eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

Informatika Fakultatea
Informatika Ingeniaritzako Gradua

■ **G**radu **A**maierako **P**roiektua ■

Konputagailuen Ingeniaritza

Sistema konkurrenteak simulatzeko kernel bat: multiprozesadore sistemak

---

Egilea: Iñigo Orrantia

Zuzendaria: F. Xabier Albizuri

2013 - iraila

informatika fakultatea
facultad de informática

LABURPENA

Jarraian, hainbat hilabetetan zehar garatutako proiektuaren deskribapena biltzen duen memoria dugu eskuragarri. Proiektu hau, sistema konkurrenteen simulazioan zentratzen da eta horretarako, mota honetako sistemen arloan hain erabiliak diren Petri Sareak lantzeaz gain, simulatzaile bat programatzeko informazio nahikoa ere barneratzen ditu.

Gertaera diskretuko simulatzaile estatistiko batean oinarrituko da proiektuaren garapena, helburua izanik Petri Sareen bidez formalizatzen diren sistemak simulatzeko softwarea osatzea. Proiektuaren helburua da objektuetara zuzendutako hizkuntzaren bidez, Java hizkuntzaren bidez alegia, simulatzailearen programazioa erraztea eta ingurune honen baliabideak erabiltzea, bereziki XML teknologiari lotutakoak.

Proiektu hau, bi zati nagusitan banatzen dela esan daiteke.

Lehenengo zatiari dagokionez, konputazio munduan simulazioa aurkeztu eta honi buruzko behar adina informazio emango da. Hau, oso erabilgarria izango da programatuko den simulatzailearen nondik norakoak ulertu eta klase desberdinen inplementazioa egin ahal izateko. Horrez gain, zorizko aldagaiak eta hauen simulazioa ere islatzen dira, simulazio prozesu hori ahalik eta era errealean gauzatzeko helburuarekin.

Ondoren, Petri Sareak aurkeztuko dira, hauen ezaugarri eta sailkapen desberdinak goraipatuz. Gainera, Petri Sareak definitzeko *XML* lengoaia erabiliko denez, mota honetako dokumentu eta eskemak aztertuko dira, hauek, garatuko den aplikazioaren oinarri izango direlarik. Bestalde, aplikazioaren muin izango diren klaseen diseinu eta inplementazioak bildu dira azken aurreko kapituluan. Alde batetik, erabili den *DOM* egituraren inguruko informazioa islatzen da eta bestetik, *XML*-tik habiatuz lortuko diren *PetriNet* instantziak maneiatzeko ezinbestekoak diren *Java* klaseen kodeak erakusten dira.

Amaitzeko, egileak ateratako ondorioez gain, proiektuaren garapen prozesuan erabili den bibliografiaren berri ere ematen da.

AURKIBIDEA

TAULA ETA IRUDIEN ZERRENDA

TAULAK

IRUDIAK

1. KAPITULUA

SARRERA

Konputazio zientzietan, sistema konkurrenteen erabilera geroz eta pisu handiagoa hartzen ari dela ukaezina da. Denborak aurrera egin ahala, diseinatzen diren sistema berrietan, zerbitzuen kalitatea hobea da, noski. Ondorioz, gaur egungo sistema berri horiek kalkulu desberdinak aldi berean egiteko eta potentzialki, elkarren artean eragiteko ahalmena izaten dute. Hori horrela izanik, ezaugarri horiek betetzen dituzten sistemek, *konkurrentzia* izenez ezagutzen den bertute edo propietatea dutela esaten da. Aipatutako aldibereko kalkulu horien kudeaketarako, hainbat eredu matematiko garatu dira azken urteotan, esaterako, Petri Sareak, kalkulu prozesua, makinaren zorizko sarbidearen eredu paraleloa, Aktore eredua edota Reo-ren koordinazio lengoaia, batzuk aipatzearren.

Proiektu honetan zehar, Petri Sareak aztertu eta erabiliko dira sistema konkurrenteen simulazioen banaketa-eredu gisa. Petri Sareak, alderdi biko grafo zuzenduak dira, non adabegiek, trantsizioak (gertaerak) eta lekuak (baldintzak) adierazten dituzten eta hauek sarrera eta irteera arkuez elkartzen diren. Hauek definitzeko, *XML* (eXtensible Markup Language) marka-lengoaia erabiliko da, horretarako, *XML* dokumentuak maneiatuko direlarik. Bestalde, aplikazioa, objektuei zuzendutako *Java* programazio lengoaiarekin garatuko da. Aplikazioaren eginbeharra, simulatu nahi diren eta *XML* formatuan dauden Petri Sareen *XML* dokumentuak jaso eta *PetriNet* java objektuak sortzea izango da, ondoren, hauek simulatu ahal izateko helburuarekin.

Konputagailuen munduan, simulazio hitzak, egoera erreal baten modelatze-saiakerari egiten dio erreferentzia, honela, sistemaren portaera aztertu ahal izango delarik. Proiektuan, simulazio prozesu eta lengoaia desberdinen inguruko informazioa barneratzen da, geroago, simulatzailea programatzeko beharrezkoak diren xehetasunak ulertu ahal izateko, hain zuzen ere.

Beraz, Petri Sareen bilakaera eta portaera aztertu ahal izateko aplikazio bat garatzeko nondik norakoak ezagutu eta aplikatu ahal izango dira jarraian eskuragarri ditugun kapituluetan zehar.

2. KAPITULUA

Proiektuaren Helburuen Dokumentua

2.1 Aurkezpena

Gaur egungo sistema informatikoetan konkurrentziak duen izugarrizko garrantzia aipatu dugu sarreran. Konkurrentzia hitza, konputazioari dagokionez, hainbat prozesu une berean exekutatzeko ahalmena duten sistemen propietate bezala defini daiteke. Berez, posible da prozesu multzo bat benetan aldi berean exekutatzea, beti ere horietako bakoitza prozesadore desberdinetan exekutatzen bada. Prozesadore bakarrean exekutatzen diren kasurako, ostera, konkurrentzia simulatua da, prozesuen arteko exekuzioan denbora-tarte oso txikiak hartzen direlarik. Honela, prozesuak aldi berean exekutatzen direla simulatzen da.

Kapitulu honetan zehar, proiektuaren nondik norakoak aztertuko dira: proiektuaren helburu nagusiak, burutuko diren atazak, hauekin igarotako denbora kopurua, sor daitezkeen arrisku-mota desberdinak, arrisku hauei aurre egiteko kontingentzia plana, lan plangintza eta metodologia, etab.

2.2 Proiektuaren helburuak

Proiektuaren helburu nagusia, sistema konkurrenteak kudeatzeko (simulatzeko batik bat) garaian Petri Sareekin lan egiten duten erabiltzaileentzat, lagungarri izango zaien aplikazio bat garatzea da. Aplikazio hau, programatu ahal izango den simulatzailearen lanetarako beharrezkoa izango da. Horrez gain, garapen hori aurrera eramateko, ezagunak diren hainbat teknologien erabileran trebetasuna lortu beharko da: *XML* (e***X****tensible* ***M****arkup* ***L****anguage*) lengoaia, *Java* programazio lengoaia edota *Petri Sare* izenez ezagutzen den eredu formala eta hauen erabileran, nagusienak aipatzearren.

Bestalde, software librearen alde egin nahi da bere erabilera bultzatuz. Proiektua, kode irekiko Ubuntu sistema eragilearen gainean garatuko da eta horrez gain, Eclipse, Openoffice, xfig, etab. bezalako programak erabiltzea sustatzen da.

Azkenik, egilearen helburu pertsonalei dagokienez, autosufizientzia bermatu nahi da, eskura dauden baliabideez baliatuz, sortzen diren arazoei aurre egin eta irtenbidea lortzeko gai dela ziurtatzeko helburuarekin.

2.3 Azpiatazen zerrenda

Azpiatazak sailkatzeko bi motatako prozesuak bereizten dira. Alde batetik *prozesu taktikoak* deitzen direnak, proiektuaren plangintzaren kudeaketa eta antolaketarekin zerikusia dutenak dira. Normalean, mota honetako prozesuak proiektu guztietara aplika daitezke. Bestetik, *prozesu operatiboak* daude, proiektuaren espezifikazio, diseinu eta garapenean oinarritzen direnak, hain zuzen ere. Hauen kasuan, helburuen arabera desberdinak izan daitezke proiektu batean edo bestean.

- **Prozesu taktikoak**

K- Kudeaketa

K1- Bilerak burutu

K1.1- Ohiko bilerak: proiektuaren zuzendariarekin, egindako lana aurkeztu eta oniritzia jasotzeko egiten direnak. Honekin batera, hitzordua finkatu edota bilera burutu bezalako ekintzak hartuko dute parte.

K1.2- Aurrerapen bilerak: proiektuaren zuzendariarekin, proiektuari jarraipena emateko burutzen diren bilerak. Ohiko bileretan parte hartzen duten ekintzek hartzen dute parte: hitzordua finkatu, bilera burutu...

K2- Artxiboa kudeatu: informazioa pilatu, antolatu, segurtasun kopiak egin, dokumentuak eguneratu,... eta antzeko ekintzak burutzen dira.

P- Planifikazioa

P1- PHD-a egin: Proiektuaren hasierako faseetan garatzen den dokumentua. Honen edukia, deskribapena, helburuak, lan metodologia, denbora-plangintza eta arriskuen zerrenda dira.

D- Dokumentazioa

D1- Memoria egin: garatu den proiektua deskribatzen duen dokumentua. Proiektua aurrera eramateko jarraitutako

prozesuak eta hautatutako soluzioaren inguruko informazioa eta deskribapena gordeko ditu. Horrez gain, hainbat eranskin ere izango ditu.

- **Prozesu operatiboak**

G- Garapena

G1- Teknologiak eta eredu formalak aztertu eta instalatu: proiektuaren garapenean zehar erabiliko diren teknologia desberdinak eta eredu formalak aztertu, hauei buruzko informazioa eskuratu eta beharrezko softwarea instalatu.

G1.1- XML teknologia aztertu

G1.2- Java teknologia aztertu

G1.3- Petri Sare eredu formala aztertu

G1.4- Simulazioa eta simulatzaile desberdinak aztertu

G1.5- Softwarea instalatu

G2- XML teknologia erabili: aplikazioak behar dituen XML dokumentuak eta hauekin bat datozen XML eskemak sortu eta manipulatu.

G2.1- XML dokumentuak erabili

G2.2- XML eskemak erabili

G3- DOM: gure aplikazioaren oinarria izango den DOM egitura ezagutu, aztertu eta maneiatu.

G4- Aplikazioa diseinatu eta inplementatu: horretarako beharrezkoak diren klaseak espezifikatu, diseinatu eta inplementatu.

G4.1- Klaseak espezifikatu eta diseinatu

G4.2- Klaseak inplementatu

G4.3- Kodea berrikusi eta txukundu

G5- Probak egin: aplikazioaren funtzionamendua bermatzeko beharrezko probak burutu.

2.4 Denboren estimazioa eta lanaren plangintza

2.4.1 Denboren estimazioa

Jarraian daukagun taulan agertzen dira proiektua osatzen duten atazak, horietako bakoitza burutzeko behar izan den ordu kopuruaren estimazio bat, deskribapen labur bat eta baita guztira igarotako ordu kopuruaren estimazioa ere:

Ataza	Deskribapena	Estimazioa (orduak)
PROZESU TAKTIKOAK	**Prozesu taktikoak guztira**	164
K- Kudeaketa	Kudeaketaren denbora guztira	**51**
K1- Bilerak burutu	Bilerekin igarotako denbora	**30**
K1.1- Ohiko bilerak	Egindako lanen oniritzia jasotzeko egindako bilerak	**16**
K1.2- Aurrerapen bilerak	Proiektuari jarraipena emateko egindako bilerak	**14**
K2- Artxiboa kudeatu	Dokumentuen kudeaketa	**21**
P- Planifikazioa	Planifikazioaren denbora guztira	**18**
P1- PHDa egin	Proiektuaren helburuen dokumentua egiten	**22**
D- Dokumentazioa	Dokumentazioaren denbora guztira	**95**
D1- Memoria egin	Proiektua deskribatzen duen dokumentua egiten	**95**
PROZESU OPERATIBOAK	**Prozesu operatiboak guztira**	**196**
G- Garapena	Garapenaren denbora guztira	**196**
G1- Teknologiak aztertu eta instalatu	XML, Java... teknologiekin igarotako denbora	**44**
G1.1- XML teknologia aztertu	XML, XPath... teknologiak aztertzen	**10**
G1.2- Java teknologia aztertu	Oracle, Java, Eclipse... teknologiak aztertzen	**14**
G1.3- Petri Sare eredu formala aztertu	Petri Sareak aztertzen	**9**
G1.4- Simulazioa eta simulatzaile desberdinak aztertu	Simulatzaileak aztertzen	**6**

G1.5- Softwarea instalatu	Java, Eclipse, xfig... instalatzen	**5**
G2- XML teknologia erabili	XML teknologien erabileran igarotako denbora	**34**
G2.1- XML dokumentuak erabili	XML dokumentuak sortu eta maneiatzen	**19**
G2.2- XML eskemak erabili	XML Schema sortu eta maneiatzen	**15**
G3- DOM	DOM ulertu eta eraldatzen emandako denbora	**12**
G4- Aplikazioa diseinatu eta inplementatu	Aplikazioa garatzen igarotako denbora	**92**
G4.1- Klaseak espezifikatu eta diseinatu	Klaseen espezifikazio eta diseinuan	**23**
G4.2- Klaseak inplementatu	Klaseen inplementazioan	**44**
G4.3- Kodea berrikusi eta txukundu	Kodea berrikusten	**25**
G5- Probak egin	Probak eta zuzenketak egiten igarotako denbora	**14**
GUZTIRA	Denbora guztira	**360**

Taula 2.4: Atazen denboren estimazioak

2.4.2 Lanaren plangintza

Proiektua aurrera eramateko bi fase nagusi bereiz daitezke planteamenduari dagokionez. Lehenengo fase horretan, kudeaketa, planifikazioa eta erabiliko diren teknologia desberdinen azterketarekin erlazionatutako atazek izango dute pisu gehien. Hori horrela izanik, hasieran, tutorearekin bilera bat egin zen proiektuaren nondik norakoak azaldu eta honen inguruko iritziak elkar-trukatzeko asmoarekin.

Ideia orokor bat egin ondoren, Proiektuaren Helburuen Dokumentuarekin jardungo da, lan-metodologia eta plangintza egiteaz gain, prozesu eta arrisku desberdinak sailkatuz. Horrez gain, aplikazioaren garapen zuzena bermatzeko beharrezkoa diren teknologien inguruan informazioa bilatuko da. Horretarako, Petri Sareen bibliografia kontsultatu ahal izango da fakultateko liburutegian eta gainera, internet bidez, Java eta XML inguruko tutorialak irakurri eta ulertzeari ekingo zaio.

Horiek guztiak lehenengo faseari dagozkien atazak lirateke. Ondoren, bigarren fasearekin hasi aurretik, eten labur bat egingo da lehenengo lauhileko azterketak modu egokian prestatu ahal izateko.

Bigarren lauhilekoan, bigarren faseari ekingo zaio. Alde batetik, XML teknologia eta bestetik, Java klaseen diseinu eta inplementazioan zentratuko naiz. Horrez gain, aplikazioaren funtzionamendua bermatzeko probak ere gauzatuko dira. Hasiera batean behintzat, ildo honetako atazak izango dira proiektuaren denboraren gehiengoa suposatuko dutenak, memoriaren idazketa kontutan hartu gabe, noski. Azken hau izango da proiektuan zehar etengabe landu beharko den ataza, ordu eta pisu gehien hartuko dituen dokumentua izango delarik.

2.5 Arriskuak eta kontingentzia plana

Proiektuaren arrakasta bermatu ahal izateko, beharrezkoa izango da helburuen lorpenean gaizki joan daitekeen guztia aurretik identifikatzea. Sor daitezkeen arazo hauek *arrisku* izenez ezagutzen dira. Hori dela eta, arazoak identifikatzeaz gain, garrantzitsua da hauei kontra egiteko *kontingentzia plana* pentsatu eta planifikatzea. Jarraian, proiektuaren garapenean ager daitezkeen arriskuen zerrenda erakusten da, hauetako bakoitza bere deskribapen, larritasun-maila, eragin, aurrezaintza eta kontingentzia planarekin batera.

1- Materiala eta teknologiarekin erlazionatutakoak

1.1- Informazio edo datuen galera.

1.1.1- Deskribapena: Zenbait datu edo informazio guztia galtzen dira.

1.1.2- Larritasun-maila: Egoeraren arabera aldakorra da. Galdutako datu-kopuruak zehaztuko du arazoaren larritasun-maila.

1.1.3- Eragina: Oso kaltegarria, datu horiek berreskuratzen ez diren kasuetan. Datuak berreskuratzea lortzen bada, denboran izango du eragin zuzena.

1.1.4- Aurrezaintza-plana: Segurtasun kopiak eta informazioa gordetzeko biltegi bat erabiliko dira.

1.1.5- Kontingentzia-plana: Informazioa lehenbailehen berreskuratzen saiatu. Berreskuratzen ez bada, galdutako lana berriro egin beharko da.

1.2- Teknologia desberdinekin lan egiteko arazoak.

1.2.1- Deskribapena: Proiektuan zehar erabiltzen diren teknologien erabilera ez da menperatzen.

1.2.2- Larritasun-maila: Baxua, ezagutzen ez den teknologia-kopurua txikia denean. Ertaina, kopuru hori handia den kasuetan.

1.2.3- Eragina: Plangintzan atzerapenak sor ditzake, teknologia guztiak menperatzea lortzen den bitartean.

1.2.4- Kontingentzia-plana: Liburu desberdinetan edota interneten, gida desberdinak bilatu eta erabiliko dira.

1.3- Baliabideak eskura ez edukitzea.

1.3.1- Deskribapena: Proiektuaren garapenean erabiliko diren baliabide eta prozeduren gabezia dago.

1.3.2- Larritasun-maila: Altua, adibidez ataza bat entregatzerako garaian internet konexiorik gabe geratzen bagara.

1.3.3- Eragina: Egin beharreko lana epez kanpo entregatzea suposa dezake.

1.3.4- Aurrezaintza-plana: Entrega bat egiteko ez da azken momentura arte itxarongo.

1.3.5: Kontingentzia-plana: Zuzendariarekin hitzarmen edo akordio batetara iristen saiatuko naiz.

2. Pertsonalak

2.1- Behin behineko baja.

2.1.1- Deskribapena: Kanpo-arrazoiengatik, momentu konkretu batean egin beharreko lana ezingo da gauzatu.

2.1.2- Larritasun-maila: Ertaina, epe motzerako bada. Altua, epe luzerako bada.

2.1.3- Eragina: Lan-pilaketak proiektuaren plangintzan eragingo du, zehazki atzerapenak sortuaz.

2.1.4- Aurrezaintza-plana: Estimatutako epeak ahal diren heinean aurreratzeak, arazoaren eragin gogorra ekidin dezake.

2.1.5- Kontingentzia-plana: Lan-pilaketa murrizteko esfortzua areagotuko da.

2.2- Atazak entregatu behar direneko epeak ez errespetatu.

2.2.1- Deskribapena: Zuzendariarekin adostutako estimatutako entregatze-datak ez dira errespetatzen.

2.2.2- Larritasun-maila: Ertaina, entregatu beharreko ataza-kopurua oso handia ez denean eta asko atzeratzen ez denean. Altua, lan asko pilatzen den kasuan.

2.2.3- Eragina: Denbora-atzerapenak lan pilaketa eragin dezake.

2.2.4- Kontingentzia-plana: Esfortzua handituko da, pilatutako lan-kopurua minimizatu eta proiektuaren martxa egokia berreskuratu ahal izateko.

2.3- Proiektuari buruzko ezjakintasuna.

2.3.1- Deskribapena: Proiektua osatzen duten atazaren bat menperatzeko arazoak daude.

2.3.2- Larritasun-maila: Ertaina, ataza-kopurua oso handia ez denerako. Geroz eta altuagoa, ezjakintasun hori handitzen doan heinean.

2.3.3- Eragina: Eskainiko den produktuaren kalitatean eragingo du.

2.3.4- Aurrezaintza-plana: Erabaki bat hartu aurretik, lan bat arrakastatsu burutzeko gai izango naizela ziurtatu.

2.3.5- Kontingentzia-plana: Esfortzua eta dedikazioa handituko dira, menperatzen ez ditudan gaien inguruan informazioa bildu eta jakituria handiagotu ahal izateko.

2.6 Lan metodologia

Lehen aipatu den moduan, proiektua, fase desberdinetan bereiz genezake lan metodologiari dagokionez. Hasiera batean, zuzendariarekin, garatu nahi zen aplikazioaren nondik norakoak erabaki ziren, proiektuaren helburuaren

ideiak argitu eta erabiliko ziren teknologiak aurkeztuz. Ondoren, teknologia horien inguruko informazio pilaketa eta trebetasunean zentratuko gara. Honekin amaitzean, aplikazioaren diseinura joko dugu eta amaitzeko, probak egiteari ekingo zaio, aplikazioaren funtzionamendu zuzena bermatu ahal izateko.

Informazio pilaketaren eta teknologia desberdinen trebetasunaren fasea aurrera eramateko, hainbat baliabide erabiliko dira, esaterako, sistema konkurrente eta Petri Sareetan oinarritzen den bibliografia, internet edota liburutegian eskuragarri dauden aldizkariak. Horrez gain, *XML* dokumentuen eta eskemen inguruko zalantzak argitzeko datu-baseak jorratzen dituzten ikasgaien apunteak eskura edukiko dira [22].

Behin zalantzak argitzen ditugunean eta behar bezala dokumentatzen garenean, aplikazioaren betebeharren eta eskaini beharreko zerbitzuen inguruan erabakiak hartuko dira zuzendariarekin batera. Garapen prozesuan, aplikazioaren klaseak programatzeko erabiliko den programazio lengoaia *Java* izango da, eta horretarako kode irekiko *Eclipse* izeneko software plataformaz baliatuko gara. Bestalde, XML dokumentuak eta XML Schema-k maneiatzeko *Oxygen XML Editor* softwarea erabiliko da eta guzti honez gain, esan beharra dago, proiektua *Ubuntu 12.10* sistema eragilearen gainean inplementatuko dela. Gainera, fitxategien eguneraketa, babes-kopiak eta konputagailu desberdinetan lan egiten denerako sinkronizazioa mantentzeko, guzti hori ahalbidetzen duen *Ubuntu One* zerbitzua erabiliko da. Honek materiala leku batetik bestera eramatean sortzen diren arazoak ekidingo ditu, sinkronizatutako fitxategiak fakultateko konputagailuetan arazorik gabe errekuperatu ahal izango baititugu, hori bai, beti ere internet konexioa eskuragarri dugunean. Amaitzeko, Petri Sareen irudiak sortzeko, *xfig* softwareaz baliatuko gara.

Azkenik, diseinatutako aplikazioaren funtzionamendua bermatuko da proba desberdinak eginez, horretarako erabili diren Java klaseak aztertuz eta egin beharreko zuzenketak burutuz. Hori horrela izanik, klase hauek memorian modu txukun eta dokumentatuan ipiniko dira.

Bestalde, edozein momentutan sortutako zalantzak zuzendariaren laguntzarekin batera argitzeko, posta bidezko komunikazioa erabiliko da bilerak adostu edo zalantza horiek posta elektronikoetan bertan argitu ahal izateko.

3. KAPITULUA

SIMULAZIOA

3.1 Zer da simulazioa?

Simulazioa, sistema baten funtzionamendua ordezkatzen duen programa baten elaborazio eta exekuzioa da [1] [2] [3]. Horretarako, simulazio batek honako betekizun hauek bete beharko ditu: sistemaren funtzionamenduaren eredu formal bat eraikitzea eta kargaren errepresentazio (eredu) edo *trazaz* hornitzea. Horrez gain, simulazio batean zenbait kontuk garrantzi handia dute. Alde batetik, eredutzat hartzen den sistemaren xehetasun-maila goraipatu behar da eta bestetik, emaitzen analisi estatistikoaz gain, esperimentuaren diseinua, azken hau egingarria izan dadin.

Orokorrean, simulazioa, konputagailu sistemekin benetako errendimendu azterketa bat aurrera eramateko konplexutasunaren irtenbide bat dela esan daiteke. Simulazio batek modelatzaileari onura potentzial asko eskaintzen dizkio, izan ere, sistema berezi baten elkarrekintza konplexuak aztertu eta esperimentatzeko aukera ematen baitu.

Simulazio batek gure sistemaren sentsibilitate-analisia egitea ahalbidetzen du, egoera desberdinen aurrean sistema horren portaera aztertzeko baliabideak eskainiz. Honi esker, benetako sistema errazago ulertzeko aukera ematen da eta sistemaren elementu desberdinek funtzionamenduan duten garrantziaz gain, elementuen arteko elkar loturak ulertzea ere ahalbidetzen du. Sistema erreala oraindik existitzen ez bada ere, honek izan ditzakeen arazoei aurre egiteko aukerak azter daitezke. Azkenik, existitzen diren sistemetan, elementu berriak gehituz edota zerbitzu ezberdinak erabiliz egiten diren neurketak goraipatu behar dira.

Simulazioa hainbat helburu lortzeko arlo desberdinetan erabili ohi da, besteak beste salerosketa, ekonomia, marketina, hezkuntza, politika, gizarte zientziak, portaera zientziak, zientzia naturalak, nazioarteko erlazioak, garraioa, gerra jokoak, legearen aplikazioa, azterketa urbanoak, sistema globalak, espazioko sistemak, konputagailu diseinua edota ebakuntzetan.

Simulazio teknikei dagokienez, **gertaera diskretuko simulazioa** [4] da aztertuko dena. Teknika hau jarraitzean, sistemaren egoera gertaera bakoitzeko eguneratuko da. Gainera, simulatzaileak etorkizuneko

gertaera-zerrenda bat maneiatzen du, gertaerak sekuentzialki prozesatzean eguneratzen dena, hain zuzen ere. Honen inguruan, gertaera diskretuen zenbait adibide aipatzekotan, bezero bat ilara batetara iristea, baliabide bat bereganatzea edota zerbitzu baten amaierari dagozkiena izan daitezke, besteak beste. Guzti honez gain, aipatu beharrekoa da, gertaera diskretuko simulazio batean ez dela lan egiten denbora-tarte konstanteen diskretizazioarekin (0, δ, 2δ, 3δ , ...).

Gertaeren inguruan sailkapen desberdinak bereizi behar dira, sistemaren *barrukoak* eta *kanpokoak* alde batetik, eta *deterministak* eta *ez-determinista*k bestetik. Horren ildotik, simulazio batean gertaera *ez-deterministen* sorrera bi eratan egin daiteke: traza bat jarraituz sistema erreal baten neurketa bidez (zerbitzariak, bideratzaileak) edo probabilitate banaketa bat jarraituz (zorizko aldagaiak simulatzen dituzten algoritmoekin).

3.2 Algoritmo determinista eta ez-deterministak

Konputazio zientzietan, algoritmo determinista bat, aurresan edo iragarri daitekeen algoritmo bat da, beti ere sarrerako datuen inguruko nahikoa informazio dagoenean. Beste era batera esanda, algoritmoaren sarrera-datuak ezagunak direnean beti irteera bera emango duen algoritmo bezala kontsidera daiteke. Adibide argi bezala funtzio matematiko bat har daiteke, izan ere, funtzioak jasotako sarrera baterako beti irteera berdina emango baitu. Hala ere, formalki, algoritmo determinista bat egoera-makina bat bezala defini daiteke, egoera batek, makina une konkretu batean egiten ari dena azalduko digularik. Aipatutako egoera-makina horiek, sarrera eta irteeraz osatutako sistemak dira, zeinetan irteerak, uneko sarrerez gain, aurreko sarrera-seinaleen eraginpean ere egongo diren.

Bestalde, algoritmo ez-deterministak, sarrera bakoitzeko hainbat irteera posible eskaintzen dituzten algoritmoak dira. Hau da, ezin izango da aurresan algoritmo ez-determinista baten exekuzioak emango duen emaitza zein izango den. Mota honetako algoritmo bat konputazionalki determinista bihurtzeko modu bat zorizko *(pseudorandom)* zenbakiak sortzeko algoritmoak erabiltzea izango da.

3.3 Simulazio prozesua

Sistema askoren artean, modelatze edo exekuzio esperimentuak egiteko konputagailu digitalaren erabilera oso arrunta izan ohi da [2]. Simulazio ingurune honetan beste teknika mota desberdinak erabiliz lor ezin daitezkeen azterketa sistematikoak egitea posible da. Simulazio programek fase diskretuak erabiliko dituzte, honela beren potentzial guztia aprobetxatzeko helburuarekin. Jarraian aipatzen dira simulazio prozesu batean jarraitzen diren pausoak:

1. Aurrean dugun arazoak simulazioaren beharra duen ala ez aztertu.
2. Arazoari irtenbidea aurkitzeko eredua zein den erabaki.
3. Arazoari dagokion simulazio-eredua zein izango den erabaki.
4. Eredua lengoaia egokian inplementatu.
5. Simulazioaren probak diseinatu.
6. Eredua balioztatu.
7. Probak burutu.

Simulazio-ereduen diseinu tipikoetan 2, 3 eta 4 pausoak izango dira diseinatzaileari buruhauste gehien emango dizkiotenak, gehienbat ereduaren sorrera, simulazio formatuaren bilakaerak eta lengoaiaren inplementazioak duten konplexutasunagatik. Horrez gain, sistemaren elementu kritikoak identifikatzea ere ez da lan erraza izango. Behin elementu horiek identifikatu eta definitu (matematikoki, portaeraren arabera, funtzionalki...) eta beren elkarrekintzen muinak aurkitu direnean, simulazio-ereduaren garapena lortzen da, sistema-ereduen definizioarekin batera. Hau da, sistema-eredu bat garatzen dugun heinean, posible da zuzenean simulazio-ereduaren egitura ere definitzea.

3.4 Denbora-kontrola

Gertaera jarraituko eta diskretuko simulazioetan, garrantzi handiena duena denboraren kontrola edo kudeaketa izango da, honen erabilerak beren aldagaietan eragin zuzena izango duelarik. Denboraren erabilera hau, gehienbat sinkronizaziorako, egoeren aldaketen kalkulurako edota elkarrekintzak kontrolatzeko izango da. Denbora-kontrola bi motatakoa izan daiteke, sinkronoa edo asinkronoa.

Sinkrono kontzeptuak, gertaeren artean denbora-tarte konstante bat finkatzen dela adierazten du. Denbora hori zein izango den erabakitzerakoan zentzuz egitea garrantzitsua da, izan ere, denbora-tarte handia hartzen bada, gertaerak era konkurrentean gerta baitaitezke. Bestalde, denbora-tarte txikiegia hartzea erabaki zorrotzegia izan daiteke, honek exekuzio denbora luzatzeaz gain, gertaeren arteko desberdintasunak eragin ditzakeelako.

Denbora asinkronoen kasuan, gertaeren artean denbora konstanteak erabili ordez aldakorrak erabiliko dira. Kontzeptua, emango diren gertaeren erregistro baten bidez, egoeraren kontrola kudeatzean datza. Honela, denbora, jarraian datorren gertaeraren arabera aldatuko da. Gertaera kronologiko hauek, normalki, zerrenda estekatuetan antolatzen dira, gertaera berrien etorrerarekin batera eguneratzen direlarik.

3.5 Sistema eta ereduak

Orain arte simulazio ereduekin erlazionatutako atributu generikoei buruz aritu gara. Aipatu ez ditugunak, eredu mota desberdinen teknikak edota simulazio inplementaziorako erabiltzen diren teknikak dira (esaterako simulazio lengoaiak). Simulazio tekniken artean goraipa genitzakeenak honakoak dira: gertaera diskretuak, aldaketa jarraitua, ilarak, eredu konbinatuak eta teknika hibridoa.

Gertaera diskretuko ereduak jarraitzen diren simulazioetan [3], entitate kontzeptua erabiltzen da sistema errealaren objektuak erreferentziatzeko. Entitate hauek definitzeko ezinbestekoak izango dira atributuak (adibidez egoeraren deskribapena definitzeko). Horrez gain, entitate horien ekintzak baldintzatzen dituzten gertaerak ere aipatu behar dira, esate baterako, etorrerak, abiatze puntuak, gelditze puntuak, itxarote denborak edota antzeko seinaleak.

Simulazio jarraituek era jarraituan aldatzen diren aldagaiekin defini daitezkeen gertaera fisikoekin (prozesuak, baldintzak, portaerak) egiten dute lan. Aipatutako aldagai hauek, prozesu fisikoak adierazteko erabiltzen diren ekuazio diferentzialetan aurkitzen dira.

Ilaretan oinarritzen diren simulazio lengoaiak (AWESIM, GPSS, Q-gert, Slam II) ere existitzen dira. Arazo asko ilaren interkonexioen laguntzaz konpontzea posible izaten da gehienetan. Lengoaia horiei esker ilarak era desberdinetan kudea eta antola daitezke, esaterako, ilaren tamaina, zerbitzari kopurua, ilara mota, ilara diziplina, zerbitzari mota, zerbitzari diziplina, bezeroen sorrera, eragiketen jarraipena edota eragiketen aurkezpenaren arabera, besteak beste.

Aurretik aipatutako teknikek duten arazorik handiena, hauen erabilera ordena izango da. Hori dela eta, konponbidea hiru teknikak biltzen dituen lengoaiaren **eredu konbinatua** sortzea izan daiteke. Honi esker, modelatzaileak maila desberdinetan ordenaturik dagoen simulazio eredua diseina dezake, zehaztasun handiagoko informazioa eskura daitekeelarik. Azkenik, **teknika hibridoari** dagokionez, ideia, aurreko tekniken hainbat ezaugarri programazio-lengoaia konbentzionalen bidez gehitzea izango da.

3.6 Simulazio lengoaiak eta softwarea

Simulazioaren erabilerak gora egin duen heinean, simulazio lengoaiak garrantzi handiagoa hartzen joan dira [2]. Lengoaia hauek arrazoi desberdinengatik garatu dira, besteak beste, denboraren nozioa mantentzeko, simulazioaren egoera egokia mantentzeko, estatistikak biltzeko edota elkarrekintzak kontrolatzeko. Lengoaia nagusienen artean *GASP IV*, *GPSS*, *Simscript* eta *Slam II* goraipa daitezke. Software modernoagoak dira *ns-3* eta *OMNeT++*, *C++* lengoaian programatutako

moduluetan oinarrituz eraikitzen direnak.

GASP IV 70. hamarkadan garatu zen eta gaur egun zenbait aldaketekin erabili ohi den lengoaia da. *FOR-TRAN* lengoaian oinarritzen da eta gertaera diskretuak, jarraituak eta konbinatuak idazteko aukera eskaintzeaz gain, simulazio jarraituen ereduak idaztea ere ahalbidetzen du. Horrez gain, erabiltzaileek denbora maneiatu, fitxategiak maneiatu (gertaera fitxategiak, biltegiratze fitxategiak,...) eta informazioa eta estatistikak pila ditzakete *GASP IV* lengoaiarekin.

```
Dimension nset (1000)
common/gcom/atrib (100/DP(100), DLL (100, DTNOW,
II,MFA, MSTOP,NCLNR,NCRDR,NPRINT,NNRUN,
NNSET,NTAPE,SS(100),SSL(100),TNEXT,TNOW,XX(100)
Common Q Set
Equivalence Nset(1),Qset(1))
NNSet=1000
NCRDR=5
NPRINT=6
NTAPE=7
Call GASP
Stop
End
```

Adibidea 3.6.1: GASP IV FORTRAN programa

GPSS (General-Purpose Simulation System), prozesuetara zuzendutako simulazio lengoaia da. Lengoaia honek gertaera arruntak eta etorkizunekoak bereizten ditu. Horrez gain, hitz klabeak erabiltzen dira, **START**, **SIMULATE**, **TERMINATE**, **GENERATE**, **RELEASE**,... simulazioa hasteko, etorkizuneko gertaerak kudeatzeko, gertaeren exekuzioa amaitzeko, etab.

```
1 *
2 Simulate
3 *
4 XPDIS function RN1,C24
5 0.0, 0.0/0.1, 0.104/0.2, 0.222/0.3, 0.355/0.4, 0.509/0.5,
0.69
6 0.6, 0.915/0.7, 1.2/0.75, 1.38/0.8, 1.6/0.84, 1.83/0.88,
2.12/0.9
7  2.3/0.92,  2.52/0.94,  2.81/0.95,  2.99/0.96,  3.2/0.97,
3.5/0.98
8 4.0/0.99, 4.6/0.995, 5.3/0.998, 6.2/0.999, 7.0/0.9997, 8
9 *
10 TISYS table MP1,0,5,20
11 *
12 * model segment
13 *
```

```
14 Generate 10, FN$XPDIS
15 Mark P1
16 Queue Waitq
17 Seize SRVR
18 Depart Waitq
19 Advance 10,5
20 Release SRVR
21 Tabulate TISYS
22 Terminate
23 *
24 * timing segment
25 *
26 Generate 480
27 Terminate 1
```

Adibidea 3.6.2: GPSS kode-zatia

Azkenik, *Simscript* 60. hamarkadan garatutako lengoaia eta *Slam II*, 70. hamarkadan garatutakoa aipatu behar dira. Lehenengoak programatzaileari askatasun asko eskaintzen dio sintaxiaren aldetik eta horrek programak ulergarriak izan daitezen laguntzen du. Gainera, esan, bi motatako entitateak bereizten direla, behin betikoak alde batetik eta behin behinekoak bestetik. *Slam II* lengoaiak, aldiz, ilarak, gertaera diskretuak eta eredu jarraituaren ideiak biltzen ditu nolabait esateko.

Petri Sareak simulatzeko garatu den simulatzailea programatzerako garaian, merkatuko zenbait simulatzaile hartu dira eredutzat. Aipatu behar da, izatez, proiektu honetan sortutako simulatzailearen azken bertsioak ez dituela hauen ideia guztiak barneratu, baino era batean edo bestean lagungarri izan dira oso. Simulatzaile horien guztien artetik bi goraipatuko nituzke, *ns-3* eta *OMNeT++* izenekoak.

ns-3 simulator

Simulatzaile hau, gure simulatzailearen antzera, gertaera diskretuetan oinarritzen den sare-simulatzailea da [5]. 2005ean *ns*-ren 3.bertsio bezala merkaturatu zen, *C++* programazio lengoaia erabiliz sortua izan zena. Simulatzaile hauen erabilerari dagokionez, gehienbat ikerketa eta irakaskuntza arloetan erabili ohi den software irekia da, GNU GPLv2-ren esku dagoena *ad-hoc* sare mugikorren ikerketa, garapen eta erabilerarako. Azken bertsio honek (ns3) haririk gabeko edota kablezko sareen hainbat protokolo inplementatzen ditu eta horrez gain, simulazioaren lan-fluxu guztia jasan dezake, konfigurazio eta tramen analisi eta bilketa, esaterako.

OMNeT++

Simulatzaile hau ere, aurreko kasuan aztertu dugun bezala, objektuetara zuzendutako eta gertaera diskretuak tratatzen dituen sare-simulatzailea da

[6]. Erabilera asko izan ditzake eta horien artean aipatu behar dira: telekomunikazio sareen trafikoa, protokolo desberdinak, multiprozesadore edota sistema banatuak, softwareen errendimenduaren ebaluazioa edo gertaera diskretuekin simulatu daitekeen edozein sistema modelatzeko erabiltzen direla.

Sare-simulatzailea dela aipatzean, kasu honetan ere haririk gabeko edota kablezko sareen simulazioetaz ari garela azpimarratu behar da, besteak beste. Horrez gain, goraipatu behar da erreminta hau LINUX zein Windows sistemetan erabilgarri dagoela eta OMNEST izenez ezagutzen den bere bertsio komertziala Simulcraft Inc. etxeak garatzen du gaur egun. Gainera, denbora errealeko simulazioetarako, sareko emulaziorako, JAVA edota C# ordezko programazio lengoaientzako, datu baseen integraziorako eta beste zenbait funtzioentzako hedadurak eskaintzen ditu.

4. KAPITULUA

PETRI SAREAK

4.1 Sarrera

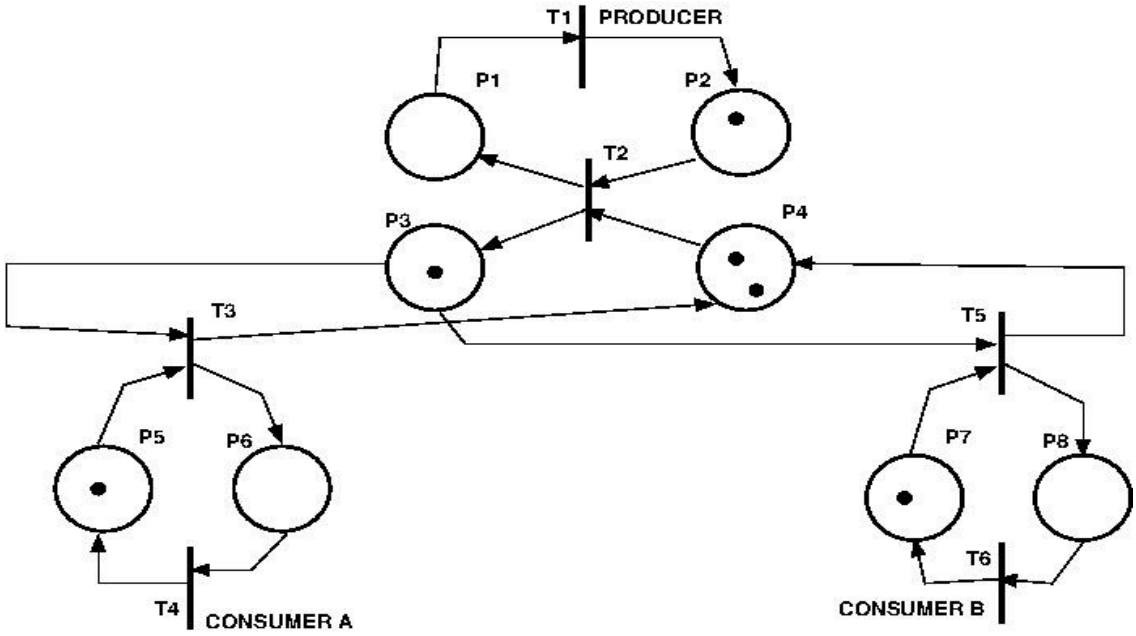
Sistema informatikoen oinarrizko ereduan, bi puntu dira goraipatu beharrekoak: baliabideak lortzeko gatazkak eta hainbat ekintza konkurrenteen sinkronizazioa gauzatzea. Ilara sareak erabiltzeak baliabideen arazoa konponduko digun arren, ekintzen sinkronizazioa ez dute menperatuko. Horretarako erabiltzen dira Petri Sareak, zehazki sistema konkurrenteen arteko sinkronizazioak gauzatzeko. Petri Sare klasikoez gain, analisi markoviarren teknikak aplikatzea ahalbidetzen duten denborazko Petri Sareak ere existitzen dira.

4.2 Petri Sare klasikoak

Petri Sare bat (*PN*) formalki laukote baten bidez definitzen da (*P, T, I, O*) [7] [8] [9]:

- *P* lekuen multzoa izango da (ingelerazko Places),
- *T* trantsizioen multzoa (ingelerazko Transitions),
- $I \subset P \times T$ trantsizio bakoitzaren sarrerak (ingelerazko Input),
- $O \subset T \; x \; P$ trantsizio bakoitzaren irteerak (ingelerazko Output).

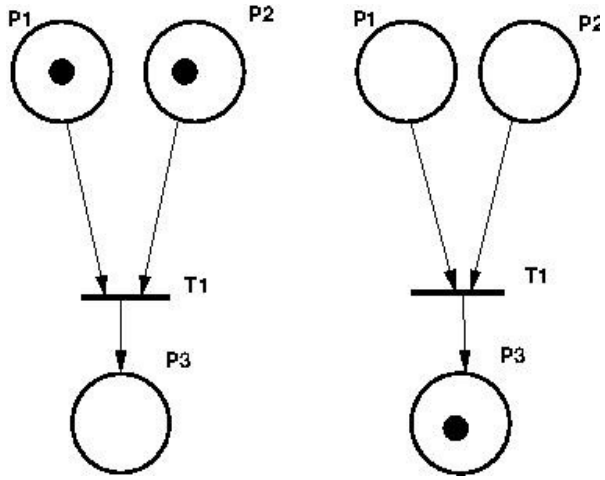
PN baten markaketa $n=(n_1,\ldots,n_m)$, $m=|P|$ leku bakoitzari ezartzen zaizkion token edo markek osatuko dute, marka hori adierazten duen zenbakia ez negatiboa izango delarik. Horrez gain, *PN* bat zirkulu, arku eta marra zuzen edo laukizuzenekin adierazten da grafikoki, honako adibidean ikus daitekeen moduan:



Adibidea 4.2.1: Petri Sarea (bezero-zerbitzari eredua)

Irudiko zirkulu bakoitzak leku bat adierazten du eta marra beltz bakoitzak, berriz, trantsizio bat. Horrez gain, gezi batekin adierazitako arkuak sarrera eta irteera-arkuak dira: zirkuluetatik marra zuzenetara (lekuetatik trantsizioetara) doazenak sarrera edo *input* arkuak eta alderantziz (trantsizioetatik lekuetara) doazenak, aldiz, irteera edo *output* arkuak.

PN baten dinamikak *token* edo markekin erlazio zuzena izango du. Esaterako, $i_1,\ldots,i_k$ sarrera-lekuak eta $i_{1,}\ldots,i_o$ irteera-lekuak dituen ***t*** trantsizio jakin baterako, trantsizioaren tiroa gertatzeko baldintza, sarrera-leku bakoitzak gutxienez marka bat izatea da. Trantsizioaren tiroaren ondorioz, sarrera-leku bakoitzak marka bat galduko du eta irteera-leku bakoitzak marka bat irabazi, guztira *K* eta *L* marka kopurua, hurrenez hurren.

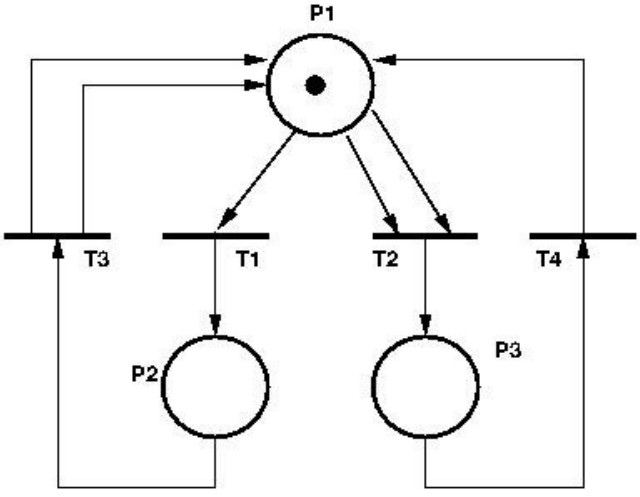


Adibidea 4.2.2: PN baten trantsizioaren tiroa

Aurreko adibidean trantsizio bat jaurtitzeko bete beharreko baldintzak aipatu dira. Baldintza horiek betetzen badira, hau da, trantsizioak tiro egiteko aukera duenean trantsizio hori *gaitua* dagoela esango da. Hainbat trantsizio aldi berean gaitzeko aukera dagoenean, PN-aren dinamika ez-determinista izango da, honek PN-aren hainbat bilakaera posible egotea suposatzen duelarik. Honela, bi trantsizioen arteko ***gatazka*** dago trantsizio bat jaurtitzearen ondorioz beste bat ezgaitzen den kasuan.

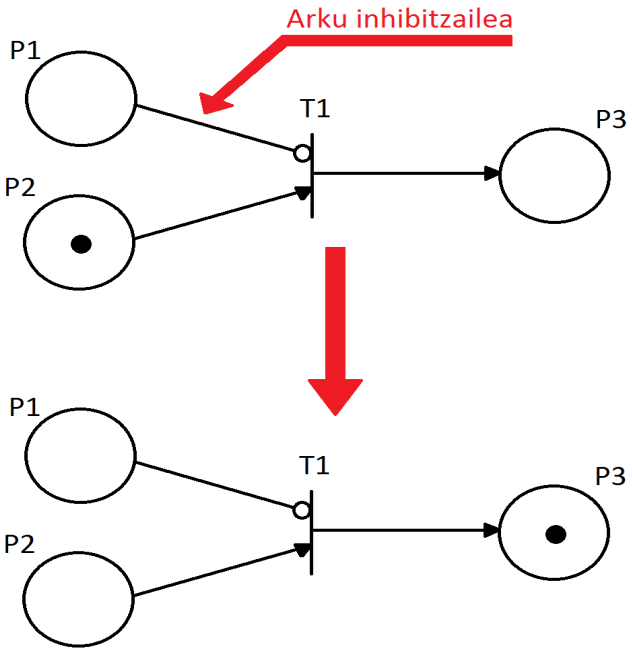
Trantsizioen etengabeko tiroek markaketa sekuentzia bat sortzen dute $n^{(i)} \mid i=0,1...$, sistemaren denboran zeharreko eboluzio bat bezala uler daitekeena (gertaera diskretuen segida bat) eta orokorrean ez-determinista izango dena: hasierako markaketa baterako trantsizioek sortutako hainbat markaketa sekuentzia posible existitzen dira.

Petri Sare klasikoen artean hedapenak ere, egon, badaude. Esaterako, trantsizio baten tiroaren ondorioz marka edo *token* bat baino gehiago galtzen duten sarrerako lekuak edota marka bat baino gehiago irabaziko duten irteera lekuak. Horren adibide, leku eta trantsizio artean (eta alderantziz) hainbat arkuz osatuta dauden Petri Sareak dira. Arkuek *multiplizitate* izeneko ezaugarri bat izango dute, zehazki arku kopurua zein den adieraziko duena.



Adibidea 4.2.3: Petri Sare klasikoa (hedapen bat)

Petri Sare mota desberdinen artean **arku inhibitzaileak** dituzten PNak ere aurki ditzakegu. Arku inhibitzaileak leku bat eta trantsizio bat elkar konektatzen dituen arkuak dira (beti lekuetatik abiatuz). Gainontzeko arkuekin gertatzen den bezala marra baten bidez adierazten dira, baino kasu honetan arkuen amaieran geziak erabili ordez borobilak irudikatzen dira dagokion trantsizioan. Irudi honetan ikus daiteke arku inhibitzailea duen Petri Sare baten portaera:



Adibidea 4.2.4: Arku inhibitzailea duen PN

Ohiko arkuak erabiltzen dituzten Petri Sareetan, lehenago aipatu den moduan, trantsizio bat gaitua egon dadin jatorrizko lekuek gutxienez marka bana eduki beharko dute. Hala ere, irudiak aztertzen badira, ikus daiteke sarrera-arku inhibitzailea duen T1 trantsizioa gaitua dagoela egoera horretan (P3-k marka irabaziko du), arku inhibitzailearen jatorri den lekuak markarik ez duelako hain zuzen ere. Beraz, arku inhibitzaileak dituzten trantsizioak gaitzeko kontrako egoeran egon beharko da Petri Sarea, hau da, P1 lekua markarik gabe eta P2 lekuak gutxienez marka bat izan beharko du.

4.3 Denborazko Petri Sareak

Petri Sare klasikoetan gertatzen ez den bezala, denborazko Petri Sareetan trantsizioei denborak esleitzen zaizkie [7] [8] [10]. Trantsizio batek tiro egin aurretik pasa beharko den denbora hori konstantea izateaz gain, mugagabea ere izan daiteke. Zenbait aukera aipatzekotan, uniformea, esponentziala edota logaritmikoa, besteak beste. Denborazko *PN* batean bi semantika mota nagusitzen dira:

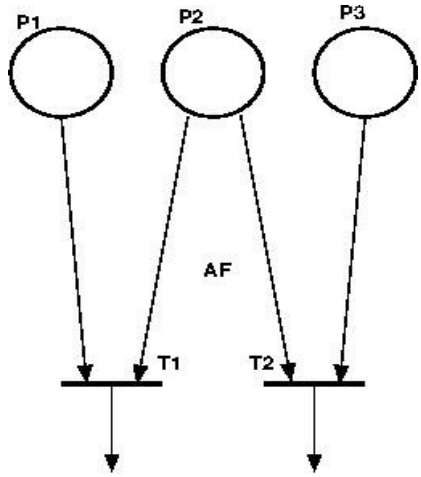
– Jaurtiketa atomikoaren semantika (*AF semantika*)

– Atomikoa ez den jaurtiketaren semantika (*NF semantika*)

■ AF semantika

Demagun *t* trantsizio konkretu bat aztertzen ari garela. Trantsizioaren tiroa bere denbora agortzean gauzatuko da, beti ere une horretan *t* gaitua baldin badago. Hori horrela bada, sarrera lekuek dagozkien markak galdu eta irteera lekuek markak irabaziko dituzte *t* trantsizioaren tiroaren ondorioz.

AF semantikan oinarritzen diren Petri Sareetan, aipatutako trantsizioaren jaurtiketa-denbora berehalakoa edo nulua duten *PN*-ak defini daitezke, grafikoki marra fin baten bidez adieraziko direnak, hain zuzen ere. Horrez gain, tiro egiteko gaitua dauden trantsizioen arteko gatazkak lehentasun edo probabilitate irizpide baten bidez ebazten dira.

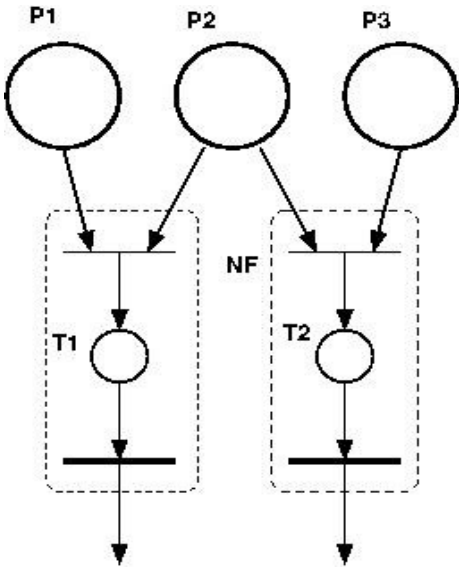


Adibidea 4.3.1: AF semantika duen PN

■ NF semantika

NF semantikaren kasuan, *t* trantsizio bat gaitzen den une berean, *t*-ri dagozkien sarrera lekuek beren markak galtzen dituzte. Ondoren, behin jaurtiketa-denbora igaro denean, irteera lekuek dagozkien markak irabaziko dituzte. Izatez, badago AF semantika eta NF semantikaren arteko erlazio zuzena. Izan ere, NF semantika jarraitzen duen *PN (P, T, I, O)* sarea berehalako leku eta trantsizioekin ordezkatzen bada, PN (*P', T', I', O'*) sare baliokidea lortzen da, AF semantika izango duena.

Berez, NF semantika duen PN bat sinpleagoa izango da AF semantika jarraitzen duen *PN* bat baino. Hala ere, NF semantikarekin adierazi ezin daitezkeen zenbait egoera existitzen dira, *lasterketa baldintza* (*race condition*) delakoa, adibidez.



Adibidea 4.3.2: NF semantika duen PN

PN bati dagokion *t* trantsizio batean ***gaikuntza aniztuna*** gertatuko da, *t*-ri dagozkion sarrera-leku bakoitzak 2 edo marka gehiago dituztenean eta *t*-ren gaikuntza bakoitzak ekintza bat ordezkatzen duenean. Ilaren terminologia erabiliz, aipatutako gaikuntza aniztun hau bi eratan adieraz daiteke:

- Zerbitzari bakarreko semantika (SS). Ekintzak zerbitzari bakar baten bidez zerbitzatzen diren kasua.
- Atzerapen zerbitzariaren semantika (DS). Ekintzak, atzerapen zentro (infinitu zerbitzari) baten bidez zerbitzatutako bezeroak diren kasua.

Azkenik, aipatu, denborazko semantikaren SS/DS sailkapena AF/NF sailkapenarekiko ortogonala dela. Proiektu honetan AF semantikako Petri Sareak kontsideratuko dira.

4.4 GSPN

GSPN, Petri Sare estokastiko orokortua, sistema markoviarrean oinarritzen den denborazko PN bat da [7] [10]. GSPN batek definitzen duen prozesu estokastikoa, denbora jarraituko Markov-en kate bat da (CTMC). CTMC honen egoera desberdinak denborazko PN-aren markaketen menpekoak dira (0, 1... bezala zerrendatzen ditugu). Horrez gain, semantikari dagokionez, denborazko semantika AF motakoa da eta bestalde, denborazko eta berehalako trantsizioak egon daitezke Petri Sarean.

Aipatutako trantsizioen denbora-banaketa esponentziala ez den kasuetan, Erlang-en banaketa adarkatuak eredu hipo-esponentzial eta hiper-esponentzialak eskaintzen ditu. Guzti horretan oinarrituta, GSPN bat

lortzeko egin beharrekoa honakoa da: denborazko PN originalari Erlang-en banaketa adarkatuaren eredua erreproduzitzen duten leku eta trantsizioak gehitzen zaizkio, PN originalaren tiroen denboretan eragin zuzena dutelarik.

Askotan, GSPN batean, berehalako trantsizioak soilik erabiltzen dira. Horren arrazoia, sinkronizazio batzuek hori besterik onartzen ez dutela da alde batetik, eta eredua sinplifikatzearena bestetik (trantsizioak berehalakoak ez badira ere denbora minimoak ezarriz). Honela, hainbat trantsizio aldi berean gaitzen diren kasuan, probabilitate batzuen arabera jaurtiko da horietako bat edo beste.

Azkenik, GSPN batean aurki ditzakegun egoerei dagokienez, hautemangarriak eta sinpleak bereizten dira. Berehalako trantsizio guztiak gaitu gabe dauden kasuan egoera bat hautemangarria izango da eta trantsizio horietako bat gutxienez aktibatuta dagoenean, berriz, egoera sinplea.

4.5 Sistema baten errendimendu parametroak

Orokorrean sistema baten simulazioa egitean, helburu nagusienetako bat errendimendu parametroak estimatzea izan ohi da [10]. Horregatik, garatutako software honen jarraipen nagusi gisa, egindako simulazioaren kalkulu estatistiko zehatzen funtzionalitatea gehitzea izan daiteke. Errendimendu parametroak definitzeko eredua zehaztea ezinbestekoa da eta Petri Sareen eredua orokorregia izan daiteke. Hori dela eta, sistema-mota berri bat aztertzea izango da irtenbideetako bat, '*kola sareak*' edo '*ilara-sareak*' izenez ezagutzen direnak, alegia.

Kola sareen eredu orokor batean, bezeroak zerbitzatzen diren estazioen multzo bat egon ohi da eta sistema itxia edo irekia izango da. Denbora jarraituko prozesu estokastiko bat har dezakegu $X(t), t \geq 0$ non $X(t)=(X_1, X_2 ... X_M)$ sistemaren egoera den eta X_i , i. estazioko bezero kopurua izango den.

Aipatutako *i* estazio horietako bakoitzerako kola sare batean kalkula ditzakegun errendimendu parametro posible batzuk honako hauek izan daitezke:

- Estazio konkretu baterako iriste-tasa λ_i , irtete-tasaren berdina izango dena eta estazioaren produktibitatea edota ingelerazko *throughput* hitzez ezagutzen dena($X_i = \lambda_i$).

- Estazioan dagoen bezero baten zerbitzu denboraren itxaropena, S_i .

- Bezero ilararen luzeraren batezbestekoa (zerbitzatzen ari den bezeroa kontuan hartuz), $Q_i = L_i$.

- Erantzun denbora, bezero bat kolan egoten deneko denboraren itxaropena (zerbitzua kontuan hartuz), $R_i = W_i$.

- Estazioaren erabilera (zerbitzatzen ari bada), estazioak zerbitzua eskaintzen dueneko denbora frakzioaren batezbestekoa, U_i .

Kola sareetan bereizi daitezkeen estazio-mota desberdinak aurkitzen dira, horien artean nagusienak aipatuko ditugularik. Alde batetik, zerbitzari bakarreko kola mota ohikoena den ahalmen finkoko zerbitzugunea daukagu. Bestalde, atzerapen-gunea edo infinitu zerbitzariren baliokidea dena; estazio mota hauetan lehen aipatutako $R_i = S_i$ berdintza emango da. Azkenik, kargaren menpeko zerbitzugunea bereizi behar da, adibidez, hainbat zerbitzari dituen estazio bat.

Horrez gain, estazio bakoitzean hainbat erlazio ematen dira eta horiek ulertzeko bi lege nabarmentzen dira. Little-n legeak dionez, sistema egonkor batean bezeroen batezbesteko denbora (Q_i), estazio zehatz batean aurkitzen den bezero kopurua (X_i) eta bezero batek sisteman zerbitzatzen irauten duen batezbesteko denboraren (R_i) arteko biderkadurak emango digu: $Q_i = X_i \cdot R_i$. Bestalde, erabilpen legearen arabera, estazioaren erabilpena (U_i) estazioko bezero kopurua (X_i) eta estazioko bezero baten zerbitzu denboraren itxaropenaren arteko biderkadurak (S_i) ematen du, $U_i = X_i \cdot S_i$.

Bezero batek kola sare batean oinarritutako sistema guztietan ziklo bat osatuko du, estazioen sekuentzia finitu bat, hain zuzen ere. Orokorrean, garraiatzen diren estazioak eta hauen kopurua zorizkoa izango da. Sistema irekia bada, ziklo horri hasiera ematen dion estazioa bezeroa iritsi denekoa izango da eta amaiera ematen diona, berriz, irten den estazioa. Bestalde, sistema itxia den kasuan, bezeroak zikloa hasi eta amaitzen duen estazio berezi bat egongo da. Hori horrela izanik, ziklo batean bezero batek *i*. estaziora egindako bisita kopuruaren itxaropena defini dezakegu (V_i).

Beraz, kola sareak erabiliz kalkulatu ahal izango diren parametroak asko dira eta proiektu honetan garatutako simulazioari funtzionalitate hau gehitzea jarraipen posible bat izango litzateke, besteak beste: estazioen eta sistemaren produktibitatea (*throughput*), bezeroen batezbestekoak, erantzun-denborak, etab.

5. KAPITULUA

MULTIPROZESADOREAK

Proiektu honetan garatutako simulatzailearen adibide gisa multiprozesadore bati dagokion GSPN Petri Sare bat eraiki dugunez beharrezkoa ikusten da kontzeptu hori definitu eta ezaugarri nagusiak aztertzea. Hori dela eta, kapitulu honetan zehar gai horren inguruan arituko gara, ondoren simulatzaileak egingo duena zehatz-mehatz ulertu ahal izateko.

5.1 Zer da multiprozesadore bat?

Multiprozesadore hitzez ezagutzen dira bi mikroprozesadore (PUZ) edo gehiago dituzten konputagailuak [11]. Makina hauek prozesu bati edo hainbat prozesuri dagozkien hariak exekutatu ahal izango dituzte aldi berean. Hala ere, multiprozesadoreek zenbait diseinu arazo izan ohi dituzte, prozesadore bakarreko konputagailuen diseinuarekin gertatzen ez den bezala. Arazo nagusia, gehienetan, aldi berean exekutatzen ari diren bi programa desberdinek memoria irakurri edo idazterako garaian dituzten koherentzia arazoetan zentratzen da. Hori horrela izanik, arazo horri aurre egiteko memoria partekatuko sistemetan bi arkitektura-mota bereizten dira, NUMA eta SMP arkitekturak hain zuzen ere.

5.2 NUMA eta SMP arkitekturak

NUMA [12] (Non-Uniform Memory Access edo Non-Uniform Memory Architecture) motako arkitekturan prozesadore bakoitzak memoria zati bat bakarrik atzitu eta kontrolatu ahal izango du. Prozesadore batek bere memoria lokala atzitzeko behar duen denbora txikia izango da memoria ez-lokala(beste prozesadore baten memoria lokala edo prozesadore guztien memoria partekatua) atzitzeko denborarekin alderatzen badugu. Horrez gain, prozesu batek memoria atzitzeko beste prozesu baten posizio erlatiboa erabiliko du.

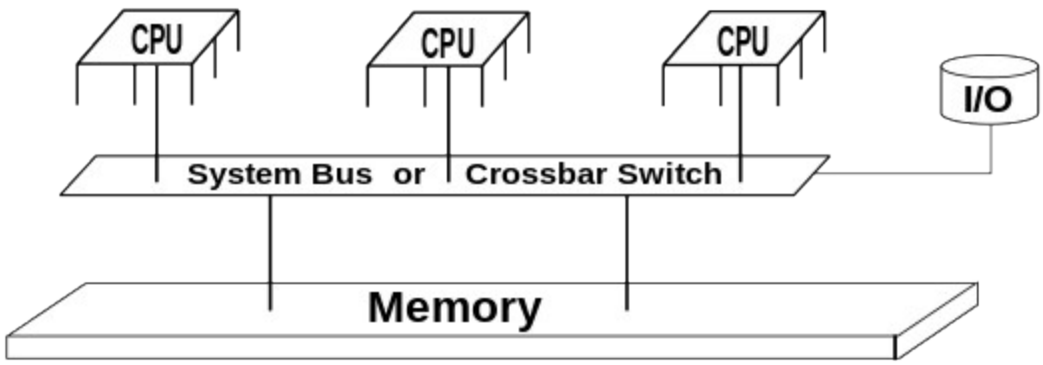
Gaur egungo errendimendu handiko konputagailuen gakoa, memoriara egindako atzipen kopurua mugatzean zentratzen da. Honek prozesadoreek Cache memoriaren abiadura handia eta Cache-ko errore-tasa txikia mantentzea behartzen ditu. Hala ere, denborak aurrera egin ahala, sistema

eragileen eta hauetan exekutatzen diren aplikazioen tamainak gora egiteak Cache-aren prozesamenduan lortu diren hobekuntza gehienak larritzea eragin du. Gainera, multiprozesadore sistemen etorrerak arazo hori areagotu dute, izan ere, oraingo sistemek hainbat prozesadore blokeatu behar baitituzte prozesadore bakar bat memoria atzitzen ari den bitartean.

NUMA arkitektura, prozesadore bakoitzarentzat memoria partekatua eskainiz, arazo hori ekiditen saiatzen da, hainbat prozesadore aldi berean memoria bera atzitzen saiatzen direnean sistemaren errendimendua mantenduz.

SMP arkitekturan [13] (UMA, Uniform Memory Access, izenez ere ezagutzen dena) oinarritzen diren konputagailuetan bi prozesadore edo gehiagok memoria nagusia partekatzen dute baldintza berdinetan eta horregatik esleitzen zaio simetriaren kontzeptua. SMP sistemek edozein prozesadorek edozein atazetan lan egitea ahalbidetzen dute hauen memoriako posizioa kontuan hartu gabe eta horrez gain, ataza horiek prozesadoreen artean modu eraginkor batean mugitu ahal izango dira. Mikroprozesadoreak memoriarekin elkar komunikatzeko bus partekatu bat erabiltzen dute. Bus hori une bakoitzean prozesadore bakarrak erabil dezakeenez, arbitro bat beharko da erabaki hori hartzeko.

Multiprozesu simetrikoen kasuan (SMP arkitektura) prozesadoreek memoria bera partekatzen dutenez, horren ondorioz horietako batean exekutatzen den kode-zati batek beste prozesadore baten memoriari eragin diezaioke. Hori dela eta, programa batean erabiltzen den aldagai baten balioa ezin izango da ziurtatu hurrengo lerroan beste prozesadore batek balio hori alda dezaken kasuan. Arazo horri aurre egiteko prozesuen arteko sinkronizazioa edota semaforoak bezalako egiturak erabil daitezke. Bestalde, prozesuen programazioan ez dugu arazo hau aurkituko, izan ere, prozesu bat normalean prozesadore bakar batean exekutatuko baita.

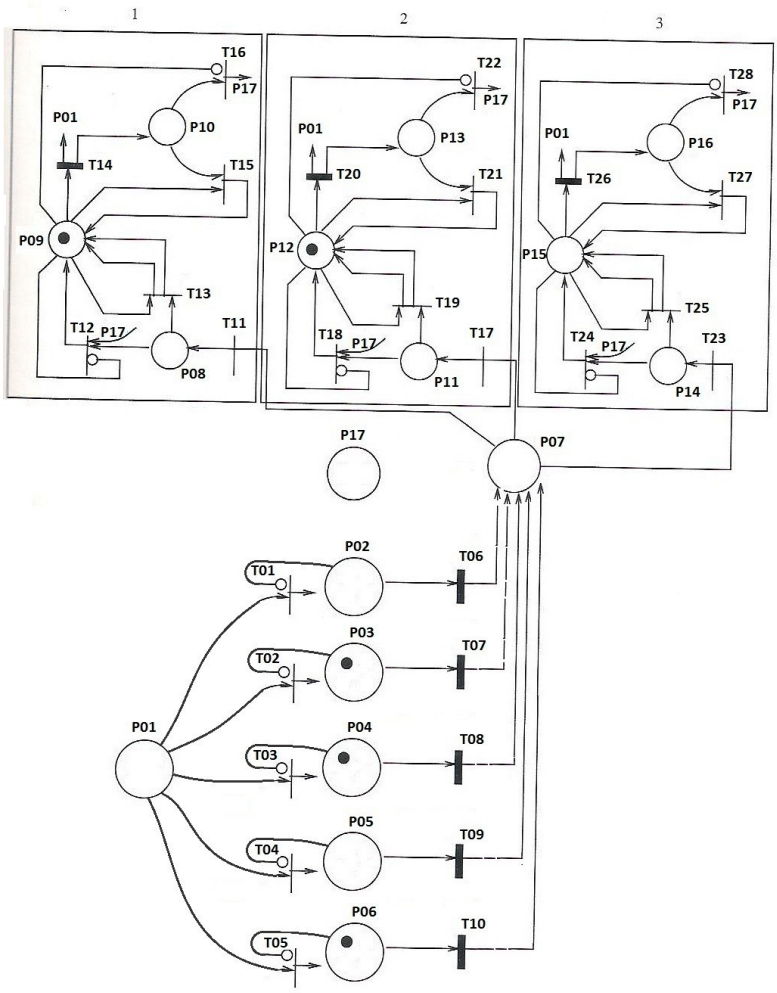


5.2.1 irudia: memoria partekatudun sistema

5.3 Multiprozesadore sistemari dagokion denborazko Petri Sarea

Proiektu honetan programatutako simulatzailea, esan bezala, multiprozesadore sistema batean prozesuen memoria atzipena simulatzeko erabiliko da. Memoriari dagokionez, hiru modulu bereizten dira eta prozesu batek ausaz hautatuko du une bakoitzean atzituko duen modulu horietako bakoitza. Hala ere, prozesu horietako batzuek cache memorietatik exekutatuko dira eta ez da beharrezkoa izango memoria-modulu partekatuetan sartzea.

Hurrengo irudian sistemaren Petri Sarearen eskema erakusten da. Multiprozesadorean bost prozesu exekutatuko dira, dagozkien lekuek eta trantsizioek irudian adierazten dutenez, eta bi bus izango dira hiru moduluen artean partekatzeko, bus kopuru hau irudian dugun hasierako markaketak definitzen du. Leku eta trantsizio bakoitzak esanahi bat izango du eta jarraian azaltzen dira horiek guztiak eta simulazio prozesua.



5.3.1 irudia: multiprozesadore sistemaren petri sarea

Cache lokaletatik exekutatzen diren prozesuak:

- P01: Memoria modulu baten erabilera amaitu duten prozesuak

- P02: Memoria modulurik erabili behar ez duen bakandutako prozesua 1. leku birtualean

- P03: Memoria modulurik erabili behar ez duen bakandutako prozesua 2. leku birtualean

- P04: Memoria modulurik erabili behar ez duen bakandutako prozesua 3.

leku birtualean

- P05: Memoria modulurik erabili behar ez duen bakandutako prozesua 4. leku birtualean

- P06: Memoria modulurik erabili behar ez duen bakandutako prozesua 5. leku birtualean

Memoria partekatua erabiliko duten prozesuak:

- P07: memoria-modulua aukeratu

A memoria moduluari dagozkien lekuak:

- P08: 1. modulua hartzeko prest

- P09: 1. modulua eta bus bat hartuta duten prozesuak

- P10: 1. moduluak hartutako busa, prozesu batek moduluaren erabilera amaitu ondoren

B memoria moduluari dagozkien lekuak:

- P11: 2. modulua hartzeko prest

- P12: 2. modulua eta bus bat hartuta duten prozesuak

- P13: 2. moduluak hartutako busa, prozesu batek moduluaren erabilera amaitu ondoren

C memoria moduluari dagozkien lekuak:

- P14: 3. modulua hartzeko prest

- P15: 3. modulua eta bus bat hartuta duten prozesuak

- P16: 3. moduluak hartutako busa, prozesu batek moduluaren erabilera amaitu ondoren

Free busses (erabilgarri dauden busak):

- P17: libre dauden busak

Trantsizioen esanahia:

- *T01: Prozesu bat bakandu 1. leku birtualera*

- *T02: Prozesu bat bakandu 2. leku birtualera*

- *T03: Prozesu bat bakandu 3. leku birtualera*

- *T04: Prozesu bat bakandu 4. leku birtualera*

- *T05: Prozesu bat bakandu 5. leku birtualera*

- *T06: 1. leku birtualeko prozesuaren operazio/itxoite denbora, memoria eskaria egin aurretik*

- *T07: 2. leku birtualeko prozesuaren operazio/itxoite denbora, memoria eskaria egin aurretik*

- *T08: 3. leku birtualeko prozesuaren operazio/itxoite denbora, memoria eskaria egin aurretik*

- *T09: 4. leku birtualeko prozesuaren operazio/itxoite denbora, memoria eskaria egin aurretik*

- *T10: 5. leku birtualeko prozesuaren operazio/itxoite denbora, memoria eskaria egin aurretik*

1. memoria moduluari dagozkion trantsizioak:

- *T11: 1. modulua aukeratu*

- *T12: bus berri bat okupatu eta modulua erabili*

- *T13: prozesu berria 1. modulua erabiltzera, aurretik gutxienez beste prozesu bat modulua erabiltzen ari dela*

- *T14: prozesu baten eragiketa denbora 1. moduluan*

- *T15: 1. moduluak hartutako busak okupatuta jarraitu beste prozesu batentzat*

- *T16: busa askatu 1. moduluak*

2. memoria moduluari dagozkion trantsizioak:

- *T17: 2. modulua aukeratu*
- *T18: busa okupatu*
- *T19: prozesuak ez du busa okupatu*
- *T20: 2. moduluan eragiketak egin*
- *T21: 2. moduluan zain dagoen prozesu batentzat busa askatu*
- *T22: busa askatu kanpoko prozesu batentzat*

3. memoria moduluari dagozkion trantsizioak:

- *T23: 3. modulua aukeratu*
- *T24: busa okupatu*
- *T25: prozesuak ez du busa okupatu*
- *T26: 3. moduluan eragiketak egin*
- *T27: 3. moduluan zain dagoen prozesu batentzat busa askatu*
- *T28: busa askatu kanpoko prozesu batentzat*

6. KAPITULUA

SIMULATZAILEAREN PROGRAMAZIOA

6.1 Zorizko aldagaiak

Simulazio prozesuan erabil daitezkeen aldagai desberdinak aipatu dira aurreko kapituluan [10] [14]. Hala ere, *zorizko aldagaietan* eta hauen *simulazioan* zentratuko gara kapitulu honetan, simulatzailearen programazioaren nondik norakoak azaltzeaz gain. Estatistika eta probabilitatearen arloan, zorizko aldagai edo aldagai estokastikoa, zorizko esperimentu batean egindako neurketetan lortzen diren balioek osatzen duten *aldagai estatistikoa* da. Hala ere, formalki, zenbaki errealei gertaerak esleitzen dizkion funtzio bezala ere ikusten da. Hau da, X zorizko aldagaia, zorizko esperimentu bati lotutako Ω lagin-espazioan definitzen den funtzio erreal bat da:

$$X : \Omega \rightarrow \Re$$

Zorizko aldagai baten balio posibleek, egiteke dagoen esperimentu baten emaitzak ordezkatzen ditu orokorrean. Intuitiboki, balio zehatz bat ez duen eta hainbat balio har ditzakeen aldagai bat bezala onar daiteke, balio desberdin horiek lortzeko probabilitatea deskribatzeko probabilitate banakuntza bat erabiltzen delarik. Horrez gain, zorizko aldagaien tratamendu estatistikorako, zorizko hainbat esperimentu erabiltzea beharrezkoa izango da, hauek emandako emaitza bakoitzari zenbaki erreal bat esleituko zaiolarik. Honela, lagin-espazioaren elementuen eta zenbaki errealen arteko erlazio funtzional bat sortu ahal izango da.

Azkenik, aipatu, mota desberdinetako zorizko aldagaiak aurki ditzakegula: zorizko aldagai diskretua alde batetik eta zorizko aldagai jarraitua bestetik.

6.2 Zorizko aldagaien simulazioa

Garatutako aplikazioan, Petri Sareen trantsizioen jaurtiketak kontrolatzeko denbora-banaketa desberdinak erabili dira. Horietako batzuk zorizko

aldagaien simulazioan oinarritzen direnez, jarraian azalduko dira banaketa uniforme, esponentzial eta log-normal izenez ezagutzen diren zorizko banaketak simulatzeko algoritmoak, Java programazio lengoaian oinarrituz.

6.2.1 Banaketa uniformea

Estatistika eta probabilitate teorian [15], banaketa uniforme jarraitua, zorizko aldagai jarraituen probabilitate-banaketa multzo bat da. Domeinua, *a* eta *b* (balio minimo eta maximoa), bi parametro zehatzek osatzen dute eta orokorrean, banaketa $U(a,b)$ edota $unif(a,b)$ modu laburtuetan idatzita ager daiteke. Honela, jarraian erakusten den *X* zorizko aldagaiak, a eta b parametroak dituen banaketa uniforme jarraitua jarraitzen duela esan daiteke:

$$X \sim U(a,b)$$

Era uniformean banatutako $(unif(0,1))$ zorizko zenbaki-sekuentzia sortzeko software bat erabil daiteke, eta gure kasuan, *java.util* paketeko *Random* klasea oso erabilgarria izango da. Algoritmo deterministan oinarrituz, iturburua izango den hasierako zenbaki osoko batek aipatutako sekuentzia zehaztuko du. Iturburu hori zehazteko aukera hauek ditugu simulazioan:

- Algoritmoari hasierako zenbakia ematen zaio,
- softwareak konputagailuaren erlojuarengana jotzen du,
- aurreko simulazio baten emaitza bezala, etab.

6.2.2 Banaketa esponentziala

Zorizko aldagai jarraituak sortzeko erabiltzen den metodoetako bat *alderantzizko transformazioaren metodoa* izenekoa da [16]. Aurreko puntuan aipatutako zorizko aldagai uniformean oinarrituta, banaketa esponentzialaren balioak simulatzea ahalbidetzen digu metodo honek, zehazki pauso hauek jarraituz:

- $(0,1)$ tartean definitutako U zorizko aldagai uniformearekin,
- $F(X)$ banaketa-funtzioa duen X zorizko aldagaiarekin,

$$X = F^{-1}(U)$$

Banaketa esponentziala simulatzeko:

- $unif(0,1)$ banaketarekin zorizko u balioa sortu eta

$x = F^{-1}(u)$ esleitu

Lortzen den zorizko aldagai esponentzialaren balioa $X = -\left(\frac{1}{\lambda}\right)\log(1-U)$ izango da, $F(x) = 1 - e^{-\lambda x}$ banaketa funtzioa duena, hain zuzen ere.

F^{-1} Alderantzizko funtziorako espresio esplizitua existitzen ez den kasurako, *ukapen metodoa* erabil daiteke. Metodo honen planteamendua $f(x)$ dentsitatea duen X zorizko aldagaia sortu nahi izatea da. Gainera, $g(y)$ dentsitatea duen Y zorizko aldagaia sor daiteke.

Hori bai, *k* konstante konkretu batek bete beharko duen baldintza honakoa da:

$$f(x) \leq kg(x), \forall x$$

Hurrengo pausoa iteratzea izango da, jarraian azaltzen diren puntuak ordena honetan errespetatuz:

1. $Y = y$ balio bat sortu,
2. $U = u$ balio bat sortu, $unif(0,1)$ banaketarekin eta $X = y$ esleitu baldin:

$$u \leq f(y)/kg(y)$$

($f(y)/kg(y)$ probabilitatearekin onartu)

6.2.3 Log-normal banaketa

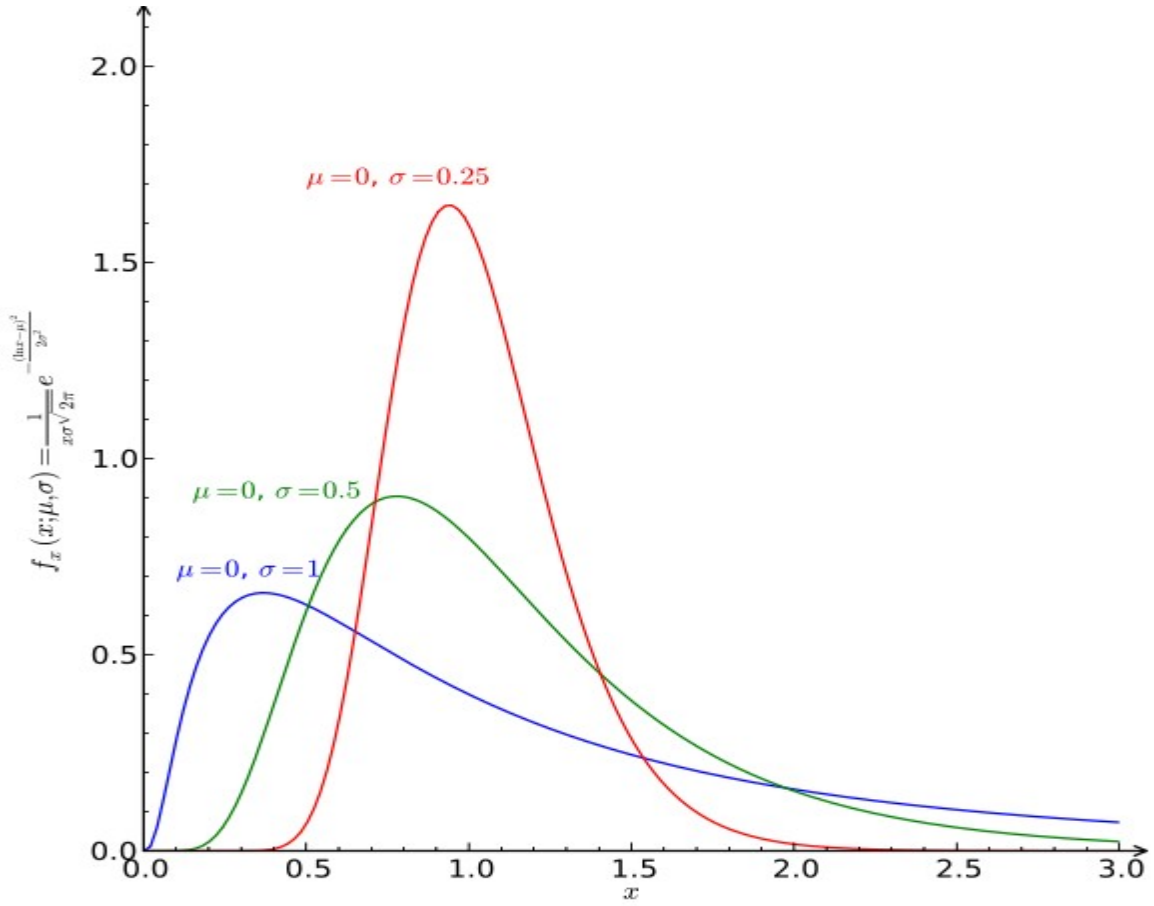
Zorizko aldagai ez-negatibo bat adierazteko banaketa esponentziala egokia ez denean, aldagaiaren bariantza hipo-esponentziala edo hiper-esponentziala delako, log-normal banaketa erabil genezake aldagaiaren probabilitate eredu bezala [14] [17].

Probabilitate-teorian, banaketa logaritmiko normala (*log-normal*), zorizko aldagaiaren logaritmoak banaketa normala jarraitzen dueneko probabilitate-banaketa da. Demagun *X*, banaketa normala jarraitzen duen zorizko aldagaia dela, orduan, $Y = \exp(X)$ espresioak *log-normal* banaketa izango du; hau da, *Y* banaketa logaritmiko normala denez, $X = \log(Y)$ espresioak banaketa normala jarraituko du (azken hau egiazkoa izango da edozein logaritmoren oinarrietarako, $\log_a(Y)$ -k banaketa normala badu, $\log_b(Y)$ -k ere banaketa normala jarraituko du, $a, b = 1$ kasuetan izan ezik).

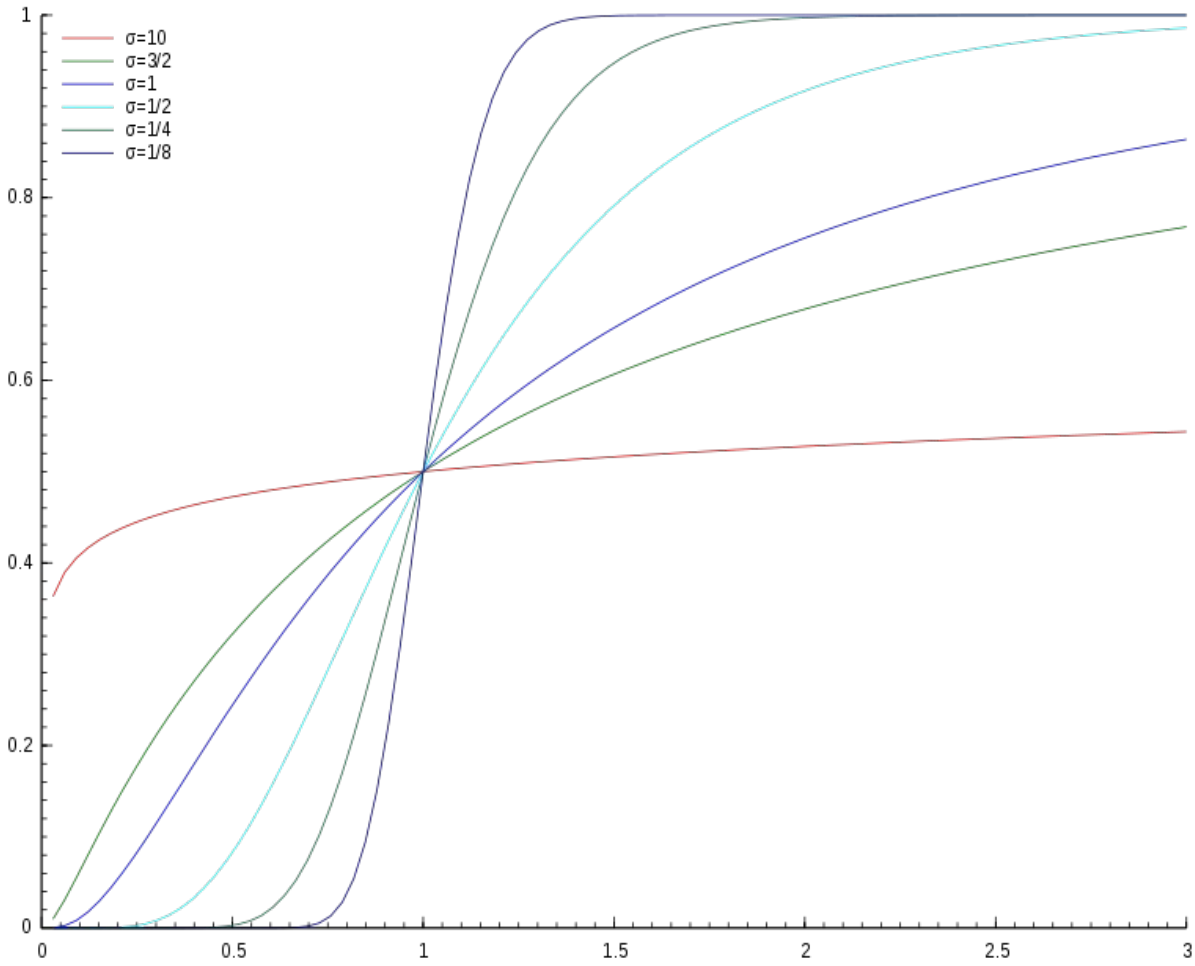
Izan bedi Z banaketa normala duen zorizko aldagaia, bere mediaren eta bariantzaren balioa 1 izanik. Orduan $\mu+\sigma Z$ aldagaiak banaketa normala izango du μ mediarekin eta σ desbiderapen estandarrarekin, eta *X* zorizko aldagaiak banaketa logaritmiko normala izango du:

$X = e^{(\mu+\sigma Z)}$, non Z , aldagai normal estandar bat den.

Probabilitate dentsitate funtzioari eta m*etatze-banaketa funtzioari dagozkien* grafikoak jarraian ikus ditzakegu. Lehenengoa, μ=0 balio finko eta σ parametroaren eskala desberdinak dituen *dentsitate-funtzioari* dagokiona da. Bigarrena, berriz, *metatze-banaketa funtzioari* dagokiona:



Grafikoa 6.2.3.1: Dentsitate-funtzioari dagokion grafikoa (μ=0)



Grafikoa 6.2.3.2: Banaketa-funtzioari dagokion grafikoa (μ=0)

Guzti honez gain, *log-normal banaketaren* balio garrantzitsuetako batzuk ere aipatu behar dira:

Batezbestekoa:

$$mean = e^{(\mu + \sigma^2/2)}$$

Bariantza:

$$Variance = e^{(\sigma^2 + 2\mu)}\left(e^{(\sigma^2)} - 1\right)$$

- ***LogNormal klasea***

Aplikazioak banaketa logaritmiko normalak kudeatzeko *LogNormal* izeneko klasea erabiltzen du. Horretarako, aurrerago ikusiko dugun *Random* klaseko *nextGaussian* metodoa ere erabiliko da.

Klasearen inplementazioa nahiko sinple eta laburra da, izan ere, eskura ditugun *mean* (batezbestekoa) eta *stdv* (desbiderapen estandarra) balioekin *mu* eta *sigma2* parametroak lortzen baitira lehenengo, eta ondoren *banaketa Gausiarra* erabiliz emaitza itzultzen baita. *LogNormal* klasearen kodea honakoa da:

```
import java.util.Random;

public abstract class LogNormal {

      public  double  nextLogNormal(Random  rand,  double  mean,
                                              double stdv) {

            double mu, sigma2;

            sigma2 = Math.log(1 + Math.pow(stdv / mean , 2));
            mu = Math.log(mean) - sigma2 / 2;
            return Math.exp(mu + Math.sqrt(sigma2) *
                                        rand.nextGaussian());

      }
}
```

6.2.4 Java.util.Random klasea

Banaketa uniformearen atalean aipatzen den moduan, *java.util* paketeko *Random* klasea erabiliko da zorizko zenbaki-sekuentziak sortzeko [18]. Klase honek, kongruentzia linealdun formula baten bidez aldatzen den 48 biteko iturburua erabiltzen du.

Klase honetako metodoen bidez simulatuko ditugu banaketa uniformea eta normala duten zorizko aldagaiak, eta gainerako aldagaien simulazioa metodo hauetan oinarrituz programatuko da.

Random klasearen bi instantzia iturburu *(seed)* berarekin eraikitzen direnean eta bakoitzarentzat egindako metodo-sekuentzia deia berdina den kasuan, eraiki eta itzuliko den zenbaki-sekuentzia berdina izango da. Honela, propietate hau mantentzekotan, *Random* klasearentzat algoritmo bereziak espezifikatzen dira. Guzti honez gain, aplikazio askok *random* metodoa *Math* klasea erabiliz dei dezakete.

Jarraian, klasearen metodo eraikitzaileak aztertuko ditugu, bakoitza bere

deskribapen eta kode zatiarekin. Eskura dauden eraikitzaileak honako taulan ikus daitezke:

Eraikitzaileak	Deskribapena
Random()	Zorizko zenbakien sortzaile berria eraikitzen du.
Random(long seed)	Zorizko zenbakien sortzaile berria eraikitzen du, algoritmoaren iturburua zehaztuz.

6.2.4.1 taula: Random klasearen eraikitzaileak

- **Random**

  Zorizko zenbaki sortzailea eraikitzen du. Iturburua finkatzeko, konputagailuaren erlojuan oinarritzen da:

```
public Random() {
      this(System.currentTimeMillis());
}
```

  Milisegundo berean sortutako zorizko bi objektuk itzultzen duten zenbaki-sekuentzia berdina izango da.

  Ikus:

  System.currentTimeMillis()

- **Random**

  Iturburu zehatz bat erabiliz, zorizko zenbaki sortzailea eraikitzen du.

```
public Random(long seed) {
      setSeed(seed);
}
```

  Parametroak:

seed- hasierako iturburua

Ikus:

setSeed

Ondorengo taulan *Random* klasea osatzen duten metodoak erakusten dira, bakoitza deskribapen labur batekin. Jarraian, horietatik, zorizko aldagaien simulaziorako erabilgarriak izango zaizkigun metodoen kodeak ere ikus ditzakegu:

Metodoak		Deskribapena
protected int	#nextInt	Hurrengo zorizko zenbakia sortzen du.
boolean	#nextBoolean	Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo balio boolearra itzultzen du.
void	nextBytes(byte[] bytes)	Zorizko byteak sortu eta erabiltzaileak hornitutako array batean gordetzen dira.
double	nextDouble()	Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun, hurrengo *double* motako eta [*0.0, 1.0)* tarteko balioa itzultzen du.
float	nextFloat()	Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun, hurrengo *float* motako eta [*0.0, 1.0)* tarteko balioa itzultzen du.
double	nextGaussian()	Zorizko zenbaki sortzailearen sekuentziako banaketa Gaussiarra duen hurrengo *double* motako, 0.0 media duen eta 1.0-ko desbiderapen estandarra duen balioa itzultzen du.
int	nextInt()	Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo *int* motako balioa itzultzen du.
int	nextInt(int n)	Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo *int* motako eta [*0.0, n)* tarteko balioa itzultzen du.
long	nextLong()	Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun

		hurrengo *long* motako balioa itzultzen du.
void	setSeed(long seed)	Iturburuaren balioa finkatzen du.

6.2.4.2 taula: Random klasearen metodoak

- **setSeed**

  Iturburuaren balioa finkatzen du. Zorizko zenbaki sortzailearen objektua aldatu egiten da argumentu bezala pasatako *seed* iturburua erabiliz; s*etSeed* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
synchronized public void setSeed(long seed) {

      this.seed = (seed ^ 0x5DEECE66DL) & ((1L << 48) - 1);
      haveNextNextGaussian = false;


}
```

  Parametroak: *seed*- iturburuaren balioa.

- **next**

  Metodo honek hurrengo *int* motako zorizko zenbakia itzultzen du. Argumentu bezala pasatako balioaren bit kopurua 1 eta 32 artekoa bada (biak barne), itzulitako balioaren maila baxuko bit asko era independentean aukeratzen dira eta horietako bakoitzak 0 edo 1 balioa izateko probabilitatea bera izango da; *next* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
synchronized protected int next(int bits) {

      seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);


      return (int)(seed >>> (48 - bits));
}
```

  Parametroak: *bits*- zorizko bitak

- **nextInt(int n)**

  Banaketa uniformea duen *int* motako zorizko zenbaki bat itzultzen du, zehazki *[0, n)* tarteko balio bat izango duena. Tarte horretako balio posible guztiek probabilitate bera izango dute; *nextInt* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
public int nextInt(int n) {
      if (n<=0)
        throw new IllegalArgumentException("n must be positive");

      if ((n & -n) == n)  // i.e., n is a power of 2
            return (int)((n * (long)next(31)) >> 31);

      int bits, val;
      do {
            bits = next(31);
            val = bits % n;
      } while(bits - val + (n-1) < 0);
            return val;
}
```

  Parametroak: *n*- itzulitako balioaren limitea. Positiboa izan behar du.

  Errorea: IllegalArgumentException- n positiboa ez den kasuan.

- **nextLong**

  Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo *long* motako balioa itzultzen du. Itzul daitezkeen 2^{64} l*ong* motako balio posible guztiek probabilitate bera izango dute; *nextLong* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
public long nextLong() {

      return ((long)next(32) << 32) + next(32);

}
```

- **nextBoolean**

  Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo balio boolearra itzultzen du. *True* eta *false* balioek, gutxi gora behera, emaitzan itzultzeko antzeko probabilitatea izango

dute; *nextBoolean* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
public boolean nextBoolean() {

      return next(1) != 0;

}
```

- **nextFloat**

Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo *float* motako eta [*0.0, 1.0)* tarteko balioa itzultzen du. $m \, x \, 2^{-24}$ (non *m*, 2^{24} baino txikiagoa den integer motako zenbakia den) formako 2^{24} *float* motako balio posible guztiek probabilitate bera izango dute; *nextFloat* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
public float nextFloat() {

      return next(24) / ((float)(1 << 24));
}
```

- **nextDouble**

Zorizko zenbaki sortzailearen sekuentziako banaketa uniformedun hurrengo *double* motako eta [*0.0, 1.0)* tarteko balioa itzultzen du. $m \, x \, 2^{-53}$ (non *m*, 2^{53} baino txikiagoa den integer motako zenbakia den) formako 2^{53} *float* motako balio posible guztiek probabilitate bera izango dute; *nextDouble* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
public double nextDouble() {

 return (((long)next(26) << 27) + next(27))/ (double)(1L << 53);

}
```

- **nextGaussian**

Zorizko zenbaki sortzailearen sekuentziako banaketa Gaussiarra duen hurrengo *double* motako, 0.0 media duen eta 1.0-ko desbiderapen estandarra duen balioa itzultzen du; *nextGaussian* metodoaren kodea honela inplementatzen du *Random* klaseak:

```
synchronized public double nextGaussian() {
      if (haveNextNextGaussian) {
            haveNextNextGaussian = false;
            return nextNextGaussian;
      } else {
            double v1, v2, s;
            do {
             v1 = 2 * nextDouble() - 1;   // between -1.0 and 1.0
             v2 = 2 * nextDouble() - 1;   // between -1.0 and 1.0
                  s = v1 * v1 + v2 * v2;
            } while (s >= 1 || s == 0);
               double multiplier = Math.sqrt(-2 * Math.log(s)/s);
                  nextNextGaussian = v2 * multiplier;
                  haveNextNextGaussian = true;
                  return v1 * multiplier;
      }
}
```

6.3 Simulatzailearen programazioa

Atal honetan, sistema konkurrenteak simulatzeko erabiliko den simulatzailearen programaren egitura azalduko da [10] [19] [20]. Horretarako, beharrezkoak diren klaseen inplementazioaren nondik norakoak argitzeaz gain, simulazioan zehar aurki ditzakegun egoera desberdinen tratamenduari ere erreparatuko zaio.

Esan bezala, kapitulu honek Java simulatzailea barneratzen du eta programazioaren oinarri izan arren, hurrengo kapituluetan azalduko dira Petri Sarearen definizioa XML dokumentuaren bidez, Petri Sarea eraikitzeko Java klase berezien espezifikazioa eta hauen instantziazioa XML dokumentutik, modu honetan azalpena ulergarriagoa delakoan.

6.3.1 Simulator klasea

Simulatzailearen nukleo gisa *Simulator* izeneko *klase estatiko nagusia* erabiliko da, klase honek, gertaera diskretu baten prozesua simulatzen duen aplikazioa inplementatzen duelarik. Simulator klaseak honako eskema jarraitzen du:

```
import java.util.LinkedList;
```

```
public class Simulator {

      public static void main(String[] args) throws Exception {

      // Create DocumentBuilderFactory and validate XML document
                  (0)

            SysEvent anEvent;
            LinkedList<SysEvent> theEventList, newEventList;
            double simulationTime = 1000.0;

            boolean printTrace = true;

            // Create the system
                  (1)
            // Order the initial event list
                  (2)
            // Iterate simulation
                  (3)
      }
      // Error handler to report errors and warnings
            (4)
}
```

- **(0) Lantokiaren konfigurazioa**

  Aplikazioa *DomEcho* programan oinarrituta dagoenez antzekotasun handiak izango ditu aipatutako aplikazioarekin, atributu estatiko berdinak erabiliko dituelarik [21].

```
  /** All output will use this encoding */
public static final String outputEncoding = "UTF-8";

/** Output goes here */
private PrintWriter out;

/** Constants used for JAXP 1.2 */
public static final String JAXP_SCHEMA_LANGUAGE =
    "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
public static final String W3C_XML_SCHEMA =
    "http://www.w3.org/2001/XMLSchema";
public static final String JAXP_SCHEMA_SOURCE =
    "http://java.sun.com/xml/jaxp/properties/schemaSource";
```

Hala ere, ez dira ahaztu behar fitxategia irakurtzeko, XML dokumentua aztertzeko edota errore-kontrola era egokian egiteko inportatu beharreko klaseak:

```
import java.io.*;
import java.io.PrintWriter;
import java.util.LinkedList;
import java.io.File;
import java.io.OutputStreamWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
```

Lehenengo pausoa aplikazioak jasoko dituen argumentu kopurua kontrolatzea izango da. Kasu honetan bi argumentu jasoko direnez honela egin daiteke:

```
if (args.length == 2){
      filename = args[0];
      simulationTime = Double.parseDouble(args[1]);
  }else{
      System.err.println("Incorrect number of arguments. Expected
                              2: XML file and simulation time. ");
  }
```

Ondoren, lantokia (*DocumentBuilderFactory*) sortzeaz gain, XML dokumentua balioztatzeko modua finkatu behar da. Aukera posibleak *DTD* bidez, *XSD* bidez (eskema) edota balioztatze hori ez egitea dira.

```
  // Step 1: create a DocumentBuilderFactory and configure it
    DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();

// Set namespaceAware to true to get a DOM Level 2 tree with nodes
// containing namesapce information. This is necessary because the
    // default value from JAXP 1.0 was defined to be false.
    dbf.setNamespaceAware(true);

    // Set the validation mode to either: no validation, DTD
    // validation, or XSD validation
    dbf.setValidating(dtdValidate || xsdValidate);
    if (xsdValidate) {
        try {
           dbf.setAttribute(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
        } catch (IllegalArgumentException x) {
        // This can happen if the parser does not support JAXP 1.2
```

```
        System.err.println("Error: JAXP DocumentBuilderFactory
        attribute not recognized: " + JAXP_SCHEMA_LANGUAGE);
        System.err.println("Check to see if parser conforms to
        JAXP 1.2 spec.");
            System.exit(1);
        }
    }
```

Jarraian, *JAXP* eskemari dagokion ***xsd*** fitxategia pasako zaio lantokiaren instantziari beste zenbait konfigurazio finkatzearekin batera. Gainera, dokumentua aztertzen hasi aurretik erroreak kontrolatzeko konfigurazioa egingo da *MyErrorHandler* klasea erabiliz. Hau egin eta gero, *DocumentBuilder* instantziako *parse* metodoari esker ***filename*** fitxategia aztertuko da, *Document* zuhaitz formako objektua itzultzeaz arduratuko dena, hain zuzen ere. Azkenik, ***PNFromXML*** klaseko ***buildPetriNet*** metodoari deia egingo zaio:

```
// Set the schema source, if any.  See the JAXP 1.2 maintenance
// update specification for more complex usages of this
    //feature.
//if (schemaSource != null) {
dbf.setAttribute(JAXP_SCHEMA_SOURCE, "petri_net.xsd");
//}

// Optional: set various configuration options
dbf.setIgnoringComments(ignoreComments);
dbf.setIgnoringElementContentWhitespace(ignoreWhitespace);
dbf.setCoalescing(putCDATAIntoText);
// The opposite of creating entity ref nodes is expanding them
                                                    inline
dbf.setExpandEntityReferences(!createEntityRefs);

// Step 2: create a DocumentBuilder that satisfies the
    //constraints
// specified by the DocumentBuilderFactory
DocumentBuilder db = dbf.newDocumentBuilder();

// Set an ErrorHandler before parsing
OutputStreamWriter errorWriter =
    new OutputStreamWriter(System.err, outputEncoding);
db.setErrorHandler(
    new MyErrorHandler(new PrintWriter(errorWriter, true)));

// Step 3: parse the input file
Document doc = db.parse(new File(filename));

// Build Petri Net
OutputStreamWriter outWriter =
    new OutputStreamWriter(System.out, outputEncoding);
PetriNet thePN = new PNFromXML().buildPetriNet(doc);
```

Ondoren, *SysEvent* motako *anEvent* aldagai laguntzailea deklaratzen da. Gainera, mota bereko zerrenda estekatuak ere deklaratuko dira, *theEventList* eta *newEventList* hurrenez hurren. Lehenengo zerrendak, prozesatzeke dauden gertaerak gordeko ditu, denboraren arabera ordenatuta egongo direlarik. Bigarrenak, aldiz, iterazio batean gertaera bat prozesatzean sortutako gertaerak jaso eta kudeatzeko erabiliko da. Amaitzeko, *simulationTime* aldagaia, simulazioaren denbora maximoa adieraziko duen balioarekin hasieratzen da (segundoak, mikrosegundoak, etab) erabiltzaileak aplikazioa exekutatzean argumentu bezala pasako dion balioarekin.

Eskeletoa osatzen duen kode osoa jarraian lau zati desberdinetan adierazten da. Lehenik eta behin sistemaren sorrera, ondoren hasierako gertaera-zerrenda ordenatzeko beharrezkoa den kode-zatia, simulazioaren iterazioa burutzeko egin beharrekoa eta azkenik, errore tratamendurako sortutako klasea adierazten dira.

- **(1) Sistemaren sorrera**

  Sistema bi pausotan sortzen da. Alde batetik *SimSystem* klasea instantziatuz simulatuko den *theSystem* sistema eraikitzen da. Bestetik, *theEvenList* zerrendak sistemaren hasierako gertaerak jasotzen ditu *theSystem*-i dagokion *start* metodoari deia eginez. Itzulitako gertaera-zerrendak ez du zertan ordenatuta egon beharrik eta kodea honakoa da:

```
SimSystem theSystem = new SimSystem(thePN);
theEventList = theSystem.start(printTrace);
```

- **(2) Hasierako gertaera-zerrenda ordenatu**

  Itzultzen den *theEventList* zerrenda gertaeren arabera ordenatuko da (non 0<= index < size()), beharrezkoa izan ala ez. Hori inplementatzen duen kode-zatia jarraian erakusten da:

```
// Order the initial event list
for(int i = 1; i < theEventList.size(); i++) {
    anEvent = theEventList.remove(i);
    int j = 0;
    while(j < i && theEventList.get(j).getTime() <
                                          anEvent.getTime()) {
        j++;
    }
```

```
            theEventList.add(j, anEvent);
        }
        if (printTrace == true) {System.out.println("Firing time of
                                                enabled transitions: ");}
        if (printTrace==true) {System.out.println();}
        for(int i = 0; i < theEventList.size(); i++) {
            if (printTrace == true)
            {System.out.format("%10.3f",theEventList.get(i).getTime());
}
            if (printTrace == true) {System.out.print(("   trans: T"+
                      (theEventList.get(i).getTransitionNumber()+1)));}
        }
        if (printTrace == true) {System.out.println();}
        if (printTrace == true) {System.out.println();}
```

- **(3) Simulazioa iteratu**

  Iteratzen hasi aurretik, zerrendako lehen gertaera ateratzen da. Ondoren, lortutako gertaeraren denbora simulazio prozesuaren denbora osoa baino txikiagoa den bitartean, pauso hauek jarraitzen dira:

  a) Uneko gertaerari dagokion trantsizioa sistemari bidaltzen zaio, honek prozesatu eta gertaera berriez osatutako zerrenda bat itzul dezan.

  b) Aurreko gertaera prozesatzearekin batera desaktibatu diren trantsizioak kendu zerrendatik.

  c) Gertaera hauek *theEventList* zerrendan denboraren arabera ordenatuta txertatzen dira.

  d) Berehalako (null) trantsizioak dauden kasuan hauen exekuzioa zoriz egiteko zerrenda berrordenatu.

  e) Aktibatuta dauden trantsizioak inprimatu eta gertaera berri bat lortzen da zikloa berriro hasi ahal izateko.

  Simulator klasea osatzen duen iterazioaren kode-zatia jarraian ikus daiteke:

```
anEvent = theEventList.removeFirst();

while (anEvent.getTime() < simulationTime) {
    if (printTrace == true) {System.out.println();}
    if (printTrace == true)
                        {System.out.println("_____________________
_____________________      ___________________
_____________________________      ___________
```

```
__");}
            if (printTrace == true) {System.out.println();}
            if (printTrace == true) {System.out.println("TRANSITION
                                                        FIRING:");}
            if (printTrace == true) {System.out.println();}
            if (printTrace == true) {System.out.print("time: ");}
            if (printTrace == true) {System.out.format("%10.3f",
                                                anEvent.getTime());}
            if
      (thePN.getTransition(anEvent.getTransitionNumber()).getTime().ge
                              tDistribution().trim().equals("null")){
                  if (printTrace == true) {System.out.println("   |
                                      transition:      T" +
(anEvent.getTransitionNumber()+1)
                          + "  {immediate}");}
            }else{
                  if (printTrace == true) {System.out.println("   |
                              transition:      T" +
                        (anEvent.getTransitionNumber()+1));}
            }
            if (printTrace == true) {System.out.println();}

            // Process an event of the system
            newEventList = theSystem.process(anEvent, theEventList);

            // Remove off events from theEventList
            theEventList = theSystem.removeOffEvents(theEventList);


            // Insert the new events in theEventList
            for(int i = 0; i < newEventList.size(); i++) {
                anEvent = newEventList.get(i);
                int j = 0;
                while(j < theEventList.size() &&
                        theEventList.get(j).getTime() <
                                              anEvent.getTime()) {
                        j++;
                }
                theEventList.add(j, anEvent);
            }

            //Order null transitions randomly at the beginning of the
            //list
            theEventList =
            theSystem.orderNullTransitionsAtRandom(theEventList,
                                              anEvent.getTime());

            if (printTrace==true){System.out.println("Firing time of
                                            enabled transitions: ");}
            if (printTrace==true){System.out.println();}
            for(int i = 0; i < theEventList.size(); i++) {
                if (printTrace == true)
                  {System.out.format("%10.3f",theEventList.get(i).getTi
                                                              me())
```

```
;}
              if (printTrace == true) {System.out.print("   trans:
                T" +
                (theEventList.get(i).getTransitionNumber()+1));}
            }
            if (printTrace == true) {System.out.println();}
            if (printTrace == true) {System.out.println();}

            //theEventList = theSystem.orderByPriority(theEventList);
            anEvent = theEventList.removeFirst();
        }
```

- **(4) Errore tratamendua**

  Erroreak kontrolatzeko *MyErrorHandler* klase estatikoa erabiliko da eta kode-zati honekin *MainClass* osaturik geratuko litzateke.

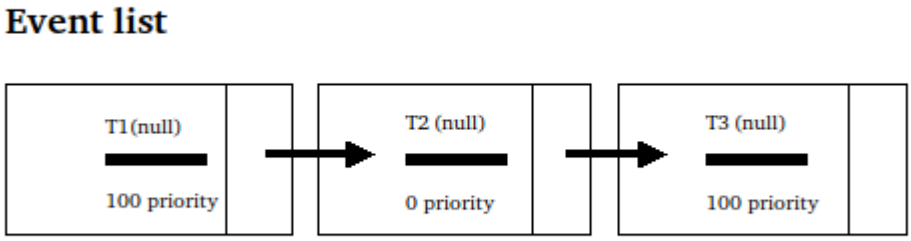
```
// Error handler to report errors and warnings
private static class MyErrorHandler implements ErrorHandler {
        /** Error handler output goes here */
        private PrintWriter out;

        MyErrorHandler(PrintWriter out) {
            this.out = out;
        }
        /**
         * Returns a string describing parse exception details
         */
        private String getParseExceptionInfo(SAXParseExceptionspe)
        {
            String systemId = spe.getSystemId();
            if (systemId == null) {
                systemId = "null";
            }
            String info = "URI=" + systemId +
                " Line=" + spe.getLineNumber() +
                ": " + spe.getMessage();
            return info;
        }

        // The following methods are standard SAX ErrorHandler
        //methods.
        // See SAX documentation for more info.
        public void warning(SAXParseException spe) throws
                                                SAXException {
            out.println("Warning: " + getParseExceptionInfo(spe));
        }

        public void error(SAXParseException spe) throws
                                                SAXException {
            String message = "Error: " +
                                getParseExceptionInfo(spe);
```

```
                    throw new SAXException(message);
            }

            public void fatalError(SAXParseException spe) throws
                                                SAXException {
                String message = "Fatal Error: " +
                                  getParseExceptionInfo(spe);
                throw new SAXException(message);
            }

    }
```

- **Denbora-mota berdinen kudeaketa**

Gertaera-zerrendan denbora banaketa-mota bereko gertaerak ditugunean arazoak sor daitezke (normalean *null* motako trantsizioekin). Hori dela eta, gertaera bat prozesatzen hasi aurretik, unekoaren eta ondorengoen denborak berdinak edota handiagoak diren aztertu beharko da. Berdinak diren kasuetarako, lehentasun-politika bat jarrai daiteke, hau da, hurrengo gertaerak unekoak baino lehentasun handiagoa badu, hori izango da prozesatuko den lehenengoa.

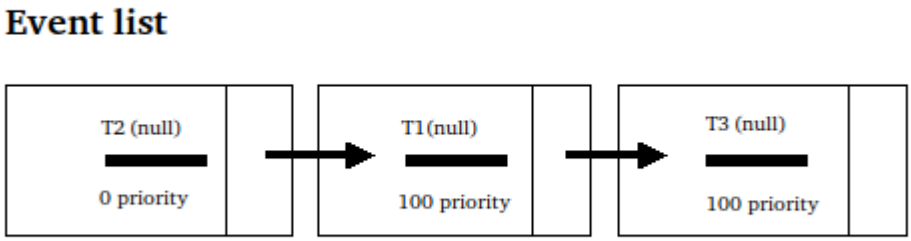
Jarraian ikus daitekeen irudian gertaerez osatutako zerrenda estekatu bat daukagu. Dakigunez, zerrendaren adabegi bakoitzak *SysEvent* objektu bat izango du. Hala ere, ulergarriagoa izan dadin, adabegi bakoitzean trantsizio bat eta horri dagozkion denbora banaketa eta lehentasun-maila agertzen dira:



Irudia 6.3.1.1: gertaeren zerrenda estekatua hasierako egoeran

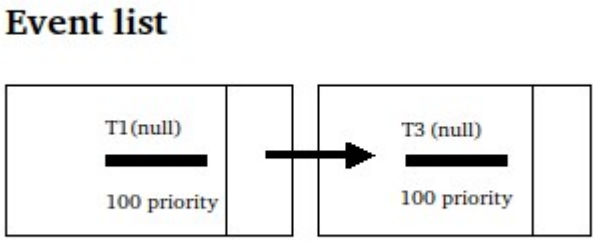
Gure algoritmoak, lehenengo gertaera (*t1* trantsizioa) prozesatzen hasi aurretik, ondorengo gertaerak (*t2* eta t3 trantsizioak) aztertuko ditu. Trantsizio guztiek denbora-banaketa berdinak dituztenez (*null*), lehentasunari erreparatuko zaio *thePriority(SysEvent anEvent)* metodoari deia eginez. Gure adibidean *t2* izango da lehentasun handiena duena (balio

guztietatik minimoa) eta beraz, gertaera hori prozesatu beharko da gainontzeko gertaerak zerrendan mantenduz. Prozesuarekin hasi aurretik, honako zerrendarekin geratuko ginateke:



Irudia 6.3.1.2: gertaeren zerrenda estekatua lehentasunaren arabera ordenaturik

T2-ren prozesua amaitzean, berriz, bi gertaera hauek osatzen dute gertaeren zerrenda estekatua:



Irudia 6.3.1.3: gertaeren zerrenda estekatua lehenengo gertaera prozesatzean

Ikus daitekeenez, *null* motako eta lehentasun bereko (100) bi gertaera geratu dira. Aurreko kasuan egin den era berean, uneko gertaera (*t1*) prozesatu aurretik ondorengoak ere aztertu beharko dira. Hori horrela izanik, adibidean bi gertaerek prozesatzeko lehentasun bera dutenez, hurrengo

gertaeraren hautapena zorizkoa izango da, bietako edozein aukeratua izateko probabilitatea berdina izango delarik.

Hala ere, simulatuko den sistema honetan lehentasun-politika hori erabiltzeak arazoak sor ditzake trantsizio batzuekin, izan ere, bat datozen berehalako trantsizioekin posible baita beti trantsizio berak tiro egitea. Horregatik, lehen kodean aipatu den moduan, tiro egingo duen hurrengo berehalako trantsizioa hautatzeko zoria erabiltzea erabaki da *orderNullTransitionsAtRandom()* metodoa erabiliz, aurrerago aztertuko duguna *SimSystem* klasearen kodean.

6.3.2 SysEvent klasea

Klase honek hiru atributu edo ezaugarri berezi izango ditu. Alde batetik, *eventTime* atributua, simulatu nahi den sistemaren gertaera zehatz bat noiz gertatuko den kudeatzeko erabiltzen dena. Bestetik, *transitionNumber* atributuak gertaeraren edo trantsizioaren zenbakia adierazten digu, gure kasuan Petri Sare bati dagokion trantsizio multzotik lortu ahal izango duguna. Hau da, gure sisteman gertaera zehatz bat definitzeko, trantsizio baten indizea eta honek tiro egingo duen uneko denbora beharko ditugu. Hala ere, gehigarri moduan, lehentasunaren atributua ere osoko moduan ipini da.

Atributu horiek metodo eraikitzailearen bidez jasoko dituzte dagozkien balioak eta horrez gain, hiru metodo ere izango ditu *SysEvent* klaseak; gertaerak kudeatzeko denbora lortzeaz arduratzen den *getTime,* gertaera zenbakia zein den jakinaraziko digun *getTransitionNumber* metodoa eta lehentasuna lortzeko *getPriority()* metodoak, hain zuzen ere. Kode-zatia jarraian erakusten da:

```
/**
 * The class of the system simulation events.
 */

public class SysEvent {

    private double eventTime;
    private int transitionNumber;
    private int priority;

    public SysEvent(double t, int transitionNumber, int
                                                priority) {
        eventTime = t;
        this.transitionNumber = transitionNumber;
        this.priority = priority;
    }

    public double getTime() {
        return eventTime;
```

```
    }

    public int getTransitionNumber() {
      return this.transitionNumber;
    }

    public int getPriority() {
      return this.priority;
    }

}
```

6.3.3 SimSystem klasea

SimSystem, simulatzailearen nukleotzat hartu dugun *Simulator* klasean instantziatzen den eta sistemaren egitura sortzeko erabiliko den klasea da. Atal honetan, klase honen inguruko ezaugarriak aipatuko dira, besteak beste.

Bi izango dira sistemaren egoera hasieratu eta gertaerak prozesatzeaz arduratuko diren metodo nagusiak, *start()* eta *process()*, hain zuzen ere. Hala ere, metodo hauez gain, *isActive()* (trantsizio bat aktibatuta dagoen ala ez jakiteko), *isInTheList()* (trantsizio bat zerrendan dagoen ala ez jakiteko), *transitionNumber()* (trantsizio bati dagokion indizea lortzeko), *removeOffEvents()* (desaktibatu diren gertaerak zerrendatik kentzeko) eta *orderNullTransitionsAtRandom()* (berehalako trantsizioak zoria erabiliz exekutarazteko) metodoak ere aipatuko dira. Klasearen eskeletoa izan litekeena jarraian erakusten da:

```
import java.util.LinkedList;
import java.util.Random;

public class SimSystem {

      public LinkedList<SysEvent> start(boolean trace) {

            LinkedList<SysEvent> startEventList =
                                    new LinkedList<SysEvent>();
            //corresponding code

            return startEventList;

      }

      public LinkedList<SysEvent> process(SysEvent theEvent) {

            LinkedList<SysEvent> newEventList =
                                    new LinkedList<SysEvent>();
            //corresponding code
```

```
            return newEventList;

        }

        private boolean isIntheList(Transition trans,
                                LinkedList<SysEvent>
                                                theEventList) {

        //corresponding code

        }

        public boolean isActive (SysEvent anEvent) {

            //corresponding code

        }

        private int transitionNumber(Transition trans) {

            //corresponding code

        }

        public int thePriority (SysEvent anEvent) {

            //corresponding code

        }

        public LinkedList<SysEvent>
            removeOffEvents(LinkedList<SysEvent> theEventList) {
            //corresponding code

        }

        public void orderNullTransitionsAtRandom
        (LinkedList<SysEvent> theEventList, double eventTime) {

            //corresponding code

        }

    }
```

Hala ere, lehenik eta behin sistemaren hasierako markaketa hartu behar da kontuan, hau da, lortutako *PetriNet* instantziatik leku bakoitzaren marka kopurua eta aktibatutako trantsizioen inguruko informazioa eskuratu behar da. Markaketa hori biltegiratzeko *array* motako datu-egitura erabiltzea da aukeretako bat, honela, gelaxka bakoitzaren indizeak Petri Sarearen leku zehatz bat adieraziko du eta edukiak, berriz, leku horrek izango dituen marka kopurua. Gainera, klaseak bereganatzen atributuak ere definitu behar ditugu jarraian erakusten den moduan:

```
// Random number generator
private Random rand;

private PetriNet pn;

private int [] markingArray;

private LogNormal lognorm = new LogNormal();

private boolean printTrace;

private String oldMarks = " ";

public SimSystem (PetriNet thePN){ // Builder method
  pn = thePN;
  markingArray = new int [17]; // 17 is the number of places
}
```

■ start() metodoa

Sistema sortu eta hasierako egoera finkatzeko *start()* metodoa erabiliko da. Metodo honek, balio boolear bat jasoko du argumentu bezala, simulazioaren traza inprimatzeko kontrolaz arduratuko dena, hain zuzen ere. Hurrengo pausoa, sortutako hasierako gertaeren zerrendari gertaerak gehitzea izango da (horretarako, aktibatuta dauden trantsizioak eta hauen tiro-denborak hartuko dira kontuan), izan ere, aktibatuta dagoen trantsizio bakoitzak gertaera bat ordezkatuko baitu. Petri Sareen kapituluan ikus daitekeen moduan, trantsizioek denbora-banaketa desberdinak izango dituzte eta horrek, era batean edo bestean, gertaerak sortzeko garaian eragin zuzena izango du. Honela, gertaera bakoitzeko, *SysEvent* instantzia bat sortu beharko da.

```
public LinkedList<SysEvent> start(boolean trace) {

      rand = new Random();
      double a = 0.0;
      double b = 0.0;
      double u = 0.0;
      double rate = 0.0;
      double mean = 0.0;
      double stdv = 0.0;
      double time = 0.0;

      …
```

Jarraian hasierako zerrenda erazagutu eta *markingArray* aldagaiean dauzkagun lekuei buruzko informazioa hasieratu eta inprimatzen da.

Informazio hori (leku bakoitzaren marka kopurua) *pn* instantziatik eskuratuko da eta bi lerrotan inprimatzen da:

```
LinkedList<SysEvent> startEventList = new
                                        LinkedList<SysEvent>();

//Initialize and print the markingArray
if (trace){
    System.out.println("Petri Net starting situation: ");
    System.out.println();
}
//for (int i=1; i < pn.getPlaces().size(); i++){
System.out.print("Places: {  ");
for (int i=0; i<markingArray.length; i++){
    System.out.print(i+1 + " ");
}
System.out.println(" }");
System.out.print("Marks:  {  ");
for (int i=0; i < pn.getPlaces().size(); i++){
    Place p = pn.getPlaces().get(i);
    markingArray[i] = p.getTokens();
    if (i>8){
          System.out.print(markingArray[i] + "  ");
          oldMarks = oldMarks + markingArray[i] + "  ";
    }else{
          if (i==8){
                System.out.print(markingArray[i] + "  ");
                oldMarks = oldMarks + markingArray[i] + "  ";
          }else if (i==16){
                System.out.print(markingArray[i]);
                oldMarks = oldMarks + markingArray[i];
          }else{
                System.out.print(markingArray[i] + " ");
                oldMarks = oldMarks + markingArray[i] + " ";
          }
    }
}
if (trace){
    System.out.println("}");
    System.out.println();
}
```

Azkenik gaiturik dauden trantsizio bakoitzeko gertaera bat sortuko da eta hauek zerrendan txertatuko dira. Gertaera horiek denbora-mota desberdinetakoak izango dira (gure Petri Sarearen adibidean *null, uniform* eta *exponential*) eta mota bakoitza bereizi egin beharko da:

```
//create events for active transitions and insert them into the
startEventList
for (int i=0; i < pn.getTransitions().size(); i++){
```

```
        Transition trans = pn.getTransition(i);
        if (this.isActive(trans)){
              //analyze time type and create the event
          if (trans.getTime().getDistribution().equals("null")){
                    //add null transition (event) to the list
                    startEventList.add(new SysEvent(0.0,
                    transitionNumber(trans), trans.getPriority()));
          }else if
        (trans.getTime().getDistribution().equals("uniform")){
                    //add uniform transition (event) to the list
                    a = trans.getTime().getA();
                    b = trans.getTime().getB();
                    startEventList.add(new SysEvent(a + (b - a) *
                      rand.nextDouble(), transitionNumber(trans),
                                          trans.getPriority()));
         }else if
        (trans.getTime().getDistribution().equals("exponential")){
                    //add exponential transition (event) to the
                    //list
                    u = rand.nextDouble();
                    rate = trans.getTime().getM();
                    startEventList.add(new
                     SysEvent(-(1/rate)*Math.log(1-u),
                    transitionNumber(trans), trans.getPriority()));
          }else if
            (trans.getTime().getDistribution().equals("constant")){
                    //add constant transition (event) to the list
                    startEventList.add(new
                      SysEvent(trans.getTime().getA(),
                    transitionNumber(trans), trans.getPriority()));
          }else if
           (trans.getTime().getDistribution().equals("lognormal")){
                    //add lognormal transition (event) to the list
                    mean = trans.getTime().getM();
                    stdv = trans.getTime().getS();
                    time = lognorm.nextLogNormal(rand, mean, stdv);
                    startEventList.add(new SysEvent(time,
                    transitionNumber(trans), trans.getPriority()));
          }
        }
    }

    printTrace = trace;


    return startEventList;

}
```

■ process() metodoa

Gertaerak prozesatzeko *process()* metodoa erabiliko da, argumentu bezala *SysEvent* (gertaera) bat eta *theEventList* zerrenda jasota, prozesu horren ondorioz sortutako egoera eta gertaera berriak kudeatzeaz arduratuko dena. Hori horrela izanik, metodoak gertaera berrien zerrenda *(newEventList)* itzuliko du emaitza gisa.

```
        public LinkedList<SysEvent> process(SysEvent theEvent,

LinkedList<SysEvent> theEventList) {

        double eventTime = theEvent.getTime();
        int loseMark = 0;
        int getMark = 0;
        rand = new Random();
        double a = 0.0;
        double b = 0.0;
        double u = 0.0;
        double rate = 0.0;
        double mean = 0.0;
        double stdv = 0.0;
        double time = 0.0;
        boolean done = false;

        LinkedList<SysEvent> newEventList = new
LinkedList<SysEvent>();
          ...
```

Sistemaren lehenengo egoeratik sortu dugun *startEventList* zerrenda ordenatua lortzean, markaketa berria kalkulatu beharko da eta gaitu diren trantsizio berriak itzuli gertaera berri bezala, horiei dagozkien *SysEvent* berriak sortu eta gertaeren zerrenda berrian (*newEventList*) txertatuz. Beraz, markaketa berria kalkulatu eta inprimatzearekin hasiko gara. Horrez gain, iterazioaren une honetan ere aurreko markaketa inprimatuko dugu, zehazki *oldMarks* aldagaiean gordetzen duguna:

```
        //calculate new marking
        for (int i=0; i < pn.getInputArcs().size(); i++){
            InputArc ia = pn.getInputArcs().get(i);
            if ((ia.getTo()) == (theEvent.getTransitionNumber()+1) &&
                                          ia.getMultiplicity()!=0){
                  loseMark = (ia.getFrom()-1);
                  markingArray[loseMark] = markingArray[loseMark] -
                                              ia.getMultiplicity();
                  //update PetriNet instance

      pn.getPlace(loseMark).setTokens(markingArray[loseMark]);
            }
```

```
  }

  for (int i=0; i < pn.getOutputArcs().size(); i++){
      OutputArc oa = pn.getOutputArcs().get(i);
      if (oa.getFrom() == (theEvent.getTransitionNumber()+1)){
            getMark = (oa.getTo()-1);
            markingArray[getMark] = markingArray[getMark] +

oa.getMultiplicity();
            //update PetriNet instance

pn.getPlace(getMark).setTokens(markingArray[getMark]);
      }
  }
  System.out.println();

  //Print the new marking array
  System.out.print("Places:     {  ");
  for (int i=0; i<markingArray.length; i++){
      System.out.print(i+1 + " ");
  }
  System.out.println(" }");

  System.out.println("Old marks:  { " + oldMarks + "}");
  oldMarks = " ";

  System.out.print("New marks:  {  ");
  for (int i=0; i<markingArray.length; i++){
      if (i>8){
            System.out.print(markingArray[i] + "  ");
            oldMarks = oldMarks + markingArray[i] + "  ";
      }else{
            if (i==8){
                  System.out.print(markingArray[i] + "  ");
                  oldMarks = oldMarks + markingArray[i] + "  ";
            }else if (i==16){
                  System.out.print(markingArray[i]);
                  oldMarks = oldMarks + markingArray[i];
            }else{
                  System.out.print(markingArray[i] + " ");
                  oldMarks = oldMarks + markingArray[i] + " ";
            }
      }
  }
  System.out.println("}");
  System.out.println();
  System.out.println();
```

Hurrengo pausoa aktibatu diren trantsizioak gertaera berrien zerrendan txertatzea izango da. Horretarako, trantsizio bat aktibatu den aztertu beharko da *isActive()* metodoari deia eginez eta horrez gain, trantsizio jakin hori *theEventList* zerrendan ez dagoela ziurtatu beharko da *isInTheList()* metodoarekin.

Simulatzen ari garen Petri Sare honetan xehetasun bat hartu behar da kontuan. Hiru memoria moduluetatik prozesu batek bat zoriz aukeratu beharko duenez, 11, 17 eta 23. trantsizioak aktibatzen diren unean horietatik bakarra hautatu beharko da, esan bezala zoria erabiliz. Hori ebazteko modu sinple bat *Random* klasearen *nextInt(x)* metodoa erabiltzea da, non metodo horrek [0, x) tarteko osoko bat itzuliko digun. Gure kasuan 3 memoria modulu ditugunez, x-en ordez 3 zenbakia erabiliko genuke.

```
//insert the new events in the newEventList
for (int i=0; i<pn.getTransitions().size(); i++){
  Transition trans = pn.getTransition(i);
  if (this.isActive(trans) && !isIntheList(trans, theEventList)
                                                    && !done){
      //analyze time type and create the event
      if ((transitionNumber(trans) == 11-1 ||
            transitionNumber(trans) == 17-1 ||
                  transitionNumber(trans) == 23-1) && !done){
            int number = rand.nextInt(3);
                  //if it returns 0 T11 will be enabled
                  //if it returns 1 T17 will be enabled
                  //if it returns 2 T23 will be enabled

            if (number == 0 && !done){
                  System.out.println("a memory module selected");
                  SysEvent event = new SysEvent(eventTime, 11-1,
                                          trans.getPriority());
                  newEventList.add(event);
                  done = true;
            }else if (number == 1 && !done){
                  System.out.println("b memory module selected");
                  SysEvent event = new SysEvent(eventTime, 17-1,
                                          trans.getPriority());
                  newEventList.add(event);
                  done = true;
            }else if (number == 2 && !done){
                  System.out.println("c memory module selected");
                  SysEvent event = new SysEvent(eventTime, 23-1,
                                          trans.getPriority());
                  newEventList.add(event);
                  done = true;
            }
```

Gainontzeko gertaerak prozesatzean, lehen esan bezala, trantsizioen tiro-denbora kalkulatzeko trantsizio-mota desberdinak bereizi beharko dira. Ikusitako *null*, *constant* eta *uniform* denborez gain, banaketa *esponentziala* eta *logaritmiko normala* ere hartu behar ditugu kontuan, nahiz eta gure sistemaren adibidean hiru besterik ez diren erabiltzen. *SysEvent* instantziak sortzean horiek kudeatu ahal izateko, *Random* klasearen metodoak erabiliko ditugu eta kapitulu honetan bertan aipatutako formulez baliatuz, kasu

bakoitzerako, zorizko balioak lortu ahal izango dira. Jarraian azaltzen dira helburu hauetako bakoitza lortzeko eman beharreko pausoak:

◆ null

Berehalako trantsizioek *null* denbora-banaketa izango dute. Gertaeren zerrenda berrian txertatuko litzatekeen *SysEvent* instantzia baten adibidea (*null* denbora-banaketarekin) honako hau izan daiteke:

```
}else{
      if (trans.getTime().getDistribution().equals("null")){
        //add null transition (event) to the list
        SysEvent event = new SysEvent(eventTime,
                        transitionNumber(trans), trans.getPriority());
        newEventList.add(event);
```

Gertaeraren denbora *eventTime* aldagaiak adierazten du. Gaitu den eta berehalakoa den trantsizioari dagokion zenbakia *transitionNumber* metodoari esker lortzen da (adibidez trantsizioa *t3* bada, aldagaiari *3* balioa dagokio).

◆ constant

Tiro denborak kudeatzeko balio konstanteak maneiatzen dituzten trantsizioak dira hauek. Gertaeren zerrenda berrian txertatuko litzatekeen *SysEvent* instantzia baten adibidea (*constant* denbora-banaketarekin) honako hau izan daiteke:

```
      }else if (trans.getTime().getDistribution().equals("constant")){
        //add constant transition (event) to the list
        SysEvent event = new SysEvent((eventTime +
            trans.getTime().getA()), transitionNumber(trans),
                                        trans.getPriority());
        newEventList.add(event);
```

Gertaeraren denbora *eventTime* eta *time* aldagaien arteko baturatik lortzen da; *time* aldagaiaren balioa *PetriNet* instantziatik atzitu ahal izango da (adibidez 10.0). Gaitu den eta tiro-denbora konstanteak erabiltzen dituen trantsizioari dagokion zenbakia *transitionNumber* metodoak itzultzen du.

◆ uniform

Banaketa uniformea jarraitzen duten trantsizioen tiro-denboretan balio minimo eta maximoa bereiziko dira. Balio hauek *PetriNet* instantziatik lortu ahal izango dira eta tarte horretan, balio guztiek probabilitate bera izango dute. Gertaeren zerrenda berrian txertatuko litzatekeen *SysEvent* instantzia baten adibidea (*uniform* denbora-banaketarekin) honako hau izan daiteke:

```
}else if (trans.getTime().getDistribution().equals("uniform")){
      //add uniform transition (event) to the list
      a = trans.getTime().getA();
      b = trans.getTime().getB();
      SysEvent event = new SysEvent(eventTime + (a + (b - a) *
      rand.nextDouble()), transitionNumber(trans),
      trans.getPriority());
      newEventList.add(event);
```

Gertaeren denbora *eventTime* eta *time* aldagaien arteko baturatik lortzen da; *time* aldagaiaren balioa *a* eta *b* balio minimo eta maximoen arteko zorizko balioa da. Gaitu den eta tiro-denbora uniformeak erabiltzen dituen trantsizioari dagokion zenbakia *transitionNumber* metodoak itzultzen du.

◆ exponential

Banaketa esponentziala jarraitzen duen zorizko aldagai baten balioa lortzeko λ parametroa beharko dugu (adibidean rate). Horrela, banaketa uniformeaz baliatuz dagokion formula aplikatu ahal izango da. Gertaeren zerrenda berrian txertatuko litzatekeen *SysEvent* instantzia baten adibidea (*exponential* denbora-banaketarekin) honako hau izan daiteke:

```
}else if
      (trans.getTime().getDistribution().equals("exponential")){
  //add exponential transition (event) to the list
  u = rand.nextDouble();
  rate = trans.getTime().getM();
  SysEvent event = new SysEvent((-(1/rate)*Math.log(1-u)) +
      eventTime, transitionNumber(trans), trans.getPriority());
  newEventList.add(event);
```

Gertaeren denbora *eventTime* eta *time* aldagaien arteko baturatik lortzen da; *time* aldagaiaren balioa aldagai esponentziala lortzeko aplikatu beharreko formularen laguntzarekin lortzen da. Gaitu den eta tiro-denbora esponentzialak erabiltzen dituen trantsizioari dagokion zenbakia *transitionNumber* metodoak itzultzen du.

◆ log-normal

Log-normal motako banaketa duten trantsizioak kudeatzeko bi argumentu jaso beharko dira *PetriNet* instantziatik: *mean* eta *stdv*, batezbestekoaren balioa eta desbiderapen estandarraren balioa, hurrenez hurren. Horretarako, kapitulu honetan azaldutako LogNormal klaseko *nextLogNormal()* metodoa erabil dezakegu. Gertaeren zerrenda berrian txertatuko litzatekeen *SysEvent* instantzia baten adibidea (*lognormal* denbora-banaketarekin) honako hau izan daiteke:

```
      double mu; double sigma;
      double mean; double stdv;
      ...
      LogNormal ln = new LogNormal();
      ...
      }else if (trans.getTime().getDistribution().equals("lognormal"))
{
            //add lognormal transition (event) to the list
            mean = trans.getTime().getM();
            stdv = trans.getTime().getS();
            time = lognorm.nextLogNormal(rand, mean, stdv);
            SysEvent event = new SysEvent(eventTime + time,
                        transitionNumber(trans), trans.getPriority());
            newEventList.add(event);
      }
    }
  }
}

return newEventList;

}
```

Gertaeren denbora *eventTime* eta *time* aldagaien arteko baturatik lortzen da; *time* aldagaiaren balioa *LogNormal* klaseko *nextLogNormal* metodoaren laguntzarekin lortzen da. Gaitu den eta tiro-denbora logaritmiko normalak erabiltzen dituen trantsizioari dagokion zenbakia *transitionNumber* metodoak itzultzen du.

■ isActive() metodoa

Gertaera bat prozesatzean, gertaera horrekin asoziatutako trantsizioa gaituta dagoela ziurtatzeko erabiltzen da *isActive()* metodoa. Metodo honek, *Transition* motako objektu bat jasoko du argumentu bezala.

Ideia, *InputArc* (sarrera-arku) guztien bektorea zeharkatzea izango da, zehazki gure trantsizioa helburutzat duten arkuak aurkitu nahi izango ditugularik. Gauza jakina da, trantsizio bat gaiturik egoteko, trantsizio horrekin lotuta dauden lekuek gutxienez marka bat eduki beharko dutena. Hala ere, trantsizio inhibitzaileak ere kontuan hartu behar ditugu, multiplizitatea 0 dutenak hain zuzen ere.

Hori horrela izanik, multiplizitatea 1 den kasuan iturburu den lekuak gutxienez marka bat badu *active* aldagaia *true* egoerara pasako da; kontrako kasuan, berriz, zuzenean gaitu gabe dagoela ziurta dezakegu. Multiplizitatea 1 baino handiagoa bada, marka kopuruak ere 1 balio hori gainditu beharko du eta multiplizitatea 0 duten trantsizio inhibitzaileetan, marka kopuru hori 0 den kasuan esleituko zaio *true* balioa *active* aldagaiari. *SimSystem* klaseko metodo honi dagokion kodea hau da:

```
public boolean isActive(Transition trans) {
      boolean active = false;
      int transNumber = this.transitionNumber(trans);
      for (int i=0; i<pn.getInputArcs().size(); i++){
            int toTrans = pn.getInputArcs().get(i).getTo();
            if (toTrans == transNumber+1){
                  int fromPlace = pn.getInputArcs().get(i).getFrom();
                  int multiplicity =
                        pn.getInputArcs().get(i).getMultiplicity();
                  // We must consider the multiplicity
                  if (multiplicity == 1) {
                        if (markingArray[fromPlace-1] > 0){
                              active = true;
                        }else{
                              return false;
                        }
                  }else if (multiplicity > 1){
                        if (markingArray[fromPlace-1] > 1){
                              active = true;
                        }else{
                              return false;
                        }
                  }else if (multiplicity == 0){
                        if (markingArray[fromPlace-1] == 0){
                              active = true;
                        }else{
                              return false;
                        }
                  }
            }
    }
    return active;
}
```

■ thePriority() metodoa

Gertaeren prozesatze ordenan sortzen diren gatazkak kudeatzeko, gertaeren zerrendako elementu bakoitzaren lehentasun-maila jakitea ezinbestekoa izango da lehentasun politika erabiltzen den kasuetan (denbora-mota berdinen kudeaketa atalean ikusi dugu) eta horretarako, *thePriority()* metodoa erabili ahal izango da. Metodo honek, argumentu bezala *SysEvent* motako objektu bat jasoko du, *PetriNet* instantziatik trantsizio konkretu bat aurkitu eta bere lehentasun-maila itzuli ahal izateko, hain zuzen ere. *SimSystem* klaseko metodo honi dagokion kodea hau da:

```
public int thePriority(SysEvent anEvent){

      return anEvent.getPriority();

}
```

■ isInTheList() metodoa

Metodo honek egingo duen gauza bakarra, trantsizio bat jaso eta gertaeren zerrendan dagoen edo ez adieraztea izango da. Horretarako, zerrendako trantsizioen indizea konparatuko da argumentu bezala pasatakoarekin. Emaitza *found* aldagaiak (*true edo false*) adierazten digu.

```
private boolean isIntheList(Transition trans,
                        LinkedList<SysEvent> theEventList) {
      // TODO Auto-generated method stub
      int tNumber = transitionNumber(trans);
      int i=0;
      boolean found=false;
      while (i < theEventList.size() && !found){
            if (theEventList.get(i).getTransitionNumber() ==
                                                      tNumber){
                  found = true;
            }else{
                  i++;
            }
      }
      return found;
}
```

■ transitionNumber() metodoa

Trantsizio bati dagokion zenbakia edo indizea jakiteko *transitionNumber()* metodoa erabil dezakegu. Trantsizio bat jasoko duenez, *pn* objektuaren

trantsizio guztiak zeharkatzen dira argumentu bezala pasatako hori jaso arte. Honela, *ans* aldagaiak gordeko du berdintza hori betetzen duen trantsizioaren zenbakia.

```
private int transitionNumber(Transition trans) {
        // TODO Auto-generated method stub

  int ans = 0;
        for (int i=0; i<pn.getTransitions().size(); i++){
              if (pn.getTransitions().get(i) == trans){
                    ans = i;
              }
        }
        return ans;
  }
```

■ removeOffEvents() metodoa

Gertaera bat prozesatzerakoan, askotan, gertaera hori prozesatu aurretik gaiturik zeuden trantsizio bat edo gehiago desaktibatzea oso ohikoa da. Hori dela eta, iterazio bakoitzean desgaitu diren gertaerak etorkizuneko gertaeren zerrendatik kendu beharko dira. Horretarako daukagu *removeOffEvents()* metodoa, *isActive()* metodoari deia eginez, une jakin batean gertaeren zerrendan egonik, desaktibatu diren gertaerak kentzeaz (*remove*) arduratzen dena.

```
public  LinkedList<SysEvent>  removeOffEvents(LinkedList<SysEvent>
                                                  theEventList) {

        // TODO Auto-generated method stub
        for (int i=0; i < theEventList.size(); i++){
              int transitionNum =
                    theEventList.get(i).getTransitionNumber();
              if (!this.isActive(pn.getTransition(transitionNum))){
                    theEventList.remove(i);
              }
        }
        return theEventList;
  }
```

■ orderNullTransitionsAtRandom() metodoa

Kapitulu honetan aipatu den moduan, gertaeren zerrendaren hasieran (denboraren arabera ordenatzen ditugulako) berehalako trantsizio bat edo gehiago izango ditugu. Hauek teorian tiroa aldi berean egingo dute *null* motakoak direlako, hala ere, horien exekuzioa/tiroa gauzatzeko zoria erabiltzea da aukera bat.

Gertaeren zerrenda eta uneko gertaerari dagokion *eventTime* denbora jasota, metodo honek, lehenik eta behin zerrendaren hasieran dauden berehalako gertaera kopurua jasotzen du *z* aldagaian. Ondoren, *Random* klaseko *nextInt(z)* metodoaren laguntzarekin, *quantity* aldagaian [0, z) tarteko osoko bat jasoko da iterazio bakoitzean. Honela, zerrenda horretatik *quantity* aldagaiak adierazten duen osagaia kenduko da eta zerrendaren lehenengo posizioan txertatuko da.

```
public LinkedList<SysEvent>
orderNullTransitionsAtRandom(LinkedList<SysEvent>
                             theEventList, double eventTime) {

  int z = 0;
  for (int i = 0; i < theEventList.size(); i++){
      // Increase z if it is null transition
      if (theEventList.get(z).getTime() == eventTime){
            z++;
      }
  }

  for (int i=0; i<z; i++){
      int quantity = rand.nextInt(z);
      SysEvent event = theEventList.remove(quantity);
      theEventList.addFirst(event);
  }
  return theEventList;

}
```

■ orderByPriority() metodoa

Nahiz eta proiektu honetan berehalako trantsizioen tiroen gatazka argitzeko lehentasun-politika ez dugun erabili, zoria erabili ordez gertaeren zerrendan dauden *null* motako trantsizioak lehentasunaren arabera ordenatzea posible egiten duen metodoa da *orderByPriority()* izenekoa.

```
      public  LinkedList<SysEvent>  orderByPriority(LinkedList<SysEvent>

      theEventList) {
            // TODO Auto-generated method stub
        int z = 0;
        boolean found = false;
        while (z < theEventList.size() || !found){
            if
(theEventList.get(z).getTime()==theEventList.get(z+1).getTime()){
                  z++;
            }else{
                  found = true;//The next event has different time
            }
        }

        // There are z events with null time distribution
        int maxPriority = 0;
        maxPriority = theEventList.get(0).getPriority();
        for (int i=0; i<=z; i++){
            if (theEventList.get(i).getPriority() > maxPriority){
                  maxPriority = theEventList.get(i).getPriority();
                  SysEvent event = theEventList.remove(i);
                  // Insert the max priiority event in the first
                  //position
                  theEventList.addFirst(event);
            }else if (theEventList.get(i).getPriority() == maxPriority)
{
                  SysEvent event = theEventList.remove(i);
                  theEventList.addFirst(event);
            }
            i ++;
        }
        return theEventList;
      }
```

7. KAPITULUA

XML LENGOAIA BIDEZKO ESPEZIFIKAZIOA

7.1 Sarrera

Proiektu honen abiapuntuan Petri Sareak XML dokumentuen bidez definitzea erabaki zen, honela simulaziorako idatzi den Java programatik bereizita izango den formatu estandar eta erraz baten bidez deskribatzen baita simulatuko den sistemaren eredu formala. Kapitulu honetan zehar, XML lengoaia aurkeztu eta aztertzeaz gain, Petri Sare batetik abiatuta eskema batean oinarrituko diren XML dokumentu horiek sortzeko jarraitu beharreko pausoak ematen dira.

XML (ingelerazko eXtensible Markup Language), markaketa lengoaia hedagarri bat da, egituratutako datuak informazio-sistema ezberdinen artean trukatzeko aukera eskaintzen duena, besteak beste. HTML lengoaiarekin antzekotasun nabarmenak dituen arren, XML-ren funtzio nagusia datuak deskribatzea da eta ez erakustea, HTML-rekin gertatzen den bezala. Horrez gain, XML lengoaiak aplikazio desberdinetatik datuak irakurtzeko aukera ere ematen digu.

XML teknologia Java ingurunean prozesatzeko erabiltzen den APIa JAXP [21] (Java API for XML Processing) izenekoa da. Honek, XML dokumentuak Java programazio lengoaian idatzitako aplikazioak erabiliz prozesatzea ahalbidetzen du. JAXP APIak eskaintzen dituen aukeren artean SAX (Simple API for XML Parsing) eta DOM (Document Object Model) nabarmentzen dira. Lehenengoak, datuak gertaera-sekuentzia moduan analizatzen ditu eta bigarrenak, aldiz, datu horiek zuhaitz egituran osatutako objektu moduan ordezkatzen ditu. Gure kasuan, DOM egitura erabiliko dugunez, hori da aurrerago azalduko duguna.

7.2 XML dokumentuak

XML, **metalengoaia** dela ere esan daiteke, izan ere, diseinatzaileei marka propioak sortzen uzteaz gain, marka horiek antolatzen ere uzten dietelako [22] [23] [25] [26]. Marka bat, irteeran azalduko ez den dokumentu baten

edozein elementu izango da, zehazki <...> ikurren artean idazten diren *etiketak*.

XML dokumentuen zenbait *ezaugarri* honako hauek dira:

- Edozein XML dokumentuk, *erro* bat dauka.
- Erroak, *umeak* edo azpielementuak ditu.
- Elementu bakoitzak beste azpielementuak izan ditzake eta *guraso* bakarra dauka.
- Hasierako etiketa guztiek bukaerako etiketa daukate.
- Etiketek era egokian habiatuta egon behar dute.
- Iruzkinak sar daitezke <!--...--> ikurren artean.
- Maiuskularen eta minuskularen erabilera desberdintzen da.
- Atributuek elementuei edo datuei buruzko informazio gehigarria ematen dute:
  · <animalia **hizkuntza=”ingelera”**> tiger </animalia>

XML dokumentu batek bi egitura nagusi ditu. Alde batetik, informazioa lortzeko erabiltzen diren Elementuak, eta bestetik, metainformazioa daramaten Atributuak. Elementuei dagokienez, sinpleak (datuen balioak dituztenak) edo konplexuak (elementu gehiago dituen hierarkia egiturak dituztenak) izan daitezke. Atributuek, esan bezala, elementuak deskribatzeko informazio gehigarria ematen dute.

7.3 Ongi eratutako XML dokumentuak

Edozein XML dokumentu, salbuespenik gabe, erabilgarria izan dadin ongi eratuta (well-formed) egon behar da. Horretarako, honako baldintza hauek bete beharko dituzte dokumentuek:

- Hasierako etiketa guztiek bukaerako etiketa daukate.
- Elementuak ez dira teilakatzen (elementuak segidan bai, baino ez gurutzatuta).
- Erro bakarra dauka.
- Atributuen balioak komatxoen artean doaz.
- Elementuek ez daukate atributu bat baino gehiago izen berarekin.
- Etiketen barruan ez da iruzkinik, ez prozesu-agindurik (<? ... ?>) agertzen.
- Zenbait karakterek “<”, “&”, e.a. ezin dute agertu elementu eta atributuetan (&alt; edo &#38 erabili behar dira “<” adierazteko, adibidez). &-ek entitateei erreferentzia egiteko balio du.

7.4 XMLren abantailak

XML erabiltzeak dakartzan *abantailen* artean honako hauek goraipa ditzakegu, besteak beste:

• **Auto-dokumentatua**: datuen esanahia ulertzeko ez da beharrezkoa eskema bat kontsultatzea.

• **Sinplea**: estandar laburra jarraitzen du (50 orri baino gutxiago). Horrez gain, testuan oinarritutako lengoaia izateak sinpleago egiten du.

• **Estandar irekia** eta plataforma eta fabrikatzaileekiko independentea: testua denez, trukerako ez dago inolako arazorik.

• **Hedagarria**: erabiltzaileari bere etiketak sortzeko ahalmena ematen dio.

• **Berrerabilgarria**: etiketen liburutegiak sor daitezke eta gainera beste aplikazioetan erabili.

• **Edukiaren eta aurkezpenaren arteko banaketa**: aurkezpenaren formatua aparteko XSL edo CSS dokumentuen bitartez deskribatzen da.

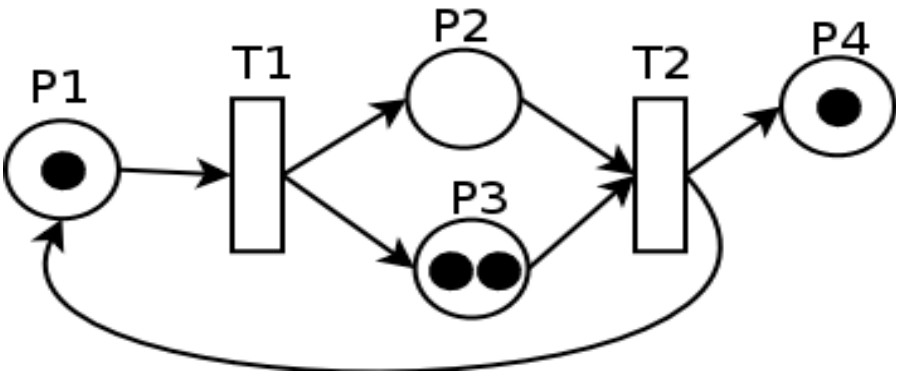
• Zerbitzarian **prozesu karga murriztea ahalbidetzen** du: datuak lokalera jaisten dira eta kalkuluak bertan burutzen dira.

• Datu heterogeneoak osatzeko euskarria.

• XMLren prozesurako **tresna** asko daude. Adibidez: bilaketa-motoreak.

7.5 Petri Sarerako XML eskema

Atal honetan XML dokumentuak balioztatzeko eskuragarri ditugun XML eskemak aztertuko ditugu eta Petri Sareen XML dokumentuetarako XML eskemak definitu [24] [25]. Hasteko, Petri Sare sinple baten adibidea ikusiko dugu. Demagun honako irudian dugun Petri Sarearen XML dokumentua sortu behar dugula:



7.5.1 irudia: Petri Sare sinplea

Adibide horri dagokion eskeletoa sortzeko leku, trantsizio eta arkuen etiketak hartu beharko ditugu kontuan:

```
<!-- Simple Petri-net-->
<petri_net>
      <places></places>
      <transitions></transitions>
      <input_arcs></input_arcs>
      <output_arcs></output_arcs>
</petri_net>
```

Ondoren, elementu nagusi bakoitza osatuko dugu dagozkien azpielementuekin. Lekuen kasuan, horietako bakoitza definitzeko izena eta marka kopurua erabiliko ditugu.

```
<places>
      <place>
            <p_name>p1</p_name>
            <tokens>1</tokens>
      </place>
      <place>
            <p_name>p2</p_name>
            <tokens>0</tokens>
      </place>
      <place>
            <p_name>p3</p_name>
            <tokens>2</tokens>
      </place>
      <place>
            <p_name>p4</p_name>
            <tokens>1</tokens>
      </place>
</places>
```

Trantsizioak definitzeko izena erabiltzeaz gain, tiro egiteko duen lehentasuna, semantika eta tiro-denbora kalkulatzeko erabilitako banaketaren inguruko informazioa gehitu behar dugu. Kasu honetan, irudian ikus daitekeenez trantsizioak ez dira berehalakoak izango, hau da, tiro-denbora kalkulatzeko denbora banaketa bat jarraitzen dute, guretzat ezezaguna dena.

Demagun T1 trantsizioak banaketa uniformea eta T2-k banaketa esponentziala jarraitzen dutela. Kasu bakoitza definitzeko azpielementu desberdinak erabiliko dira.

```
<transitions>
      <transition>
            <tr_name>t1</tr_name>
            <time>
                  <distribution>uniform</distribution>
                  <a>1.1</a>
                  <b>5.2</b>
            </time>
            <priority>0</priority>
            <semantics>single</semantics>
      </transition>
      <transition>
            <tr_name>t2</tr_name>
            <time>
                  <distribution>exponential</distribution>
                  <m>2.3</m>
            </time>
            <priority>0</priority>
            <semantics>single</semantics>
      </transition>
</transitions>
```

Azkenik, lekuetatik trantsizioetara doazen sarrera-arkuak eta alderantzizko bidea egiten duten irteera-arkuak definitzea besterik ez zaigu geratzen.

```
<input_arcs>
<!-- from place to transition -->
      <input_arc>
            <from>01</from>
            <to>01</to>
      </input_arc>
      <input_arc>
            <from>02</from>
            <to>02</to>
      </input_arc>
```

```
      <input_arc>
            <from>03</from>
            <to>02</to>
      </input_arc>
</input_arcs>

<output_arcs>
<!-- from transition to place>
      <output_arc>
            <from>01</from>
            <to>02</to>
      </output_arc>
      <output_arc>
            <from>01</from>
            <to>03</to>
      </output_arc>
      <output_arc>
            <from>02</from>
            <to>01</to>
      </output_arc>
      <output_arc>
            <from>02</from>
            <to>04</to>
      </output_arc>
</output_arcs>
```

XML dokumentuak balioztatzeko erabiltzen diren eskema-lengoaien artean *DTD* (*Document Type Definition*) eta *W3C* ***XML Schema*** goraipatu behar dira. Ezaugarri nagusienetakoa, eskema batek XML dokumentu baten sintaxia definitzen duena da. Dokumentua eskema konkretu batekiko baliozkoa izan dadin honako baldintza hauek bete beharko ditu:

- Ongi eratutako dokumentua da.
- Elementu guztiak eskeman definituta daude.
- Elementuak eskemaren arabera egituratuta daude.

● XML Schema

XML Schema, XML dokumentu baten dokumentu egitura definitzeko balio duen datuak definitzeko lengoaia da. Lengoaia honek *WC3*k definitutako *estandarra* jarraitzeaz gain, eskemako elementu mota bakoitza nola definitzen den eta elementuak zein datu-mota lotuta duen zehazten du.

XML Schemak hainbat ezaugarri ditu, besteak beste:

- XML sintaxia duen XML dokumentu bat da. XML kodea irakurtzeko erabiltzen diren tresnak erabil daitezke XML eskema bat editatzeko eta prozesatzeko.

- DTDak dituen mugak gainditzen saiatzen da.

- Erabiltzaileak bere datu-motak defini ditzake.

- Elementuetan agertzen den testua, mota espezifiko batzuei murriztuta egon daiteke.

- Motak murrizteko aukera ematen du espezializatutako motak sortzeko.

- Izen-espazioak definitzea ahalbidetzen du.

- Kardinalitate murriztapenak definitzea ahalbidetzen du.

- Modulartasuna: erabiltzaile motak toki desberdinetan erabil daitezke eta beste XML Schematako zatiak berrerabil ditzake.

● Petri Sareetarako XML eskema

Aplikazioak erabiliko dituen XML dokumentuak baliozkoak izateko XML Schema bat edo gehiago jarraitu beharko dute. Kasu honetan eskema bakarra erabili da, ***petri_net.xsd*** izenekoa, hain zuzen ere. Aipatutako fitxategian elementu sinple eta konplexuen arteko bereizketa egiten da eta eskemak berak ere bere hasierako eta bukaerako etiketak izango ditu, edozein XML dokumentutan gertatzen den bezala. Eskemaren eskeletoa honakoa izango da:

```
<xs:schema elementFormDefault="qualified">
      <!-- definition of complex elements -->
      <xs:element name="petri_net">
            <xs:complexType>
                  <xs:sequence>
                        <xs:element ref="places"/>
                        <xs:element ref="transitions"/>
                        <xs:element ref="input_arcs"/>
                        <xs:element ref="output_arcs"/>
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
      …
</xs:schema>
```

Elementu konplexu bakoitza adierazteko etiketa bereziak erabiltzen dira (*<xs:complexType>* eta *</xs:complexType>*) eta horrez gain, kasu honetan *petri_net* elementuaren azpielementu diren elementuen sekuentzia adierazteko ere *<xs:sequence>* eta *</xs:sequence>* hasierako eta

bukaerako etiketak erabiltzen dira. XML Schemak *ref* atributua erabiltzeko aukera ere ematen du, elementu bati erreferentzia egiteko, hain zuzen ere.

Ondoren, elementu horietako bakoitza azaldu behar da; *places* elementu konplexuak place elementuei egingo die erreferentzia. Hori horrela izanik, Petri Sareak izango duen leku kopurua zehazteko *minOccurs="1"* eta *maxOccurs="unbounded"* (zehaztu gabea) atributuak beharko dira. Bestalde, *place* elementu bakoitzaren *p_name* eta *tokens* azpielementu sinpleak ere zehazten dira lekuen definizioarekin amaitzeko.

```
<xs:element name="places">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="place"  minOccurs="1"
                                        maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="place">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="p_name"/>
            <xs:element ref="tokens" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Trantsizio bakoitza lau elementuk osatzen dute: izena, denbora, lehentasuna eta semantika. Elementu horietatik izena, lehentasuna eta semantika sinpleak izango dira, baino denboraren kasuan ez da gauza bera gertatzen, elementu hau konplexua izango baita, hots, zenbait azpielementuz osatua egongo da. Azpielementu horiek denbora-mota desberdinak kudeatzeko beharko ditugu: denbora-tarteak, media edota desbiderapen estandarra definitzeko, besteak beste.

```
<xs:element name="transitions">
    <xs:complexType>
        <xs:sequence>
          <xs:element ref="transition" minOccurs="1"
```

```
                                                maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="transition">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="tr_name"/>
                <xs:element ref="time"/>
                <xs:element ref="priority" minOccurs="0"/>
                <xs:element ref="semantics" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

    <xs:element name="time">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="distribution"/>
                <xs:element ref="m" minOccurs="0"/>
                <xs:element ref="a" minOccurs="0"/>
                <xs:element ref="b" minOccurs="0"/>
                <xs:element ref="s" minOccurs="0"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

Azkenik, sarrera eta irteera arkuak definitzeko jarraitu beharreko eskema eta falta diren elementu sinpleak gehitu beharko dira. Sarrera eta irteera arkuak 3 azpielementu sinplez osaturiko elementu konplexuak izango dira. Sarrera arkuen kasuan *from* elementuak iturburu lekua adierazten du eta *to* elementuak, berriz, helburuko trantsizioa. Horrez gain, *multiplicity* atributua ere definitu beharko da, hau da, leku eta trantsizioa elkar lotzen dituen arku kopurua.

Irteera arkuen kasuan elementuak aurrekoen berdinak izango dira *from* eta *to* elementuen esanahia trukatuz. Kasu honetan *from* elementuak

iturburu trantsizioari egiten dio erreferentzia eta *to* elementuak, aldiz, helburuko lekuari.

```
<xs:element name="input_arcs">
    <xs:complexType>
        <xs:sequence>
           <xs:element ref="input_arc" minOccurs="1"
                                              maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="input_arc">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="from"/>
            <xs:element ref="to"/>
            <xs:element ref="multiplicity" minOccurs="0"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="output_arcs">
    <xs:complexType>
        <xs:sequence>
          <xs:element ref="output_arc" minOccurs="1"
                                              maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="output_arc">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="from"/>
            <xs:element ref="to"/>
            <xs:element ref="multiplicity" minOccurs="0"/>
```

```
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Aipatutako elementu sinpleen definizioa eskemaren hasieran idatziko dugu:

```
<!-- definition of simple elements -->
<xs:element name="p_name" type="xs:string"/>
<xs:element name="tokens" type="xs:integer" default="0"/>
<xs:element name="tr_name" type="xs:string"/>
<xs:element name="priority" type="xs:integer" default="0"/>
<xs:element name="semantics" default="single">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="single|delay"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="distribution">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="null"/>
            <xs:enumeration value="constant"/>
            <xs:enumeration value="uniform"/>
            <xs:enumeration value="exponential"/>
            <xs:enumeration value="lognormal"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
<xs:element name="m" type="xs:double"/>
<xs:element name="a" type="xs:double"/>
<xs:element name="b" type="xs:double"/>
<xs:element name="s" type="xs:double"/>
<xs:element name="from" type="xs:integer"/>
<xs:element name="to" type="xs:integer"/>
```

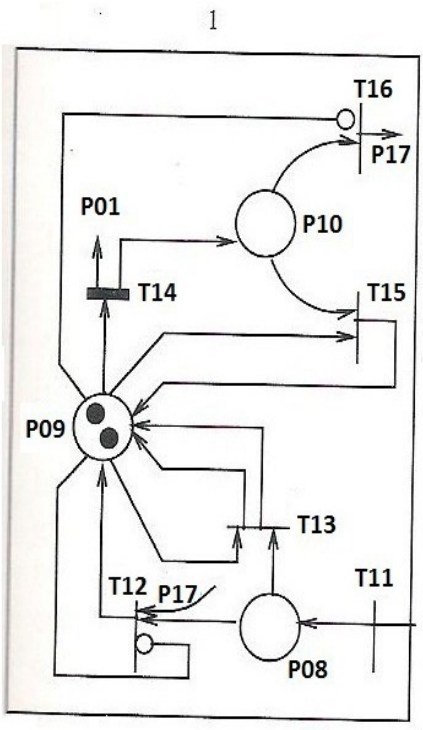
```
<xs:element name="multiplicity" type="xs:integer" default="1"/>
```

Errendimendu parametroen espezifikazioa

Dakigun bezala, proiektu honetan simulatutako sistema multiprozesadorea (5.3 atala) eta haren Petri Sarea (5.3.1 irudia) definitzeko XML dokumentua erabili dugu. Sistema honetan 3 memoria modulu bereizten dira, prozesuak banatzeaz arduratzen direnak hain zuzen ere. Hori horrela izanik, aukera bat '*kola sarea*' kontsideratuz kudeatzea izango da, memoria modulu bakoitza kola bat bezala hartuz. Kola edo ilara sareak, ilara bat baino gehiagoz osaturiko sistemak dira, non ilara horietako bakoitzaren irteera beste baten sarrera bezala onartzen den. Bezeroek erabileren arabera bisitatzen dituzte orokorrean desberdinak izan ohi diren kolak eta gainera, gertaeren arteko dependentziak direla eta, kola sistema hauen ereduak nahiko konplexuak izaten dira.

Oinarrizko errendimendu parametro bezala multiprozesadorearen memoria-modulu bakoitzaren produktibitatea aztertzea da aukeretako bat. Gure ereduan moduluak guztira hiru direnez hiru parametro kalkulatuko genituzke eta hori Petri Sarearen ereduan trantsizio zehatz baten tiro-tasak emango digu (kasu bakoitzerako T14, T20 eta T26 trantsizioak), hau da, denbora unitatean batez bestean eragiketa moduluan zenbat prozesuk amaitzen duten.

Irudian A moduluari dagokion eskema ikus daiteke. Bertan, 11, 12, 13, 15 eta 16. trantsizioak berehalakoak dira eta prozesu batek (edo gehiagok) modulu horretan igarotako denboraren kalkulua T14 trantsizioaren tiro-denborak baldintzatuko du, baina moduluaren erantzun denbora kalkulatzeko tiro-denboraz gain jakin behar da prozesua noiz iritsi den modulura, beraz azterketa sakondu beharko litzateke Petri Sarean honelako estatistikoen definizio orokorra egiteko. Aldiz, moduluaren produktibitatea, T14 trantsizioaren tiro tasa, oso erraz definituko dugu XML dokumentuan, T14 trantsizioa ematea nahikoa izango baita.



7.5.2 irudia: A modulua

Bestalde, sistemaren produktibitate totala, orokorrean moduluen produktibitatearen batura izango litzateke moduluak homogeneoak diren kasuan. Hala ere, heterogeneoak diren kasuan produktibitate totalaren definizioaren egokitasuna aztertu beharko litzateke.

Guzti hori estatistiko bat kalkulatzeko ideia bat besterik ez litzateke, modu berean beste hainbeste kalkulu egin ahal izango direlarik, adibidez, estatistikoen zerrenda bat beharrezkoa den kodearekin XML dokumentuetan gehituz egitura hau jarraituz:

```
<statistics>
      <statistic>
            <throughput>
                  <tr_name>T14</tr_name>
            </throughput>
      </statistic>
      <statistic>
            ...
      </statistic>
</statistics>
```

Adibide horretan oinarrituz ez litzateke konplexua izango XSD eskeman definitu nahi diren estatistikoak gehitzea, produktibitateaz gain egon daitezkeen beste estatistikoak ere zehaztu ondoren.

7.6 Multiprozesadorearen XML espezifikazioa

Jarraian, garatutako aplikazioan erabilitako multiprozesadore sistemaren *XML dokumentuaren* adibidea erakusten da [24] [25]. Petri Sareen atalean azaldutako elementuak erabiltzen dira, hau da, oinarria, lekuak (places), trantsizioak (transitions), sarrera-arkuak (input arcs) eta irteera-arkuak (output arcs) izango dira. XML dokumentuaren eskeletoa honakoa izango da:

```
<!-- Petri-net -->
<petri_net>
      <places>...</places>
      <transitions>...</transitions>
      <input_arcs>...</input_arcs>
      <output_arcs>...</output_arcs>
</petri_net>
```

Lehenengo lerroaren edukia iruzkin bat da, <!-- ... --> ikurrak erabilita. Dokumentuaren *erroa* ***petri_net*** etiketa izango da eta erro horrek hainbat

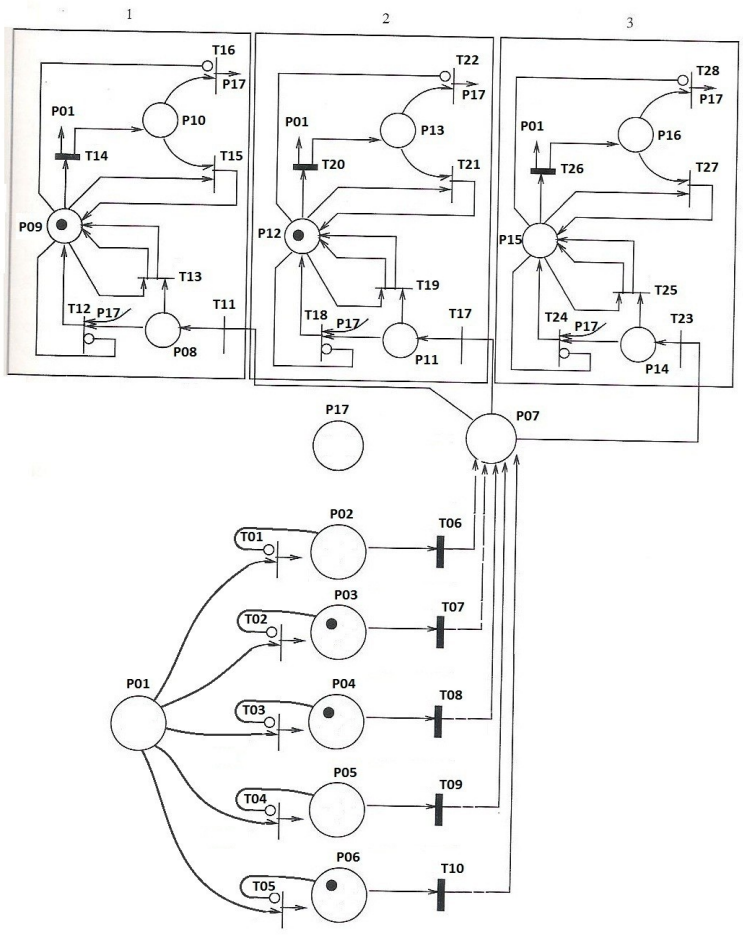
azpielementu *konplexu* izango ditu; ***places***, ***transitions***, ***input_arcs*** eta ***output_arcs***, bakoitza bere hasierako eta bukaerako etiketekin.

Ondoren, elementu konplexu horietako bakoitzaren azpielementuak osatu behar dira. Lehenik eta behin, *places* elementua aztertuko dugu. Honek ***place*** izeneko hainbat azpielementu konplexu izango ditu (Petri Sareak dituen leku kopurua adina elementu) eta leku horietako bakoitza, ***p_name*** (izena edo deskribapena) eta ***tokens*** (lekuaren marka kopurua) elementu sinpleek osatuko dute.

Simulatuko dugun Petri Sarean 17 leku izango ditugu, leku horietako bakoitza definitzeko, aipatutako ***p_name*** elementua erabiltzen delarik. Lekuari dagokion zenbaki edo identifikatzaileaz gain deskribapen bat gehitu zaio horietako bakoitzari. Esate baterako, 9. lekua definitzeko, "*P09: 1. modulua eta bus bat hartuta duten prozesuak*" deskribapena erabiltzen da eta gainontzeko lekuak modu berean definitzen dira.

Bestalde, Petri Sare bat osatzen duten trantsizio bakoitzak, bere izenaz (***tr_name***) gain, tiroa kontrolatuko duen denbora (***time***), lehentasuna (***priority***) eta semantika (***semantics***) elementuak izango ditu. Gainera, denbora-mota desberdinak kudeatzen direnez, time elementua konplexua izango da, hau da, beste hainbat elementuz osatuta egon daiteke, adibidean erabilitako ***distribution***, ***a***, ***b***, ***m*** edota ***s*** elementuen antzekoak. Adibidez 6. trantsizioa banaketa uniformean oinarritzen da aktibatzen den bakoitzean tiro-denbora sortzeko. Trantsizio hau definitzeko "*T06: 1. leku birtualeko prozesuaren operazio/itxoite denbora, memoria eskaria egin aurretik*" deskribapena erabiltzen da eta ***time*** elementu konplexua osatzeko (*2.3, 7.1)* tartea definitzen da.

Geratzen diren elementuak (sarrera-arkuak eta irteera-arkuak) antzeko eran osatzen dira, izan ere, arku bakoitzak bi azpielementu izango baititu (***from***, ***to***). Sarrera-arkuen kasuan, ***from*** elementu sinplearen balioak jatorrizko lekuaren zenbakia adierazten du eta ***to*** elementu sinpleak, berriz, helburuko trantsizio zenbakia. Irteera arkuen kasuan, ordea, balioak aldatu egiten dira, hau da, ***from***-ek jatorrizko trantsizioa adierazten du eta ***to*** elementuaren balioak, aldiz, helburuko lekuaren zenbakia. Hala ere, hurrengo irudian ikus dezakegun Petri Sarea definitzeko sortu den XML dokumentuaren kode osoa 7.7 ataleko eranskinean ikus dezakegu.



7.6.1 irudia: multiprozesadore sistemaren Petri Sarea

7.7 Eranskina

A) Multiprozesadorearen XML dokumentua

```
<!-- Fig. 12-11: GSPN model of a multiprocessor system -->
<petri_net>
    <places>

       <!-- Cache lokaletatik exekutatzen diren prozesuak -->
```

```
       <place>
      <p_name>P01: Memoria modulu baten erabilera amaitu duten
prozesuak</p_name>
         </place>


       <!--  -->
       <place>
         <p_name>P02: Memoria modulurik erabili behar ez duen
bakandutako prozesua 1. leku birtualean</p_name>
       </place>
       <place>
           <p_name>P03: Memoria modulurik erabili behar ez duen
bakandutako prozesua 2. leku birtualean</p_name>
         <tokens>1</tokens>
       </place>
       <place>
           <p_name>P04: Memoria modulurik erabili behar ez duen
bakandutako prozesua 3. leku birtualean</p_name>
         <tokens>1</tokens>
       </place>
       <place>
           <p_name>P05: Memoria modulurik erabili behar ez duen
bakandutako prozesua 4. leku birtualean</p_name>
       </place>
       <place>
           <p_name>P06: Memoria modulurik erabili behar ez duen
bakandutako prozesua 5. leku birtualean</p_name>
         <tokens>1</tokens>
       </place>


       <!-- Memoria konpartitua erabiliko duten prozesuak -->
         <place>
             <p_name>P07: memoria-modulua aukeratu</p_name>
               <tokens>0</tokens>
         </place>


       <!-- A memoria moduluari dagozkien lekuak -->
         <place>
```

```
            <p_name>P08: 1. modulua hartzeko prest</p_name>
          </place>
        <place>
            <p_name>P09: 1. modulua eta bus bat hartuta duten
prozesuak</p_name>
          <tokens>1</tokens>
        </place>
        <place>
            <p_name>P10: 1. moduluak hartutako busa, prozesu batek
mouduaren erabilera amaitu ondoren</p_name>
        </place>

        <!-- B memoria moduluari dagozkien lekuak -->
          <place>
              <p_name>P11: 2. modulua hartzeko prest</p_name>
          </place>
        <place>
            <p_name>P12: 2. modulua eta bus bat hartuta duten
prozesuak</p_name>
          <tokens>1</tokens>
        </place>
        <place>
            <p_name>P13: 2. moduluak hartutako busa, prozesu batek
mouduaren erabilera amaitu ondoren</p_name>
        </place>

        <!-- C memoria moduluari dagozkien lekuak -->
          <place>
              <p_name>P14: 3. modulua hartzeko prest</p_name>
          </place>
        <place>
            <p_name>P15: 3. modulua eta bus bat hartuta duten
prozesuak</p_name>
        </place>
        <place>
            <p_name>P16: 3. moduluak hartutako busa, prozesu batek
mouduaren erabilera amaitu ondoren</p_name>
        </place>
```

```
        <!-- Free busses (erabilgarri dauden busak) -->
        <place>
            <p_name>P17: libre dauden busak</p_name>
        </place>
    </places>

    <transitions>

        <!--  -->
         <transition>
             <tr_name>T01: Prozesu bat bakandu 1. leku
birtualera</tr_name>
              <time>
                 <distribution>null</distribution>
              </time>
         </transition>
       <transition>
           <tr_name>T02: Prozesu bat bakandu 2. leku
birtualera</tr_name>
           <time>
               <distribution>null</distribution>
           </time>
       </transition>
       <transition>
           <tr_name>T03: Prozesu bat bakandu 3. leku
birtualera</tr_name>
           <time>
               <distribution>null</distribution>
           </time>
       </transition>
       <transition>
           <tr_name>T04: Prozesu bat bakandu 4. leku
birtualera</tr_name>
           <time>
               <distribution>null</distribution>
           </time>
```

```
        </transition>
        <transition>
            <tr_name>T05: Prozesu bat bakandu 5. leku
birtualera</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>


         <!--  -->
        <transition>
            <tr_name>T06: 1. leku birtualeko prozesuaren
operazio/itxoite denbora, memoria eskaria egin aurretik</tr_name>
                <time>
                   <distribution>uniform</distribution>
                  <a>1.2</a>
                  <b>4.7</b>
                </time>
          </transition>
        <transition>
            <tr_name>T07: 2. leku birtualeko prozesuaren
operazio/itxoite denbora, memoria eskaria egin aurretik</tr_name>
            <time>
                <distribution>uniform</distribution>
                <a>2.3</a>
                <b>5.0</b>
            </time>
        </transition>
        <transition>
            <tr_name>T08: 3. leku birtualeko prozesuaren
operazio/itxoite denbora, memoria eskaria egin aurretik</tr_name>
            <time>
                <distribution>uniform</distribution>
                <a>3.3</a>
                <b>8.6</b>
            </time>
        </transition>
```

```
       <transition>
           <tr_name>T09: 4. leku birtualeko prozesuaren
operazio/itxoite denbora, memoria eskaria egin aurretik</tr_name>
           <time>
               <distribution>uniform</distribution>
               <a>2.1</a>
               <b>6.7</b>
           </time>
       </transition>
       <transition>
           <tr_name>T10: 5. leku birtualeko prozesuaren
operazio/itxoite denbora, memoria eskaria egin aurretik</tr_name>
           <time>
               <distribution>uniform</distribution>
               <a>1.4</a>
               <b>5.9</b>
           </time>
       </transition>

        <!-- 1. memoria moduluari dagozkion trantsizioak -->
       <transition>
             <tr_name>T11: 1. modulua aukeratu</tr_name>
             <time>
                <distribution>null</distribution>
             </time>
         </transition>
       <transition>
           <tr_name>T12: bus berri bat okupatu eta modulua
erabili</tr_name>
           <time>
               <distribution>null</distribution>
           </time>
       </transition>
       <transition>
           <tr_name>T13: prozesu berria 1. modulua erabiltzera,
aurretik gutxienez beste prozesu bat modulua erabiltzen ari
dela</tr_name>
           <time>
```

```
                <distribution>null</distribution>
            </time>
          </transition>
        <transition>
            <tr_name>T14: prozesu baten eragiketa denbora 1.
moduluan</tr_name>
            <time>
                <distribution>exponential</distribution>
                <m>2.0</m>
            </time>
          </transition>
        <transition>
            <tr_name>T15: 1. moduluak hartutako busak okupatuta
jarraitu beste prozesu batentzat</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
         <transition>
             <tr_name>T16: busa askatu 1. moduluak</tr_name>
             <time>
                 <distribution>null</distribution>
             </time>
         </transition>

         <!-- 2. memoria moduluari dagozkion trantsizioak -->
         <transition>
             <tr_name>T17: 2. modulua aukeratu</tr_name>
             <time>
                 <distribution>null</distribution>
             </time>
         </transition>
         <transition>
             <tr_name>T18: busa okupatu</tr_name>
             <time>
                 <distribution>null</distribution>
             </time>
```

```
        </transition>
        <transition>
            <tr_name>T19: prozesuak ez du busa okupatu</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
        <transition>
            <tr_name>T20: 2. moduluan eragiketak egin</tr_name>
            <time>
                <distribution>exponential</distribution>
                <m>2.0</m>
            </time>
        </transition>
        <transition>
            <tr_name>T21: 2. moduluan zain dagoen prozesu batentzat
busa askatu</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
        <transition>
            <tr_name>T22: busa askatu kanpoko prozesu
batentzat</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>

        <!-- 3. memoria moduluari dagozkion trantsizioak -->
        <transition>
            <tr_name>T23: 3. modulua aukeratu</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
        <transition>
```

```
            <tr_name>T24: busa okupatu</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
        <transition>
            <tr_name>T25: prozesuak ez du busa okupatu</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
        <transition>
            <tr_name>T26: 3. moduluan eragiketak egin</tr_name>
            <time>
                <distribution>exponential</distribution>
                <m>2.0</m>
            </time>
        </transition>
        <transition>
            <tr_name>T27: 3. moduluan zain dagoen prozesu batentzat
busa askatu</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
        <transition>
            <tr_name>T28: busa askatu kanpoko prozesu
batentzat</tr_name>
            <time>
                <distribution>null</distribution>
            </time>
        </transition>
    </transitions>

    <!-- from Place to Transition -->
    <input_arcs>
        <!-- from P01 to T01, T02, T03, T04, T05 -->
```

```
<input_arc>
      <from>01</from>
      <to>01</to>
  </input_arc>
  <input_arc>
      <from>01</from>
      <to>02</to>
  </input_arc>
  <input_arc>
      <from>01</from>
      <to>03</to>
  </input_arc>
  <input_arc>
      <from>01</from>
      <to>04</to>
  </input_arc>
  <input_arc>
      <from>01</from>
      <to>05</to>
  </input_arc>

  <!-- Inhibitzaileak -->
  <input_arc>
      <from>02</from>
      <to>01</to>
      <multiplicity>0</multiplicity>
  </input_arc>
  <input_arc>
      <from>03</from>
      <to>02</to>
      <multiplicity>0</multiplicity>
  </input_arc>
  <input_arc>
      <from>04</from>
      <to>03</to>
      <multiplicity>0</multiplicity>
```

```
</input_arc>
<input_arc>
    <from>05</from>
    <to>04</to>
    <multiplicity>0</multiplicity>
</input_arc>
<input_arc>
    <from>06</from>
    <to>05</to>
    <multiplicity>0</multiplicity>
</input_arc>

<!-- from P02 to T06, from P03 to T07... -->
<input_arc>
    <from>02</from>
    <to>06</to>
</input_arc>
<input_arc>
    <from>03</from>
    <to>07</to>
</input_arc>
<input_arc>
    <from>04</from>
    <to>08</to>
</input_arc>
<input_arc>
    <from>05</from>
    <to>09</to>
</input_arc>
<input_arc>
    <from>06</from>
    <to>10</to>
</input_arc>

<!-- from P07 to T11, T17, T23 -->
<input_arc>
```

```
    <from>07</from>
    <to>11</to>
</input_arc>
<input_arc>
    <from>07</from>
    <to>17</to>
</input_arc>
<input_arc>
    <from>07</from>
    <to>23</to>
</input_arc>

<!-- 1.moduluko sarrera-arkuak -->
<!-- from P08 -->
<input_arc>
    <from>08</from>
    <to>12</to>
</input_arc>
<input_arc>
    <from>08</from>
    <to>13</to>
</input_arc>
<!-- from P09 (inhibitzailea) -->
<input_arc>
    <!-- inhibitzailea -->
    <from>09</from>
    <to>12</to>
    <multiplicity>0</multiplicity>
</input_arc>
<input_arc>
    <from>09</from>
    <to>13</to>
</input_arc>
<input_arc>
    <from>09</from>
    <to>14</to>
```

```
</input_arc>
<input_arc>
    <from>09</from>
    <to>15</to>
</input_arc>
<input_arc>
    <!-- inhibitzailea -->
    <from>09</from>
    <to>16</to>
    <multiplicity>0</multiplicity>
</input_arc>
<!-- from P10 -->
<input_arc>
    <from>10</from>
    <to>15</to>
</input_arc>
<input_arc>
    <from>10</from>
    <to>16</to>
</input_arc>

<!-- 2. moduluko sarrera-arkuak -->
<!-- from P11 -->
<input_arc>
    <from>11</from>
    <to>18</to>
</input_arc>
<input_arc>
    <from>11</from>
    <to>19</to>
</input_arc>
<!-- from P12 -->
<input_arc>
    <!-- inhibitzailea -->
    <from>12</from>
    <to>18</to>
```

```
    <multiplicity>0</multiplicity>
</input_arc>
<input_arc>
    <from>12</from>
    <to>19</to>
</input_arc>
<input_arc>
    <from>12</from>
    <to>20</to>
</input_arc>
<input_arc>
    <from>12</from>
    <to>21</to>
</input_arc>
<input_arc>
    <!-- inhibitzailea -->
    <from>12</from>
    <to>22</to>
    <multiplicity>0</multiplicity>
</input_arc>
<!-- from P13 -->
<input_arc>
    <from>13</from>
    <to>21</to>
</input_arc>
<input_arc>
    <from>13</from>
    <to>22</to>
</input_arc>

<!-- 3. moduluko sarrera-arkuak -->
<!-- from P14 -->
<input_arc>
    <from>14</from>
    <to>24</to>
</input_arc>
```

```
<input_arc>
    <from>14</from>
    <to>25</to>
</input_arc>
<!-- from P15 -->
<input_arc>
    <!-- inhibitzailea -->
    <from>15</from>
    <to>24</to>
    <multiplicity>0</multiplicity>
</input_arc>
<input_arc>
    <from>15</from>
    <to>25</to>
</input_arc>
<input_arc>
    <from>15</from>
    <to>26</to>
</input_arc>
<input_arc>
    <from>15</from>
    <to>27</to>
</input_arc>
<input_arc>
    <!-- inhibitzailea -->
    <from>15</from>
    <to>28</to>
    <multiplicity>0</multiplicity>
</input_arc>
<!-- from P16 -->
<input_arc>
    <from>16</from>
    <to>27</to>
</input_arc>
<input_arc>
    <from>16</from>
```

```
            <to>28</to>
        </input_arc>

        <!-- from P17 to T12, T18, T24 -->
        <input_arc>
            <from>17</from>
            <to>12</to>
        </input_arc>
        <input_arc>
            <from>17</from>
            <to>18</to>
        </input_arc>
        <input_arc>
            <from>17</from>
            <to>24</to>
        </input_arc>
    </input_arcs>

    <!-- from Transition to Place -->
    <output_arcs>
        <!-- from T01 to P02, from T02 to P03... -->
        <output_arc>
            <from>01</from>
            <to>02</to>
        </output_arc>
        <output_arc>
            <from>02</from>
            <to>03</to>
        </output_arc>
        <output_arc>
            <from>03</from>
            <to>04</to>
        </output_arc>
        <output_arc>
            <from>04</from>
            <to>05</to>
```

```
</output_arc>
<output_arc>
    <from>05</from>
    <to>06</to>
</output_arc>

<!-- to P07 -->
<output_arc>
    <from>06</from>
    <to>07</to>
</output_arc>
<output_arc>
    <from>07</from>
    <to>07</to>
</output_arc>
<output_arc>
    <from>08</from>
    <to>07</to>
</output_arc>
<output_arc>
    <from>09</from>
    <to>07</to>
</output_arc>
<output_arc>
    <from>10</from>
    <to>07</to>
</output_arc>

<!-- 1. moduluko irteera-arkuak -->
<!-- from T11 to P08 -->
<output_arc>
    <from>11</from>
    <to>08</to>
</output_arc>

<!-- from T12 to P09 -->
```

```
<output_arc>
    <from>12</from>
    <to>09</to>
</output_arc>
<!-- from T13 to P09 -->
<output_arc>
    <from>13</from>
    <to>09</to>
    <multiplicity>2</multiplicity>
</output_arc>

<!-- from T14 to P01, P10 -->
<output_arc>
    <from>14</from>
    <to>01</to>
</output_arc>
<output_arc>
    <from>14</from>
    <to>10</to>
</output_arc>

<!-- from T15 to P09 -->
<output_arc>
    <from>15</from>
    <to>09</to>
</output_arc>

<!-- from T16 to P17 -->
<output_arc>
    <from>16</from>
    <to>17</to>
</output_arc>

<!-- 2. moduluko irteera-arkuak -->
<!-- from T17 to P1 -->
<output_arc>
```

```
    <from>17</from>
    <to>11</to>
</output_arc>

<!-- from T18 to P12 -->
<output_arc>
    <from>18</from>
    <to>12</to>
</output_arc>

<!-- from T19 to P12 -->
<output_arc>
    <from>19</from>
    <to>12</to>
    <multiplicity>2</multiplicity>
</output_arc>

<!-- from T20 to P01, P13 -->
<output_arc>
    <from>20</from>
    <to>01</to>
</output_arc>
<output_arc>
    <from>20</from>
    <to>13</to>
</output_arc>

<!-- from T21 to P12 -->
<output_arc>
    <from>21</from>
    <to>12</to>
</output_arc>

<!-- from T22 to P17 -->
<output_arc>
    <from>22</from>
```

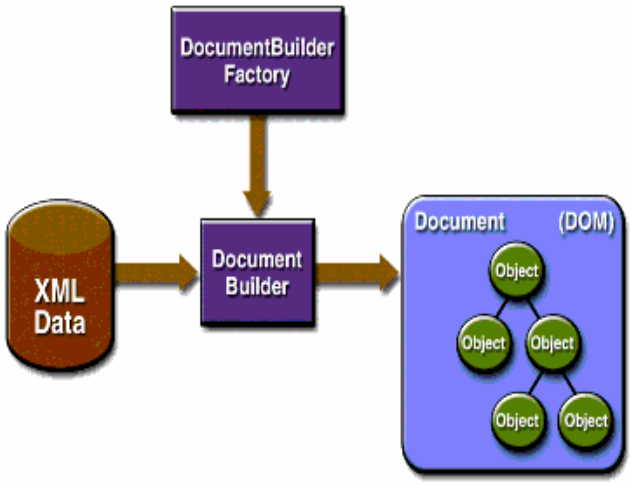
```
    <to>17</to>
</output_arc>

<!-- 3. moduluko irteera-arkuak -->
<!-- from T23 to P14 -->
<output_arc>
    <from>23</from>
    <to>14</to>
</output_arc>

<!-- from T24 to P15 -->
<output_arc>
    <from>24</from>
    <to>15</to>
</output_arc>

<!-- from T25 to P15 -->
<output_arc>
    <from>25</from>
    <to>15</to>
    <multiplicity>2</multiplicity>
</output_arc>

<!-- from T26 to P01, P16 -->
<output_arc>
    <from>26</from>
    <to>01</to>
</output_arc>
<output_arc>
    <from>26</from>
    <to>16</to>
</output_arc>

<!-- from T27 to P15 -->
<output_arc>
    <from>27</from>
```

```
            <to>15</to>
        </output_arc>

        <!-- from T28 to P17 -->
        <output_arc>
            <from>28</from>
            <to>17</to>
        </output_arc>
    </output_arcs>
</petri_net>
```

8. KAPITULUA

PETRI SAREETARAKO KLASEAK

8.1 Sarrera



Gure aplikazioa osatuko duten klaseak garatzen hasi aurretik, XML dokumentuak nola prozesatzen diren aztertuko dugu, hau da, XML bidez zehaztutako ereduetatik Java klaseen instantzia izango diren objektuak nola sortzen diren. Hasteko, interesgarria da DOM egitura aztertu eta ulertzea [27] [28]. DOM bat zuhaitz egitura bat da, non bere adabegi bakoitzak XML egitura bateko osagai bat izango duen. Zuhaitz hori osatzen duten adabegi moten artean, ohikoenak elementu eta testu motakoak dira. Gainera, DOM funtzio desberdinekin adabegiak sortu, ezeztatu, balioak aldatu edota hauen edukia atzitu ahal izateko aukera eskaintzen da.

DOM bat sortzeko eman beharreko pausoak aztertu aurretik, DOM baten egitura nolakoa den ulertzea lagungarri suertatzen da. Horretarako, JAXP APIa instalatu [21] eta ***INSTALL_DIR/JAXP-versión/samples/dom*** direktorioan aurkitzen den DomEcho programan oinarritu gaitezke. Hala ere, interesgarriak izango dira DomEcho programaren sorrerarako eman beharreko pausoak aztertzea.

8.2 DOM baten sorrera

- **Eskeletoaren sorrera**

Lehenik eta behin, XML dokumentu bat DOM batean irakurri eta ondoren inprimatuko duen programa eraiki behar da. Horretarako, oinarrizko kodea honako hau izango da, komando lerrotik argumentu bat pasako dela

ziurtatzeaz arduratzen dena:

```
public class DOMEcho {

      static final String outputEncoding = "UTF-8";

      private static void usage() {
      // ...
      }

      public static void main(String[] args) throws Exception {
            String filename = null;

            for (int i = 0; i < args.length; i++) {
                  if (...) {
                  // ...
            }
                  else {
                        filename = args[i];
                        if (i != args.length - 1) {
                              usage();
                        }
                  }
            }

            if (filename == null) {
                  usage();
            }
      }
}
```

- **Beharrezko klaseak inportatu**

Hauek dira DomEcho programak erabiltzen dituen JAXP APIak:

```
package dom;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
```

XML dokumentu bat aztertzerakoan sor daitezkeen salbuespenetarako klaseak:

```
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
```

```
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.*
```

XML dokumentua irakurri eta sarrera-irteera maneiatzeko:

```
import java.io.File;
import java.io.OutputStreamWriter;
import java.io.Printwriter;
```

W3C definizioak, DOM salbuespenak, entitateak eta adabegiak inportatzeko:

```
import org.w3c.dom.Document;
import org.w3c.dom.DocumentType;
import org.w3c.dom.Entity;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
```

● Errore kontrola

DomEcho-k XML dokumentu bat aztertzerakoan errore-kontrol bat erabiltzea beharrezkoa da. Izan ere, JAXP dokumentu eraikitzaileak arazoak dituenean, SAX salbuespenen inguruan informatzea behartuta dago. Jarraian datorren kodea da DOM aplikazioetan erabili ohi dena:

```
private static class MyErrorHandler implements ErrorHandler {

      private PrintWriter out;

      MyErrorHandler(PrintWriter out) {
            this.out = out;
      }

   private String getParseExceptionInfo(SAXParseException spe) {
            String systemId = spe.getSystemId();
            if (systemId == null) {
                  systemId = "null";
            }

            String info = "URI=" + systemId + " Line=" +
                                    spe.getLineNumber() + ": " +
                                                spe.getMessage();
            return info;
      }

      public void warning(SAXParseException spe) throws
                                                      SAXException {
            out.println("Warning:" + getParseExceptionInfo(spe));
      }
```

```
        public void error(SAXParseException spe) throws
                                                SAXException {
              String message = "Error: " +
                               getParseExceptionInfo(spe);
              throw new SAXException(message);
        }

        public void fatalError(SAXParseException spe) throws
                                                SAXException {
              String message = "Fatal Error: " +
                               getParseExceptionInfo(spe);
              throw new SAXException(message);
        }
  }
```

● Lantegia instantziatu

Lantegi baten instantziatik abiatuta, dokumentu eraikitzaile bat lortu ahal izateko, metodo estatiko nagusian gehitu beharreko kodea hau da:

```
public static void main(String[] args) throws Exception {
      DocumentBuilderFactory dbf =
                        DocumentBuilderFactory.newInstance();

      // ...
}
```

● Aztertzailea lortu eta fitxategia aztertu

Jarraian dagoen kodea *main* metodoan gehituz gero, eraikitzailearen instantzia bat lortu ahal izango da, zehaztutako fitxategia aztertzen lagunduko diguna, hain zuzen ere:

```
DocumentBuilderFactory dbf =
                  DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse(new File(filename));
```

filename izeneko aldagaian gordeko da DomEcho programa exekutatzean argumentu bezala pasako den fitxategiaren izena.

- **Lantegiaren konfigurazioa**

DomEcho-k jasoko duen XML fitxategiaz gain, dokumentu hori balioztatuko duen DTD edo XML SCHEMA fitxategiak ere komando lerrotik argumentu bezala pasatzea posible da. Hala ere, XML dokumentu bat balioztatzeko era desberdinak erabil badaitezke ere, lantegiaren oinarrizko konfigurazioa ondorengo kodeak egiten du:

```
public static void main(String[] args) throws Exception {

      String filename = null;
      boolean dtdValidate = false;
      boolean xsdValidate = false;
      String schemaSource = null;

      for (int i = 0; i < args.length; i++) {
            if (args[i].equals("-dtd"))  {
            dtdValidate = true;
            }
            else if (args[i].equals("-xsd")) {
                  xsdValidate = true;
            }
            else if (args[i].equals("-xsdss")) {
                  if (i == args.length - 1) {
                        usage();
                  }
                  xsdValidate = true;
                  schemaSource = args[++i];
            }
            else {
                  filename = args[i];
                  if (i != args.length - 1) {
                        usage();
                  }
            }
      }

      if (filename == null) {
            usage();
      }

      DocumentBuilderFactory dbf =
                        DocumentBuilderFactory.newInstance();

      dbf.setNamespaceAware(true);
      dbf.setValidating(dtdValidate || xsdValidate);

      // ...

      DocumentBuilder db = dbf.newDocumentBuilder();
      Document doc = db.parse(new File(filename));
}
```

• DOM adabegiak pantailaratu

Behin DOM egitura sortzen denean, egitura bera, edukia eta bere funtzionamendua ulertzeko erarik aproposena adabegiak pantailaratu eta informazioa aztertzea da. Jarraian erakusten den taula oso erabilgarria da DOM batekin lanean aritzeko garaian:

Node	**nodeName**	**nodeValue**	**Attributes**
Attr	Name of attribute	Value ofattribute	null
CDATASection	#cdata-section	Content of the CDATA section	null
Comment	#comment	Content of the comment	null
Document	#document	null	null
DocumentFragment	#documentFragment	null	null
DocumentType	Document Type name	null	null
Element	Tag name	null	null
Entity	Entity name	null	null
EntityReference	Name of entity referenced	null	null
Notation	Notation name	null	null
ProcessingInstruction	Target	Entire content excluding the target	null
Text	#text	Content of the text node	null

Taula 8.2: Adabegi moten taula

- **Adabegi motaren informazioa eskuratu**

org.w3c.dom.Node klaseko metodoez baliatuz, adabegi bakoitzari dagokion informazioa lor daiteke. DomEcho programan horretarako erabiltzen den metodoa *printlnCommon* izenekoa da eta hau da bere kodea:

```
private void printlnCommon(Node n) {
      out.print(" nodeName=\"" + n.getNodeName() + "\"");

      String val = n.getNamespaceURI();
      if (val != null) {
            out.print(" uri=\"" + val + "\"");
      }

      val = n.getPrefix();

      if (val != null) {
            out.print(" pre=\"" + val + "\"");
      }

      val = n.getLocalName();
      if (val != null) {
            out.print(" local=\"" + val + "\"");
      }

      val = n.getNodeValue();
      if (val != null) {
            out.print(" nodeValue=");
            if (val.trim().equals("")) {
                  // Whitespace
                  out.print("[WS]");
            }
            else {
                  out.print("\"" + n.getNodeValue() + "\"");
            }
      }
      out.println();
}
```

Bertan ikus daitekeenez, Node motako argumentua jasoko du eta metodo desberdinak erabiliz, adabegi jakin bati dagozkion mota, izena edota balioa lortzea ahalbidetuko du metodo honek.

- **DOM zuhaitzeko adabegiak inprimatu**

Esan bezala, aplikazio bat garatzerakoan oso erabilgarria izango da *DomEcho* aplikazioak sortzen duen egituraren errepresentazioa pantailaratzea. Horretarako, adabegi guztiak inprimatu beharko dira beraien informazioarekin. Garrantzitsua da jakitea DOM zuhaitz batean gehien azalduko diren adabegiak *Element* eta *Text* motakoak izango direla. Baina ez hori bakarrik, *Text* motakoak beti *Element* adabegi ume izateaz gain, atzitu nahiko ditugun datuen jabe ere izango dira.

Honako *echo* izeneko metodoak, adabegi konkretu baten informazioa pantailaratzen lagunduko digu, horretarako out izeneko *PrintWriter* motako aldagaia erabiliko delarik:

```
private void echo(Node n) {
      outputIndentation();
      int type = n.getNodeType();

      switch (type) {
            case Node.ATTRIBUTE_NODE:
                  out.print("ATTR:");
                  printlnCommon(n);
                  break;

            case Node.CDATA_SECTION_NODE:
                  out.print("CDATA:");
                  printlnCommon(n);
                  break;

            case Node.COMMENT_NODE:
                  out.print("COMM:");
                  printlnCommon(n);
                  break;

            case Node.DOCUMENT_FRAGMENT_NODE:
                  out.print("DOC_FRAG:");
                  printlnCommon(n);
                  break;

            case Node.DOCUMENT_NODE:
                  out.print("DOC:");
                  printlnCommon(n);
                  break;

            case Node.DOCUMENT_TYPE_NODE:
                  out.print("DOC_TYPE:");
                  printlnCommon(n);
                  NamedNodeMap nodeMap =
                        ((DocumentType)n).getEntities();
                  indent += 2;
                  for (int i = 0; i < nodeMap.getLength(); i++) {
                        Entity entity = (Entity)nodeMap.item(i);
```

```
                        echo(entity);
                    }
                    indent -= 2;
                    break;

                case Node.ELEMENT_NODE:
                    out.print("ELEM:");
                    printlnCommon(n);

                    NamedNodeMap atts = n.getAttributes();
                    indent += 2;
                    for (int i = 0; i < atts.getLength(); i++) {
                        Node att = atts.item(i);
                        echo(att);
                    }
                    indent -= 2;
                    break;

                case Node.ENTITY_NODE:
                    out.print("ENT:");
                    printlnCommon(n);
                    break;

                case Node.ENTITY_REFERENCE_NODE:
                    out.print("ENT_REF:");
                    printlnCommon(n);
                    break;

                case Node.NOTATION_NODE:
                    out.print("NOTATION:");
                    printlnCommon(n);
                    break;

                case Node.PROCESSING_INSTRUCTION_NODE:
                    out.print("PROC_INST:");
                    printlnCommon(n);
                    break;

                case Node.TEXT_NODE:
                    out.print("TEXT:");
                    printlnCommon(n);
                    break;

                default:
                    out.print("UNSUPPORTED NODE: " + type);
                    printlnCommon(n);
                    break;
            }
            indent++;
            for (Node child = n.getFirstChild(); child != null;
                 child = child.getNextSibling()) {
                echo(child);
            }
            indent--;
        }
```

8.3 XML schema bidezko balioztatzea

XML dokumentu bat balioztatzeko era desberdinen artean aurkitzen da jarraian azaldu eta aplikazioa garatzean erabili dena, *XML Schema*, hain zuzen ere. Balioztatze-prozesu horretan sor daitezkeen erroren informazioa lortu ahal izateko, ezinbestekoak dira honako hauek:

- Lantegiak errore-kontrol zuzena konfiguratu eta aplikatu behar du.
- Dokumentua eskema bat edo gehiagorekin erlazionatuta egon behar da.

- **DocumentBuilder Factory lantegiaren konfigurazioa**

Erabilgarriena lantegiaren konfiguraziorako erabiliko diren konstanteen definizioa finkatzearekin hastea da:

```
static final String JAXP_SCHEMA_LANGUAGE =

      "http://java.sun.com/xml/jaxp/properties/schemaLanguage";

static final String W3C_XML_SCHEMA =
      "http://www.w3.org/2001/XMLSchema";
```

Ondoren, *DocumentBuilderFactory* klaseko *setValidating* metodoa erabiliz, izen-espazioak sortzeko egokituko da *DocumentBuilderFactory* lantokia, kode hau gehituz:

```
// ...
dbf.setNamespaceAware(true);
dbf.setValidating(dtdValidate || xsdValidate);

if (xsdValidate) {
      try {
            dbf.setAttribute(JAXP_SCHEMA_LANGUAGE,
                                        W3C_XML_SCHEMA);
      }
      catch (IllegalArgumentException x) {
            System.err.println("Error: JAXP
                        DocumentBuilderFactory attribute " + "not
                        recognized: " + JAXP_SCHEMA_LANGUAGE);

            System.err.println("Check to see if parser conforms
                                        to JAXP 1.2 spec.");
```

```
                System.exit(1);
            }
        }
        // ...
```

● Dokumentua eta eskemaren arteko lotura finkatu

XML dokumentu bat XML Schema bat edo gehiagorekin erlazionatzen dela ziurtatzeko bi modu nagusi bereizten dira. Alde batetik, XML dokumentuan eskemaren berri eman daiteke kode egokia gehituz, eta bestetik, aplikazioan bertan ezartzea ere posible da. Aipatu beharrekoa da, bigarren aukera hautatuz gero, dokumentuan erabilitako edozein eskema deklarazio baztertuko dela.

Eskemaren definizioa dokumentuan finkatzeko, honelako kode-lerroa jarri beharko litzateke:

```
        <documentRoot
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation='YourSchemaDefinition.xsd'>
[…]
```

Lehen atributuak, XML izen-espazioaren balioa adierazten du, XML eskemaren adibidea esanahia duena. Bigarren lerroak izen-espaziorik gabeko dokumentuko elementuentzako erabiliko den eskemaren berri emango digu, normalean edozein oinarrizko XML dokumentutan erabiltzen diren elementuentzat, hain zuzen ere.

Hala ere, lehen aipatu dugun eran, posible da eskemaren fitxategi konkretu bat aplikazioan bertan espezifikatzea, *DomEcho*-k egiten duen modu berean:

```
        static final String JAXP_SCHEMA_SOURCE =

              "http://java.sun.com/xml/jaxp/properties/schemaSource";

        // ...

        dbf.setValidating(dtdValidate || xsdValidate);
        if (xsdValidate) {
              // ...
        }
        if (schemaSource != null) {
              dbf.setAttribute(JAXP_SCHEMA_SOURCE, new
                                          File(schemaSource));
        }
```

8.4 XPath

Xpath (XML Path Language) XML dokumentu bat prozesatu eta korritzeko espresioak eraikitzea ahalbidetzen duen lengoaia da [29]. Ideia espresio erregularrenaren antzekoa da, baino kasu honetan XML-ri aplikatuta.

Behin XML dokumentutik abiatuta DOM zuhaitza sortu denean, oso erabilgarria izango da *Xpath* lengoaia, izan ere, posible baita marka bakoitza direktorio bat bezala kontsideratzea. Espresio desberdinak helbide absolutu moduan erabiliz zuhaitzeko adabegiak multzokatzeko aukera eskaintzen digu. Adibidez, ***/markaNagusia/umearenMarka*** helbidearekin, *Xpath*-ek gure XML-ko ***umearenMarka*** hautatuko luke.

• Xpath DOM-ekin erabiltzen

DOM-ekin *Document* motako objektuak sortzeko lantokia edo DocumentBuilderFactory erabiltzen den bezala, *Xpath*-ek ere *XpathFactory* izeneko lantokia erabiltzen du. Horrela, *Xpath* instantzia eskuratzean *evaluate* metodoa eskaintzen digu DOM-ekin batera lan egin ahal izateko. Honako adibidean ikus dezakegu nola erabil daitekeen:

```
NodeList placeTags = (NodeList)
(XPathFactory.newInstance().newXPath().evaluate("//place",
                              doc,  XPathConstants.NODESET));
```

evaluate metodoak jasoko dituen parametroak 3 izango dira. Alde batetik, lehen aipatutako helbide absolutuaren espresioa (kasu honetan "***place***" marka duten adabegiak multzokatu nahi dira). Bestetik, jada sortu dugun *Document* objektua, XML dokumentuaren edukia memorian zuhaitz forman gordeta dagoena. Azkenik, metodoaren exekuzioarekin batera jasotzea espero dena; kasu honetan ***NODESET***-ek adabegien zerrenda bat itzuliko digula adierazten du. Hala ere, adabegi zerrendaz gain, adabegi bat, boolear bat, zenbaki bat edota karaktere-kate bat jasotzea ere posible da horretarako egokiak diren espresioak erabiliz.

8.5 Petri Sareak programatzeko klaseak

Java ingurunean XML teknologiak maneiatu behar direnean, oso lagungarria izango da gure aplikazioaren oinarritzat lehen aipatutako *DomEcho* programa hartzea [28]. Aurretik ikusi dugu lortzen den DOM instantziaren edukia pantailaratzeak bere funtzionamendua ulertzen laguntzen duela erabat, baina orain, eraiki den DOM objektutik abiatuta, Petri sareak eraikitzeko pausoa eman behar da.

Aipatuko ditugun klaseak, ***PNFromXML***, ***PetriNet*** eta Petri Sare baten oinarri diren lekuak (***Place***), trantsizioak (***Transition***), sarrera arkuak (***InputArc***) eta irteera arkuak (***OutputArc***) eraikitzeko beharrezkoak diren klaseak dira.

8.5.1 PNFromXML klasea

Klase honek *buildPetriNet* izeneko oinarrizko metodo bat izango du, *Document* motako objektu bat argumentu bezala jasota, Petri Sarea eraikitzeaz arduratuko dena. Metodoaren eskeletoa jarraian agertzen dena izango da:

```
public PetriNet buildPetriNet(Document doc) throws
                        XPathExpressionException {

      PetriNet pn = new PetriNet();

      //Lekuak eraikitzeko kodea

      //Trantsizioak eraikitzeko kodea

      //Sarrerako arkuak eraikitzeko kodea

      //Irteerako arkuak eraikitzeko kodea

      return pn;
}
```

Metodoa lau zati nagusitan banatzen da; lekuak, trantsizioak, sarrera-arkuak eta irteera arkuak sortzen diren zatietan, hain zuzen ere. Horrez gain, hasieran Petri Sare baten instantzia bat sortuko da (***pn***), gero bukaeran itzuliko dena (***return pn***).

● Lekuak eraikitzen

Lekuak eraikitzerako garaian, lehen aipatutako *XPath* teknologiaz baliatuko gara DOM zuhaitzetik interesatzen zaizkigun adabegiak multzokatzeko [29].

Kasu honetan lekuekin zerikusia duten elementuak atzitu nahiko ditugu, baino aurrerago ikusiko dugun moduan, trantsizioak edota arkuak sailkatzeko ere erabilgarria izango da teknologia hau. Honela, *Document* motako egitura aztertu eta lekuak eraikitzeko beharrezkoa den kode-zatia honakoa da:

```
NodeList placeTags = (NodeList)
                  (XpathFactory.newInstance().newXPath().evaluate
                  ("//place", doc, XPathConstants.NODESET));
for (int i=0; i<placeTags.getLength(); i++){
      Node node = placeTags.item(i);
      try{
            if (node instanceof Element &&
                  node.getLocalName().trim().equals("place")){
                  NodeList children = node.getChildNodes();
                  Place place = buildPlace(children);
                  pn.addPlace(place);
            }
      }catch(Exception e){
            e.printStackTrace();
      }
}
```

placeTags izeneko aldagaiak, lekuekin zerikusia duten adabegiak gordeko ditu eta ondoren hauek banan bana aztertuko dira. Gainera, *Element* motako leku bakoitzaren umeen zerrendarekin (***children***) eta ***buildPlace*** metodoaren bidez, lekua eraiki eta horietako bakoitza ***pn*** izeneko Petri Sarean gehituko da ***addPlace*** metodoaz baliatuz.

● Trantsizioak eraikitzen

Lekuak eraikitzeko erabili den eskema antzekoa jarraitzen da trantsizioak eraikitzerako garaian. Hemen ere *XPath* teknologia erabiliz trantsizio adabegiak sailkatuko dira ***transitionTags*** aldagaiean eta hauetako bakoitzaren ume adabegiak argumentu bezala pasako zaizkio ***buildTransition*** metodoari, *Element* motakoak badira, noski. Jarraian erakusten dira zein diren jarraitu beharreko pausoak:

```
NodeList transitionTags = (NodeList)
                  (XpathFactory.newInstance().newXPath().evaluate
                  ("//transition", doc, XPathConstants.NODESET));
for (int i=0; i<transitionTags.getLength(); i++){
      Node node = transitionTags.item(i);
      try{
            if (node instanceof Element &&

                  node.getLocalName().trim().equals("transition")
```

```
){
                        NodeList children = node.getChildNodes();
                        Transition transition =
                                          buildTransition(children);
                        pn.addTransition(transition);
                  }
            }catch(Exception e){
                  e.printStackTrace();
            }
      }
```

● Arkuak eraikitzen

Lekuak eta trantsizioak sortzeko kodean oinarrituz, sarrera eta irteera arkuen eraikuntza ez da asko aldatuko. Hauek ***inputArcTags*** eta ***outputArcTags*** aldagaiak erabiltzen dituzten *XPath*-en ***evaluate*** metodotik jasoko dituzten adabegi-zerrendak gordetzeko. Horrez gain, ***buildInputArc*** eta ***buildOutputArc*** metodoak dira sarrera eta irteera arku bakoitzaren sorrerarako erabilitako metodoak, hurrenez hurren:

```
      NodeList inputArcTags = (NodeList)
                        (XpathFactory.newInstance().newXPath().evaluate
                        ("//input_arc", doc, XPathConstants.NODESET));
      for (int i=0; i<inputArcTags.getLength(); i++){
            Node node = inputArcTags.item(i);
            try{
                  if (node instanceof Element &&

                       node.getLocalName().trim().equals("input_arc")){
                        NodeList children = node.getChildNodes();
                        InputArc inputArc = buildInputArc(children);
                        pn.addInputArc(inputArc);
                  }
            }catch(Exception e){
                  e.printStackTrace();
            }
      }

      NodeList outputArcTags = (NodeList)
(XPathFactory.newInstance().newXPath().evaluate("//output_arc", doc,
                                          XPathConstants.NODESET));
      for (int i=0; i<outputArcTags.getLength(); i++){
            Node node = outputArcTags.item(i);
            try{
                  if (node instanceof Element &&
                  node.getLocalName().trim().equals("output_arc")){
                        NodeList children = node.getChildNodes();
                        OutputArc outputArc = buildOutputArc(children);
                        pn.addOutputArc(outputArc);
                  }
            }catch(Exception e){
```

```
                e.printStackTrace();
            }
        }
```

● Leku bat eraikitzen

Lehen azaldutako lekuen eraikuntzan *buildPlace* izeneko metodoa aipatu dugu, leku adabegi baten ume adabegi zerrendatik abiatuta *Place* instantzia bat sortzeaz arduratzen dena. Gauza jakina da horretarako eskuratu nahi den informazioa testu motako adabegien azpian gordetzen dena, eta espero diren balioak lekuaren izena (String) eta leku horri dagozkion markak (int) direna:

```
for (int i=0; i<children.getLength(); i++){
      Node node = children.item(i);
      if (node instanceof Element &&
                  node.getLocalName().trim().equals("p_name")){
            Node child = node.getFirstChild();
            pName = child.getNodeValue();
      }else if (node instanceof Element &&
                  node.getLocalName().trim().equals("tokens")){
            try{
                  Node child = node.getFirstChild();
                  tokens =
                  Integer.parseInt(child.getNodeValue().trim());
            }catch (Exception e){
                  e.printStackTrace();
            }
      }
}
return (new Place(pName, tokens));
```

Metodoak "***p_name***" eta "***tokens***" izeneko *Element* motako adabegiren bat aurkitzen duenean, bere ume adabegiari (Testu motakoa) eskatuko dio behar duen informazioa, ***pName*** (izena) edota ***tokens*** (marka kopurua), hain zuzen ere. Amaitzeko, lekuaren instantzia bat sortuko du eskuratutako datuekin (***new Place***).

● Trantsizio bat eraikitzen

Trantsizio bat eraikitzeko erabiltzen den metodoa ***buildTransition*** izenekoa da. Lekuen kasuan gertatzen den modu berean, metodo honek ere adabegi zerrenda bat jasoko du eta *Transition* instantzia eraiki bere izen (***tr_name***), denbora (***time***), lehentasun (***priority***) eta semantikarekin (***semantics***)

batera.

```
      for (int i=0; i<children.getLength(); i++){
            Node node = children.item(i);
            if (node instanceof Element &&
                        node.getLocalName().trim().equals("tr_name")){
                  Node child = node.getFirstChild();
                  name = child.getNodeValue().trim();
            }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("time")){
                  NodeList timeChildren = node.getChildNodes();
                  time = buildTime(timeChildren);
            }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("priority")){
                  try{
                        Node child = node.getFirstChild();
                        priority=Integer.parseInt(child.getNodeValue().
                                                            trim());
                        hasPriority = true;
                  }catch(Exception e){
                        e.printStackTrace();
                  }
            }else if (node instanceof Element &&
                  node.getLocalName().trim().equals("semantics")){
                  Node child = node.getFirstChild();
                  semantics = child.getNodeValue();
                  hasSemantics = true;
            }
      }
```

Ezinbestekoa izango da trantsizioen instantziak sortzerako garaian lehentasun eta semantikaren agerpena aztertzea. Hauek hautazkoak direnez, posible da Petri Sarea osatzen duen XML dokumentuak hauen baliorik ez islatzea. Hori dela eta, lehenetsitako balioak esleituko zaizkie, 0 lehentasunarentzat eta “***single***” semantikarentzat. Beraz, jarraian datorren kode-zatiarekin amaitzen da metodoaren eginbeharra:

```
      if (!hasPriority && !hasSemantics){ //default value for priority
is 0 and "single" for semantics return (new Transition(name, time));
      }else if (!hasPriority && hasSemantics){
            return (new Transition(name, time, semantics));
      }else if (hasPriority && !hasSemantics){
            return (new Transition(name, time, priority));
      }else{
            return (new Transition(name, time, priority, semantics));
      }
```

Kodean ikus daitekeenez trantsizio baten denbora eraikitzea nahiko lan konplexua izango da, izan ere, denbora-mota desberdinak bereizten baitira.

Aukera horien artean, *null*, *constant*, *uniform*, *exponential* eta *lognormal* dauzkagu eta horiek eraikitzeaz arduratzen den metodoa ***buildTime*** izango da.

```
for (int i=0; i<timeChildren.getLength(); i++){
      Node node = timeChildren.item(i);
      if (node instanceof Element &&
            node.getLocalName().trim().equals("distribution")){
            Node child = node.getFirstChild();
            distribution = child.getNodeValue().trim();
      }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("a")){
            Node child = node.getFirstChild();
            a = Double.parseDouble(child.getNodeValue().trim());
            hasA = true;
      }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("b")){
            Node child = node.getFirstChild();
            b = Double.parseDouble(child.getNodeValue().trim());
            hasB = true;
      }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("m")){
            Node child = node.getFirstChild();
            m = Double.parseDouble(child.getNodeValue().trim());
            hasM = true;
      }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("s")){
            Node child = node.getFirstChild();
            s = Double.parseDouble(child.getNodeValue().trim());
            hasS = true;
      }
}
```

Orain, kodearen azken zatia osatzea falta zaigu, *Time* instantzia desberdinak eraikitzea, hain zuzen ere. Hori denbora-banaketen araberakoa izango denez, horretarako bost aukera desberdin egongo dira, denbora-mota bakoitza bereiziko delarik.

```
if (distribution.equals("constant")){
      if (hasA){
            return (new Time(distribution, a));
      }else{
            System.err.println("Error: incorrect number of
                  arguments for constant time distribution.");
            }
}else if (distribution.equals("uniform")){
      if (hasA && hasB){
            return (new Time(distribution, a, b));
      }else{
            System.err.println("Error: incorrect number of
```

```
                  arguments for Uniform time distribution.");
      }

}else if (distribution.equals("exponential")){
      if (hasM){
            return (new Time(distribution, m));
      }else{
            System.err.println("Error: incorrect number of
                  argument for exponential time distribution.");
      }

}else if (distribution.equals("lognormal")){
      if (hasM && hasS){
            return (new Time(distribution, m, s));
      }else{
            System.err.println("Error: incorrect number of
                  arguments for lognormal time distribution.");
      }

}else{ //null time distribution
      if (!hasA && !hasB && !hasM && !hasS){
            return (new Time(distribution));
      }else{
            System.err.println("Error: incorrect number of
                        argument for null time distribution.");
      }
}
return null;
```

PNFromXML klasea osatzeko, sarrera eta irteera arkuak eraikitzeko metodoak geratzen dira, ***buildInputArc*** eta ***buildOutputArc***, hurrenez hurren. Metodo hauek oso antzekoak izango dira, izan ere eremu berdinek osatzen baitituzte. Alde batetik ***from*** eta ***to***, *int* motako eremuak, sarrera arkuen kasuan zein lekutatik zein trantsiziotara doazen eta irteera arkuen kasuan zein trantsiziotatik zein lekutara doazen adieraziko dutenak. Bestetik, hautazkoa izango den multiplizitatearen balioa (***multiplicity***), lehenetsitako balioa ***1*** izango duena, hain zuzen ere. Beraz, trantsizioekin gertatzen den antzeko eran, metodo hauek ***multiplicity*** eremuaren agerpena kontrolatu beharko dute instantziak sortzerako momentuan. Sarrera arku bat eraikitzeko oinarrizko kodea honakoa da:

```
for (int i=0; i<children.getLength(); i++){
      Node node = children.item(i);
      if (node instanceof Element &&
                  node.getLocalName().trim().equals("from")){
            try{
                  Node child = node.getFirstChild();
                  from =
                  Integer.parseInt(child.getNodeValue().trim());
            }catch(Exception e){
                  e.printStackTrace();
```

```
            }

      }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("to")){
            try{
                  Node child = node.getFirstChild();
                  to =
                  Integer.parseInt(child.getNodeValue().trim());
            }catch(Exception e){
                  e.printStackTrace();
            }

      }else if (node instanceof Element &&
              node.getLocalName().trim().equals("multiplicity")){
            try{
                  Node child = node.getFirstChild();
                  multiplicity =
                  Integer.parseInt(child.getNodeValue().trim());
                  hasMultiplicity = true;
            }catch(Exception e){
                  e.printStackTrace();
            }
      }
}

if (!hasMultiplicity){ //default value for multiplicity is 1
      return (new InputArc(from, to));
}else{ //hasMultiplicity==true
      return (new InputArc(from, to, multiplicity));
}
```

Irteera arku bat eraikitzeko, berriz, hau da *buildOutputArc* metodoaren inplementazioa:

```
for (int i=0; i<children.getLength(); i++){
      Node node = children.item(i);
      if (node instanceof Element &&
                  node.getLocalName().trim().equals("from")){
            try{
                  Node child = node.getFirstChild();
                  from =
                  Integer.parseInt(child.getNodeValue().trim());
            }catch(Exception e){
                  e.printStackTrace();
            }

      }else if (node instanceof Element &&
                        node.getLocalName().trim().equals("to")){
            try{
                  Node child = node.getFirstChild();
                  to =
                  Integer.parseInt(child.getNodeValue().trim());
```

```
                }catch(Exception e){
                      e.printStackTrace();
                }

          }else if (node instanceof Element &&
                node.getLocalName().trim().equals("multiplicity")){
                try{
                      Node child = node.getFirstChild();
                      multiplicity =
                      Integer.parseInt(child.getNodeValue().trim());
                      hasMultiplicity = true;
                }catch(Exception e){
                      e.printStackTrace();
                }
          }
    }

    if (!hasMultiplicity){ //default value for multiplicity is 1

          return (new OutputArc(from, to));

    }else{ //hasMultiplicity==true

          return (new OutputArc(from, to, multiplicity));

    }
```

8.5.2 PetriNet klasea

PetriNet instantzia baten oinarrizko elementuak lekuak, trantsizioak, sarrera-arkuak eta irteera-arkuak izango dira. Honela, informazio hau egituratzeko era desberdinak egon daitezkeen arren, *bektoreak* erabiltzea era sinple eta eraginkorra izan daiteke.

```
private Vector<Place> places;
private Vector<Transition> transitions;
private Vector<InputArc> inputArcs;
private Vector<OutputArc> outputArcs;

public PetriNet(){
      places = new Vector<Place>();
      transitions = new Vector<Transition>();
      inputArcs = new Vector<InputArc>();
      outputArcs = new Vector<OutputArc>();
}
```

Hori bai, horretarako beharrezkoa izango da *Vector* klasearen inportazioa:

```
import java.util.Vector;
```

Metodo eraikitzaileaz gain, lekuak, trantsizioak eta arkuak bektoreetatik lortu eta hauek bektoreetan gehitu ahal izateko metodoak ere programatu behar dira, ingelerazko *Getters and Setters* metodoak, alegia.

```
public Place getPlace(int index) {
      return places.get(index);
}

public void addPlace(Place place) {
      this.places.add(place);
}

public Transition getTransition(int index) {
      return transitions.get(index);
}

public void addTransition(Transition transition){
      this.transitions.add(transition);
}

public InputArc getInputArc(int index){
      return inputArcs.get(index);
}

public void addInputArc(InputArc inputArc){
      this.inputArcs.add(inputArc);
}

public OutputArc getOutputArc(int index){
      return outputArcs.get(index);
}

public void addOutputArc(OutputArc outputArc){
      this.outputArcs.add(outputArc);
}



public Vector<Place> getPlaces() {
      return places;
}

public void setPlaces(Vector<Place> places) {
      this.places = places;
}

public Vector<Transition> getTransitions() {
      return transitions;
}

public void setTransitions(Vector<Transition> transitions) {
```

```
        this.transitions = transitions;
    }

    public Vector<InputArc> getInputArcs() {
        return inputArcs;
    }

    public void setInputArcs(Vector<InputArc> inputArcs) {
        this.inputArcs = inputArcs;
    }

    public Vector<OutputArc> getOutputArcs() {
        return outputArcs;
    }

    public void setOutputArcs(Vector<OutputArc> outputArcs) {
        this.outputArcs = outputArcs;
    }
```

Guzti hauez gain, nahiz eta gure trazan ez dugun erabiliko, PetriNet klaseak bere edukia inprimatzeko print klasea ere izango du, honako eskeletoarekin:

```
    public void print() {

        //printing places

        //printing transitions

        //printing input arcs

        //printing output arcs

    }
```

Petri Sarea osatzen duten lekuen inprimaketarako kode-zatia honakoa da:

```
    //printing places
    for (int i=0; i<places.size(); i++){
        String pName = places.get(i).getName();
        int tokens = places.get(i).getTokens();
        if (tokens==1){
            System.out.print("The place " + pName + " has " +
            tokens + "                                token.");
            System.out.println();
        }else{
            System.out.print("The place " + pName + " has " +
            tokens + "                                tokens." );
            System.out.println();
        }
```

```
}

System.out.println();
```

Trantsizioak inprimatzea ahalbidetzen duena, aldiz, hau:

```
//printing transitions
for (int i=0; i<transitions.size(); i++){
      String trName = transitions.get(i).getName();
      Time time = transitions.get(i).getTime();
      String distribution = time.getDistribution();
      int priority = transitions.get(i).getPriority();
      String semantics = transitions.get(i).getSemantics();

      if (distribution.equals("constant")){
            double a = time.getA();
            System.out.print("The transition " + trName + " has "
            + distribution + " time distribution with " +
                        a + " value, " + priority + " priority
                              and " + semantics + " semantic.");
            System.out.println();
      } else if (distribution.equals("uniform")){
            double a = time.getA();
            double b = time.getB();
            System.out.print("The transition " + trName + " has "
                  + distribution + " time distribution in [" + a
                        + ", " + b + "], " + priority + "
                  priority and " + semantics + " semantic.");
            System.out.println();

      }else if (distribution.equals("exponential")){
            double m = time.getM();
            System.out.print("The transition " + trName + " has "
            + distribution + " time distribution with rate
                  " + m + ", " + priority + " priority and " +
                              semantics + " semantic.");
            System.out.println();

      }else if (distribution.equals("lognormal")){
            double m = time.getM();
            double s = time.getS();
            System.out.print("The transition " + trName + " has "
                  + distribution + " time distribution with mean
                              " + m + " and standard deviation "
            + s + ", " + priority + " priority and " + semantics
                                    + " semantic.");
            System.out.println();
      }else{
            System.out.print("The transition " + trName + " has "
                  + distribution + " time distribution, " +
                  priority + " priority and " + semantics + "
                                          semantic.");
```

```
                System.out.println();
        }
}

System.out.println();
```

Amaitzeko, sarrera eta irteera arkuak inprimatzeko kode-zatia falta zaigu:

```
//printing input arcs
for (int i=0; i<inputArcs.size(); i++){
      int from = inputArcs.get(i).getFrom();
      int to = inputArcs.get(i).getTo();
      int multiplicity = inputArcs.get(i).getMultiplicity();
      System.out.print("An input arc which multipicity is " +
      multiplicity + " goes from place number " + from + " to
                              transition number " + to + ".");
      System.out.println();
}
System.out.println();
//printing output arcs
for (int i=0; i<outputArcs.size(); i++){
      int from = outputArcs.get(i).getFrom();
      int to = outputArcs.get(i).getTo();
      int multiplicity = outputArcs.get(i).getMultiplicity();
      System.out.print("An output arc which multipicity is " +
            multiplicity + "  goes from transition number " +
                        from + " to place number " + to + ".");
      System.out.println();
}
```

8.5.3 Place klasea

Aipatu den bezala, *Place* klaseak bi eremu izango ditu. Alde batetik izena (***name)*** ordezkatzen duen *String* bat eta bestetik leku bakoitzak izango dituen *int* motako markak (***tokens***).

```
private String name;
private int tokens;
```

Metodo eraikitzaileei dagokienez, *Place* instantzia bat sortzeko bi aukera posible egongo direnez, bi eraikitzaile gehituko dizkiogu, horietako bat argumenturik gabekoa.

```
public Place(){
}
```

```
public Place(String name, int tokens){
      this.name = name;
      this.tokens = tokens;
}
```

Azkenik, eremu horiek atzitu eta ezartzeko *Getters and Setters* metodoak gehitu behar dira:

```
public String getName() {
      return name;
}

public void setName(String name) {
      this.name = name;
}

public int getTokens() {
      return tokens;
}

public void setTokens(int tokens) {
      this.tokens = tokens;
}
```

8.5.4 Transition klasea

Trantsizio baten kasuan, lau eremu beharko ditugu, izena, denbora, lehentasuna eta semantika islatu ahal izateko.

```
private String name;
private Time time;
private int priority;
private String semantics;
```

Metodo eraikitzaileak bost izango dira, lehenetsitako eraikitzailea kontutan hartuta, izan ere, lehentasun eta semantika eremuak hautazkoak izateak *Transition* objektu baten instantzia sortzea baldintzatzen baitu. Aukera posible guztiak hauek dira:

```
public Transition(){

}

public Transition(String name, Time time){
      this.name = name;
      this.time = time;
}
```

```
public Transition(String name, Time time, int priority){
      this.name = name;
      this.time = time;
      this.priority = priority;
}

public Transition(String name, Time time, String semantics){
      this.name = name;
      this.time = time;
      this.semantics = semantics;
}

public Transition(String name, Time time, int priority, String
                                                      semantics){
      this.name = name;
      this.time = time;
      this.priority = priority;
      this.semantics = semantics;
}
```

Amaitzeko, eremuen *Getters and Setters* metodoak osatzen dute klasea:

```
public String getName() {
      return name;
}


public void setName(String name) {
      this.name = name;
}

public Time getTime() {
      return time;
}

public void setTime(Time time) {
      this.time = time;
}

public int getPriority() {
      return priority;
}

public void setPriority(int priority) {
      this.priority = priority;
}

public String getSemantics() {
      return semantics;
}
```

```
public void setSemantics(String semantics) {
      this.semantics = semantics;
}
```

8.5.5 Time klasea

Denbora-banaketa desberdinak maneiatzen direnez, eremu desberdinak beharko dira denbora guztiak kudeatzeko. Horrez gain, banaketa horren berri emango digun *distribution* eremua ere beharko du Time klaseak.

```
private String distribution;
private double a;
private double b;
private double m;
private double s;
```

Eraikitzaileak ere denbora-banaketaren menpekoak dira, hiru metodo desberdin egongo direlarik (denbora uniforme eta logaritmikoek metodo bera konpartituko dute, konstante eta esponentzialekin gertatzen den bezala):

```
public Time (String distribution){
      this.distribution = distribution;
}

public Time (String distribution, double a, double b){
      this.distribution = distribution;
      if (distribution.equals("uniform")){
            this.a = a;
            this.b = b;
      }else if (distribution.equals("lognormal")){
            this.m = a;
            this.s = b;
      }
}

public Time (String distribution, double m){
      this.distribution = distribution;
      if (distribution.equals("constant")){
            this.a = m;
      }else if(distribution.equals("exponential")){
            this.m = m;
      }
}
```

Azkenik, *Getters eta Setters* metodoak gehituko dira:

```
public double getA() {
      return a;
}

public void setA(double a) {
      this.a = a;
}

public double getB() {
      return b;
}

public void setB(double b) {
      this.b = b;
}

public double getM() {
      return m;
}

public void setM(double m) {
      this.m = m;
}

public double getS() {
      return s;
}

public void setS(double s) {
      this.s = s;
}
```

8.5.6 InputArc klasea

Sarrera-arkua osatzen duten eremuak ***from***, ***to*** eta ***multiplicity*** izango dira. Sarrera-arku bat zein lekutatik (***from***) zein trantsiziotara (***to***) doan definitzeko *int* motako eremuak erabiliko dira. Horrez gain, ***multiplicity*** eremuaren lehenetsitako balioa ***1*** balioarekin finkatuko da, honen agerpena hautazkoa izango baita:

```
private int from;
private int to;
private int multiplicity = 1;
```

Multiplizitate eremuaren hautazko agerpen horrek, eraikitzaile kopurua baldintzatzen du:

```
public InputArc(){

}

public InputArc(int from, int to){
      this.from = from;
      this.to = to;
}

public InputArc(int from, int to, int multiplicity){
      this.from = from;
      this.to = to;
      this.multiplicity = multiplicity;
}
```

Bukatzeko, *Getters and Setters* metodoak gehitu beharko dira:

```
public int getFrom() {
      return from;
}

public void setFrom(int from) {
      this.from = from;
}

public int getTo() {
      return to;
}

public void setTo(int to) {
      this.to = to;
}

public int getMultiplicity() {
      return multiplicity;
}

public void setMultiplicity(int multiplicity) {
      this.multiplicity = multiplicity;
}
```

8.5.7 OutputArc klasea

Klase honek sarrera-arkuen eremu berdinak erabiltzen ditu. Izatez, oso antzekoak dira bi klaseen inplementazioak, baino desberdintasun esanguratsu bat dago bien artean. Irteera-arkuen ***from*** eremuak trantsizio bati egingo dio erreferentzia, sarrera-arkuen ***from*** eremuak ez bezala, eta ***to*** eremuak, berriz, leku bati.

InputArc klasean gertatzen den era berean, eraikitzaileak

multiplizitatearen agerpenaren arabera erabiliko dira:

```
public OutputArc(){

}

public OutputArc(int from, int to){
      this.from = from;
      this.to = to;
}

public OutputArc(int from, int to, int multiplicity){
      this.from = from;
      this.to = to;
      this.multiplicity = multiplicity;
}
```

Klasea osatzeko, eremuen *Getters and Setters* metodoak falta dira:

```
public int getFrom() {
      return from;
}

public void setFrom(int from) {
      this.from = from;
}

public int getTo() {
      return to;
}

public void setTo(int to) {
      this.to = to;
}

public int getMultiplicity() {
      return multiplicity;
}

public void setMultiplicity(int multiplicity) {
      this.multiplicity = multiplicity;
}
```

8.6 Iturburu fitxategiak eta aplikazioaren exekuzioa

Proiektuan zehar erabili eta garatutako kodea eskuragarri dago honako web orri honetan (*zip* formatuan):

http://www.sc.ehu.es/ccwalirx/usr/13InigoOrrantia.zip

8.6.1 13InigoOrrantia.zip fitxategiaren edukia

Aipatutako *zip* formatuko fitxategiak, **13InigoOrrantia** izeneko karpeta nagusia du. Honen edukia, 3 karpeta berri dira:

Alde batetik, **Klase lagungarriak** izenekoak, simulatzailearen programazioa garatzeko oinarri bezala erabili diren *Simulator.java*, *SimSystem.java*, *SysEvent.java* eta *LogNormal.java* fitxategiak barneratzen ditu.

Bestalde, **petrinet** izeneko katalogoan, proiektuaren lehenengo zatian Petri Sareak deskribatu eta instantziak sortzeko egileak programatutako aplikazioaren muin diren *Java* klaseak daude eskuragarri, horien artean:

- Metodo estatiko nagusia duen *MainClass.java* klase nagusia.

- *DOM* egituran oinarritzen den *PNFromXML.java* klasea. Petri Sarea eraikitzeaz arduratzen den *buildPetriNet()* metodoa klase honetan aurkitzen da.

- Petri Sare bati dagokion instantzia eraikitzeko beharrezkoak diren *PetriNet.java*, *Place.java*, *Transition.java*, *Time.java*, *InputArc.java* eta *OutputArc.java* klaseen kodea.

Horrez gain, **simulator** katalogoak barneratzen ditu simulatzailearen programaziorako eta aplikazioa exekutatzeko beharrezkoak diren klaseak. Hauen artean **Klase lagungarriak** eta **petrinet** katalogoetako klaseak egongo dira eraldatuta:

- Metodo estatikoa duen *Simulator.java* klase nagusia.

- Simulatzailea osatzen duten *SimSystem.java*, *SysEvent.java* eta *LogNormal.java* klaseak.

- *DOM* egituran oinarritzen den *PNFromXML.java* klasea. Petri Sarea eraikitzeaz arduratzen den *buildPetriNet()* metodoa klase honetan aurkitzen da.

- Petri Sare bati dagokion instantzia eraikitzeko beharrezkoak diren *PetriNet.java*, *Place.java*, *Transition.java*, *Time.java*, *InputArc.java* eta *OutputArc.java* klaseen kodea.

Azkenik, **XML fitxategiak** karpetan, Petri Sare desberdinen definizioak dauzkagu *xml* formatuan (*petri_net1.xml*, *petri_net2.xml*, ...) eta horrez gain, Petri Sareetarako XML eskema ere bai *xsd* formatuan (*petri_net.xsd*). Simulazioa egiteko erabili den XML dokumentua ere (*petri_net_multiproz_berr.xml*) bertan aurki dezakegu.

8.6.2 Fitxategien instalazio, konpilazio eta exekuzioa

Demagun **13InigoOrrantia** katalogoa, gure Ubuntu sistemako */home/user1* katalogoan finkatzea erabakitzen dugula. Simulazioa egin aurretik, **petrinet** karpetako *.java* fitxategiak konpilatu eta exekuta ditzakegu Petri Sarearen egitura ondo sortzen dela ziurtatzeko. Horretarako, beharrezkoak diren komandoak honakoak dira:

a) Katalogo egokian kokatu:

```
cd /home/user1/13InigoOrrantia
```

b) **petrinet** karpetako *.java* fitxategi guztiak konpilatu:

```
javac petrinet/*
```

Pauso hauek eman ondoren, aplikazioaren lehen zatia exekutatuko ahal izango da. Honek, argumentu bezala pasako diogun XML bidezko Petri Sarearen espezifikazioari (adibidean XML\ fitxategiak/petri_net_multiproz_berr.xml) dagokion instantzia sortu eta egituraren osagaiak pantailaratuko dizkigu:

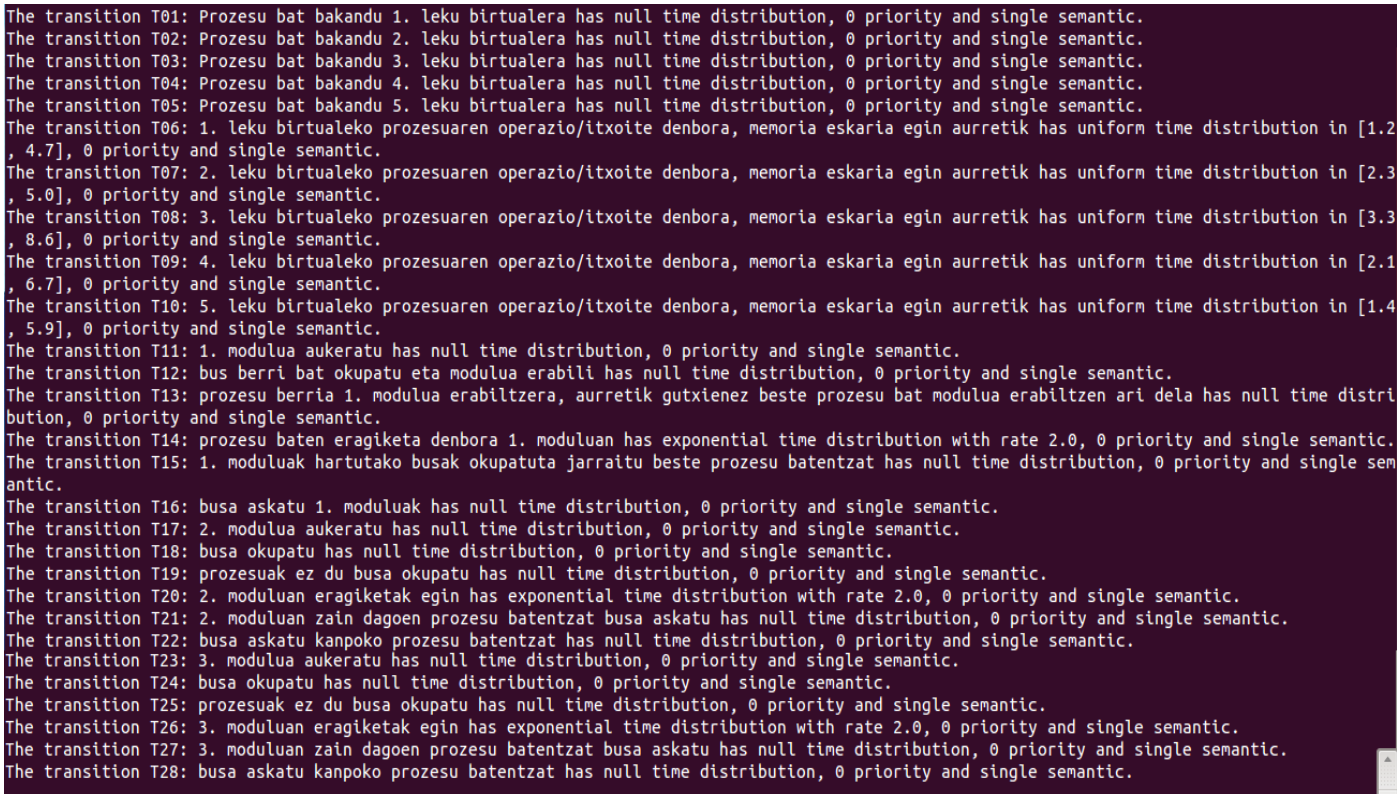
```
java petrinet/MainClass XML\ fitxategiak/petri_net_multiproz_berr.xml
```

Komandoak eta lekuei buruzko informazioa:

```
hugamen@GAP-simuladorea:~/Escritorio/GAP/12InigoOrrantia$ rm petrinet/*.class
hugamen@GAP-simuladorea:~/Escritorio/GAP/12InigoOrrantia$ javac petrinet/*
hugamen@GAP-simuladorea:~/Escritorio/GAP/12InigoOrrantia$ java petrinet.MainClass XML\ fitxategiak/petri_net_multiproz_berr.xml
The place P01: Memoria modulu baten erabilera amaitu duten prozesuak has 0 tokens.
The place P02: Memoria modulurik erabili behar ez duen bakandutako prozesua 1. leku birtualean has 0 tokens.
The place P03: Memoria modulurik erabili behar ez duen bakandutako prozesua 2. leku birtualean has 1 token.
The place P04: Memoria modulurik erabili behar ez duen bakandutako prozesua 3. leku birtualean has 1 token.
The place P05: Memoria modulurik erabili behar ez duen bakandutako prozesua 4. leku birtualean has 0 tokens.
The place P06: Memoria modulurik erabili behar ez duen bakandutako prozesua 5. leku birtualean has 1 token.
The place P07: memoria-modulua aukeratu has 0 tokens.
The place P08: 1. modulua hartzeko prest has 0 tokens.
The place P09: 1. modulua eta bus bat hartuta duten prozesuak has 1 token.
The place P10: 1. moduluak hartutako busa, prozesu batek mouduaren erabilera amaitu ondoren has 0 tokens.
The place P11: 2. modulua hartzeko prest has 0 tokens.
The place P12: 2. modulua eta bus bat hartuta duten prozesuak has 1 token.
The place P13: 2. moduluak hartutako busa, prozesu batek mouduaren erabilera amaitu ondoren has 0 tokens.
The place P14: 3. modulua hartzeko prest has 0 tokens.
The place P15: 3. modulua eta bus bat hartuta duten prozesuak has 0 tokens.
The place P16: 3. moduluak hartutako busa, prozesu batek mouduaren erabilera amaitu ondoren has 0 tokens.
The place P17: libre dauden busak has 0 tokens.
```

8.6.2.1 irudia: Petri Sarearen osagaiak (lekuak)

Trantsizioei buruzko informazioa:



8.6.2.2 irudia: Petri Sarearen osagaiak (trantsizioak)

Sarrera-arkuei buruzko informazioa:

```
An input arc which multipicity is 1 goes from place number 1 to transition number 1.
An input arc which multipicity is 1 goes from place number 1 to transition number 2.
An input arc which multipicity is 1 goes from place number 1 to transition number 3.
An input arc which multipicity is 1 goes from place number 1 to transition number 4.
An input arc which multipicity is 1 goes from place number 1 to transition number 5.
An input arc which multipicity is 0 goes from place number 2 to transition number 1.
An input arc which multipicity is 0 goes from place number 3 to transition number 2.
An input arc which multipicity is 0 goes from place number 4 to transition number 3.
An input arc which multipicity is 0 goes from place number 5 to transition number 4.
An input arc which multipicity is 0 goes from place number 6 to transition number 5.
An input arc which multipicity is 1 goes from place number 2 to transition number 6.
An input arc which multipicity is 1 goes from place number 3 to transition number 7.
An input arc which multipicity is 1 goes from place number 4 to transition number 8.
An input arc which multipicity is 1 goes from place number 5 to transition number 9.
An input arc which multipicity is 1 goes from place number 6 to transition number 10.
An input arc which multipicity is 1 goes from place number 7 to transition number 11.
An input arc which multipicity is 1 goes from place number 7 to transition number 17.
An input arc which multipicity is 1 goes from place number 7 to transition number 23.
An input arc which multipicity is 1 goes from place number 8 to transition number 12.
An input arc which multipicity is 1 goes from place number 8 to transition number 13.
An input arc which multipicity is 0 goes from place number 9 to transition number 12.
An input arc which multipicity is 1 goes from place number 9 to transition number 13.
An input arc which multipicity is 1 goes from place number 9 to transition number 14.
An input arc which multipicity is 1 goes from place number 9 to transition number 15.
An input arc which multipicity is 0 goes from place number 9 to transition number 16.
An input arc which multipicity is 1 goes from place number 10 to transition number 15.
An input arc which multipicity is 1 goes from place number 10 to transition number 16.
An input arc which multipicity is 1 goes from place number 11 to transition number 18.
An input arc which multipicity is 1 goes from place number 11 to transition number 19.
An input arc which multipicity is 0 goes from place number 12 to transition number 18.
An input arc which multipicity is 1 goes from place number 12 to transition number 19.
An input arc which multipicity is 1 goes from place number 12 to transition number 20.
An input arc which multipicity is 1 goes from place number 12 to transition number 21.
An input arc which multipicity is 0 goes from place number 12 to transition number 22.
An input arc which multipicity is 1 goes from place number 13 to transition number 21.
An input arc which multipicity is 1 goes from place number 13 to transition number 22.
An input arc which multipicity is 1 goes from place number 14 to transition number 24.
An input arc which multipicity is 1 goes from place number 14 to transition number 25.
An input arc which multipicity is 0 goes from place number 15 to transition number 24.
An input arc which multipicity is 1 goes from place number 15 to transition number 25.
An input arc which multipicity is 1 goes from place number 15 to transition number 26.
```

8.6.2.3 irudia: Petri Sarearen osagaiak (sarrera-arkuak)

Irteera-arkuei buruzko informazioa:

```
An output arc which multipicity is 1 goes from transition number 1 to place number 2.
An output arc which multipicity is 1 goes from transition number 2 to place number 3.
An output arc which multipicity is 1 goes from transition number 3 to place number 4.
An output arc which multipicity is 1 goes from transition number 4 to place number 5.
An output arc which multipicity is 1 goes from transition number 5 to place number 6.
An output arc which multipicity is 1 goes from transition number 6 to place number 7.
An output arc which multipicity is 1 goes from transition number 7 to place number 7.
An output arc which multipicity is 1 goes from transition number 8 to place number 7.
An output arc which multipicity is 1 goes from transition number 9 to place number 7.
An output arc which multipicity is 1 goes from transition number 10 to place number 7.
An output arc which multipicity is 1 goes from transition number 11 to place number 8.
An output arc which multipicity is 1 goes from transition number 12 to place number 9.
An output arc which multipicity is 2 goes from transition number 13 to place number 9.
An output arc which multipicity is 1 goes from transition number 14 to place number 1.
An output arc which multipicity is 1 goes from transition number 14 to place number 10.
An output arc which multipicity is 1 goes from transition number 15 to place number 9.
An output arc which multipicity is 1 goes from transition number 16 to place number 17.
An output arc which multipicity is 1 goes from transition number 17 to place number 11.
An output arc which multipicity is 1 goes from transition number 18 to place number 12.
An output arc which multipicity is 2 goes from transition number 19 to place number 12.
An output arc which multipicity is 1 goes from transition number 20 to place number 1.
An output arc which multipicity is 1 goes from transition number 20 to place number 13.
An output arc which multipicity is 1 goes from transition number 21 to place number 12.
An output arc which multipicity is 1 goes from transition number 22 to place number 17.
An output arc which multipicity is 1 goes from transition number 23 to place number 14.
An output arc which multipicity is 1 goes from transition number 24 to place number 15.
An output arc which multipicity is 2 goes from transition number 25 to place number 15.
An output arc which multipicity is 1 goes from transition number 26 to place number 1.
An output arc which multipicity is 1 goes from transition number 26 to place number 16.
An output arc which multipicity is 1 goes from transition number 27 to place number 15.
An output arc which multipicity is 1 goes from transition number 28 to place number 17.
```

8.6.2.4 irudia: Petri Sarearen osagaiak (irteera-arkuak)

Petri Sarearen egitura ondo eraiki dela ziurtatzen dugunean simulazio prozesuari ekin diezaiokegu. Horretarako, kasu honetan, **simulator** katalogoko klaseak konpilatu eta exekutatu beharko dira pauso hauek jarraituz:

a) Katalogo egokian kokatu:

```
cd /home/user1/13InigoOrrantia
```

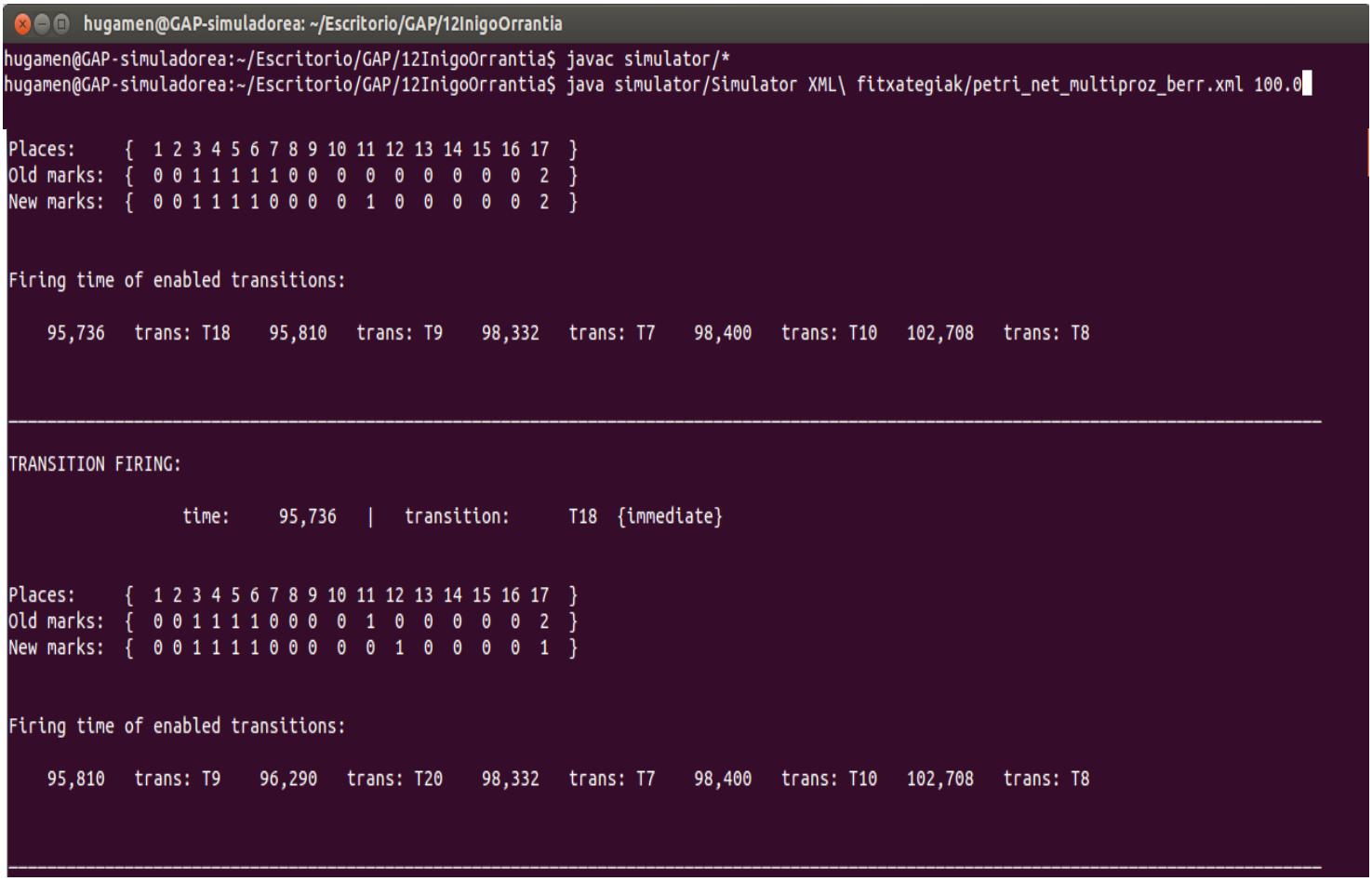
b) **simulator** karpetako *.java* fitxategi guztiak konpilatu:

```
javac simulator/*
```

Orain simulazioaren traza lortzeko exekutatu beharreko komandoak

aztertuko ditugu. Argumentu bezala pasako dizkiogun XML bidezko Petri Sarearen espezifikazioari (adibidean XML\ fitxategiak/petri_net_multiproz_berr.xml) eta simulazio denborari (adibidean 100.0) dagokien traza sortu eta pantailaratuko digun komandoa honakoa da:

```
java simulator/Simulator XML\fitxategiak/petri_net_multiproz_berr.xml
100.0
```



8.6.2.5 irudia: Petri Sarearen simulazioa

Hala ere, erosoagoa izango da traza guztia fitxategi batean biltzea zehatz-mehatz aztertu ahal izateko. Horregatik, terminalean bertan exekutatu ordez informazio guztia testu fitxategi batera bidal dezakegu komandoa editatuz:

```
java simulator/Simulator XML\ fitxategiak/petri_net_multiproz_berr.xml
>           /home/user1/simulazioaren_traza.txt 100.0
```

9. KAPITULUA

ONDORIOAK

9.1 Orokorrak

Proiektuan zehar hainbat atal interesgarri aipatu eta landu ahal izan dira. Alde batetik, sistema konkurrenteak sakonago aztertzeko aukera eskaintzen du. Horretarako, hainbat sistema motaren eredu formal bezala erabiltzen diren Petri Sareak aurkezten dira, beren ezaugarri eta erabilerak goraipatuz. Gainera, *XML* lengoaiak izan dezakeen potentzia egiaztatu ahal izan da, honen laguntzarekin, aipatutako Petri Sareak definitu eta maneiatzeko aukera ematen zaigula ikusiz. Guzti honez gain, *XML* teknologiarekin lan egiteko *Java* hizkuntzaren inguruneak eskaintzen dituen tresnak ere azpimarratu nahi nituzke.

Bestalde, simulazio eta simulatzaileen programazioari dagokienez, estatistika eta probabilitate arloko zorizko aldagai desberdinak ezagutzeko ematen den aukera ere aipatu beharrekoa da, hauei esker, Petri Sare baten portaera simulatu eta aztertzea posible egiten dutelarik. Gainera, gertaera diskretuko simulazioa programatzeko objektuetara zuzendutako lengoaien egokitasuna goraipatu behar da eta bereziki, *Java* lengoaiak simulazio estatistikoa programatzeko duen egokitasuna.

Proiektu honetan garatu den kernelak hainbat jarraipen interesgarri izan ditzake etorkizunari begira. Alde batetik, erabiltzailearen eskakizunei dagokienez, simulatzailea erabili ahal izateko interfazeen diseinuak garatu ahal izango dira kernel honetan oinarrituz. Aplikazioak erabiliko dituen *XML* dokumentuak eta eskemak kargatzeko, simulatzaileak eskatzen dituen parametro desberdinak sartzeko edota simulazio prozesuak itzulitako emaitzak era garbi eta ulergarrian ikusi ahal izateko interfazeak diseinatzeko aukera eskaintzen du, besteak beste. Horrez gain, simulazioaren ondorioz egindako kalkulu estatistiko horiek datuen analisian izan dezakete erabilpena, adibidez, bildutako datu hauez baliatuz estimazioaren zehaztasuna kuantifikatzean edota arazoak aztertu eta identifikatzean. Bestalde, proiektu honen jarraipen gisa ere, azpisistema desberdinen programazioa aipa dezakegu. Hala ere, jarraipen bat baino gehiago, ekintza hauek, simulatzailearen funtzionamendu zuzena bermatuko duten sistema garrantzitsuen aurre-programazio bezala ulertu behar dira, bereziki kolak

edota antzeko azpisistemak goraipatuz, erabiltzaileak zuzenean erabili ahal izateko helburuarekin.

Azkenik, proiektuan garatu den kernelaren hedapen interesgarria litzateke errendimendu parametroei dagozkien estatistikoak gehitzea. Proiektuaren planteamenduari jarraituz, honek eskatzen du lehenik Petri Sarearen eredu abstraktuari gehitzea estatistiko hauen definizioa eta XML dokumenturako eskeman ere estatitisko hauen formatua zehaztea, ondoren simulatzailearen muina den Java kodea ere hedapen honetarako egokituko litzateke.

9.2 Pertsonalak

Orokorrean, balorazio guztiz positiboa egin dezaket proiektu honen inguruan eta zenbait puntu edo atal goraipatuko nituzke honi esker ikasitakoa aipatu ahal izateko.

Alde batetik, oso gustura geratu naiz lan pertsonalari dagokionez, tutorearekin batera adostutako helburuak era batean edo bestean lortzeko gai izan naizela uste baitut. Hala ere, helburu horiek lortzeko prozesuan, tutorearen etengabeko prestakizun eta laguntzeko ekimena aipatu eta eskertu nahi nituzke, arazoen aurrean lasaitu ederrak hartu ahal izan baititut bere laguntzari esker. Horrez gain, proiektuan zehar sortutako arazo horiei irtenbideak aurkitzerako garaian irabazitako esperientzia, etorkizunean oso lagungarri izango zaidala ere iruditzen zait.

Bestalde, ikasgai desberdinetako hainbat atal landu ahal izan ditut hilabete hauetan zehar. Arlo estatistiko eta matematikoetan, zorizko aldagaiak eta hauen simulazioaz gain, banaketa desberdinen erabilera ere oso lagungarri izan zaizkit proiektuaren garapenean. Sistema konkurrenteei dagokienez, Petri Sareen inguruan ikasitakoa aplikatzeak, helburuak errazago lortzeko aukera eskaini didala esan dezaket. Horrez gain, programazio arloan *Java* lengoaian trebetasuna irabazi dudala iruditzen zait, garatutako aplikazioaren gehiengoa hizkuntza honetan oinarritzen baita. Gainera, *XML* lengoaia interesgarri eta erabilgarria ere landu ahal izan dut proiektuan zehar, orain arte ezagutzen ez nituen zenbait baliagarritasun ezagutuz.

Beraz, hasieran esan bezala, etorkizunari begira oso interesgarria iruditu zait hilabete hauetan zehar egindako lan guztia, sortutako buruhauste eta arazoetatik hainbat gauza ikasi ahal izan baitut eta honek balorazio guztiz positiboa egitera eramaten nau, ezbairik gabe.

ERANSKINAK

A. ERREFERENTZIAK

Simulazioaren oinarriak barneratzen hasi eta teknika desberdinak ezagutzeko [2] eta [3] liburuak aurkitu ditut. Lehenengoa nahiko oinarrizkoa iruditu zait eta bigarrena, berriz, maila aurreratuago batean lan egin nahi duen erabiltzaileei gomendatuko nieke. [5] eta [6] simulatzaileak oinarritzat hartu ditut nire simulatzailea programatzerako garaian haien artean antzekotasun batzuk aurki baitaitezke. Horiez gain, simulazioaren inguruan interneteko web orrietan ez da batere zaila informazioa aurkitzea [1] [4].

Petri Sareei dagokienez, [7] liburua oso egokia da mundu horretan barneratzen den edonorentzat, liburu hau guztiz irakurterraza iruditu baitzait. [8] liburua, berriz, aurrekoa baino osatuagoa dela esango nuke Petri Sareen adibide argi eta ulergarriak barneratzen baititu; oso gomendagarria dela esango nuke. [9] web orrian informazio mordo eta nahiko txukuna aurki daiteke eta [10] apunteak guztiz erabilgarriak izan zaizkit memoria osoan zehar informazioa ulertu eta gehitu ahal izateko. Bestalde, trantsizioen tiro-denborak maneiatzeko banaketa jarraituak erabili direnez [14], [15], [16] eta [17] erreferentziekin aritu naiz lanean.

XML teknologiarekin hasieratik hasten den edozeinek [22], [24] eta [25] erreferentzietan aurkitu ahal izango du behar adina informazio (apunte eta tutorial erabilgarriak). Horrez gain, gaur egun nahi adina informazio euskaraz aurkitzea lan zaila denez, [26] gida goraipatzea gustatuko litzaidake. Bestalde, proiektu honetan jardun aurretik ezagutzen ez nuen *xpath* teknologia erabili ahal izateko [29] erreferentzia oso lagungarri izan zait. XML atalarekin amaitzeko, Java eta XML lengoaiekin aldi berean lan egin behar duenari gomendatuko nioken liburua da [23].a, azalpen argiak eta adibidez josiak barneratzen ditu eta.

Simulatzailea programatzerako garaian agian erreferentzia erabilienak izan dira Java programazio lengoaiarekin erlazionaturikoak. Nire klaseen oinarri den DOM egitura ulertu eta aldi berean erabiltzeko dauzkagu [27] eta [28] estekak. Gainera, [18] dokumentazio teknikoa eta Java eta XML erlazionatzen dituen [21] erreferentzia ere aplikazioaren oinarri izan dira. Hala ere, unibertsitatean landutako [20] apunteak berreskuratzeak ere asko lagundu dit Java lengoaian klaseak inplementatzerako garaian. Azkenik, simulatzaileekin erlazionatutako [19] liburu elektronikoa ere aipatu nahi nuke.

[1] Simulazioa konputagailuetan (Wikipedia)

http://en.wikipedia.org/wiki/Computer_simulation

[2] Kant, Krishna (1992): *Introduction to computer system performance evaluation*, McGraw-Hill, New York [etc.]

[3] Paul J. Fortier eta Howard E. Michel (2003): *Computer systems performance evaluation and prediction*, Digital Press, Burlington, Massachusetts.

[4] Gertaera diskretuko simulazioa (Wikipedia)

http://en.wikipedia.org/wiki/Computer_simulation

[5] ns-3 simulatzailearen webgunea

http://www.nsnam.org/

[6] OMNeT ++ simulatzailearen webgunea

http://www.omnetpp.org/

[7] Kant, Krishna (1992): *Introduction to computer system performance evaluation*, McGraw-Hill, New York [etc.]

[8] Paul J. Fortier eta Howard E. Michel (2003): *Computer systems performance evaluation and prediction*, Digital Press, Burlington, Massachusetts.

[9] Petri net (Wikipedia)

http://en.wikipedia.org/wiki/Petri_net

[10] Moujahid A. eta Albizuri F. X. (2012): *Evaluación del rendimiento de sistemas informáticos* (Apunteak)

http://www.sc.ehu.es/ccwalirx/docs/ersi.html

[11] Multiprozesadore (Wikipedia)

http://eu.wikipedia.org/wiki/Multiprozesadore

[12] NUMA arkitektura (Wikipedia)

http://en.wikipedia.org/wiki/Non-Uniform_Memory_Access

[13] SMP arkitektura (Wikipedia)

http://en.wikipedia.org/wiki/Symmetric_Multiprocessor

[14] Banaketa jarraituak

http://mathworld.wolfram.com/topics/ContinuousDistributions.html

[15] Banaketa uniformea (Wikipedia)

http://en.wikipedia.org/wiki/Uniform_distribution_(continuous)

[16] Banaketa esponentziala (Wikipedia)

http://en.wikipedia.org/wiki/Exponential_distribution

[17] Banaketa logaritmiko normala (Wikipedia)

http://en.wikipedia.org/wiki/Log-normal_distribution

[18] Java SE dokumentazio teknikoa

http://docs.oracle.com/javase/

[19] Klaus Wehrle, Mesut Günes eta James Gross (2010): *Modeling and Tools for Network Simulation (Online)*, Springer Berlin Heidelberg

http://link.springer.com/book/10.1007/978-3-642-12331-3/page/1

[20] Pereira Juanan eta Iturrioz Jon (2009): *Datu egiturak eta algoritmoak Javan* (Apunteak)

http://cvb.ehu.es/open_course_ware/euskera/irakaskuntza_teknikoak/datu_egiturak_eta_algoritmoak_javan/Course_listing.html

[21] Java API for XML Processing (JAXP)

Gida

http://docs.oracle.com/javase/tutorial/jaxp/index.html

Instalazioa

http://jaxp.java.net/downloads.html

[22] Oronoz, Maite (2009): *Datu baseen oinarriak* (Apunteak)

[23] McLaughlin, Brett eta Edelson, Justin (2006): *Java and XML*, O'Reilly Media, Inc.

[24] XML-ren tutorialen web orria

http://www.w3schools.com/xml/

[25] XML Schema-ren tutorialaren web orria

http://www.w3schools.com/schema/default.asp

[26] XML gida euskaraz (*programatzen, programazio euskalduna sustatzen*)

http://www.blogak.com/programatzen/xml-gida

[27] DOM (Wikipedia)

http://en.wikipedia.org/wiki/Document_Object_Model

[28] DOM erabiltzeko kode lagungarria

http://docs.oracle.com/javase/tutorial/jaxp/dom/index.html

[29] XPath

- zvon web orrian: http://www.zvon.org/xxl/XPathTutorial/
- w3chools web orrian: http://www.w3schools.com/xpath/