vertexSet)
  end if
end for
```

L'algoritme itera sobre tots els vèrtexs del graf, és a dir, els events. Si algun vèrtex no pertany a cap component, es fa una *Cerca en Amplada* de tots els vèrtex connectats a ell, i es crea un nou component amb els vèrtexs trobats.

La Figura 5.4 mostra com quedaria el Graf de Causalitats usat com exemple, si s'apliqués l'Algoritme de Components Connexes.

5.2.3 Algoritme de Cliques

En aquest algoritme la idea és semblant: buscar events que estiguin relacionats, per mostrar el seu comportament en una mateixa projecció. Però a diferència de l'algoritme de compo-

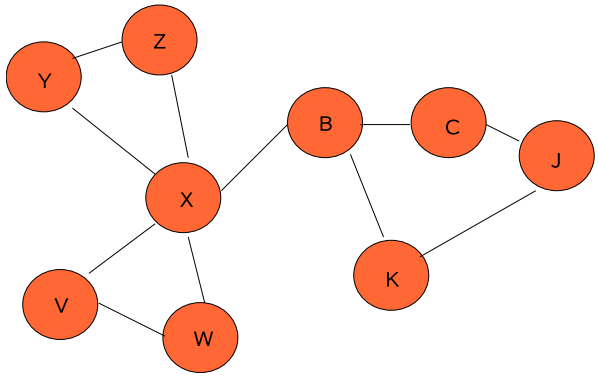


Figura 5.4: Graf de Causalitats amb l'Algoritme de Components Connexos

nents connexos, en aquest algoritme es preten buscar components que estiguin més fortament relacionats. Per fer això, s'utilitza el concepte de *Clique*.

Clique Un clique en un graf no dirigit G és un conjunt de vèrtex V tals que per a tot parell de vèrtexs de V, existeix una aresta que les connecta. En altres paraules, un clique és un subgraf en que cada vèrtex està conectat a cadascun dels altres vèrtex del subgraf.

La Figura 5.5 mostra un exemple de clique. Malgrat un clique de només dos vèrtex és un clique vàlid, en aquest algoritme no s'han tingut en compte aquest tipus de cliques, ja que són massa simples, no aporten gaire informació i compliquen la situació a l'hora de fer projeccions.

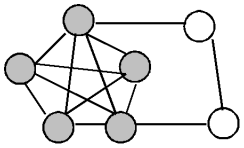


Figura 5.5: Exemple de clique

Un cop definit el concepte de clique és facil veure que events que pertanyin al mateix clique, son events fortament connectats, i per tant pot resultar interessants veure com interactuen junts.

Per realitzar el procés de trobar els cliques d'un graf s'ha fet servir una implementació del *Algoritme de detecció de cliques de Bron-Kerbosch*, tal i com es descriu en [16], proporcionada per la llibreria *JGraphT*.

La Figura 5.6 mostra com quedaria el Graf de Causalitats usat com exemple, si s'apliqués l'Algoritme de Cliques. L'event X pertany a les dues projeccions, una per cada clique: (y-z-x) i (v-w-x).

5.2.4 Algoritme de Clustering

Finalment, l'últim algoritme de Pre-Projecció del que consta *GenetGUI* també pretén agrupar els events segons les seves relacions entre ells. Però en aquest cas, aquesta partició és vol fer

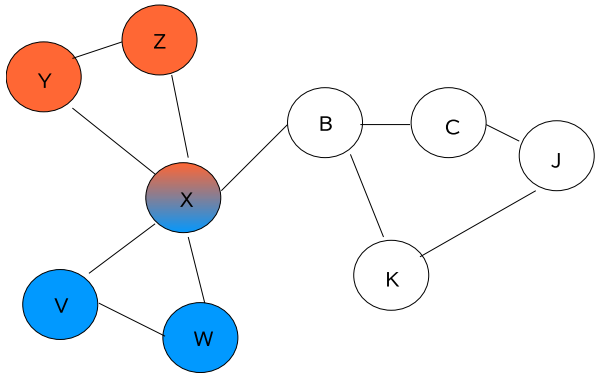


Figura 5.6: Graf de Causalitats amb l'Algoritme de Cliques

fent servir un *Algoritme de Clustering*. Un Algoritme de Clustering [11] és un algoritme que agrupa conjunts d'events més o menys pròxims, anomenats *clústers*,i que separa aquest grups dels altres, amb els que no tenen tanta relació.

La Figura 5.7 il·lustra el concepte de *clústers*.

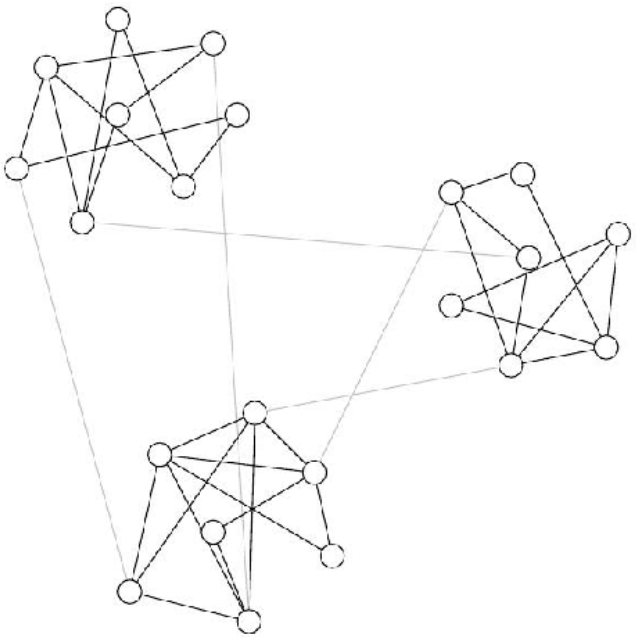


Figura 5.7: Exemple de clústers

Per dur a terme aquesta descomposició en clústers, l'algoritme es basa en el concepte de *Betweenness* d'una aresta.

Betweenness La Betweenness d'una aresta es pot definir com la freqüència en la que aquesta aresta apareix en el camí més curt entre tots els parelles de nodes.

L'Algoritme 5.5 mostra com s'utilitza aquest concepte a l'hora de crear els clústers. Es calcula la *betweenness* de totes les arestes[8], i es treu la que tingui una *betweenness* més alta, i per tan, la aresta més propensa a ser la que uneix els diferents grups. Aquest procés es realitza tants cops com arestes es vulguin treure per formar els clústers.

Algoritme 5.5 Algoritme de Clustering

for $1 \leq k \leq numEdgesToRemove$ **do**
  computeBetweenness()
  edge := getHighestBetweenness()
  removeEdge(edge)
end for

La Figura 5.8 mostra com quedaria el Graf de Causalitats usat com exemple, si s'apliqués l'Algoritme de Clustering només treient una aresta.

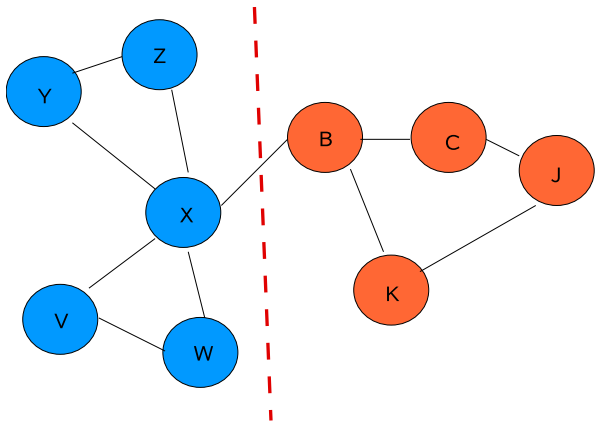


Figura 5.8: Graf de Causalitats amb l'Algoritme de Clustering

6
Exemple d'ús

En aquest capítol es mostra, a traves d'un exemple pràctic, el funcionament de l'aplicació, pas per pas. Aquest exemple esta basat en l'anàlisi dels logs d'un sistema web. En concret, es tracta del sistema encarregat d'allotjar i administrar el *Racó de la FIB*, la intranet per a estudiants i professors de la Facultat d'Informàtica de Barcelona. El servidor web Apache encarregat del Racó, així com la majoria de servidors web del mercat, disposen dels elements necessaris per a exportar els logs de forma fàcil i ràpida.

Després d'aconseguir el log, el següent pas va ser manipular-lo per poder ser usat per GenetGUI. En concret, i donada la enorme dimensió del log, es va decidir considerar només els events que fessin referència al *fòrum* i als *perfils*, dues de les seccions del Racó. Un cop fet això, es va procedir a transformar el format del log (un format propi dels servidors Apache) en un format acceptat per GenetGUI (MXML en aquest cas). Per a fer això, es va fer ús de l'eina *ProMImport* [6], i del seu plugin per al format Apache. Finalment, i un cop el log va estar en format MXML, es va decidir traduir els noms dels events (noms molt llargs, al tractar-se d'un àmbit web) en noms més curts i més fàcils de tractar.

Un cop es va tenir el log en el format adequat, es va obrir amb l'aplicació. Per a obrir-lo es va utilitzar l'opció *Open Trace/Log*, que permet especificar els parametres de la transformació del log en Sistema de Transicions. Els parametres triats van ser *Passat*, *Seqüència*, i un *Horitzó Màxim* d'infinit. El Sistema de Transicions resultant es mostra en la Figura 6.2.

Un cop carregat el Sistema de Transicions, es va decidir fer servir l'opció de *Mining* de Genet per a generar les Xarxes de Petri corresponents. A més, es va decidir fer servir l'opció de *Projecció*. En concret, es va triar fer dues projeccions: una corresponent als events relacionats amb el *fòrum* i una altre amb els events de la secció *perfils* del Racó (Figura 6.3).

Tenint en compte les opcions triades i les projeccions creades, es van generar dues Xarxes de Petri, una per cada projecció, tal com mostren les Figures 6.4 i 6.5. L'eina va permetre veure la evolució d'aquestes xarxes de forma separada, i analitzar així el seu comportament.

Un cop generades les Xarxes de Petri, es va decidir composar-les, forman així una única xarxa, tal com mostra la Figura 6.6. D'aquesta manera es va poder analitzar la evolució del *fòrum* i dels *perfils* de forma conjunta.

Finalment, es va decidir exportar, en mode text, el Sistema de Transicions i les tres Xarxes de Petri (les dues intermèdies i la composada), per així poder ser processades posteriorment per altres eines.

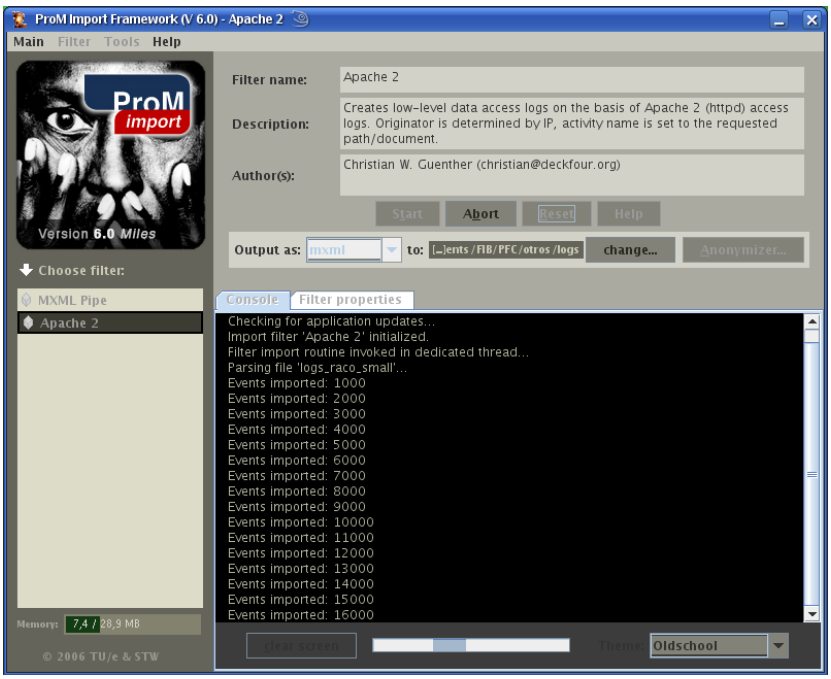


Figura 6.1: Convertir el log a MXML amb ProMImport

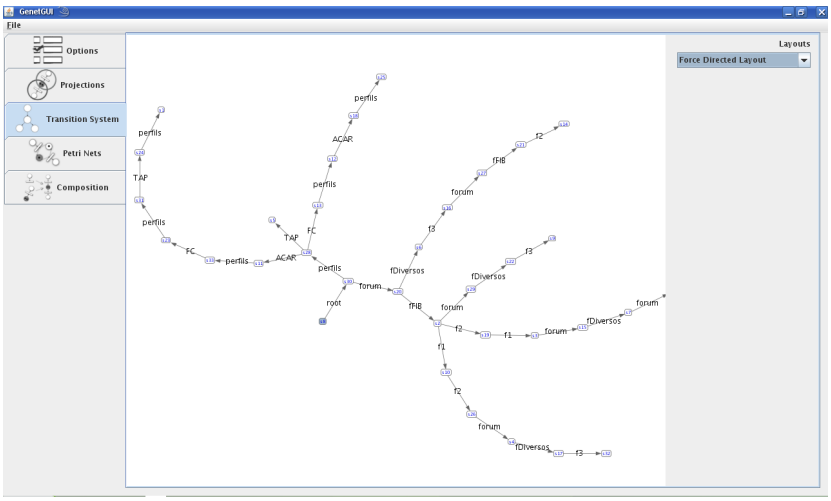


Figura 6.2: Sistema de Transicions del Racó

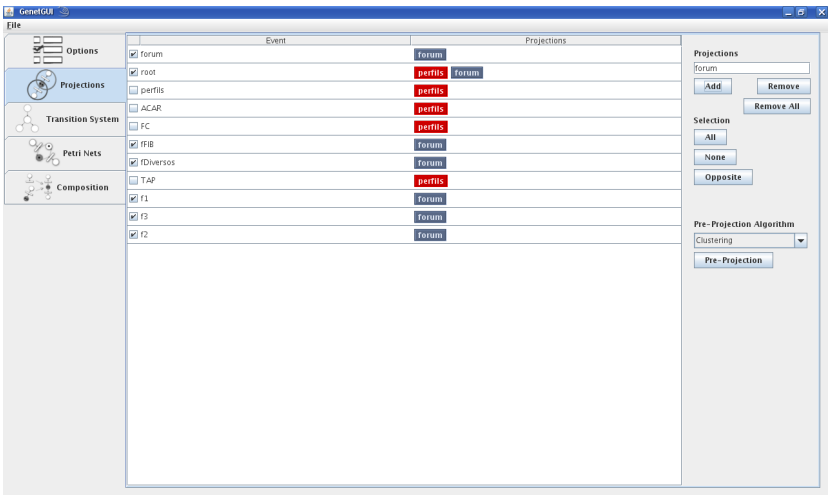


Figura 6.3: Projeccions sobre el Racó

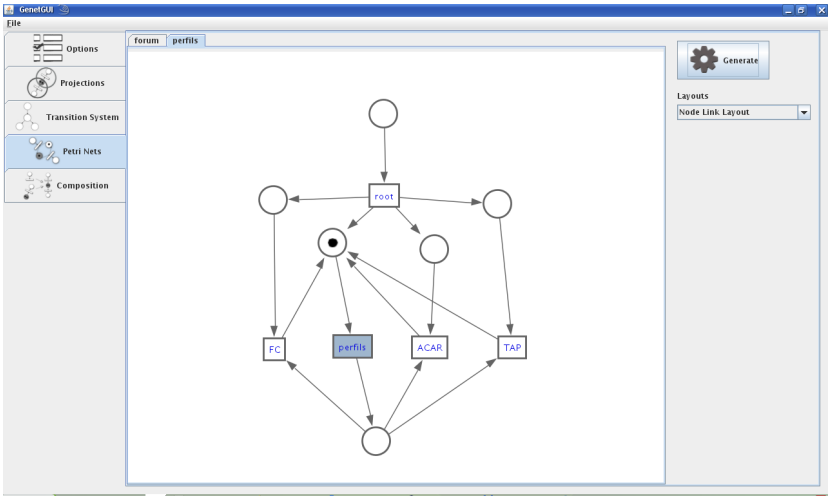


Figura 6.4: Xarxa de Petri Perfils

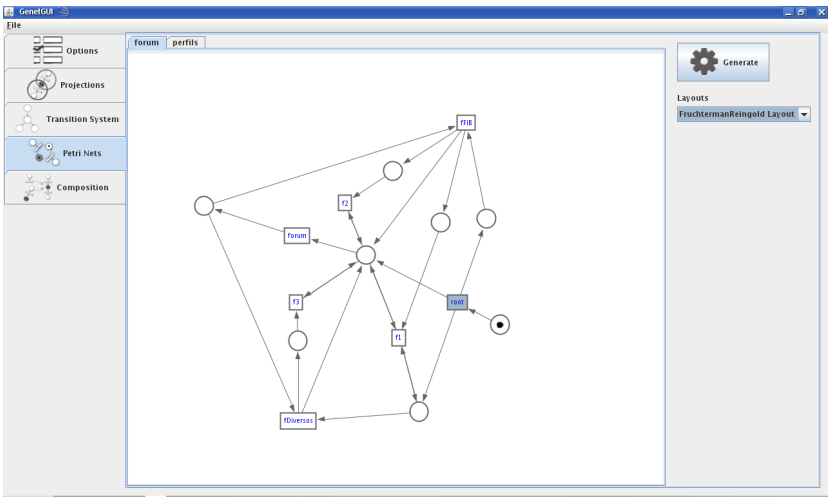


Figura 6.5: Xarxa de Petri Fòrum

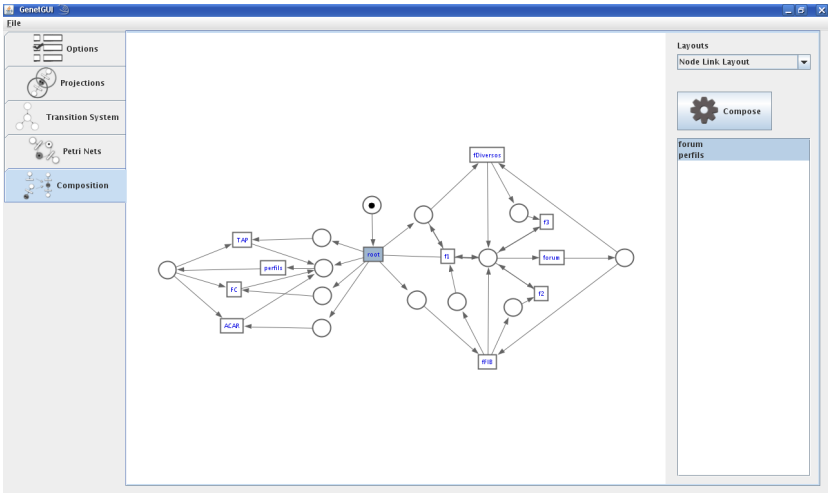


Figura 6.6: Xarxa de Petri composada del Racó

7 Planificació i anàlisi econòmic del projecte

En aquest capítol es fa un anàlisi de la planificació temporal del projecte. S'analitzen les diferents etapes principals en el desenvolupament del projecte, fent una comparativa entre el temps que s'havia previst emprar en la consecució de l'etapa i el temps que finalment s'ha necessitat per completar-la.

A continuació es fa també un anàlisi de les despeses econòmiques del projecte, tant pel que fa a hores de treball personal com les eines utilitzades.

7.1 Planificació temporal

El desenvolupament del projecte s'ha dut a terme en diverses etapes. Inicialment es va fer una previsió temporal de les hores necessàries per a cada una d'aquestes etapes, previsió que no s'ha complert per la dificultat que representa fer aquesta mena de planificació la primera vegada que es du a terme un projecte d'aquesta magnitud. Les etapes en les que s'ha dividit el projecte són les següents:

Etapa 1. Breu estudi teòric sobre els conceptes a utilitzar

Etapa 2. Anàlisi de requeriments.

Etapa 3. Especificació i Disseny.

Etapa 4. Estudi de les possibles tecnologies a utilitzar per la implementació.

Etapa 5. Implementació.

Etapa 6. Proves.

Etapa 7. Documentació

A la taula 7.1 es pot observar la planificació de cada una de les etapes. S'hi poden veure les hores que es van planificar inicialment i també, les hores que realment han calgut per completar-les.

Com podem observar hi ha una desviació total de 95 hores a l'engròs. Aquesta desviació correspon bàsicament a l'etapa d'implementació, a la que s'han dedicat 100 hores més de les inicialment planificades. La causa d'aquest desviament és la clara subestimació que es va

Etapa	Hores planificades	Hores reals
1	15	20
2	5	10
3	160	120
4	35	60
5	350	450
6	50	40
7	150	160
Total	750	860

Taula 7.1: Planificació temporal

fer inicialment de la quantitat de coses a fer, i en segon lloc, de la sèrie de dificultats que s'han trobat per la manca d'experiència en la programació de projectes d'aquesta magnitud. L'esforç dedicat a les tecnologies usades per la implementació també ha superat el previst, arribant fins i tot a doblar les hores previstes inicialment.

En canvi és curiós veure com, en les etapes referents a l'Enginyeria del Software, l'especificació i el disseny, s'han dedicat menys hores de les inicialment planificades.

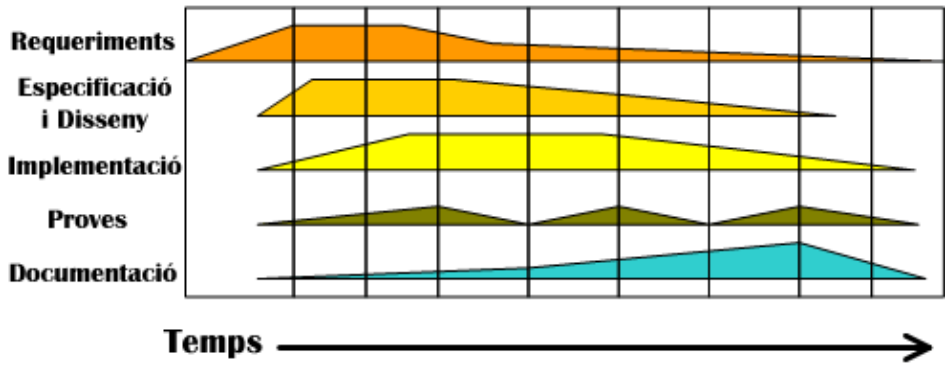


Figura 7.1: Distribució de les etapes en el temps

Pel que fa a la distribució de les etapes en el temps, aquestes s'estenen al llarg de tot el desenvolupament tal i com mostra la Figura 7.1. Això es produeix degut a l'ús d'un cicle de vida *iteratiu i incremental* pel projecte, refinant-lo iteració a iteració, tal i com s'explica en l'apartat 3.

7.2 Anàlisi econòmic

Per fer l'anàlisi econòmic tindrem en compte dos factors:

- Les hores de treball.
- Eines utilitzades.

Pel que fa a les hores de treball, s'han agrupat diverses etapes en funció del nivell de valoració laboral que aquestes tenen. Així doncs, separarem les hores d'analista del que serien les hores de programador. A la categoria d'anàlisi hi posarem les etapes 1, 2, 3 i 7. Les etapes 4, 5 i 6 seran considerades hores de programador. La taula 7.1 mostra la planificació temporal.

Etapa	Hores	Preu per hora	Preu total
1	20	36€	720€
2	10	36€	360€
3	120	36€	4320€
4	60	24€	1440€
5	450	24€	10800€
6	40	24€	960€
7	160	36€	5760€
Total	24360€		

Taula 7.2: Cost de les hores de treball

Pel que fa a les eines, la taula 7.3 mostra les despeses necessàries del projecte. Al ser un projecte purament software, les despeses per hardware no són significatives. A més, al centrar els esforços en l'ús de software lliure pel seu desenvolupament, els gastos en llicencies no tenen cap repercussió.

Eina	Preu
Ordinador domèstic	1000€
Bibliografia i Documentació	gratuït
Sistema operatiu Linux	gratuït
Kit de desenvolupament de software de Java	gratuït
Software d'enginyeria del software Umbrello	gratuït
Genet [1]	gratuït
PComp [3]	gratuït
Altres llibreries	gratuït
Sistema de control de versions Subversion	gratuït
IDE Netbeans	gratuït
Total	1000€

Taula 7.3: Cost de les eines utilitzades

Tenint en compte les hores de treball i el cost de les eines, el cost total del projecte és de 25360€.

7.2.1 Beca de Col·laboració

El 10 de desembre de 2008 a aquest projecte se li va atorgar la *Beca de Col·laboració COLAB 2008* que concedeix l'*Agència de Gestió d'Ajuts Universitaris i de Recerca (AGAUR)*. Aquest

ajut, pensat per al desenvolupament de projectes en l'ambit universitari, va constar d'un únic pagament de 3000€.

7.2.2 Presentació en un fòrum internacional

Aquest projecte estarà representat a la *9th International Conference on Application of Concurrency to System Design 2009* que tindrà lloc a Augsburg (Alemania) del 1 al 3 de juliol. GenetGUI formarà part de l'exposició de l'eina *Genet* que es farà en aquesta conferencia.

8 Conclusions

8.1 Objectius assolits

Al finalitzar aquest projecte, i analitzar el resultat, podem arribar a la conclusió que tots els objectius que s'havien definit al principi del projecte han estat assolits.

Per una banda s'ha aconseguit l'objectiu d'implementar l'eina, i que aquesta consti de totes les funcionalitats definides i acordades amb el director del projecte.

A més de l'eina, també s'han complert tots els requisits relacionats amb la documentació, tan del projecte com de l'eina en sí. Resultat d'això ha estat la redacció de la memòria del projecte, que detalla tot el procés, i la redacció de la documentació propia del codi de l'eina. A més, i com a complement a aquesta documentació, s'han elaborat dos documents més: El primer és el manual d'usuari de l'aplicació que permetrà entendre el funcionament de l'eina. El segon consisteix en un tutorial sobre el funcionament de la llibreria *Prefuse* utilitzada en aquest projecte, i que pretén pal·liar parcialment la manca de documentació que té aquesta llibreria.

8.2 Treball futur

Tot i ser un projecte acabat, no és pas un projecte tancat. Encara resten obertes moltes possibilitats en les que treballar en el futur. A continuació s'esmenten algunes d'aquestes possibilitats:

8.2.1 Nous models d'entrada

Actualment el programa accepta com a entrada models especificats com a Sistemes de Transicions, Traces i Logs d'Events codificats en MXML.

Seria interessant que en un futur s'estudiés la possibilitat que el programa pogués acceptar nous models. De fet, *GenetGUI* està pensat per poder incloure nous conversors de forma fàcil. A més de models nous, també podria ser interessant incorporar conversors pels models ja existents, però usant un procés diferent. Dins d'aquesta categoria podrien estar conversors de traces que utilitzessin una informació diferent per derivar els estats, o conversors de fitxers MXML que utilitzessin alguns dels aspectes que aquest format ofereix.

8.2.2 Nous algoritmes de Pre-Projecció

El procés de fer les projeccions és un procés complex, i que depèn d'infinitats de factors. És per això que pot ser interessant també l'estudi de nous algoritmes que ajudin en aquesta tasca.

Aquests nous algoritmes de Pre-Projecció poden usar també el *Graf de Causalitats* com a font d'informació per a realitzar les projeccions. A més, es poden estudiar altres algoritmes que utilitzin una altra informació, que pot ser més adient en altres situacions.

8.2.3 Integració amb ProM

Com ja s'ha comentat en aquesta memòria, una de les eines més populars en el camp del *Process Mining* és *ProM*. ProM [5] és un *framework* de codi lliure que aglutina mecanismes pel *Process Mining* realitzats per persones de tot el món.

És per això que una de les coses a tenir en compte en el futur és la possibilitat d'integrar algunes parts de *GenetGUI* en *ProM*. O fins i tot integrar el propi *Genet*.

8.3 Conclusions

El balanç del projecte, contemplat en la seva totalitat, ha estat molt positiu.

D'una banda està el fet de realitzar la producció d'una aplicació software no trivial de principi a fi. Aquest fet m'ha permés aprofundir en els coneixements apresos durant la carrera pel que fa a les diferents etapes del desenvolupament de software: començant per l'anàlisi de requisits i l'especificació del sistema, passant pel seu disseny i la seva implementació i acabant amb el testeix i la documentació.

L'etapa d'implementació es mereix una menció a banda. Per la consecució d'aquesta etapa m'he vist obligat a aprendre tot una sèrie de conceptes nous, i molt diferents entre ells. Alguns d'aquests conceptes van des del parsejar documents fent ús de gramàtiques, fins al disseny de la part gràfica del projecte, tant de la interfície com de la visualització dels models.

També caldria destacar l'etapa de documentació, i en especial la redacció de la memòria del projecte. És veritat que hi ha diversos responsables d'assignatures que intenten reforçar positivament la redacció de documentació tècnica per part dels alumnes, i més des de l'arribada del nou marc europeu que pretén afavorir el desenvolupament de les anomenades "competències transversals o genèriques". Però tot i això no hi ha cap activitat realitzada durant la carrera que s'equipari en complexitat i extensió a la memòria d'un PFC. Això fa que els coneixements i habilitats adquirits durant la redacció d'aquest siguin molt valuoses de cara al futur, ja sigui en marc d'altres activitats acadèmiques o en el món laboral.

D'altra banda, a part de tots els coneixements pràctics de gestió i desenvolupament de projectes, aquest *PFC* també m'ha aportat quelcom més: l'atracció pel món del *Process Mining*. Ha estat durant la realització d'aquest projecte quan he tingut el meu primer contacte amb aquesta àrea de coneixement. I tot i que aquest contacte ha estat molt superficial, m'ha permés veure tot l'esforç de les persones que hi ha al darrera, i les grans oportunitats que aquest camp pot oferir. Això ha fet que em decidís a realitzar el *Doctorat en Computació*, i així poder seguir amb la recerca en aquesta àrea.

Bibliografia

[1] Genet web site. http://www.lsi.upc.edu/~jcarmona/genet.html.

[2] Mining XML specification. http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd.

[3] PComp web site. http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/pcomp/.

[4] Process Mining Web. http://www.processmining.org.

[5] ProM web site. http://prom.win.tue.nl/tools/prom/.

[6] ProMImport web site. http://sourceforge.net/projects/promimport.

[7] Jean Bovet and Terence Parr. Antlrworks: an antlr grammar development environment. *Softw., Pract. Exper.*, 38(12):1305–1332, 2008.

[8] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[9] Josep Carmona, Jordi Cortadella, and Michael Kishinevsky. Genet: a tool for the synthesis and mining of petri nets(tool paper). In *Ninth International Conference on Application of Concurrency to System Design (ACSD 2009), 1-3 July 2009, Augsburg, Germany*, 2009.

[10] Dolors Costal, Xavier Franch, M. Ribera Sancho, and Ernest Teniente. *Enginyeria del software Especificació*. Edicions UPC, 2005.

[11] Michelle Girvan and M. E. J. Newman. Community structure in social and biological networks. *PROC.NATL.ACAD.SCI.USA*, 99:7821, 2002.

[12] Christian W. Günther. Process Mining TV - Episode 1: What Is Process Mining? http://prom.win.tue.nl/pmtv/index.php?episode=0.

[13] Cristina Gómez, Enric Mayol, Antoni Olivé, and Ernest Teniente. *Enginyeria del software Disseny I*. Edicions UPC, 2003.

[14] R. S. Mans, Helen Schonenberg, Minseok Song, Wil M. P. van der Aalst, and Piet J. M. Bakker. Application of process mining in healthcare - a case study in a dutch hospital. In Ana L. N. Fred, Joaquim Filipe, and Hugo Gamboa, editors, *BIOSTEC (Selected Papers)*, volume 25 of *Communications in Computer and Information Science*, pages 425–438. Springer, 2008.

[15] Terence Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. The Pragmatic Bookshelf, Raleigh, 2007.

[16] R. Samudrala and J. Moult. A graph-theoretic algorithm for comparative modeling of protein structure. *Journal of molecular biology*, 279(1):287–302, May 1998.

[17] Steven S. Skiena. *The Algorithm Design Manual*. Springer-Verlag, 1998.

[18] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 2009.

[19] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[20] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The prom framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.

A

GenetGUI: User Manual

GenetGUI: User Manual

Jorge Muñoz Gama

Contents

1 About GenetGUI

GenetGUI is a program to help on Process Mining [3], techniques that allow for extracting information from event logs. For example, the audit trails of a workflow management system or the transaction logs of an enterprise resource planning system can be used to discover models describing processes, organizations, and products.

Specifically, GenetGUI allows to transform Event Logs or Transition Systems into Petri Nets, and display them. It was developed by Jorge Muñoz as part of the Degree Final Project "Process Mining: descobrint models formals a partir de logs d'un sistema" in the "Universitat Politecnica de Catalunya" (UPC). GenetGUI is based on *Genet* [1], a tool for the synthesis and mining of Petri nets from transition systems. The tool is based on the theory of regions.

2 Using of GenetGUI

Use GenetGUI is really easy. This is the ordinary workflow of the program:

Open a file Open a new model to work with. This model can be a Transition System, a Event Log or a set of Traces.

Visualize the Transition System See and interact with the Transition System generated from the file you has opened.

Set the options The options that will use Genet to transform the Transition System into Petri Nets.

Manage Projections Divide the Transition System in parts (or projections) to create several Petri Nets.

Generate Petri Nets Generate Petri Nets according to the options set and projections created before. You can also visualize the Petri Nets and interact with them.

Compose Petri Nets Compose several of the Petri Nets in a unique composed Petri Net, and visualize it.

Export models The textual description of the different models can be exported.

Obviously, from one step you can go back to a previous step to change or add something, repeating the process again. All the details of each step will be explained in the following sections.

3 Parts of GenetGUI

3.1 Open a file

The list of types of models that can be opened with GenetGUI is shown in the Table 1:

Model	Extension
Transition System	.sg
Traces	.tr
Event Log	.xml

Table 1: Files and their extensions supported

The formats to specify those models are explained in the Section 4. Two options of the menu can be used to Open a file:

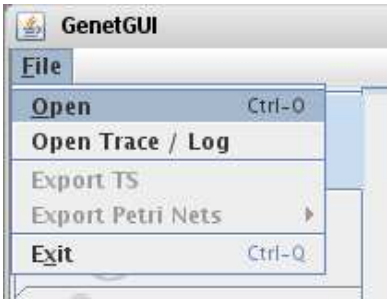


Figure 1: Open a file Menu

Open It can be used to Open everything: Transition Systems, Traces or Event Logs. But as far as Genet only works with Transition Systems, the Trace and Logs will be transformed in Transition Systems before open it. To Transform them, GenetGUI will use the options set in the system (the last options used)

Open Trace / Log It can be used to Open Traces or Logs, but choosing the options that we want to use to transform those Traces or Logs into Transition System. The details of those options and that transformation are explained in the Section 5

3.2 Visualize the Transition System

Once a file has been opened, the corresponding Transition System can be graphically visualised using GenetGUI.

Several elements are added to help to visualize it:

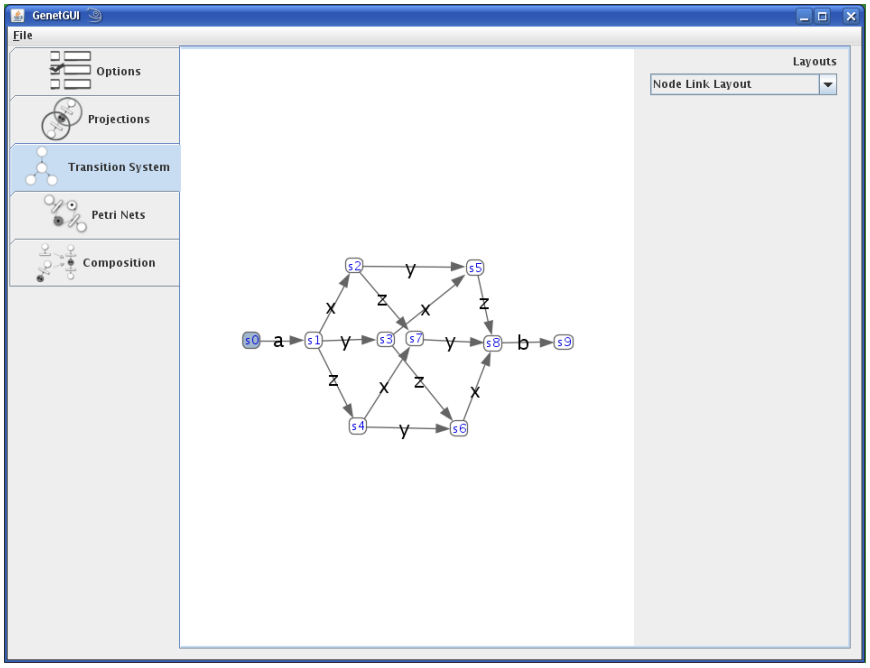


Figure 2: Transition System Screen

- Initial state in a different color.
- Change the layout of the elements of the Transition System.
- Color green and red for input and output states of the under-cursor state.
- Drag and Drop elements of the Transition System.
- Pan, Zoom and ZoomToFit
  - Pan: clicking on the background of a visualization with the left mouse button and then dragging.
  - Zoom: with the mouse wheel or pressing the right mouse button on the background of the visualization and dragging the mouse up or down.
  - ZoomToFit: clicking the right mouse button once, with no dragging.

3.3 Set Genet Options

GenetGUI uses Genet[1] to generate the Petri Nets. To understand the Genet options it is needed to explain more about Genet. The input of the

Genet is a Transition System specifying a behaviour. The tool derives a Petri net (PN) that has some relation with the input transition system. Depending on the usage of the tool, two possible outcomes are possible:

Synthesis The PN derived has a reachability graph bisimilar to the input transition system.

Mining the set of possible traces in the PN derived is a superset of the ones possible in the input transition system. Usually, the PN generated with the mining option is less complex (in terms of places, arcs and transitions) than the one derived for synthesis.

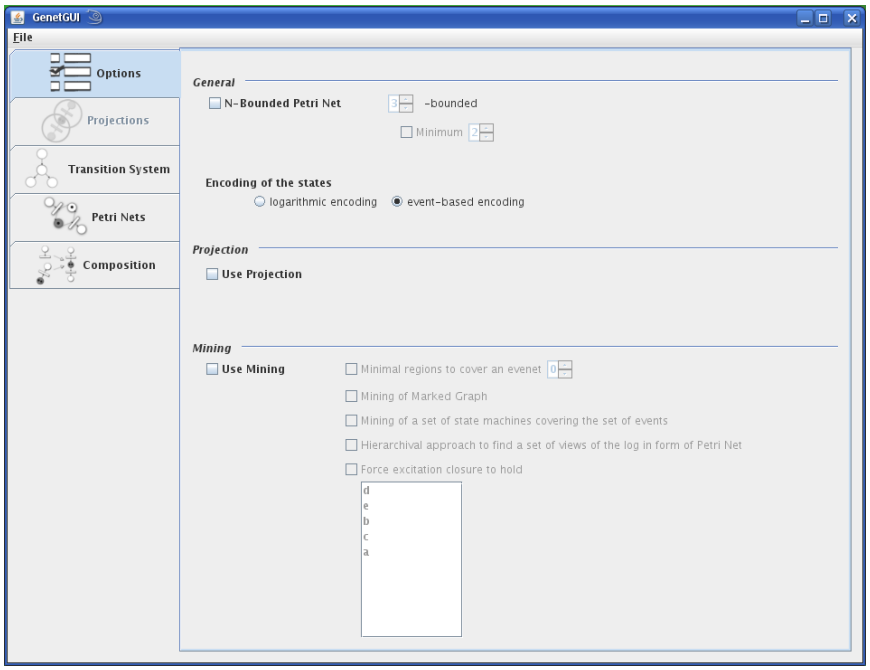


Figure 3: Options Screen

The options of Genet can be classified in *General Options (Synthesis & Mining)* or *Mining Options*:

GENERAL OPTIONS (SYNTHESIS & MINING)

- To look for an n-bounded Petri net
- Start the search for regions with bound $n \geq k \geq s$, where n has been assigned by previous option . This may speed up the search.
- encoding of the states using event-based encoding or logarithmic encoding.

- Enable the Projection Tab to projects the initial transition system into the events selected by the user.

MINING OPTIONS

- To apply petri net mining (approximation of the excitation closure).
- Number of minimal regions to cover an event. This might be useful when the set of minimal regions is large and therefore one can weaken the condition to cover an event with few regions.
- Mining of a marked graph. This is experimental, and not very debugged option (self-loops still not treated properly).
- Mining of a set of state machines (1-bounded Petri nets exhibiting no concurrency) covering the set of events.
- Hierarchical approach to find a set of views of the log in form of Petri nets. By using some causal dependencies between the events, it collects groups of events that are tightly related and projects the log on each group. This option must be used in combination with the synthesis, mining or conservative components derivation.
- By splitting necessary labels, forces excitation closure to hold for the events selected.

3.4 Manage Projections

If the "Use Projection" option is enable, the program allows the access to the Projection Tab, where the Projections can be managed. A Projection is a partition of the initial Transition System in groups of events, to create several Petri Nets, one for each projection. Projecting makes possible to analyse large Transition Systems.

The "Projection Tab" allows you to:

- Add events to a projection: Select the events, write the name of the projection and click "Add".
- Remove events from a projection: Select the events, write the name of the projections and click "Remove". To remove a single event from a projection, double-clicking over the "label" in the event projections cell is also possible.
- Remove all the projections: Just click "Remove All"
- Use Pre-Projection: Pre-Projection is a set of algorithms to create projections in a automatic and heuristic way. To use them just select an algorithm and click "Pre-Projection". The Pre-Projection Algorithms

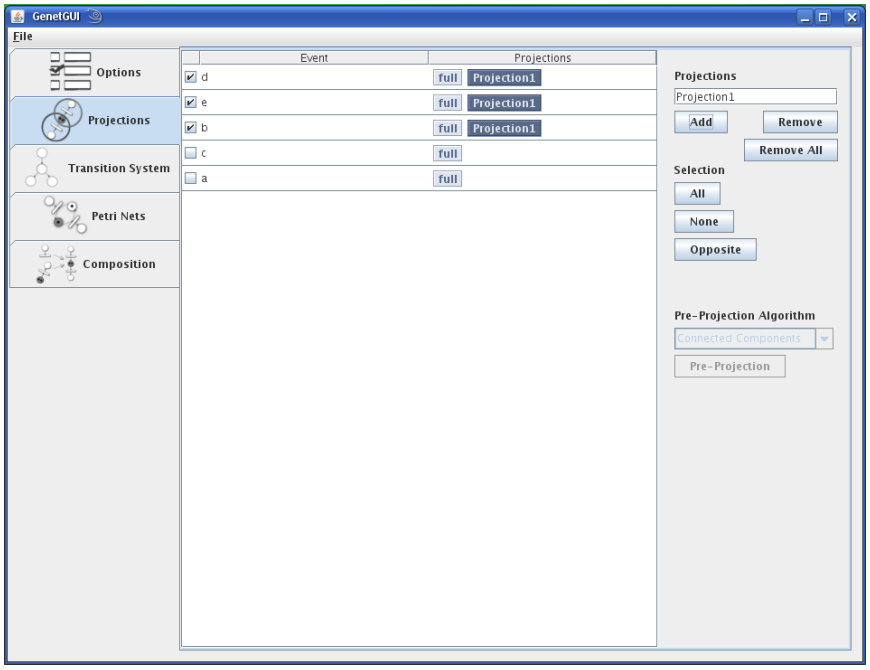


Figure 4: Projections Screen

are only available for Traces/Logs files with the CausalGraph option enabled (see section 5).

To make easier the selection of events to add or remove from a projection, GenetGUI provides the buttons of:

- Select ALL the events.
- Select NONE of the events.
- Select the OPPOSITE events, the enable events become disable, and vice versa.

3.5 Generate Petri Nets

After opening a file and setting all the options and projections, the Petri Nets can be generated. To do this just click "Generate" button. All the resulting Petri Nets can be displayed, one on each tab.

Several elements are added to help to visualize:

- Enable Transition are displayed in a different color.
- Fire Enable Transitions double-clicking over it.

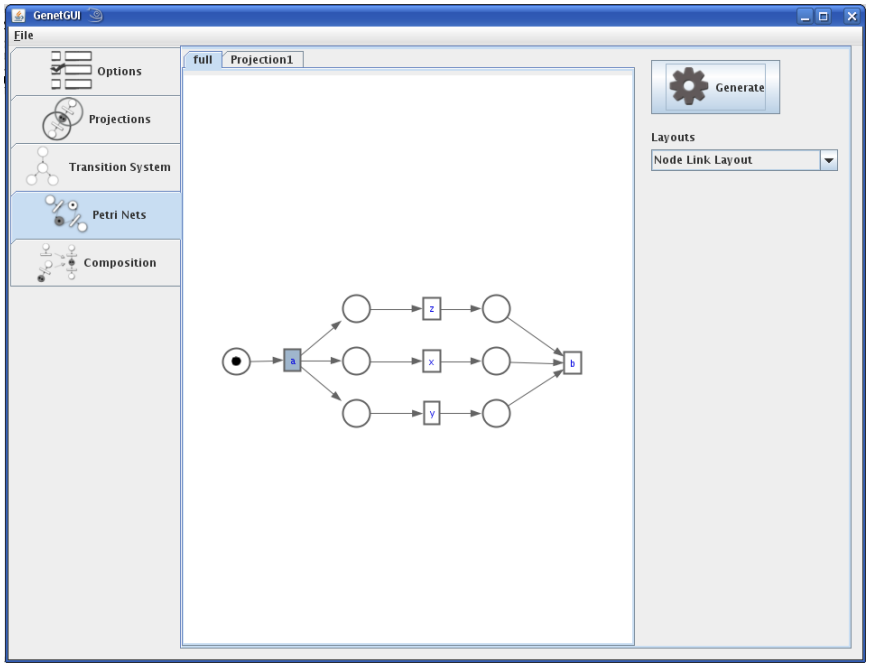


Figure 5: Petri Net Display Screen

- Change the layout of the elements of the Petri Net.
- Color green and red for input and output elements of the under-cursor element.
- Drag and Drop elements of the Petri Net.
- Pan, Zoom and ZoomToFit
  - Pan: clicking on the background of a visualization with the left mouse button and then dragging.
  - Zoom: with the mouse wheel or pressing the right mouse button on the background of the visualization and dragging the mouse up or down.
  - ZoomToFit: clicking the right mouse button once, with no dragging.

3.6 Compose Petri Nets

After generating several Petri Nets, a set of them could be composed creating a unique composed Petri Net. To do this just select the Petri Nets to be composed, and click “Compose”. The composed Petri Net will be

displayed. The same elements to help in the visualization of the generated Petri Nets, are in the composed Petri Net display. GenetGUI uses *PComp* [2] to compose the Petri Nets.

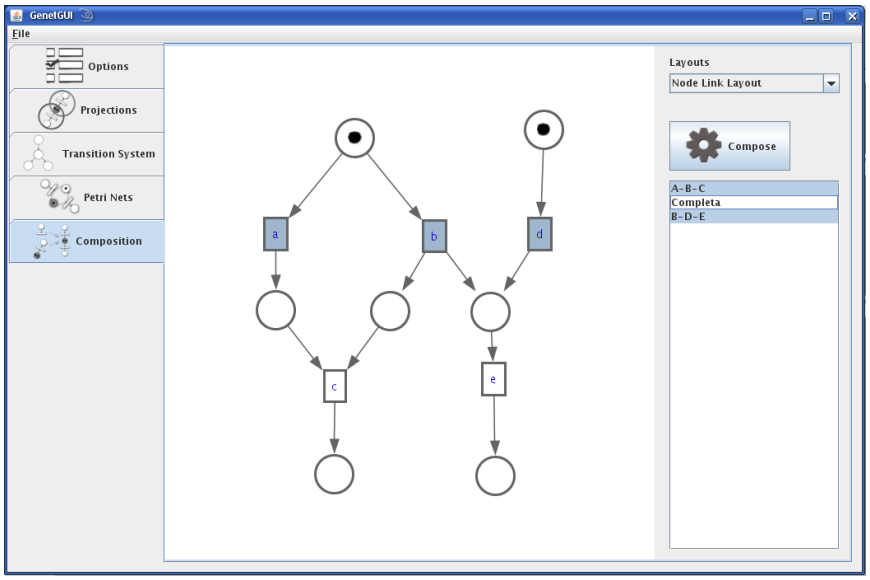


Figure 6: Petri Net Composition Display Screen

3.7 Export models

Finally, the textual description of all the models used in the process can be exported. To do this, just click the chosen button of the menu bar. The format of each textual description can be seen in the Section 4.

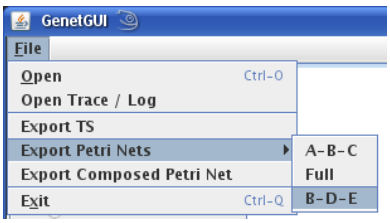


Figure 7: Export Menu

The models that can be exported are the following:

- The Transition System
- All the generated Petri Nets

- The composed Petri Net

4 Formats of GenetGUI

In this section are specified the format of the models description, used in GenetGUI.

4.1 Petri Nets format

The format used to encode the description of the Petri Nets and the composed Petri Net is determined by the Genet and PComp tools. Next there is an example of that format, and its explanation.

Listing 1: Petri Net format

```
#Example of a Petri Net
.outputs t1 t2 t3
.inputs t4
.graph
p5 t1(2)
p3 t1
p6 t2
p4 t3
p0 t4
t1 p5(2)
t2 p5
t2 p8
t3 p1
t3 p7
t4 p2
t4 p5
.capacity p0=3 p1=3 p2=3 p3=3 p4=3 p5=3 p6=3 p7=3 p8=3
.marking { p1 p2 p5=3 p7 p8 }
.end
```

.outputs/.inputs *Transitions* of the Petri Nets.

.graph Type of model, *.graph* in case it is a Petri Net.

.marking Initial token assignment of the *places*.

.capacity The maximal capacity for the *places* of the Petri net.

.end End of the Petri Net specification.

x y The other lines contains the definition of a (possibly weighted) *arc*. An arc can be from a transition to a place or vice versa. The arc can be weighted, as it is shown in the line "p5 t1(2)", representing that transition "e1" removes two tokens from place "p5".

4.2 Transition System format

The format used to encode the description of a Transition System is determined by input accepted by Genet. Next there is an example of that format, and its explanation.

Listing 2: Transition System format

```
# Example of a Transition System
.model example
.outputs a b c d e
.state graph
s0 a s1
s0 b s2
s1 b s5
s2 a s5
s1 c s3
s2 c s4
s5 c s6
s3 b s6
s4 a s6
s6 d s7
s7 e s0
.marking {s0}
.end
```

.model Name of the Transition System (optional).

.outputs *Events* of the Transition System.

.state Type of model, *graph* in case of Transition Systems.

.marking Initial *State*.

.end End of Transition System specification.

x y z The rest of lines contains the definition of a Transition System *arc* (source_state, event, target_state).

4.3 Traces format

The format for simple sets of Traces is really intuitive. Each Trace in a line. The events of the Trace are separated by white spaces. Next there is a example of that format:

Listing 3: Traces format

```
# Example of a set of Traces
```

```
a b c d e
b c
a c d e
a b c a a
b c j k
v w x y z
```

4.4 Event Logs format

The Event Logs' format is a bit different. The logs should be encoded using *Mining XML*. MXML is a variation of XML specification language, used for powerful Process Mining tools like *ProM*. Moreover, it exists a tool to convert from any log format to MXML, called *ProM Import*[`https://sourceforge.net/projects/promimport`].

MXML is a very complete format, but for GenetGUI purpose just a few fields will be considered. Next, there is an example that explain that fields. To see the complete specification of MXML format visit `http://is.tm.tue.nl/research/processmining/WorkflowLog.xsd`.

Listing 4: Event Logs format

```
<WorkflowLog>
  <Process id="0" description="a42f0n00_5.trc">
    <ProcessInstance id="0" description="">
      <AuditTrailEntry>
        <WorkflowModelElement>S</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a1</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a31</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
    </ProcessInstance>
    <ProcessInstance id="1" description="">
      <AuditTrailEntry>
        <WorkflowModelElement>S</WorkflowModelElement>
        <EventType>complete</EventType>
      </AuditTrailEntry>
      <AuditTrailEntry>
        <WorkflowModelElement>a1</WorkflowModelElement>
        <EventType>complete</EventType>
```

```
        </AuditTrailEntry>
        <AuditTrailEntry>
          <WorkflowModelElement>a34</WorkflowModelElement>
          <EventType>complete</EventType>
        </AuditTrailEntry>
      </ProcessInstance>
    </Process>
</WorkflowLog>
```

Process Define a process.

ProcessInstance Define a instance of the process.

AuditTrailEntry Define an event into the instance of the process. The order of the events is determined by the order that their appear in the file.

WorkflowModelElement Name of the event.

EventType Type of the event. GenetGUI will only consider *complete* process.

5 Transform Traces/Event Logs into TS

While GenetGUI accepts as input Traces, Event Logs and Transition Systems, Genet only accepts Transition Systems. Therefore, GenetGUI must convert Traces/Event Logs into Transition Systems. The convertion of Event Logs into Traces is straight forward. Thus, in that section we will refer only to the Traces.

This transformation is based on [4].

A Transition System is a state-based model. Therefore, the first step must be define the states. To define the states we will choose:

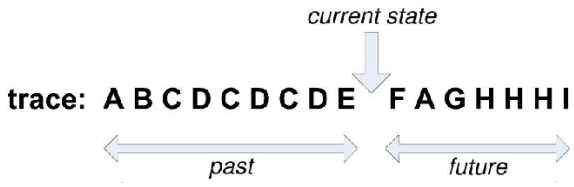


Figure 8: Past and Future of a current state

Past/Future As it is shown in the Figure 8, given the current position of the trace we can define the current stat using the events already happened (past) or the coming events (future).

Maximal Horizon The number of events to consider (in the past or in the future)

Q Parameter The order of the events matters (sequence) or doesn't (set).

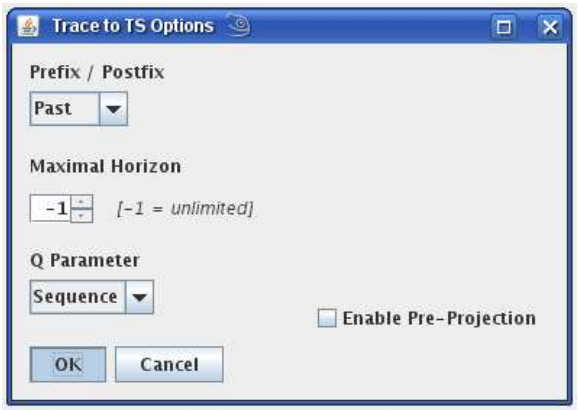


Figure 9: Traces to Transition System Options

An extra option non direct related with the conversion appears in that panel: the CasualGraph check box. The CasualGraph is necessary to use the Pre-Projections Algorithms. Therefore, if the check box is not selected of the convertion, then the Pre-Projection Algorithms will not be able.

References

[1] Genet web site. `http://www.lsi.upc.edu/~jcarmona/genet.html`.

[2] PComp web site. `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/pcomp/`.

[3] Process Mining Web. `http://www.processmining.org`.

[4] W. van der Aalst, V. Rubin, H. Verbeek, B. van Dongen, E. Kindler, and C. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and Systems Modeling*, 2009.

B

Building a Petri Net using Prefuse

Building a Petri Net using Prefuse

Jorge Muñoz Gama

Contents

1 Introduction

The aim of this tutorial is to give a quick overview to the Prefuse library using the creation of a Petri Net visualization as example (visit `http://en.wikipedia.org/wiki/Petri_net` to know more about what a Petri Net is). With that example we will see how to create a simple Prefuse visualization and how to deal with the specific problems that appears, thing that can be useful in other Prefuse applications.

This tutorial is part of the Final Degree Project "Process Mining: descobrint models formals a partir de logs d'un sistema". The aim of that project was to build an application to handler with Petri Nets, called GenetGUI.

I am sure that this tutorial can be improved, extended, and corrected. For that reason, the sources and the code are provided too. Feel free to modified, use and share with everybody your changes.

2 What is Prefuse?

Prefuse is a Java-based toolkit for building interactive information visualization applications. It supports a rich set of features for data modeling, visualization, and interaction. It provides optimized data structures for tables, graphs, and trees, a host of layout and visual encoding techniques, and support for animation, dynamic queries, integrated search, and database connectivity.

Prefuse is written in Java, using the Java 2D graphics library, and is easily integrated into Java Swing applications or web applets. Prefuse is licensed under the terms of a BSD license, and can be freely used for both commercial and non-commercial purposes.

The Figure 1 shows the pipeline to construct a prefuse visualization.

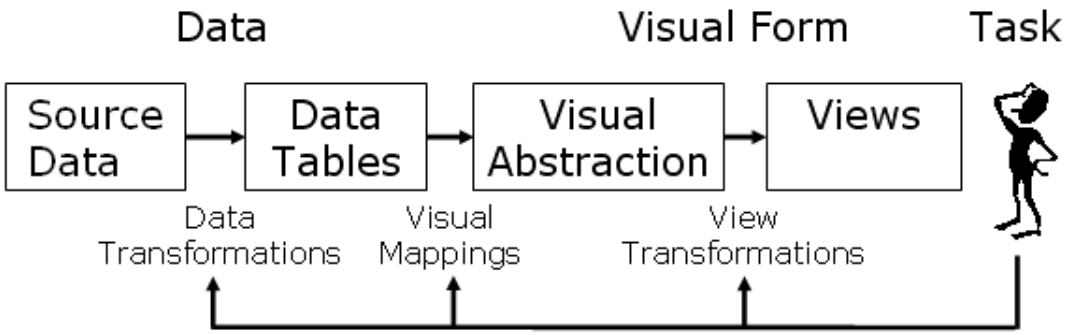


Figure 1: Prefuse Visualization Pipeline

The data to be displayed is imported to Prefuse internal structures, transforming them if it is necessary. Then, the structures are mapped to a Visual Abstraction. This Visual Abstraction will contain information about how to display the elements of the visualization (shape, color, layout, ...). After that, a view (or more than one) is created for this Visual Abstraction (or for just a subset of their elements). At the end, several tasks are added to perform transformation in some elements of the pipeline.

3 Building a Petri Net

3.1 Importing your source data

The first step that you must do to build a Petri Net using Prefuse is importing the data of the Petri Net into internal Prefuse structures. Prefuse provides you several ways to do that task easily. There are functions to import automatically data from XML files, like *GraphMLReader*, or you can overwrite the *AbstractGraphReader* for import your specific data. Don't forget to take a look to Prefuse's API.

In our case, and because we used an other tool to parser the data, we decided to create a class that contains the internal Prefuse structure and a few functions to fill it. The class, called *PetriNet*, has 2 attributes: a Prefuse Graph structure (the one chosen to represent a Petri Net) and a Map, to get a node easily using its name.

Listing 1: Petri Net's Attributes

```
private Graph graph;
private Map<String,Node> nodeMap;
```

When you create a new instance of a Petri Net, a new Prefuse Graph is created and then all the fields of the nodes and edges, using the *addColumn*. In our Petri Net. The fields are:

Name Name of the nodes

Place Boolean that shows if the node is a Place or a Transition

Capacity The capacity of each place

Marking The tokens of each place

Marking Name We will use a image to show the tokens inside a Place. That field shows the path and name of the image that displays the tokens.

Weight Weight of the edges (number of tokens that they will consume or produce)

The Listing 2 shows the code to create the Petri Net. See that we use constants in the place of simple strings, to have a cleaner and easy-to-extend code.

Listing 2: Creating a Petri Net Structure

```
public PetriNet() {
    //First, inizializate the nodeMap
    this.nodeMap = new HashMap<String,Node>();

    //Second, inizializate the Graph
    this.graph = new Graph(true);//true = Directed Graph
    // General Node attributes
    this.graph.addColumn(Constants.NAME, String.class);
    this.graph.addColumn(Constants.PLACE, boolean.class);
    //Place Node attributes
    this.graph.addColumn(Constants.CAPACITY, Integer.class);
    this.graph.addColumn(Constants.MARKING, Integer.class);
    this.graph.addColumn(Constants.MARKING_IMAGE, String.class);
    //Edge Attributes
    this.graph.addColumn(Constants.WEIGHT, Integer.class);
}
```

The last part of importing the data is create functions to fill the structure with that data. And example of one of this functions is shown in the Listing 3. It is used to add a new Transition Node to the Petri Net. The rest of the functions can be seen in the Chapter 4.

Listing 3: Add Transition Node Function

```
public void addTransition(String name) {
    Node node = this.graph.addNode();
    node.set(Constants.NAME, name);
    node.set(Constants.PLACE, false);
    this.nodeMap.put(name, node);
}
```

3.2 Map the data to visual abstraction

The second step of the process is add the data imported in the previous step to a visual abstraction. This visual abstraction includes features such as the spatial layout, color, size and shape of the elements. The most important Prefuse class for that task is *Visualization*. The Listing 4 shows how to do that mapping to a visualization object.

Listing 4: Add a Graph to a Visualization

```
m_vis.add(GRAPH, graph);
```

After that, we have to set up the renders. The renders will indicate how to display each element of the visualization. Prefuse provides a few easy-to-use renders. But Petri Net's case is a bit special: we need special shapes for the nodes depending on if they are places or transitions. For that reason we will implement our own *RendererFactory*. That RendererFactory will contain the different renderers for the edges, transitions, empty places and places with tokens, and it will return each one in each case. The Listing 5 shows the implementation.

Listing 5: Implementation of the Renderer Factory

```
public class PetriNetRendererFactory extends DefaultRendererFactory{

    public static final int PLACE_SIZE = 30;

    private EdgeRenderer edgeR;
    private LabelRenderer transR;
    private ShapeRenderer placeR;
    private PlaceRenderer placeMarkedR;

    /**
     * Creator that create a DefaultRendererFactory
     */
    public PetriNetRendererFactory(Visualization vis) {

        //create a DefaultRendererFactory
        super();

        //create the Edge Renderer
        edgeR = new EdgeRenderer(Constants.EDGE_TYPE_LINE, Constants.
            EDGE_ARROW_FORWARD);

        //create the Transition Node Renderer
        transR = new LabelRenderer(genetgui.Constants.NAME);
        transR.setHorizontalPadding(6);
        transR.setVerticalPadding(6);
```

```
        //create the Place Node Renderer
        placeR = new ShapeRenderer();
        placeR.setBaseSize(PLACE_SIZE);

        //create a Marked Place Node Renderer
        placeMarkedR = new PlaceRenderer(PLACE_SIZE, vis);
    }

    /**
     * Redefined getRenderer to return different renderers depending on if it's
     * a Edge, a Transition Node o a Place Node
     * @param item Item asked for its renderer
     * @return Renderer of the item asked
     */
    @Override
    public prefuse.render.Renderer getRenderer(VisualItem item) {

        Renderer returnR = null;

        //If the item is a Edge, return EdgeRenderer
        if (item instanceof EdgeItem){
            returnR = edgeR;
        }

        //If the item is a Node, depending on if it's Place o Transitions
        else {
            boolean isPlace = ((Boolean)item.get(genetgui.Constants.PLACE)).
                booleanValue();

            //Code for the Edge Decorators
            if(item.isInGroup(genetgui.Constants.EDGE_DECORATORS)){
                if ((item.get(genetgui.Constants.WEIGHT)) == null) return new
                    NullRenderer();
                else return new LabelRenderer(genetgui.Constants.WEIGHT);
            }

            if (isPlace) {
                item.setShape(Constants.SHAPE_ELLIPSE);

                if ((item.get(genetgui.Constants.MARKING)) == null){
                    returnR = placeR;
                }
                else{
                    returnR = placeMarkedR;
                }
            }
            else if (!isPlace) {
                returnR = transR;
            }
        }

        return returnR;
    }
}
```

See that we use our own renderer, called *PlaceRenderer*, to display the places with tokens (the other renderers used are provided by Prefuse). That renderer is a combination of three simple renderers: a *ShapeRenderer* to display a circle, a *LabelRenderer* to display the image of

the tokens, and an other *LabelRenderer* to write the number of tokens if they are too many to be display as an image. The implementation can be seen in the Chapter 4. The Listing 6 shows how to set the RendererFactory to the Visualization.

Listing 6: Set the RendererFactory to the Visualization

```
PetriNetRendererFactory pNRF = new PetriNetRendererFactory(m_vis);
m_vis.setRendererFactory(pNRF);
```

An other important part of our Petri Net Display is to show a label in the edges when the number of tokens consumed or produced for firing a Transition is greater than 1. To do this, we use *Decorators*. Decorators are VisualItem instances intended to "decorate" another VisualItem, such as providing a label or dedicated interactive control. The code of our Petri Net shows how to use decorators to display labels in the middle of the edges.

The next step of the Petri Net building process is set the colors of the elements. This will be done using *ColorActions* and adding all that actions to a *ActionList*. In our case, we have decided to add a useful feature to our Petri Net: when the cursor is over a node, its color is orange, the color of the nodes that point to it is green, and the nodes that are the output of the node are in red. For that reason, there are more than one color depending on the situation of the node. In the section 3.3 we will see how to implement that behaviour.

Listing 7: ColorActions

```
// Fill Nodes Action
ColorAction fillN = new PetriNetNodeColorAction(NODES,
        VisualItem.FILLCOLOR, ColorLib.gray(255));
fillN.add(new InGroupPredicate(INNODES), ColorLib.rgb(127, 255, 100));
fillN.add(VisualItem.FIXED, ColorLib.rgb(255,200,125));
fillN.add(VisualItem.HIGHLIGHT, ColorLib.rgb(255,100,100));

//Fill Edges Action
ColorAction fillE = new ColorAction(EDGES, VisualItem.FILLCOLOR, ColorLib.
    gray(100));
fillE.add(new InGroupPredicate(INEDGES), ColorLib.rgb(127, 255, 100));
fillE.add(VisualItem.FIXED, ColorLib.rgb(255,100,100));
fillE.add(VisualItem.HIGHLIGHT, ColorLib.rgb(255,100,100));

//Stroke Edges Action
ColorAction strokeE = new ColorAction(EDGES, VisualItem.STROKECOLOR,
    ColorLib.gray(100));
strokeE.add(new InGroupPredicate(INEDGES), ColorLib.rgb(127, 255, 100));
strokeE.add(VisualItem.FIXED, ColorLib.rgb(255,100,100));
strokeE.add(VisualItem.HIGHLIGHT, ColorLib.rgb(255,100,100));

//Draw Action
ActionList draw = new ActionList();
draw.add(fillN);
draw.add(fillE);
draw.add(strokeE);
draw.add(new ColorAction(NODES, VisualItem.STROKECOLOR, ColorLib.gray(100)
    ));
draw.add(new ColorAction(NODES, VisualItem.TEXTCOLOR, ColorLib.rgb(0, 0,
    225)));
draw.add(new StrokeAction(NODES,new BasicStroke(2)));
```

See that the *PetriNetNodeColorAction* used in the Listing 7 is a special Color Action created by ourself that fill the node with a special color if it is a enable transition (a transition that can be fired). The implementation can be seen in the Chapter 4.

One of the others things that has to be set is the *Layout*. The layout indicates how and where have to be display the elements of the graph. Prefuse provides some basic layouts, and others elements to create your own layout. In our Petri Net example, we have create a function to choose between some of Prefuse default layouts. The Listing 8 shows how to create the ActionList and set up the most common layouts.

Listing 8: Layouts

```
private ActionList getLayout(String layoutName) {
    ActionList layout = new ActionList();

    if (layoutName.equalsIgnoreCase(genetgui.Constants.NODE_LINK_TREE_LAYOUT))
        {
        layout.add(new NodeLinkTreeLayout(GRAPH));
    }
    else if (layoutName.equalsIgnoreCase(genetgui.Constants.RADIAL_TREE_LAYOUT
        )) {
        layout.add(new RadialTreeLayout(GRAPH,100));
    }
    else if (layoutName.equalsIgnoreCase(genetgui.Constants.
        FRUCHTERMANREINGOLD_LAYOUT)) {
        layout.add(new FruchtermanReingoldLayout(GRAPH));
    }
    else if (layoutName.equalsIgnoreCase(genetgui.Constants.
        FORCE_DIRECTED_LAYOUT)) {
        //Set Infinity Duration
        layout.setDuration(Activity.INFINITY);
        ForceDirectedLayout lay = new ForceDirectedLayout(GRAPH);
        //More distance between nodes
        ForceSimulator fsim = lay.getForceSimulator();
        fsim.getForces()[2].setParameter(1, 100);
        //fsim.getForces()[0].setParameter(0, -1.2f);
        layout.add(lay);
    }

    return layout;
}
```

The last task is register all the Actions to the Visualization. The Listing 9 show how can be done.

Listing 9: Register the Actions

```
//Register the Actions
m_vis.putAction("draw", draw);
m_vis.putAction("layout", layout);
m_vis.putAction("animate", animate);
m_vis.runAfter("draw", "layout");
m_vis.runAfter("layout", "animate");
```

3.3 Generate a view of the visual abstraction

Once the source data has been imported, and a visual abstraction has been create, the next step is create one (or more) views for that visual abstraction. The view may be zoomed to show only a subset of the data in the visual abstraction or multiple views may be generated to show the same visual abstraction from different perspectives. The most important Prefuse class for creating a view is *Display*. Display object provides a "window" into the contents of the Visualization object. This object draws all the items within its current view, and can support

panning, zooming and rotation. A single Visualization object can support multiple Display objects simultaneously, allowing multiple views into the same data set. For example, a program can be written with one view showing an overview of the data, and a second with a detailed view into specific elements. Modifications to the Display objects do not affect one another; however, modifications of the Visualization object, or the visual abstractions would be visible in both displays.

In our example, we just need a simple Display to show the Petri Net. The easier way is creating a class that extends from Display, and inside that class a visualization would be created and setted up (as shown in the previous steps). In the Chapter 4 you can see that class.

The next thing that must be done is set up the *Controls*. The Controls are the way that offers prefuse to provide especial behaviors to the Display, like zoom or pan. Prefuse has some really powerful Controls that can be setted up with just a code line. The Listing 10 show the controls used for our Petri Net Display. All are Prefuse default controls except *NeighborHiglight-InOutDisplayControl* and *PetriNetDisplayContro*, two Controls created by us for our Petri Net.

Listing 10: Controls

```
// 5. Add interactive controls for visualization
addControlListener(new DragControl());
addControlListener(new ZoomControl());
addControlListener(new PanControl());
addControlListener(new WheelZoomControl());
addControlListener(new ZoomToFitControl());
addControlListener(new NeighborHiglightInOutDisplayControl(m_vis,INEDGES,
    INNODES));
addControlListener(new PetriNetDisplayControl());
```

The *NeighborHiglightInOutDisplayControl* is a Control that fill the neighbors of the node which is under the cursor with different colors, depending on if they are input or output nodes. The implementation can be seen in the Chapter 4.

The *PetriNetDisplayControl* is a Control that allows you to fire transitions in a Petri Net. To fire a Transition it is just needed to double-click in a enable Transition. Firing a Transition will change the tokens of the places. The implementation of that Control can be seen in the Chapter 4.

The last step to create a Display is set up the display and run the correct Visualization Action to begin the display of everything. The Listing 11 shows that.

Listing 11: Set up the Display and Run the Action

```
// 6. Set up the Display
setSize(width,height);
pan((width/2),(height/2));
setHighQuality(true);

// 7. Set things running
m_vis.run("draw");
```

4 Code

4.1 MarkLayout

Listing 12: MarkLayout

```
/*
 * MarkLayout.java
 */

package genetgui.prefuse;

import java.awt.geom.Rectangle2D;
import java.util.Iterator;
import prefuse.action.layout.Layout;
import prefuse.visual.DecoratorItem;
import prefuse.visual.VisualItem;

/**
 *
 * @author Jorge ñMuoz Gama
 * @version 1.0
 */
public class MarkLayout extends Layout{
    public MarkLayout(String group) {
        super(group);
    }
    public void run(double frac) {
        Iterator iter = m_vis.items(m_group);
        while ( iter.hasNext() ) {
            DecoratorItem decorator = (DecoratorItem)iter.next();
            VisualItem decoratedItem = decorator.getDecoratedItem();
            Rectangle2D bounds = decoratedItem.getBounds();

            double x = bounds.getCenterX();
            double y = bounds.getCenterY();

            /* modification to move edge labels more to the arrow head
            double x2 = 0, y2 = 0;
            if (decoratedItem instanceof EdgeItem){
              VisualItem dest = ((EdgeItem)decoratedItem).getTargetItem();
              x2 = dest.getX();
              y2 = dest.getY();
              x = (x + x2) / 2;
              y = (y + y2) / 2;
            }
            */

            setX(decorator, null, x);
            setY(decorator, null, y);
        }
    }
} // end of inner class LabelLayout
```

4.2 NeighborHiglightInOutDisplayControl

Listing 13: NeighborHiglightInOutDisplayControl

```java
/*
 * NeighborHiglightInOutDisplayControl.java
 */

package genetgui.prefuse;

import java.awt.event.MouseEvent;
import java.util.Iterator;
import prefuse.Visualization;
import prefuse.controls.NeighborHighlightControl;
import prefuse.data.Tuple;
import prefuse.data.tuple.TupleSet;
import prefuse.visual.EdgeItem;
import prefuse.visual.NodeItem;
import prefuse.visual.VisualItem;

/**
 * Class to allow diferents colors between in and out edges and nodes
 * @author Jorge ñMuoz Gama
 * @version 1.0
 */
public class NeighborHiglightInOutDisplayControl extends NeighborHighlightControl
    {

    private Visualization vis;
    private String inEdges, inNodes;
    private TupleSet inEdgesTupleSet,inNodesTupleSet;
    private Iterator iterInEdges;

    public NeighborHiglightInOutDisplayControl(Visualization vis, String inEdges,
        String inNodes) {
        super();

        this.vis = vis;
        this.inEdges = inEdges;
        this.inNodes = inNodes;
        vis.addFocusGroup(this.inEdges);
        vis.addFocusGroup(this.inNodes);
      this.inEdgesTupleSet = vis.getFocusGroup(this.inEdges);
        this.inNodesTupleSet = vis.getFocusGroup(this.inNodes);
    }

    @Override
    public void itemEntered(VisualItem item, MouseEvent e) {
    //TODO Different color for in-out nodes

        if ( item instanceof NodeItem ){

            //Process In Edges and Nodes
            iterInEdges = ((NodeItem)item).inEdges();
            while (iterInEdges.hasNext()) {
                EdgeItem edge = (EdgeItem)iterInEdges.next();
                NodeItem sourceNode = edge.getSourceItem();
                this.inEdgesTupleSet.addTuple(edge);
                this.inNodesTupleSet.addTuple(sourceNode);
            }

            //Super class function
            setNeighborHighlight((NodeItem)item, true);
        }
```

```
    }

    @Override
    public void itemExited(VisualItem item, MouseEvent e) {
        if ( item instanceof NodeItem ){
            this.inEdgesTupleSet.clear();
            this.inNodesTupleSet.clear();
            //Superclass function
            setNeighborHighlight((NodeItem)item, false);
        }
    }


}
```

4.3 PetriNetDisplayControl

Listing 14: PetriNetDisplayControl

```
/*
 * PetriNetDisplayControl.java
 */

package genetgui.prefuse;

import java.awt.event.MouseEvent;
import java.util.Iterator;
import javax.swing.SwingUtilities;
import prefuse.controls.ControlAdapter;
import prefuse.visual.EdgeItem;
import prefuse.visual.NodeItem;
import prefuse.visual.VisualItem;

/**
 * Class with the actions to control the Display
 * @author Jorge ñMuoz Gama
 * @version 1.0
 */
public class PetriNetDisplayControl extends ControlAdapter{

    @Override
    public void itemClicked(VisualItem item, MouseEvent e) {

        //If Double click with left button
        if (SwingUtilities.isLeftMouseButton(e) && e.getClickCount() == 2) {

            if(item instanceof NodeItem){
                //If the Node is a Transition Node
                boolean isTransition = !((Boolean)item.get(genetgui.Constants.
                    PLACE)).booleanValue();
                if(isTransition) {
                    fireTransition((NodeItem)item);
                }
            }
        }
    }
```

```
    /**
     * Check if it's possible to fire a Transition, and if it's, it fire it
     * @param item Item of the Transition Node to be fired
     */
    private void fireTransition(NodeItem item) {
        if (canFire(item)) {
            fire(item);
        }
    }

    /**
     * Check if a transition can be fired (enough tokens in the input places)
     * @param item Item of the Transition Node to be check
     * @return true if the transition can be fired
     */
    private boolean canFire(NodeItem item) {
    //TODO Check for the capacity of the output node
        boolean canFire = true;

        //If there are more input edges and fire the transitions it's even
            possible
        Iterator iterInEdges = item.inEdges();
        while (iterInEdges.hasNext() && canFire) {

            //Get the input edge and the input Node
            EdgeItem edge = (EdgeItem) iterInEdges.next();
            NodeItem sourceNode = edge.getSourceItem();

            //Get the weight of the Arc
            Integer weight = (Integer) edge.get(genetgui.Constants.WEIGHT);
            if(weight == null) weight = new Integer(1);

            //Get the tokens of the Input Place
            Integer marking = (Integer) sourceNode.get(genetgui.Constants.MARKING)
                ;
            if(marking == null) marking = new Integer(0);

            //Check if the number of tokens is enough
            if(marking < weight) canFire = false;
      }

        return canFire;
    }

    /**
     * Fire a transition (wasting the input tokens and setting token in the output
     * places)
     * @param item Item of the Transition Node to be fired
     */
    private void fire(NodeItem item) {

        //Remove the input tokens
        Iterator iterInEdges = item.inEdges();
        while (iterInEdges.hasNext()) {

            //Get the input edge and the input Node
            EdgeItem edge = (EdgeItem) iterInEdges.next();
            NodeItem sourceNode = edge.getSourceItem();

            //Get the weight of the Arc
```

```
            Integer weight = (Integer) edge.get(genetgui.Constants.WEIGHT);
            if(weight == null) weight = new Integer(1);

            //Get the tokens of the Input Place
            Integer marking = (Integer) sourceNode.get(genetgui.Constants.MARKING)
                ;
            if(marking == null) marking = new Integer(0);

            //Set the new marking
            setMarking(sourceNode,marking-weight);
        }

        //Add the output tokens
        Iterator iterOutEdges = item.outEdges();
        while(iterOutEdges.hasNext()) {

            //Get the output edge and the output Node
            EdgeItem edge = (EdgeItem) iterOutEdges.next();
            NodeItem targetNode = edge.getTargetItem();

            //Get the weight of the Arc
            Integer weight = (Integer) edge.get(genetgui.Constants.WEIGHT);
            if(weight == null) weight = new Integer(1);

            //Get the tokens of the Output Place
            Integer marking = (Integer) targetNode.get(genetgui.Constants.MARKING)
                ;
            if(marking == null) marking = new Integer(0);

            //Set the new marking
            setMarking(targetNode,marking+weight);
        }
    }

    /**
     * Set the Marking and the Marking Image of a Place Node
     * @param node Place to set the Marking
     * @param marking Marking to be set
     */
    private void setMarking(NodeItem node, Integer marking) {
    //TODO Put the setMarkingImage as a function of the Prefuse Node (extends)

        //If the place is empty, fill with null
        if (marking == 0) marking = null;

        //Set the Marking
        node.set(genetgui.Constants.MARKING, marking);

        //Set the Marking Image
        if (marking == null) {
            node.set(genetgui.Constants.MARKING_IMAGE, null);
        }
        else if (marking == 1) {
            node.set(genetgui.Constants.MARKING_IMAGE, genetgui.Constants.
                IMAGES_DIR+"GenetguiDot1.png");
        }
        else if (marking == 2) {
            node.set(genetgui.Constants.MARKING_IMAGE, genetgui.Constants.
                IMAGES_DIR+"GenetguiDot2.png");
        }
```

```
        else if (marking == 3) {
            node.set(genetgui.Constants.MARKING_IMAGE, genetgui.Constants.
                IMAGES_DIR+"GenetguiDot3.png");
        }
        else {
            node.set(genetgui.Constants.MARKING_IMAGE, genetgui.Constants.
                IMAGES_DIR+"GenetguiDot1.png");
        }
    }
}
```

4.4 PetriNetDisplay

Listing 15: PetriNetDisplay

```
/*
 * PetriNetDisplay.java
 */

package genetgui.prefuse;

import java.awt.BasicStroke;
import java.util.logging.Level;
import java.util.logging.Logger;
import prefuse.Display;
import prefuse.Visualization;
import prefuse.action.ActionList;
import prefuse.action.RepaintAction;
import prefuse.action.assignment.ColorAction;
import prefuse.action.assignment.StrokeAction;
import prefuse.action.layout.graph.ForceDirectedLayout;
import prefuse.action.layout.graph.FruchtermanReingoldLayout;
import prefuse.action.layout.graph.NodeLinkTreeLayout;
import prefuse.action.layout.graph.RadialTreeLayout;
import prefuse.activity.Activity;
import prefuse.controls.DragControl;
import prefuse.controls.PanControl;
import prefuse.controls.WheelZoomControl;
import prefuse.controls.ZoomControl;
import prefuse.controls.ZoomToFitControl;
import prefuse.data.Graph;
import prefuse.data.Schema;
import prefuse.util.ColorLib;
import prefuse.util.FontLib;
import prefuse.util.PrefuseLib;
import prefuse.util.force.ForceSimulator;
import prefuse.visual.VisualItem;
import prefuse.visual.expression.InGroupPredicate;

/**
 * Prefuse Display subclass with set up to show Petri Nets correctly
 * @author Jorge ñMuoz
 * @version 1.0
 */
public class PetriNetDisplay extends Display {

    public static final String NODES = "petrinet.nodes";
    public static final String EDGES = "petrinet.edges";
    public static final String GRAPH = "petrinet";
```

```
    private static final String INEDGES = "inEdges";
    private static final String INNODES = "inNodes";

    private static final Schema DECORATOR_SCHEMA = PrefuseLib.getVisualItemSchema
        ();
    static {
      DECORATOR_SCHEMA.setDefault(VisualItem.INTERACTIVE, false);
      DECORATOR_SCHEMA.setDefault(VisualItem.TEXTCOLOR, ColorLib.gray(128));
      DECORATOR_SCHEMA.setDefault(VisualItem.FONT, FontLib.getFont("Tahoma",16));
    }


    /**
     * Create a Display to show the Petri Net
     * @param graph Petri Net , as a prefuse graph, to be shown
     */
    public PetriNetDisplay(Graph graph, int height, int width, String layoutName)
        {

        //1. Create the default display and visualization
        super(new Visualization());

        //Disable Info Warnings
        Logger.getLogger("prefuse.data.expression.parser").setLevel(Level.OFF);

        //2. Add the graph to the visualization
        m_vis.add(GRAPH, graph);

        //3. Set up the renders
        PetriNetRendererFactory pNRF = new PetriNetRendererFactory(m_vis);
        m_vis.setRendererFactory(pNRF);

        DECORATOR_SCHEMA.setDefault(VisualItem.TEXTCOLOR, ColorLib.gray(0));
        m_vis.addDecorators(genetgui.Constants.EDGE_DECORATORS, EDGES,
            DECORATOR_SCHEMA);

        //4. Process the actions

        // Fill Nodes Action
        ColorAction fillN = new PetriNetNodeColorAction(NODES,
                VisualItem.FILLCOLOR, ColorLib.gray(255));
        fillN.add(new InGroupPredicate(INNODES), ColorLib.rgb(127, 255, 100));
        fillN.add(VisualItem.FIXED, ColorLib.rgb(255,200,125));
        fillN.add(VisualItem.HIGHLIGHT, ColorLib.rgb(255,100,100));

        //Fill Edges Action
        ColorAction fillE = new ColorAction(EDGES, VisualItem.FILLCOLOR, ColorLib.
            gray(100));
        fillE.add(new InGroupPredicate(INEDGES), ColorLib.rgb(127, 255, 100));
        fillE.add(VisualItem.FIXED, ColorLib.rgb(255,100,100));
        fillE.add(VisualItem.HIGHLIGHT, ColorLib.rgb(255,100,100));

        //Stroke Edges Action
        ColorAction strokeE = new ColorAction(EDGES, VisualItem.STROKECOLOR,
            ColorLib.gray(100));
        strokeE.add(new InGroupPredicate(INEDGES), ColorLib.rgb(127, 255, 100));
        strokeE.add(VisualItem.FIXED, ColorLib.rgb(255,100,100));
        strokeE.add(VisualItem.HIGHLIGHT, ColorLib.rgb(255,100,100));
```

```
        //Draw Action
        ActionList draw = new ActionList();
        draw.add(fillN);
        draw.add(fillE);
        draw.add(strokeE);
        draw.add(new ColorAction(NODES, VisualItem.STROKECOLOR, ColorLib.gray(100)
            ));
        draw.add(new ColorAction(NODES, VisualItem.TEXTCOLOR, ColorLib.rgb(0, 0,
            225)));
        draw.add(new StrokeAction(NODES,new BasicStroke(2)));

        //Layout Action
        ActionList layout = getLayout(layoutName);

        //Animate Action
        ActionList animate = new ActionList(Activity.INFINITY);
        animate.add(fillN);
        animate.add(fillE);
        animate.add(strokeE);
        animate.add(new RepaintAction());
        animate.add(new MarkLayout(genetgui.Constants.EDGE_DECORATORS));

        //Register the Actions
        m_vis.putAction("draw", draw);
        m_vis.putAction("layout", layout);
        m_vis.putAction("animate", animate);
        m_vis.runAfter("draw", "layout");
        m_vis.runAfter("layout", "animate");


        // 5. Add interactive controls for visualization
        addControlListener(new DragControl());
        addControlListener(new ZoomControl());
        addControlListener(new PanControl());
        addControlListener(new WheelZoomControl());
        addControlListener(new ZoomToFitControl());
        addControlListener(new NeighborHiglightInOutDisplayControl(m_vis,INEDGES,
            INNODES));
        addControlListener(new PetriNetDisplayControl());

        // 6. Set up the Display
        setSize(width,height);
        pan((width/2),(height/2));
        setHighQuality(true);

        // 7. Set things running
        m_vis.run("draw");
    }

    /**
     * Function to change the layout of a display
     * @param layoutName Name of the new layout to be display
     */
    public void changeLayout(String layoutName) {
        //Cancel and Remove old Layout action
        m_vis.getAction("layout").cancel();
        m_vis.removeAction("layout");

        //Create and run the new layout action
        m_vis.putAction("layout", getLayout(layoutName));
```

```
        m_vis.run("layout");
    }


    /**
     * Function that return the layout specified as a parameter
     * @param layoutName Name of the layout
     * @return ActionList with the layout
     */
    private ActionList getLayout(String layoutName) {
        ActionList layout = new ActionList();

        if (layoutName.equalsIgnoreCase(genetgui.Constants.NODE_LINK_TREE_LAYOUT))
            {
            layout.add(new NodeLinkTreeLayout(GRAPH));
        }
        else if (layoutName.equalsIgnoreCase(genetgui.Constants.RADIAL_TREE_LAYOUT
            )) {
            layout.add(new RadialTreeLayout(GRAPH,100));
        }
        else if (layoutName.equalsIgnoreCase(genetgui.Constants.
            FRUCHTERMANREINGOLD_LAYOUT)) {
            layout.add(new FruchtermanReingoldLayout(GRAPH));
        }
        else if (layoutName.equalsIgnoreCase(genetgui.Constants.
            FORCE_DIRECTED_LAYOUT)) {
            //Set Infinity Duration
            layout.setDuration(Activity.INFINITY);
            ForceDirectedLayout lay = new ForceDirectedLayout(GRAPH);
            //More distance between nodes
            ForceSimulator fsim = lay.getForceSimulator();
            fsim.getForces()[2].setParameter(1, 100);
            //fsim.getForces()[0].setParameter(0, -1.2f);
            layout.add(lay);
        }

        return layout;
    }
}
```

4.5 PetriNet

Listing 16: PetriNet

```
/*
 * PetriNet.java
 */

package genetgui.data;

import genetgui.Constants;
import java.util.HashMap;
import java.util.Map;
import prefuse.data.Edge;
import prefuse.data.Graph;
import prefuse.data.Node;

/**
 * Represent a Petri Net
```

```
 * @author Jorge ñMuoz Gama
 * @version 1.0
 */
public class PetriNet {

    //ATTRIBUTES
    private Graph graph;
    private Map<String,Node> nodeMap;


    //CONSTRUCTORS----------------------------------------
    /**
     * Construct a PetriNet, creating all the internal structures
     */
    public PetriNet() {
        //First, inizializate the nodeMap
        this.nodeMap = new HashMap<String,Node>();

        //Second, inizializate the Graph
        this.graph = new Graph(true);//true = Directed Graph
        // General Node attributes
        this.graph.addColumn(Constants.NAME, String.class);
        this.graph.addColumn(Constants.PLACE, boolean.class);
        //Place Node attributes
        this.graph.addColumn(Constants.CAPACITY, Integer.class);
        this.graph.addColumn(Constants.MARKING, Integer.class);
        this.graph.addColumn(Constants.MARKING_IMAGE,String.class);
        //Edge Attributes
        this.graph.addColumn(Constants.WEIGHT, Integer.class);
    }

    //PUBLIC FUNCTIONS------------------------------------
    /**
     * Create an arc, and create the nodes if they don't exist yet
     * @param sourceName Name of the source Node
     * @param targetName Name of the target Node
     * @param weight Weight of the arc (or null if it doesn't exist)
     */
    public void addArc(String sourceName, String targetName, String weight ) {
        //TODO Add a Prefuse group for Places and Transitions

        //Process the source
        Node sourceNode = this.nodeMap.get(sourceName);
        if(sourceNode == null){
            sourceNode = this.graph.addNode();
            sourceNode.set(Constants.NAME,sourceName);
            sourceNode.set(Constants.PLACE,true);
            this.nodeMap.put(sourceName, sourceNode);
        }

        //Process the target
        Node targetNode = this.nodeMap.get(targetName);
        if(targetNode == null){
            targetNode = this.graph.addNode();
            targetNode.set(Constants.NAME,targetName);
            targetNode.set(Constants.PLACE,true);
            this.nodeMap.put(targetName, targetNode);
        }

        //Create the arc
```

```
        Edge arc = this.graph.addEdge(sourceNode, targetNode);
        if(weight != null) arc.set(Constants.WEIGHT, Integer.valueOf(weight));
    }

    /**
     * Add a Transition as a Node, knowing that is a Transition
     * @param name Name of the transition to be added
     */
    public void addTransition(String name) {
        Node node = this.graph.addNode();
        node.set(Constants.NAME, name);
        node.set(Constants.PLACE, false);
        this.nodeMap.put(name, node);
    }

    /**
     * Set the capacity value to a Node
     * @param name Name of the node
     * @param capacity Capacity to be set
     */
    public void setCapacity(String name, String capacity) {
        Node node = this.nodeMap.get(name);
        node.set(Constants.CAPACITY, Integer.valueOf(capacity));
    }

    /**
     * Set the marking value to a Node
     * @param name Name of the Node
     * @param marking Marking to be set
     */
    public void setMarking(String name, String marking) {
        int mark;

        if(marking == null) mark = 1;
        else mark=Integer.parseInt(marking);

        Node node = this.nodeMap.get(name);
        node.set(Constants.MARKING, Integer.valueOf(mark));

        switch (mark) {
            case 1:
                node.set(Constants.MARKING_IMAGE, Constants.IMAGES_DIR+"
                    GenetguiDot1.png");
                break;
            case 2:
                node.set(Constants.MARKING_IMAGE, Constants.IMAGES_DIR+"
                    GenetguiDot2.png");
                break;
            case 3:
                node.set(Constants.MARKING_IMAGE, Constants.IMAGES_DIR+"
                    GenetguiDot3.png");
                break;
            default:
                node.set(Constants.MARKING_IMAGE, Constants.IMAGES_DIR+"
                    GenetguiDot1.png");
                break;
        }
    }
```

```
    //GETTERS AND SETTERS----------------------------------------
    public Graph getGraph() {
        return graph;
    }

}
```

4.6 PetriNetNodeColorAction

Listing 17: PetriNetNodeColorAction

```
/*
 * PetriNetNodeColorAction.java
 */

package genetgui.prefuse;

import java.util.Iterator;
import prefuse.action.assignment.ColorAction;
import prefuse.util.ColorLib;
import prefuse.visual.EdgeItem;
import prefuse.visual.NodeItem;
import prefuse.visual.VisualItem;

/**
 *
 * @author Jorge ñMuoz Gama
 * @version 1.0
 */
public class PetriNetNodeColorAction extends ColorAction {

    public PetriNetNodeColorAction(String group, String field, int color) {
        super(group,field,color);
    }

    @Override
    public int getColor(VisualItem item) {
        if(item instanceof NodeItem){
            //If the Node is a Transition Node
            boolean isTransition = !((Boolean)item.get(genetgui.Constants.PLACE)).
                booleanValue();
            if(isTransition) {
                //If its a enable Transition (can be fired)
                if(canFire((NodeItem)item)){return ColorLib.rgb(159, 182, 205);}
                else{return super.getColor(item);}
            }
            else {return super.getColor(item);}
        }
        else{
            return super.getColor(item);
        }
    }

    /**
     * Check if a transition can be fired (enough tokens in the input places)
     * @param item Item of the Transition Node to be check
     * @return true if the transition can be fired
     */
    private boolean canFire(NodeItem item) {
```

```
    //TODO Check for the capacity of the output node
        boolean canFire = true;

        //If there are more input edges and fire the transitions it's even
            possible
        Iterator iterInEdges = item.inEdges();
        while (iterInEdges.hasNext() && canFire) {

            //Get the input edge and the input Node
            EdgeItem edge = (EdgeItem) iterInEdges.next();
            NodeItem sourceNode = edge.getSourceItem();

            //Get the weight of the Arc
            Integer weight = (Integer) edge.get(genetgui.Constants.WEIGHT);
            if(weight == null) weight = new Integer(1);

            //Get the tokens of the Input Place
            Integer marking = (Integer) sourceNode.get(genetgui.Constants.MARKING)
                ;
            if(marking == null) marking = new Integer(0);

            //Check if the number of tokens is enough
            if(marking < weight) canFire = false;
      }

        return canFire;
    }

}
```

4.7 PetriNetRendererFactory

Listing 18: PetriNetRendererFactory

```
/*
 * PetriNetRendererFactory.java
 */

package genetgui.prefuse;

import prefuse.Constants;
import prefuse.Visualization;
import prefuse.render.DefaultRendererFactory;
import prefuse.render.EdgeRenderer;
import prefuse.render.LabelRenderer;
import prefuse.render.NullRenderer;
import prefuse.render.Renderer;
import prefuse.render.ShapeRenderer;
import prefuse.visual.EdgeItem;
import prefuse.visual.VisualItem;

/**
 * Redefined DefaultRendererFactory to give differents Renders depending on
 * if it's Place or Transitions
 * @author Jorge Munoz Gama
 * @version 1.0
 */
public class PetriNetRendererFactory extends DefaultRendererFactory{
```

```
    public static final int PLACE_SIZE = 30;

    private EdgeRenderer edgeR;
    private LabelRenderer transR;
    private ShapeRenderer placeR;
    private PlaceRenderer placeMarkedR;

    /**
     * Creator that create a DefaultRendererFactory
     */
    public PetriNetRendererFactory(Visualization vis) {

        //create a DefaultRendererFactory
        super();

        //create the Edge Renderer
        edgeR = new EdgeRenderer(Constants.EDGE_TYPE_LINE, Constants.
            EDGE_ARROW_FORWARD);

        //create the Transition Node Renderer
        transR = new LabelRenderer(genetgui.Constants.NAME);
        transR.setHorizontalPadding(6);
        transR.setVerticalPadding(6);

        //create the Place Node Renderer
        placeR = new ShapeRenderer();
        placeR.setBaseSize(PLACE_SIZE);

        //create a Marked Place Node Renderer
        placeMarkedR = new PlaceRenderer(PLACE_SIZE, vis);
    }

    /**
     * Redefined getRenderer to return different renderers depending on if it's
     * a Edge, a Transition Node o a Place Node
     * @param item Item asked for its renderer
     * @return Renderer of the item asked
     */
    @Override
    public prefuse.render.Renderer getRenderer(VisualItem item) {

        Renderer returnR = null;

        //If the item is a Edge, return EdgeRenderer
        if (item instanceof EdgeItem){
            returnR = edgeR;
        }

        //If the item is a Node, depending on if it's Place o Transitions
        else {
            boolean isPlace = ((Boolean)item.get(genetgui.Constants.PLACE)).
                booleanValue();

            //Code for the Edge Decorators
            if(item.isInGroup(genetgui.Constants.EDGE_DECORATORS)){
                if ((item.get(genetgui.Constants.WEIGHT)) == null) return new
                    NullRenderer();
                else return new LabelRenderer(genetgui.Constants.WEIGHT);
            }
```

```
            if (isPlace) {
                item.setShape(Constants.SHAPE_ELLIPSE);

                if ((item.get(genetgui.Constants.MARKING)) == null){
                    returnR = placeR;
                }
                else{
                    returnR = placeMarkedR;
                }
            }
            else if (!isPlace) {
                returnR = transR;
            }
        }

        return returnR;
    }
}
```

4.8 PlaceRenderer

Listing 19: PlaceRenderer

```
/*
 * PlaceRenderer.java
 */

package genetgui.prefuse;

/**
 * Renderer that draw a Petri Net Place marked
 * @author Jorge ñMuoz Gama
 * @version 1.0
 */
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Shape;

import prefuse.Visualization;
import prefuse.render.AbstractShapeRenderer;
import prefuse.render.LabelRenderer;
import prefuse.render.ShapeRenderer;
import prefuse.util.ColorLib;
import prefuse.visual.VisualItem;

public class PlaceRenderer extends AbstractShapeRenderer {

private ShapeRenderer circleR;
private LabelRenderer dotR;
private LabelRenderer textR;


public PlaceRenderer(int placeSize, Visualization vis) {
    circleR = new ShapeRenderer(placeSize);
    dotR = new LabelRenderer(null, genetgui.Constants.MARKING_IMAGE);
    dotR.setRenderType(AbstractShapeRenderer.RENDER_TYPE_NONE);
    textR = new LabelRenderer(genetgui.Constants.MARKING);
    textR.setRenderType(AbstractShapeRenderer.RENDER_TYPE_NONE);
```

```
    dotR.getImageFactory().preloadImages(vis.items(),genetgui.Constants.
        MARKING_IMAGE);
}

@Override
public void render(Graphics2D g, VisualItem item) {
    circleR.render(g, item);
    dotR.render(g, item);
    Integer mark = (Integer) item.get(genetgui.Constants.MARKING);

    //If the number of tokens is greater than 3, put only 1 and display the number
         over
    if(mark != null && mark > 3){
        item.setTextColor(ColorLib.rgb(255, 0, 0));
        item.setFont(new Font(null,Font.BOLD,20));
        textR.render(g, item);
    }
}


@Override
///XXX getRawShape should return the "raw", not transformed shape!
//Actually, it just returns the (already transformed) ShapeRenderers shape.
protected Shape getRawShape(VisualItem item) {
    return circleR.getShape(item);
}

}
```

5 More information

5.1 Official Sources

- `http://prefuse.org/`
  Prefuse web site
- `http://prefuse.org/gallery/`
  Prefuse example gallery. The examples and their code can be found inside the Prefuse source
- `http://prefuse.org/doc/api/`
  Prefuse API
- `https://sourceforge.net/forum/forum.php?forum_id=343013`
  Prefuse Help Forum. Your best friend. There are examples and ideas to do almost everything with Prefuse

5.2 Other Sources

- `http://prefuse.blogspot.com/`
  Useful overview of Prefuse
- `http://www.cs.mun.ca/~hoeber/teaching/cs4767/notes/04-prefuse/`
  Small and useful Prefuse introduction and tutorial
- `http://www.infovis-wiki.net/index.php/Prefuse`
  Another Prefuse introduction
- `http://goosebumps4all.net/34all/bb/forumdisplay.php?fid=18`
  Unofficial Prefuse Forum. With some good examples