TRANSFORMANT MODELS BPMN A LLENGUATGE NATURAL

Memòria final

FIB

Autor: Genís Martín Coca

Data: 18 de gener de 2016

Directors: Josep Carmona i Lluís Padró

Índex

1. Context

Aquest projecte tracta majoritàriament sobre llenguatge natural, tot i que també toca aspectes de la Intel·ligència artificial o altres disciplines en menor mesura, com per exemple la generació d'autòmats. Es tracta d'un projecte en el que es parteix d'un model de processos del qual n'ha d'acabar sortint un text intel·ligible per a una persona que no tingui coneixements tècnics i que necessiti una planificació de la seva empresa. Aquest planificació quedarà millor estructurada degut a la generació de textos. En una època en la que la transparència és molt important cal que tothom sàpiga que ha de fer. A més a més, això pot ajudar aquestes persones a tenir més coneixements sobre els processos de negoci en una institució.

1.1 BPMN

El BPMN significa Bussiness Process Model and Notation o, traduït al català, Model i Notació de Processos de Negocis, i consta d'un conjunt de processos, en el qual es descriu el procés per a dur una funció descrivint l'ordre entre activitats i actors.

Per exemple, es pot utilitzar un cas senzill com una reunió d'empresa. Per al cas de l'empleat, ell va a la reunió, escolta les idees dels altres mentre explica les seves i després marxa. En el cas del cap, ell va a la reunió, explica el que cal fer, escolta les idees de l'empleat i s'apunta les millors abans de marxar de la reunió.

1.2 Llenguatge natural

El llenguatge natural, mentrestant, és un camp de la Informàtica, de la Intel·ligència Artificial i de la Lingüística que estudia les interaccions entre els ordinadors i el llenguatge humà. Tracta de dissenyar mètodes que siguin eficaços en el sentit de l'ordinador i que puguin simular comunicació[1]. És a dir, es tracta de crear un codi o un programa que pugui passar qualsevol cosa de la lògica computacional a un text que un ésser humà sigui capaç d'entendre.

És en aquesta part on es centra més aquest projecte. Per tal de poder fer el programa cal saber distingir bé les diferents parts d'un BPMN i el funcionament dels BPMNs.

1.3 Actors

Aquest projecte manté els quatre tipus d'actors amb els seus rols originals:

- Desenvolupador: Jo he estat l'únic desenvolupador del projecte. Per tant, no només m'he encarregat de fer el programa, si no també de buscar informació, fer la memòria, el disseny, l'anàlisi i les proves.
- Directors de projecte: El projecte ha tingut dos directors, el Josep Carmona i el Lluís Padró, tots dos professors dels departament de Ciències de la Computació de la Universitat Politècnica de Catalunya. El rol que desenvolupaven era el d'ajudar el desenvolupador en cas de tenir dubtes o problemes i de guiar-lo i donar-li informació suplementària.
- Usuaris: Aquest projecte estarà orientat a empreses que necessitin una manera de traslladar una planificació en activitats a un text que tothom pugui entendre. En principi, hauria de ser útil tant per a grans empreses com per a mitjanes o petites.
- Actors beneficiats: Els usuaris serien els únics actors beneficiats. Podríem concretar, però, que potser hi hauria dos departaments que serien els beneficiats en major mesura, que serien el de Recursos Humans, ja que s'estalviaria feina de com repartir les tasques, i el de Tresoreria, degut a que es podrien reduir els costos en paper.

2. Elements del projecte

En aquest apartat s'explicarà una mica millor alguns dels elements més importants que conformen aquest projecte a partir del qual s'ha generat el text. Aquests són, en concret, el BPMN, el RPST, les xarxes de Petri i els Patrons.

2.1 BPMN

Un BPMN és un tipus de model de processos. Aquest tipus de model de processos està orientat a negocis. Totes les activitats estan interconnectades entre elles mitjançant fluxos, que indiquen precedència o successió, o bé pas de missatges. Algunes vegades es faran unes o altres activitats depenent de certs factors determinats per les portes exclusives, inclusives o basades en esdeveniments. D'altres, però, hi haurà diferents activitats que es realitzarien alhora separades per portes paral·leles.

En la figura 1 es poden veure diferents tipus d'elements dins del que és un BPMN. Els dos objectes circulars són esdeveniments. El més fi és un esdeveniment inicial, mentre que el més gros és un final.

Els elements rectangulars, per la seva banda, són tasques. En aquest cas es veu que en totes elles tenen una icona d'una persona en l'interior d'aquesta tasca. Hi ha quatre tipus de tasca. La normal, la d'usuari, la de recepció i la d'enviament. En el cas de la normal no hi apareix cada icona, mentre que la d'enviament té un sobre negre i la de recepció el té blanc.

Els objectes en forma de rombe són portes, i són les que separen o uneixen diferents branques dins d'un mateix procés. En aquest model se'n veuen tres dels quatre tipus principals. Les exclusives (una aspa en l'interior), que obliguen a escollir un dels camins de sortida, les paral·leles (una creu semblant al signe +), on les dues branques es fan alhora i les inclusives (un cercle), on es segueix per una o més de les branques. El quart tipus són les portes basades en esdeveniments, les quals tenen un pentàgon dibuixat a l'interior del rombe, que tenen un comportament similar a les exclusives, amb la diferència que es basen en qüestions externes.

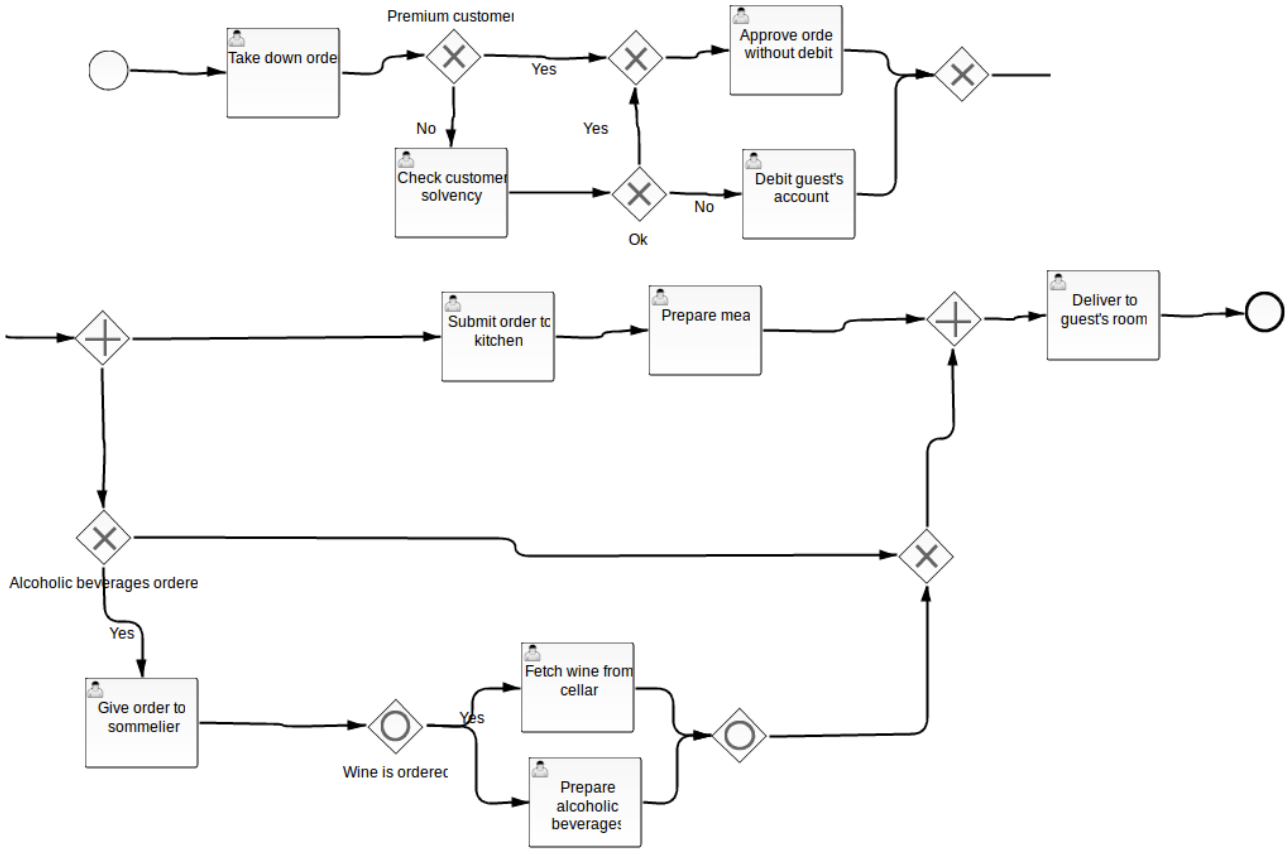


Figura 1. Un model BPMN dels utilitzats per a aquest projecte

Per últim, les fletxes són els fluxos, en aquest cas tots són de seqüència i la punta de la fletxa marca el camí que segueixen. Aquests fluxos són els que li donen un ordre a les execucions dels models.

2.2 RPST

Un RPST (Refined Program Structure Tree) és un diagrama jeràrquic que mostra les relacions niades de fragments o regions d'entrada única i sortida única (o SESE), mostrant l'organització d'un programa d'ordinador[2].

El RPST té informació de l'ordre que tenen els diferents elements o de com s'han de separar les diferents branques. Els RPST estan formats per fragments, que són parts del model de processos que tenen una única entrada i una única sortida. Un RPST té quatre tipus de fragments diferents:

- Els trivials són els més bàsics, i acostumen a ser fluxos de seqüència.

- Els polígons són seqüències d'altres fragments, siguin del tipus que sigui, sense importar les dimensions. Un procés també es considera com un polígon.
- Els vincles són fragments on es prenen decisions, iniciades sempre per una porta, i tancades per una altra porta o per un esdeveniment final. Els bucles s'inclouen en aquesta categoria. La restricció és, però, que si hi ha porta de sortida, els fluxos que surten de les portes d'entrada no es trobin fins a la porta de sortida. Els vincles poden contenir tant fragments trivials com polígons. Els subprocessos també són considerats com a vincles.
- Els rígids són vincles més complexos, on dues o més de les branques que surten de la porta d'entrada es troben a mig camí abans d'arribar a la porta de sortida. En aquest cas només es poden posar els fragments trivials que es troben entre les dues portes com a fills del rígid.

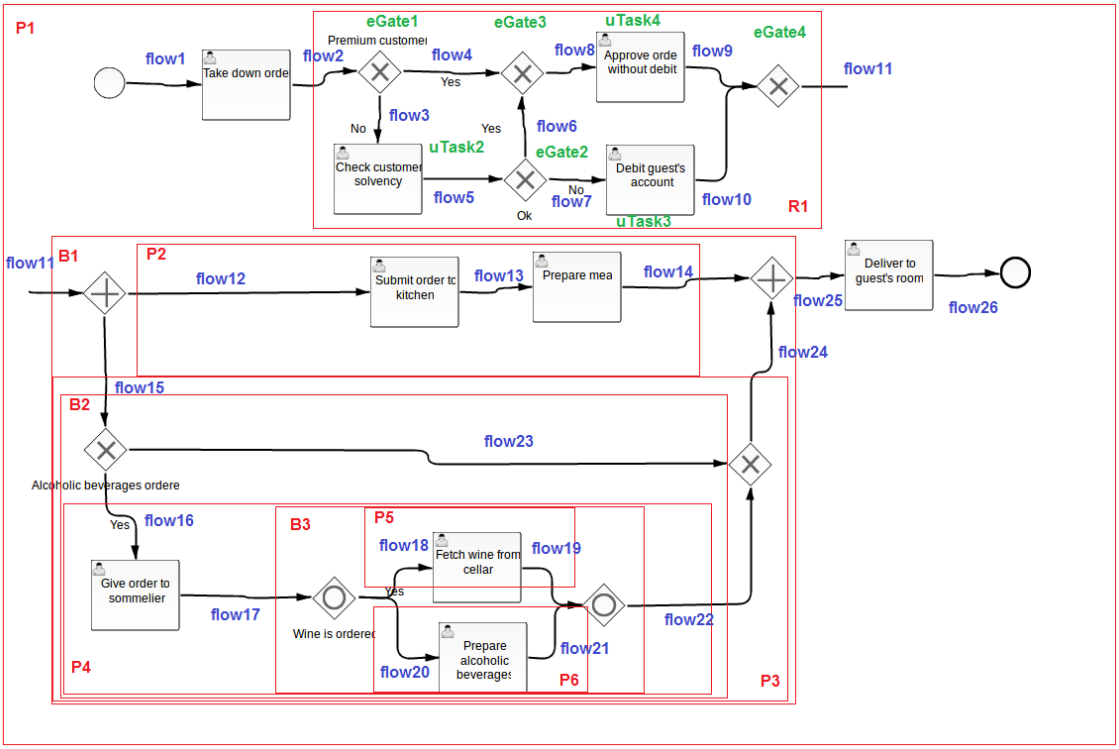


Figura 2. El model de processos anterior s'ha organitzat en els blocs que formaran el RPST

Hem aprofitat la figura 1 per a poder explicar de forma més visual com són aquests polígons. Com es pot veure a la figura 2, el procés sencer és un

polígon (iniciat per P). Veiem que certs fluxos només són dins de P1, com poden ser el 1, el 2, el 25 o el 26. Aquests fluxos seran, doncs, fills del polígon P1. Es pot comprovar també com a la part de baix hi ha un rectangle vermell que encercla altres de més petits. Aquest (B1) és un vincle, i com es pot veure té dos polígons que són els més grans de la resta (P2 i P3), essent aquests els seus fills. Sempre que un flux es pugui incloure en un polígon es farà, tret de si és flux de tornada a bucle o de salt. També es pot veure que cada porta d'entrada genera un vincle, el qual tindrà tants fills com sortides tingui. En canvi, es pot comprovar com el rígid (R1), a la part superior de la figura, només inclou els seus fragments trivials.

A la figura 3 es poden veure els resultats d'aquesta divisió en blocs. El procés, P1, té tan el rígid, com el vincle principal, com 5 fragments trivials com a fills directes. Com s'ha explicat anteriorment, rarament els fluxos estaran fora de polígons, a excepció de si és tornada de bucle o un salt. En aquest cas això li passa al flux 23, que passa directe de la porta d'entrada a la de sortida. També cal fixar-nos que per a cada porta els vincles iniciats en aquesta tenen tants fills com sortides té la porta. B1 té dos fills per que la seva porta d'entrada té dos fluxos de sortida. Això mateix s'aplica a B2 i a B3. També s'ha d'afegir que els fragments trivials sempre seran nodes fulla, i només els fragments trivials ho seran.

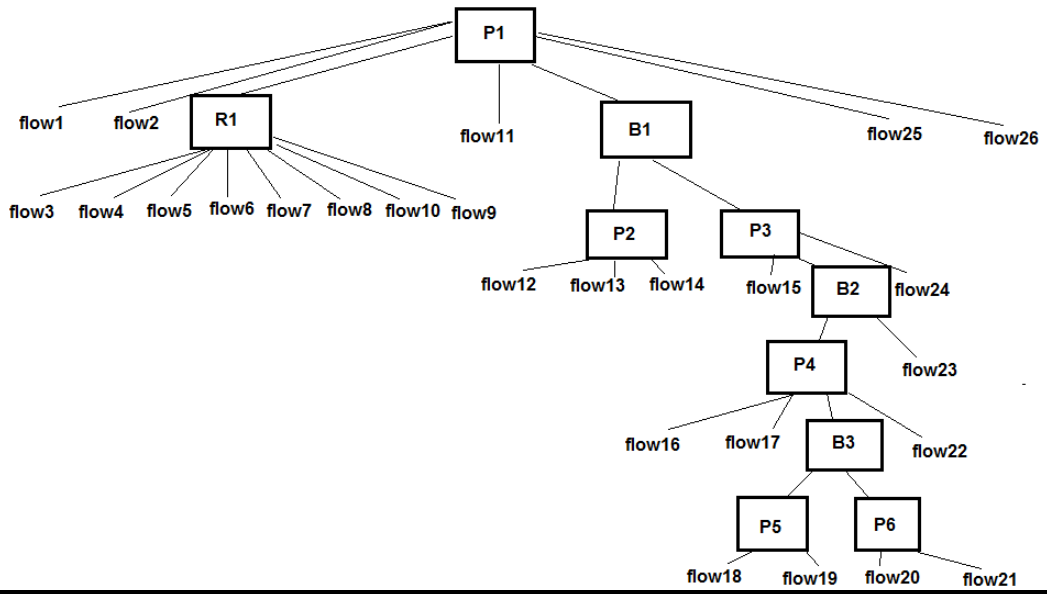


Figura 3. L'arbre RPST resultant del procés de creació de l'arbre

2.3 Xarxes de Petri

Una xarxa de Petri és un graf bipartit dirigit on els nodes representen transicions i llocs. Els arcs dirigits descriuen llocs i pre o post condicions per a les transicions[3]. Petri Net és un de molts llenguatges de modelat matemàtic per a la descripció de sistemes distribuïts. En el cas de la xarxa de Petri no sempre ha estat constant, sinó que primer es feia d'una forma més ineficient, passant posteriorment a crear-se en forma d'autòmat. Les xarxes de Petri s'han utilitzat per tal de desfer els fragments que no s'han pogut desfer anteriorment. En l'exemple que s'està comentant, aquest és el cas del bloc R1, com es pot veure en la figura 3.

2.3.1 Xarxa de Petri original

En la figura 4 es pot observar quin era el resultat de descompondre R1 en la xarxa de Petri original. En aquesta forma original de crear una nova estructura es necessitaven moltes dades per cada node, com quin era l'antic nom de la porta, quins successors tenia i quines vies passaven per cada flux. Tot i això, el resultat mantenia l'estructura original del rígid en quan a portes i tasques reals.

Això, però, era l'únic punt positiu, ja que es creaven sempre nodes ficticis que no existien, ja que sempre calia que hi hagués una tasca entre porta i porta i una porta entre tasca i tasca. En la figura es poden apreciar dues portes (p2 i p4) i 4 tasques (Task01, Task02, Task03 i Task04) amb els noms de color vermell. Aquestes són les tasques fictícies. Per tant de 7 nodes es passava a 13. A més a més, cap de les portes fictícies tenia informació. Les tasques només tenien com a informació el flux de sortida. En color blau es veuen les portes i fluxos reals, i les activitats reals són en negre. Després calia assignar-li una via, com a mínim, per a cadascun dels fluxos. El número en color verd indica quina via era cadascuna.

Aquesta tècnica tenia un problema. Que no imprimia les condicions de les portes per separat, i tot i que una condició estigués a la meitat es posaven totes seguides, sense dir abans les tasques que es feien entre les dues portes. Això feia que el text perdés el sentit de l'ordre en les parts on hi hagués rígids.

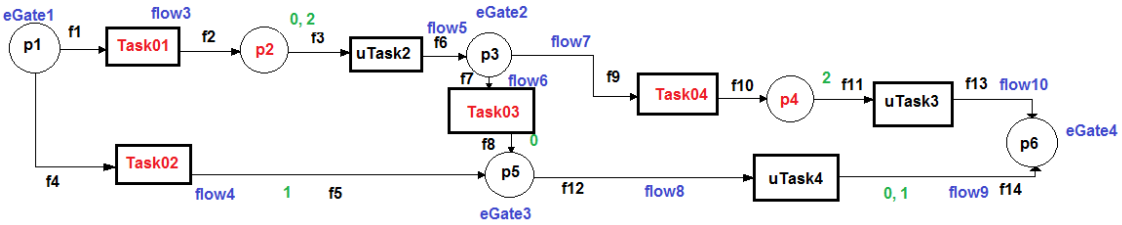


Figura 4. Aquest era el resultat de la creació de les xarxes de Petri

2.3.2 Arbres adaptats a l'autòmat

Degut a la ineficàcia i a la ineficiència de l'altre forma de crear una xarxa de Petri es va pensar en fer-ne una de nova, més eficient i que li aportés més granularitat als rígids. Aquesta era passar-li aquest rígid a un programa extern anomenat Petrify. Però per tal de fer-ho abans calia crear un arbre de successions que pogués assemblar-se a una màquina d'estats o un autòmat.

Per tal de fer-ho es comença per la porta d'inici. Per a cada porta es miren tots els fluxos de sortida, fins que es troba una porta disjuntiva o una tasca. En el segon cas es posa la tasca com a aresta, incorporant-hi la seva informació i la branca de la porta. En cas que sigui una altra porta es crea l'aresta com el flux que les connecta. Després d'una tasca si ve una porta es posa aquesta com a nou node. En cas que el que ve després d'una tasca sigui una altra tasca caldrà posar un node fictici, de la qual sortirà la tasca següent. Això es fa de forma recursiva fins a arribar al final.

Com es pot observar es passa de 13 nodes en el primer a només 5 en l'actual. Aquí es pot veure que la porta que no aportava informació real, eGate2, ha desaparegut de l'arbre, i com les tres tasques (uTask2, uTask3 i uTask4) han passat a ser arestes de l'arbre. Es pot veure també que la porta de sortida, eGate4, surt tres vegades i sempre al final dels estats. Es pot comprovar, comparant amb la figura 3 com els fluxos que surten de eGate1 acaben a les tasques uTask2 i uTask4.

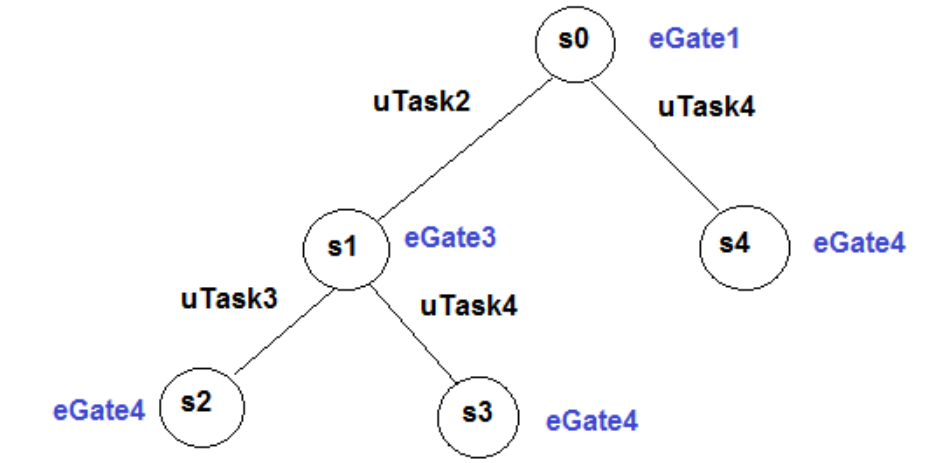


Figura 5. Els arbres de l'autòmat contenen certa informació per a la impressió

2.3.3 Resultat del Petrify

Un cop l'arbre de la figura 5 s'ha transformat en un fitxer, es passa aquest al Petrify, que el llegirà. Com es pot veure en la figura 6, ara és una xarxa de Petri simplificada, on hi ha portes (p0 i p1) i tasques (uTask2, uTask3 i uTask4). Ara, però, veiem que la porta que tancava el rígid també desapareix en totes les ocasions on sortia, quedant només aquelles que realment divideixen les branques. També es pot observar que les tasques han deixat de ser arestes per a passar a ser nodes. En canvi, veiem que l'estructura en quan a ordre és prou similar entre l'autòmat i la nova xarxa de Petri, ja que les tasques tenen un antecessor i, com a molt, un successor, mentre que les portes en poden tenir més d'un successor, tot i que només un antecessor.

Si comparem també amb el resultat de la figura 4 es pot veure també que està millor separat, i degut a que l'estructura del resultat del Petrify té símils amb la de l'autòmat, que ha guardat la informació necessària, es podrà imprimir el text gairebé com si fos un vincle més. Es podria dir, veient que uTask4 apareix dos cops, que ha duplicat les parts on el flux pot arribar a passar en més d'una ocasió.

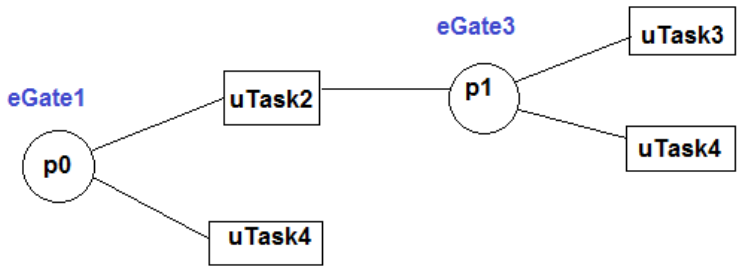


Figura 6. La xarxa de Petri simplificada resultant del pas pel Petrify

2.4 Patrons

L'última part important a destacar són els patrons que s'utilitzen per al programa. De patrons se'n troben de molts tipus, dels quals se n'ha posat un exemple al segon annex, per a veure el funcionament. Les marques que es posen s'explicaran, d'altra banda, més endavant, a l'apartat implementació i problemes. Es pot comprovar com hi ha diverses paraules que estan totes escrites amb majúscula. Aquestes són paraules clau, indicadores per al programa per a llegir els tipus de patró. Algunes paraules clau marquen inici i final de patró o sèrie de patrons, altres són simplement el nom de la sèrie de patrons, mentre que la resta són restriccions aplicades per a un patró concret.

A partir del RPST o bé de la xarxa de Petri existeix una funció que mira en quin tipus de node ens trobem. Si és un flux mirarà quina tasca la tanca i li posarà un patró d'acció, o bé un de temps si és esdeveniment intermedi. En cas dels vincles s'utilitzen els d'introducció de les branques o de bucle, i en els polígons s'utilitzaran alhora els de branca de vincle i els de seqüència. En cas que sigui una seqüència d'un únic element es posa $1 directament com a patró.

3. Estat de l'art

3.1 Resultats de treballs anteriors

Hi ha hagut alguns treballs anteriors amb els quals s'ha treballat sobre la generació de llenguatge natural amb tècniques ben diferents les unes de les altres. A més a més de amb models de processos, també s'ha creat llenguatge natural a partir tant de UML com amb models d'objectes. Tots tres tipus seran explicats a continuació.

3.1.1 Llenguatge natural sobre models d'objectes

Al document "The ModelExplainer"`[4] es pot veure una implementació que passa un model d'objectes a llenguatge natural. Els models d'objectes en el món de la informàtica són dues coses diferents: [5]

- Les propietats d'objectes en general en un llenguatge de programació específic. Aquests models d'objectes pel general es defineixen utilitzant conceptes com classe, missatge, herència, polimorfisme i encapsulació.
- Una col·lecció d'objectes o classe per les quals un programa pot examinar i manipular algunes parts específiques del seu món.

En diversos estudis previs es va veure que els llenguatges gràfics suposaven una dificultat per als usuaris, ja fossin usuaris normals o fins i tot analistes experts. Els autors van arribar a la conclusió que calia complementar-se amb una vista alternativa. Per això van crear la eina ModelExplainer o ModEx.

Les característiques del ModEx inclouen tenir exemples en els seus textos, tenint també descripcions convencionals, utilitzar interfície d'hipertext que permet als usuaris cercar pel model basada en tecnologia WWW, i un llenguatge de modelat simple, basat en ODL, que és la oficina d'aprenentatge digital. Una de les seves mancances és que no té accés al coneixement del domini del model de dades.

L'estudi que ells fan es basa en un escenari on una universitat contracta un analista de consultoria per a construir un sistema d'informació per a la seva administració. Aquest fa un model de dades i li passa a un membre de l'administració per a que el validi. L'administrador, però, no és familiar amb aquesta notació. Invoca ModEx per a generar una descripció textual en anglès. Ells fan que es vegi a través d'una pàgina web. També s'inclouen enllaços d'hipertext, segons on es cliqui. Això pot aclarir també dubtes sobre els nombres que hi hagués en cada relació segons el model. També permet arreglar els errors que s'hagin comés en aquestes.

ModEx s'implementava utilitzant l'arquitectura de "pipeline". Opera com a servidor web que genera fitxers HTML que es poden veure per qualsevol servidor web.

Té la virtut de ser portable, però això significa que no detecta errors de modelat semàntics. Dóna, però, una representació diferent del model. L'absència de coneixement del domini fa que no pugui escollir la para-frase correcta. Pot ser que surti una frase que no tenia res a veure amb el context original. Això es camufla mitjançant un requeriment a l'usuari en respecte a l'etiquetat de relacions i objectes. Això té un segon propòsit: imposar disciplina en el nomenament, degut a que es necessita consistència.

Està implementada en C++ sobre plataformes de PC i UNIX, i integrat en ADM i KBSA.

3.1.2 Llenguatge natural sobre diagrames UML

El text creat a partir de diagrames UML s'ha fet, com es pot demostrar en el document Generating Natural specifications from UML class diagrams[6]. Ells utilitzen WordNet com a ontologia lingüística per a eliminar ambigüitats en estructures lèxiques del UML i crear frases que realment semblin frases. El sistema el fan amb Java. El seu objectiu és traduir diagrames UML a llenguatge natural.

Una de les tasques que fan és cercar sistemes de generació de llenguatge natural. S'omplen certs patrons. També escriuen un diagrama de classes com a

una col·lecció de classes interconnectades per una llista de relacions. Al nombre d'objectes involucrats se'l coneix com a multiplicitat.

Ells havien examinat 45 diagrames de classe. Van observar dues propietats: que el llenguatge utilitzat a UML és un subconjunt controlat de llenguatge natural i que a la literatura observada els desenvolupadors de software utilitzen convencions per a anomenar els diversos components de diagrames de classe. Ells utilitzen unes convencions del món acadèmic per a anomenar les classes i relacions.

L'arquitectura del GeNLangUML (el nom abreujat de Generating Natural specifications from UML) es composa de quatre components, que són el pre-processador i etiquetat, planificador de frases, realitzador i estructurador de document. Es va desenvolupar, però, un pas previ que era fer una eina per a dibuixar components. S'utilitza XML com a representació interna dels diagrames de classe UML.

El pre-processat i etiquetat es fa degut a l'ambigüitat del llenguatge natural. S'anomena etiquetat al mecanisme d'anomenar una part del discurs. S'utilitza un algorisme de triatge que escull el més probable.

El planificador de frases rep les etiquetes i els mots corresponents de l'etiquetat generant frases seguint una aproximació basada en plantilles, combinant les paraules en atributs, operacions, classes i associacions. Segons el tipus de part, atribut o operació s'apliquen diferents rutines segons els casos d'aquestes parts.

En quant a generació per a relacions hi ha tres tipus de relacions: associacions, agregacions i generalitzacions. Una associació és una relació funcional entre dos o més objectes. Una agregació s'utilitza per a relacions on un objecte és la part o el tot d'un altre. Una generalització vol dir que un objecte és un cas especial d'un altre.

El realitzador agafa les frases fetes al planificador per a assegurar concordança entre paraules i agregacions. Dins d'aquí tenen un processador morfològic, que s'encarrega de la cardinalitat entre frases de noms o entre frase de noms i verb. També fa canvis segons el verb sigui present o passiu. L'agregat agafa

frases del processador morfològic i els agrupa segons sigui frases amb té o no i també combina les frases de verbs eliminant duplicitats.

L'estructurador del document el que fa és fer més llegibles les frases que han sortit de la fase anterior. Es processen amb associacions i frases que es refereixen a la mateixa entitat, agrupant les que es refereixen al mateix i separant-les de la resta. Per últim posen un analitzador de semàntica, que s'encarrega de verificar que les frases són correctes semànticament i de mostrar entitats relacionades a tots els noms principals i noms d'objectes a la classe UML.

Les frases generades d'atributs i operacions sempre tenen un verb. La verificació inclou comprovar la validesa de certes frases de nom que es trobin al voltant d'un verb específic. Es categoritzen les frases de noms segons individu, cosa o col·lectiu, extraient-ne el nom principal. Hi ha certes excepcions: els verbs tenir, aconseguir, posar, canviar, actualitzar i insertar.

Per a fer les proves ells van agafar un diagrama de classes d'un sistema d'universitat que té funcionalitat estàndard. Eren un conjunt de casos d'ús, el que implicava que es miraven els resultats per cada cas d'ús. Els noms de classe i relacions són 100% compatibles amb les regles, i el 94.5% dels atributs també, essent la resta també ben etiquetat. El 89.8% dels noms d'operació eren compatibles, i els no compatibles tenien la forma verb-nom-adjectiu.

Es descobreix que el 92.8 % de les cadenes de paraules són compatibles amb les regles i que del 35.7 % de cadenes que resulten ser ambigües se'n resolen el 84% correctament degut a l'alta compatibilitat. Una gran part de la informació dels casos dels diagrames d'ús s'ha pogut traduir en frases, degut a que la majoria d'operacions tenien un sol verb. Totes les cadenes de paraules resoltes acaben sent frases, però aquelles cadenes de paraules que no s'han pogut resoldre bé acaben sent frases sense sentit.

Les mancances del seu projecte són, però, que s'ha desenvolupat en un entorn acadèmic, pel qual les convencions d'anomenat s'han utilitzat d'acord a llibres acadèmics, podent ser ben diferent en el món industrial, té un nivell

d'abstracció similar al diagrama de classe original i que els diagrames de classes no són l'únic model desenvolupat.

3.1.3 Llenguatge natural sobre models de processos

Hi ha hagut, però, experiments interessants que han treballat tant llenguatge natural com models de processos. Aquest és el cas explicat al document "Supporting Process Model Validation through Natural Language Generation"[7]. Com es pot veure, passen per sis fases per tal d'acabar extraient un text. Aquestes fases són: extracció d'informació lingüística, generació d'un RPST amb anotacions, estructuració del text, generació d'un missatge DSynT, refinament del missatge i realització amb RealPro. Aquest experiment i el que jo he fet tenen bastants elements en comú, degut a que les bases s'han agafat a partir d'aquest document.

L'extracció d'informació lingüística consisteix en fer inferència en informació lingüística en tots els elements etiquetats del procés. També enriqueixen elements com activitats, esdeveniments o portes. En alguns casos les portes són considerades condicions. La generació d'un RPST amb anotacions es fa derivant cada actor de la entrada, que contindrien una jerarquia de subgrafs derivats del model original, als quals se'ls hi anomena fragments.

En l'estructuració del text, el text ha d'acabar subdividit en diferents paràgrafs de forma heurística. Algunes aproximacions sofisticades, però, intenten usar distribució de semblança per a identificar l'òptim. Primer es comuniquen bé les branques, quedant semànticament relacionades. Les branques paral·leles o alternatives també són explicades. La generació de missatge DSynT transforma el RPST amb anotacions en missatges intermedis per a cada node. Es guarda la informació en un arbre, essent això un lexema complet, sense verbs ni articles. Es posen aquí relacions entre dos nodes. A partir d'aquests arbres es treuen els missatges intermedis. Segons el tipus de node es fa una acció o una altra.

El refinat del missatge afegeix expressions i inserció de marcadors de discurs. En aquest etapa s'agafa la sèrie de missatges procedent de la anterior, afegint-

hi els rols, els objectes de negoci i les accions. La realització de superfície es fa amb RealPro. La part de llenguatge natural resultant s'afegeix al final del text de sortida.

Les proves les fan a partir de 46 models: Els models són 4 de la universitat Humboldt de Berlin, 2 de la universitat tècnica de Berlin, 8 de la universitat de tecnologia de Queensland, 1 de la de Eindhoven, 3 de tutorials de venedors, 4 de Inubit AG, un d'un consultor BPM, 3 d'un llibre de pràctiques de BPMN, 6 d'una guia de BPMN i 14 del sector públic. Ho fan segons nombre mitjà de frases, paraules per frase, clàusules per frase, unitats T per frase, unitats T complexes per frase, frases per node, cobertura d'activitat, d'esdeveniment, de porta, de flux, de seqüència, frases amb contingut, meta frases i frases d'informació.

Inicialment els que més nodes per model tenen són els de la universitat tècnica de Berlin, igual que les activitats, esdeveniments, portes, fluxos, actors i línies per node. Només és superat per les guies de BPMN en nombre de símbols diferents de símbols BPMN. En la banda oposada tenim el consultor de BPMN, que té menys Nodes, activitats, portes i actors per model. El que menys esdeveniments i línies per node té és la universitat de Queensland.

Això es veu en el temps que triguen en generar-se. De mitjana el del consultor de BPMN només triga 4,19 segons, mentre que els de la universitat tècnica de Berlín en triguen 9 i mig. També repercuteix en els resultats finals. Pot comprovar-se que el del consultor manté o augmenta lleugerament els valors originals, essent 7 frases originals per 8 finals, 8.1 paraules per frase originals per 10 finals, 1.3 clàusules per frase finals per 1.1 originals, 0.9 t-unitats finals per 1 inicial, passant de 0.1 a 0.2 les complexes i de 0.9 a 1 frase per node. D'altra banda podem veure que excepte en nombre de frases, de t-unitats i frases per node el de la universitat tècnica de Berlín redueix força els seus valors. Passa de tenir 34 a 42.2 frases, però les paraules per frase baixen de 20 a 8.9, les clàusules de 1.8 a 1.2, les t-unitats de 1 a 0.9, les complexes cauen del 0.6 al 0.1 i les frases per node pugen de 0.6 a 1. De fet, el nombre de frases per node està en tots els casos al voltant del 0.9, entre 0.8 i 1 en els casos finals, demostrant així la semblança.

Finalment, la cobertura d'activitats, esdeveniments, portes i fluxos sempre acaba essent del 100%, quelcom que inicialment les activitats només ho feien un 99% i els esdeveniments un 96%. En canvi, les frases de contingut baixen del 91 al 85, mentre que es disparen les meta frases del 1% al 15%. Les de informació passen del 8 a desaparèixer. Per últim, cal esmentar que en una prova de retraduït van fer que 11 estudiants provessin d'entendre tres models cadascun. El pitjor cas va ser un encert del 83%, mentre que dues terceres parts van ser del 100%.

També es pot, però, trobar projectes anteriors que fan el pas invers, és a dir, passar d'unes oracions en llenguatge natural a un procés en BPMN, és a dir, passar de paraules o textos a models de procés. Com es pot veure en el document de "Process Model Generation from Natural Language Text"[8], es treballa a partir del llenguatge natural, per a finalment crear un BPMN.

Com es pot veure aquí els autors comencen amb textos, a vegades poc estructurats, i aconsegueixen un model. Empren mètodes de lingüística computacional i processament de llenguatge natural. A partir d'aquí es crea un arbre sintàctic i les relacions gramàtiques entre les parts de cada frase. Ells utilitzen l'analitzador sintàctic de Stanford, que és una eina d'anàlisi sintàctic per a determinar un arbre sintàctic. Per a l'anàlisi semàntic s'utilitzen FrameNet i WordNet. Troben quatre problemes principals: la llibertat d'acció sintàctica, l'atomicitat, la rellevància i les referències.

Dins l'anàlisi de la frase primer descomponen la frase, i després extreuen els actors, els verbs, i després els objectes combinant-los amb els verbs, combinen cada actor amb cada acció i finalment comproven les conjuncions globals. En l'anàlisi a nivell de text es divideix en cinc fases: primer es resolen referències, després detecten marcadors, combinen accions, determinen enllaços i finalment construeixen fluxos. Després generen el model de procés. Dins d'això primer creen els nodes, després els fluxos de seqüència, eliminen nodes inútils, finalitzen les obertures, processen meta activitats, si cal creen les pools (o accions per actor), si cal també construeixen objectes de dades i finalment el model.

Per a avaluar el model agafen 47 parelles text-model, els mateixos que abans, al qual se li afegeix un venedor més. Es mira quant similars són el BPMN fet a mà pels autors i el resultant. En cas que tots els nodes es trobin el resultat serà 1, i en cas que cap hi sigui, 0. Hi ha nivells intermedis entre aquests dos valors. El 77% dels nodes es troben, i es pot arribar fins al 96% en els millors casos. Els errors es deuen a sorolls, nivells d'abstracció diferents i problemes de processament en el sistema. El pitjor cas en mitjana és el de la universitat d'Eindhoven amb una semblança d'un 41.54% i el millor el sector públic, amb un 89.81%.

3.2 Conclusions

Com es pot veure, el llenguatge natural és una ciència que es pot fer de diferents maneres i amb gran varietat de models d'origen o de destí, i tot i que ja hi ha un gran nombre de models en els que ja s'ha provat, segur que no s'ha provat sobre tot tipus de models. El coneixement d'aquest món és bastant ampli, però no s'ha aconseguit mai fins ara una precisió del 100%. Això és degut a que el llenguatge natural té moltes ambigüitats que fan gairebé impossible aconseguir la perfecció.

Considerant les diferents opcions vistes anteriorment, vaig considerar que la millor opció era basar-me en part en la idea proposada en el document de Leopold, Mendling i Polyvyanyy[7], però fent-hi variacions i ampliacions, fent que el programa pogués servir per a més d'un sol idioma (el projecte explicat anteriorment només serveix per a l'idioma anglès), creant més d'una frase per a cada tipus de porta o de seqüència, aprofitant-ne, però, les ja existents, o bé permetent als usuaris a adaptar-lo a la seva forma d'expressar-se, així com passar els rígids a un BPMN mitjançant autòmats, quelcom que al document es passa a una estructura de dades diferent a la que s'aplica en el meu projecte.

El propòsit d'aquest projecte, però, no era només el de crear uns textos a partir de BPMNs o ampliar l'altre treball, sinó també el d'experimentar sobre com actuarien aquest textos si se li afegís varietat que li donés més riquesa i menys automatismes a les frases, sabent si és una opció per a millorar la precisió o si

allò investigat fins ara era millor que els resultats d'aquesta prova. Això feia que aquest experiment fos prou interessant com per portar-lo a terme.

4. Abast

4.1 Objectius

L'objectiu final era que a partir d'un model de processos, que constés d'activitats o portes connectades entre sí per fluxos s'aconseguís crear un text que la persona interessada pogués entendre. Això es va pensar de cara a fer-ho per a tres idiomes, anglès, català i espanyol, tot i que finalment, s'ha acabat renunciant a aquest últim per falta de temps, però manté la possibilitat de ser adaptable a qualsevol altre llengua. Dins d'aquest objectiu principal, s'han inclòs uns quants de secundaris que es detallaran a continuació:

- Passar de BPMN a RPST: L'objectiu aquí era fer un algorisme que permetés passar qualsevol model de processos a un RPST. En el nostre cas, concretament, un node representa un flux o un bloc contenidor. Aquests últims tenen supeditats altres blocs o fluxos.
- Passar parts d'un RPST a un de més eficient mitjançant autòmats: En les parts on el RPST perd concreció (això passa quan el fragment és rígid) cal crear una xarxa de Petri. En el cas d'aquest projecte, inicialment els nodes representaven portes i activitats tant reals com fictícies, reduint-se finalment a una xarxa més eficient amb només portes i tasques reals, Es concep com a una màquina abstracta que pot ser en un dels estats finits.[9]
- Lligar els RPSTs i les activitats d'aquests amb les plantilles per casos per a crear un text: A partir del RPST (original o xarxa de Petri simplificada, segons el cas) l'objectiu era el d'agafar la plantilla d'entrada i substituir els forats especificats dins de cada cas necessari de la plantilla per l'activitat que el succeeix o pels nodes fill de l'arbre, de tal forma que quedés un text el més semblant possible a un text real.
- Creació de fitxer de sortida en text pla: L'objectiu en aquesta part era poder crear un fitxer de XML i posar-hi contingut textual de tal forma que quedés ben estructurat per a ser llegible per a l'usuari, i també de crear un fitxer de text pla. Finalment no s'ha fet el fitxer en XML, degut a que s'ha considerat massa llunyà als objectius principals del projecte, el qual

s'acosta més a passar de models a text, que no pas a crear fitxers en altres llenguatges de programació.

En la següent figura es pot veure els passos que fa el programa des del model original fins al text final

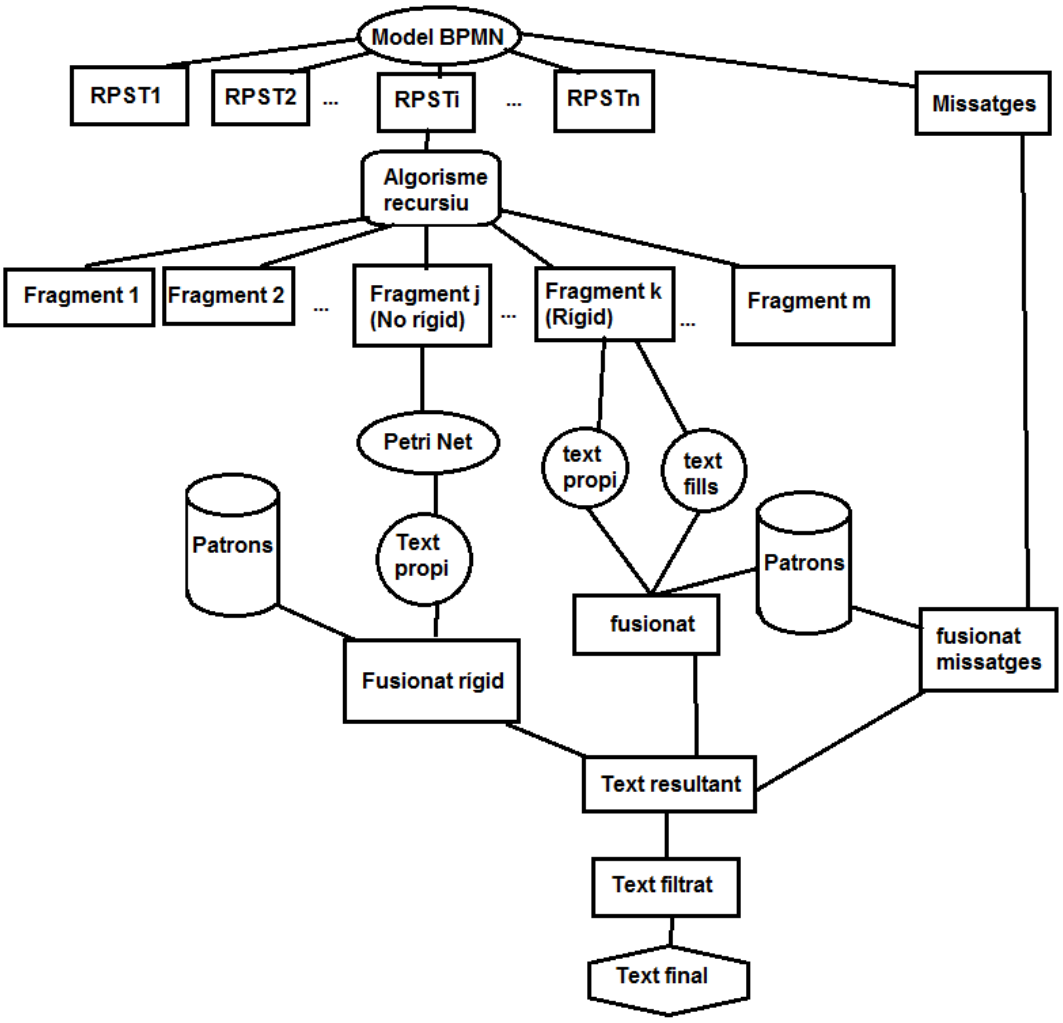


Figura 7. Aquest és el procés que pateix un model BPMN fins a arribar al text final

Per a cada execució hi havia inicialment dos fitxers d'entrada. Per una banda un model BPMN que tingués les relacions de fluxos i per l'altra banda una plantilla en anglès que indicava com es volien fer les frases i els forats on anirien les parts que s'anessin extraient en l'algorisme. Finalment, s'hi han incorporat una altra plantilla per al català, i també un altre fitxer d'entrada que conté els paràmetres que es poden escollir, que són idioma, model, tenir o no

cometes, posar allò que és al mateix nivell de profunditat amb la mateixa indentació o no i posar-ho tot com a llistes, paràgrafs o bé una barreja dels dos.

4.2 Explicació de l'abast

L'abast d'aquest projecte ha inclòs vuit passos, ordenats aquí per l'ordre ideal d'implementació:

- El primer pas era conèixer tot allò amb el que es treballava, ja fos Sistema Operatiu (Ubuntu), llenguatges de programació (Java i XML) o Plug-ins (Activiti, Petrify i Camunda), estudiant-ho en profunditat i practicant-hi.
- Lectura d'un fitxer d'entrada: Llegir un fitxer d'entrada en XML i Activiti. Es pot veure un exemple d'un d'aquests models en XML al quart annex.
- La construcció dels RPSTs: Amb totes les dades, a partir del BPMN es construeix un RPST mitjançant un algorisme, canviant els fluxos per nodes d'un arbre, amb algunes excepcions.
- Fabricació d'un RPST alternatiu mitjançant autòmats: En cas que un bloc del RPST fos massa complex, es descompon aquest bloc mitjançant un algorisme que transforma aquest bloc complex en una xarxa de blocs més senzilla i tractable. Per a això ha calgut una eina externa, Petrify.
- La creació del text: A partir del RPST, xarxes de Petri (si cal), les plantilles de frases i els missatges entre processos, s'agafa una frase aleatòria de la plantilla segons el tipus de relació entre els elements i amb un algorisme s'afegeix el que es demana en el lloc de les etiquetes, sigui el cas.
- Polit i paràgrafs: S'eliminen repeticions i duplicitats del text, siguin marques de paràgrafs, signes de puntuació, tabulacions o espais per mitjà d'un algorisme que busca marques que sobren.
- Creació dels fitxers de sortida: Es crea un fitxer de sortida en text pla. Finalment no s'ha creat en XML.
- Jocs de proves: Es fan unes plantilles i uns BPMNs de prova utilitzant XML i Activiti en el segon cas i text pla en el primer, emparellats entre

ells. Posteriorment s'han provat, s'han analitzat els seus resultats, els quals es fan servir per a validar el projecte.

Es necessita, però, que aquest programa valgui per a qualsevol fitxer que sigui en XML, llenguatge en el qual també es farà de sortida.

Aquest projecte compta finalment, amb dues plantilles completes en català i anglès, a partir de les qual es podran crear de noves per a aquests dos idiomes i per a d'altres, en comptes de la plantilla bàsica plantejada inicialment. Per altre banda, degut a que cada consumidor té unes necessitats diferents, no s'inclou a l'abast els BPMN d'entrada degut a que els models de processos són els que necessitin els consumidors.

4.3 Compliment dels objectius

Gairebé tot allò que es va prometre fer en la fita inicial d'aquest projecte s'ha complert. Tot i així hi va haver un canvi que és relativament menor a les dimensions d'aquest projecte. Aquest ha estat la creació dels fitxers de sortida. Degut a que a mig projecte es va considerar que no calia passar el text resultant a XML després que ja fos text pla per estar massa allunyat de la formulació d'aquest problema, es va substituir per un simple fitxer de text pla .txt, el qual a més a més s'ensenyarà per consola. Tampoc s'ha fet una plantilla per al castellà, tot i que el programa està adaptat per tal de poder-ho fer. Simplement és agafar una de les altres plantilles, canviar-li el nom del fitxer i les frases, mantenint, però, les paraules clau.

4.4 Obstacles i limitacions

Inicialment, en les proves només s'incloïen opcions per a poder fer textos en català, espanyol o anglès. Finalment, s'ha hagut de renunciar a l'idioma espanyol. Tot i així, en principi és un programa adaptable a altres idiomes, inclòs l'espanyol, com per exemple el francès, l'alemany o l'italià.

La limitació principal, però, era la del llenguatge de programació amb el que es llegeixen els models: només es podien llegir models BPMN que estiguessin

adaptats al format XML i que, alhora, fossin comptables amb Activiti. L'execució del codi fallava en cas que no fos un XML, o bé no donava cap resultat si era un XML no compatible amb Activiti, ja que es produïa un error en el qual no es llegia cap procés. També hi ha limitacions amb les plantilles, ja que hi ha certes etiquetes que són per a encaixar les diferents peces del text o per a crear els paràgrafs segons seqüències o blocs. Inicialment s'ignoraven els esdeveniments intermedis, però finalment s'apliquen només per a esdeveniments de temps.

Aquest projecte tampoc detecta tots els tipus d'element, pel que potser no s'hi podrien aplicar tots els models. Tot i així, els elements no inclosos en el projecte són elements que s'utilitzen rarament, com bases de dades.

L'últim problema ha estat a l'hora de fer el programa. Un error significava un retard a l'hora d'acabar una tasca, i degut a que el temps era limitat, fins el 18 de gener, gairebé ha significat un problema.

4.5 Metodologia i rigor

La metodologia i el rigor del projecte no han variat gaire des de la fase inicial, degut a que ja era una metodologia adequada.

L'únic que ha canviat en realitat és que s'han afegit plugins amb els quals es treballen. Tot i que es mantenen l'editor Eclipse en llenguatge Java, Activiti i XML, amb el qual es fan l'entrada i la sortida, s'han afegit Camunda, per a crear models BPMN, i Petrify, per a simplificar xarxes BPMN complexes a xarxes de Petri simples. A mig projecte es va plantejar la possibilitat d'incorporar Freeling, que hagués millorat l'anàlisi semàntic. Finalment, aquesta eina no s'ha utilitzat. El mètode de treball ha estat el proposat al principi, treballar cada setmana de manera regular, ja que és així com es fan els projectes reals.

Les fases del projecte s'han tractat també tal i com es va plantejat inicialment, ja que és indispensable per a la programació. S'havia de pensar per tal que quedés amb la implementació més eficient i eficaç possible. Un cop pensat s'implementava el codi. Per provar que cada part funcionava es feia primer un exemple manualment a partir del qual es feia la prova amb el programa.

S'havia de mirar si el resultat era raonablement similar a l'esperat. En cas que no funcionés el programa es corregien els errors. Si donava un resultat massa diferent de l'esperat es comprovava el perquè i es corregien els errors si s'esqueia.

El seguiment d'aquest projecte s'ha fet mitjançant reunions regulars amb els dos directors del mateix, com era d'esperar, o amb un d'ells si l'altre no hi podia assistir, cada setmana dimarts a les 10, per tenir suficient temps per a fer la feina i no passés tampoc un temps excessiu abans de tenir noves tasques a fer, fins a mitjans de desembre, mantenint posteriorment el contacte per correu electrònic. S'ha fet també així per a comprovar que el projecte progressava adequadament i es feien les diferents parts del treball d'una forma correcta.

En el que fa al mètode de validació, s'ha aplicat un mètode similar al que s'aplica en el del document de Leopold, Mendling i Polyvyanyy[7]. Això és agafar parelles formades per un text i per un BPMN. Es comprova mirant si realment el text resultant del BPMN és similar al text original. Aquests parells són extrets d'una font externa que conté algunes de les proves que també van utilitzar ells, recomanades pels directors i fetes al Camunda per mi mateix. També s'han fet proves experimentals fetes per mi mateix, les quals, juntament amb les de la font externa, es detallaran més endavant. Totes s'han traduït posteriorment al català. La raó de les fonts era trobar exemples reals que donessin uns resultats similars, que a més a més tinguessin diferències entre uns altres, o posar exemples reals d'un estudiant com per exemple un projecte de TFG o l'avaluació d'un examen.

5. Integració de coneixements i anàlisi d'alternatives

5.1 Coneixements previs

La creació d'arbres n-aris, apresa a l'assignatura d'Algorísmia, m'ha servit per a poder fer els RPSTs. Això es deu a que un RPST és un arbre n-ari. El que cal fer en aquests arbres n-aris seria afegir-hi informació de tipus estructura que pugui relacionar els nodes entre ells i, alhora, pugui distingir els uns dels altres, a més a més de posar-hi informació relacionada amb el BPMN original. També em servia la mateixa assignatura per a fer la implementació antiga de les xarxes de Petri, degut a que aquests s'implementaven amb llistes de nodes, quelcom que també es va aprendre aquí. Aquestes, però, eren més complexes, ja que els nodes no s'afegien un a un i que cada node podia tenir més d'un successor o antecessor.

Per a la nova versió de les xarxes de Petri, en canvi s'utilitzaran coneixements de Llenguatges de Programació i de Teoria de la Computació. En aquest cas, el primer que es fa és crear nodes en forma de autòmat, quelcom que es va aprendre en aquesta última assignatura. Això, però, es farà d'una forma lleugerament diferent a la que vaig aprendre a l'assignatura, degut a que en aquest cas els estats són portes, siguin del tipus que siguin i les arestes són els fluxos. En alguns casos però, entre aresta i aresta ha calgut posar una porta falsa, que posteriorment acabarà desapareixent un cop passi a ser a xarxa de Petri simplificada. A més a més, aquesta variació no té estat final únic, si no que en té diversos. Com aquest autòmat es fa mitjançant una espècie de "parser", aquí s'apliquen els coneixements de Llenguatges de Programació. Un cop acabat l'autòmat és passat a un programa extern, Petrify, i a partir d'aquí es recull un BPMN que utilitzarà allò explicat per als arbres n-aris explicats anteriorment.

Per a fer la creació del text es poden aprofitar els coneixements apresos també a l'assignatura de Llenguatges de Programació. Aquí se'ns va ensenyar com fer un analitzador sintàctic o "parser", el qual agafa tot el que entra i ho transforma a quelcom diferent, com passa amb els autòmats explicats prèviament. Aquí es fa la variació de passar-ho a text. Ens hauria servit també

si s'hagués fet també el fitxer XML, ja que s'hauria hagut de traduir de text pla a codi d'un llenguatge, pel que ens servia per a les dues parts. També s'aprenia l'anàlisi de semàntica, que és el que utilitzava el Freeling per tal de separar subjecte de predicat, que finalment no s'ha utilitzat.

Per últim, els coneixements adquirits a Intel·ligència Artificial es podrien aplicar a aquest projecte de manera lleugera, sobretot per l'aprenentatge computacional, ja que és una tècnica que s'aprèn en aquesta assignatura. Aquí s'apliquen els algorismes pseudoaleatoris per a agafar patrons que s'utilitzen per a crear frases variades, i amb els "flags" que s'apliquen, posar-hi restriccions per tal que només apareguin frases que tinguin relació amb les que les envolten, i per a que encaixi amb la resta del text.

5.2 Adequació a computació

Aquest projecte és adequat a l'especialitat de computació perquè inclou diverses característiques típiques d'aquesta especialitat. Unes de les característiques més evidents és l'existència de molts algorismes i d'estructures de dades complexos. Un exemple d'això és l'arbre RPST, que és una estructura de dades complexa en la que cada node té molta informació que la diferencia de la resta. El mateix passaria tant amb la xarxa de Petri antiga com amb la nova, ja que la primera era una estructura que tenia molts paràmetres i que la seva estructura en sí també era complexa, degut al nombre variable d'antecessors i successors que tenia cada node, mentre que la segona consta de la generació d'un autòmat que conté una altra estructura de dades, que és intermèdia entre les dues anteriors. En tots tres casos s'utilitzen punters a altres nodes de l'estructura, quelcom típic també de l'especialitat.

L'algorisme de pas del BPMN al RPST també pot ser complex, ja que requereix transformar un model de processos en un arbre, i això comporta aplicar recursivitat i, en algunes parts, com per exemple en els vincles o en els rígids, pot necessitar una llarga estona de càlcul, ja que caldrà distingir el tipus de fragment. Encara més complicat era el pas de RPST a l'antiga xarxa de Petri, ja que calia fer des de l'inici del fragment fins al final la llista especial abans esmentada per cada node de la xarxa de Petri, creant també nodes ficticis que

no tenien cap significat, requerint també recursivitat. Després calia assignar-li un camí fins haver passat per totes les arestes. En la nova xarxa de Petri, és un algorisme recursiu similar al del RPST que crea un autòmat, que té un sol predecessor, però, en cas dels estats, pot tenir qualsevol nombre de successors, i guarden la informació necessària, la qual la passem al Petrify i ens retorna un nou BPMN, pel qual s'aplicaria un procediment similar al de RPST per a imprimir la informació.

El pas a text també és un pas que pot arribar a ser difícil. Per als casos no rígids, es cerca en la plantilla el patró del cas concret, s'implementa un algorisme d'aleatorietat amb limitacions per a que tregui un resultat que sigui adient a les circumstàncies en les que es troba el node. Un cop es té el patró es fa un algorisme iteratiu per a cercar on es troben els forats per a inserir els resultats del fill o de l'activitat que segueix el flux o, en casos particulars, el precedent. En aquest cas de nou s'utilitzaria recursivitat per tal que sortís tot el text, des de l'arrel fins a les fulles. A més a més, agafar el patró comporta tenir algorismes provinents de la Intel·ligència Artificial per tal d'aportar-hi la major variabilitat possible, sempre dintre d'uns límits. En el cas de l'antiga xarxa de Petri es buscaven totes les opcions iterativament que hi havia i es cercaven les condicions i activitats reals incloses en cadascuna per a imprimir-les per pantalla mitjançant els patrons. Actualment, però, degut a que els rígids són més similars que abans a un BPMN capaç de transformar-se en un RPST, es fan 4 iteracions recursives per a cada estat no final, agafant un patró més exterior i uns quants d'interiors que vagin encaixant, segons el nombre de textos que arribin dels seus fills o de les arestes que en surten.

Si s'hagués fet el pas al fitxer XML, hagués requerit una conversió de tot el text pla a un llenguatge de programació, pel que hagués requerit d'un altre algorisme que hagués creat les diferents parts del XML segons la disposició del text, és a dir, com estaven organitzats els paràgrafs, les llistes i els diferents apartats. Aquest principi, però, sí s'ha aplicat al canvi de llenguatge en el pas de XML a Activiti per tal de poder executar el programa.

5.3 Competències tècniques

Aquest projecte inclou vuit competències tècniques:

1. Demostrar coneixement dels fonaments, dels paradigmes i de les tècniques pròpies dels sistemes intel·ligents, i analitzar, dissenyar i construir sistemes, serveis i aplicacions informàtiques que utilitzin aquestes tècniques en qualsevol àmbit d'aplicació (CCO2.1). El nivell és en profunditat, degut a que s'apliquen tècniques d'Intel·ligència Artificial per a generar el text, aplicant patrons que es trien de forma pseudoaleatòria a partir d'una plantilla de patrons i posant-hi les parts introduïdes per l'usuari i unes certes restriccions per cada cas.
2. Capacitat per a adquirir, obtenir, formalitzar i representar el coneixement humà d'una forma computable per a la resolució de problemes mitjançant un sistema informàtic en qualsevol àmbit d'aplicació (CCO2.2). En profunditat, ja que és un requisit indispensable per tal de poder convertir en dades tot el coneixement humà necessari, en major part la informació del RPST, de la xarxa de Petri antiga, els patrons i l'autòmat. Per tal de passar del coneixement humà a informació computable s'utilitzen els diferents algorismes explicats anteriorment.
3. Avaluar la complexitat computacional d'un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d'acord amb els requisits establerts (CCO1.1). El nivell és bastant, ja que s'han de buscar diferents algorismes per a poder aplicar solucions a tots els objectius parcials i també el final, buscant una eficiència raonable en la seva implementació, a més a més de la seva eficàcia.
4. Definir, avaluar i seleccionar plataformes de desenvolupament i producció hardware i software per al desenvolupament d'aplicacions i serveis informàtics de diversa complexitat (CCO1.3). S'han de buscar diferents formes fiables per a poder programar sense tenir gaires problemes tècnics, cercant també problemes que siguin compatibles amb el software disponible. Per això el nivell és de bastant.

5.4 Anàlisi d'alternatives

5.4.1 Rígids

El cas més evident de l'anàlisi d'alternatives ha estat el dels rígids. Es van tenir dues idees principals en ment.

La primera d'elles era una variació lleugera d'un dels documents en el que em vaig basar, "Validation through Natural Language Generation"[7]. En aquest cas, a partir del fragment rígid, entre l'inici i el final el que es feia era crear una xarxa de Petri la qual tenia tasques i portes fictícies que no tenien cap tipus d'informació i era un mer formalisme per al funcionament correcte.

L'altra alternativa possible era el pas a un autòmat i l'ús d'una eina externa que ens ajudés a simplificar aquesta xarxa, o més concretament, descompondre el BPMN en trossos llegibles. En aquest cas l'autòmat només posaria informació útil, tota agrupada en arestes i nodes, essent les primeres tasques o fluxos i els segons, portes. L'eina externa retornaria el resultat de l'autòmat convertit a una xarxa de Petri simplificada per tal de fer-ho més eficient.

Finalment es va triar la segona opció per diverses raons. La primera és que es necessiten menys iteracions, 2 per contra de 3, i s'implementen menys nodes, degut a la inexistència de portes fictícies i el reduït nombre d'arestes falses. La xarxa de Petri original té un gran desavantatge, el qual ha estat decisiu, i és que ajuntava totes les condicions al principi i les tasques al final, el qual donava un resultat caòtic. Tampoc ajudava a que l'estructura de la xarxa de Petri fos una mica més complexa que la del BPMN descompost, a més de quedar resultats massa diferents que en d'altres casos millor descompostos, com poden ser els vincles.

L'eina que s'ha decidit utilitzar és el Petrify degut a ser una eina fàcil d'entendre i per a la qual és fàcil crear un autòmat, i passar aquest a un fitxer de text. Petrify només necessita saber les arestes que existeixen i els seus ordres de precedència i successió, si en tenen. A partir d'aquí, el resultat que retorna Petrify és fàcil de comprendre i fàcil de traduir a un BPMN simple, ja que mantindrà sempre la mateixa estructura de programa.

5.4.2 Patrons

Un altre punt del projecte que ha anat variant gairebé durant tot el projecte són els patrons, que s'han anat unificant o separant per casos, alhora que aquests també es canviaven, s'afegien o s'esborraven, com els antics patrons dels rígids.

Aquí també hi va haver tres alternatives principals: posar-los de manera uniforme i posar uns patrons que fossin molt variables, o bé incorporar-hi també "flags". La primera alternativa, l'explicada en el document que s'utilitza de base, era simplement que per a un cas concret agafés sempre la mateixa frase, independentment dels continguts que aquesta pogués tenir. La segona opció era crear més d'una frase per a cada tipus de situació, ja siguin portes, vincles, seqüències, etc., però que es tingués en compte segons els textos d'aquesta part. La tercera alternativa era agrupar diferents situacions similars en un sol patró amb restriccions i "flags" per tal que no es barregessin els resultats finals, tot i ésser més complex. En tots tres casos, però, les característiques comunes eren l'existència de marques i la característica de ser patrons multi línia.

Com volíem que el text sortint no fos massa repetitiu la primera opció va ser descartada quasi automàticament, ja que es volia aportar varietat a les frases. Quedaven, doncs la segona i la tercera opció. Es va provar amb totes dues i es va arribar a la conclusió de que no calia utilitzar íntegrament una de les dues opcions si no que es podia fer una combinació d'ambdues. Es va decidir agrupar aquelles situacions on l'únic que diferia era el nombre de textos a posar o bé la forma d'imprimir-se el text per a cada porta o seqüència. Tot i així, han quedat alguns casos, com per exemple el de salt, el de final directe o el de bucle que segueixen sense restriccions, ja que aquests són casos particulars.

Això s'ha fet així degut a que sempre que es pot simplificar un algorisme cal fer-ho. I també és ideal posar el menor nombre de casos possible. S'ha pensat que la alternativa combinada és millor, ja que redueix molts casos, té una complexitat menor per als casos que són senzills i té rapidesa i eficàcia degut a que les restriccions dels "flags" són aportades pels textos d'entrada. Un dels casos més flagrants d'aquestes fusions de patrons, per exemple, és la dels

antics patrons rígids, que han acabat desapareixent i integrant-se en la dels patrons dels vincles.

5.4.3 Millora de la comprensió

Aquí s'expliquen dues opcions per a millorar la comprensió del text. Les cometes i el suport semàntic.

En el cas de les cometes, la idea va sorgir per tal de millorar la comprensió del text resultat. Les alternatives eren posar-les o no posar-les. Finalment es va decidir posar-les per tal de millorar la comprensió de les parts més importants del text, fent-se això just davant i darrere d'aquestes, degut a que tampoc és gaire complicat d'implementar.

En el cas del suport semàntic hi havia tres alternatives: la primera era posar unes poques paraules de separació entre subjecte i predicat. La segona era posar subjecte i predicat seguits, sense cap paraula entre mig. Per últim, la opció d'incorporar-hi un suport semàntic extern que donés més concordança, com per exemple Freeling.

En el primer cas es va comprovar que el resultat quedava molt estrany i li feia perdre molt sentit a les frases, quedant, per exemple, El tècnic fa arregla l'ordinador", el qual fa que quedi una frase amb menys sentit. Donat aquesta causa, es va descartar. Finalment es va optar per posar subjecte i predicat seguits, ja que utilitzar un suport semàntic extern ens hagués portat massa temps d'adaptació, a més a més de no estar ja previst a l'inici del projecte. Es va decidir renunciar potser de forma lleugera a l'excel·lència per a poder tenir un text que es pugui entendre sense necessitat de tenir dos programes externs. A més a més, la connexió entre subjecte i predicat només milloraria en cas que en el model no concordessin.

Aquest últim punt s'ha fet per a donar-li més naturalitat a les frases que surten i, per últim cal destacar que en el projecte del qual n'agafo la base utilitzen un altre software, DSynT, que separa tota la frase sintàcticament. A nosaltres, però ens interessa més el subjecte i el predicat, ja que el rol ja ve separat

anteriorment, degut a que es posa en el BPMN l'usuari com a usuari d'una tasca o bé a la pool que conté la tasca.

5.4.4 Arbre RPST

Com en el cas dels rígids, la idea dels RPSTs va ser extreta del paper "Validation through Natural Language Generation"[7]. Tot i això hi havia dues alternatives per a fer-ho. Les dues alternatives tenien diverses coses en comú. En tots dos casos es mostra la jerarquia de l'arbre i es posen els fragments de la mateixa manera. Les diferències apareixen a l'hora de posar la informació als nodes. La primera alternativa era posar informació referent a la representació del BPMN a l'arbre, és a dir, quin esdeveniment, porta o tasca inicia el fragment i quin ho acaba, el qual conté informació sobre el tipus. La segona era posar directament els textos dels nodes sobre l'arbre, d'una manera similar als que s'implementen per als rígids.

La decisió va ser fer-ho de la primera manera. Hi ha però, una causa principal. La possible existència de fluxos de missatge i de tasques de recepció. Si s'hagués fet de la segona manera es podria haver perdut certa informació, ja que per a les tasques de recepció sense text no s'hauria dit res. En canvi, en l'alternativa escollida primer s'haurien creat tots els RPSTs dels processos. A partir d'aquí si una tasca de recepció no té text sempre es podrà buscar per què existeix aquesta tasca a través d'aquest flux de missatge. Aquesta manera de fer-ho també guarda el tipus d'element que envolta un fragment, fent-lo més ràpid per a trobar un dels patrons i donant-hi també més informació. En aquest cas s'ha buscat l'eficàcia per davant de l'eficiència. Tot i així, s'ha de dir que l'altra alternativa hagués estat bona per a models amb un sol procés.

Un altre problema afegit de la segona alternativa era que guardava informació no vital, com finals d'una seqüència, els quals es poden obviar de la manera en la que finalment s'ha decidit fer.

5.4.5 Lectura dels fitxers

Per a la lectura del fitxer del model hi havia dues opcions. Una d'elles era utilitzar JBPT i agafar models que tinguessin aquest tipus de tecnologia o bé agafar fitxers en XML que tinguessin Activiti. Es va optar per aquesta última opció, degut a que en el cas de JBPT es trigava més en crear models. A més a més, alhora de llegir aquests models donava errors a l'hora de crear-los.

Un cop decidit de quina forma llegíem els fitxers calia pensar en com passar aquestes dades a quelcom amb el que es pogués treballar. La opció era deixar una llista d'elements o bé separar-ho per tipus. Per tal de millorar l'eficiència la opció triada va ser la segona, ja que amb tots els elements separats per tipus era més ràpid buscar un element d'un tipus que no pas a l'inrevés. La contrapartida d'utilitzar més variables queda compensada, ja que aquestes ocuparan el mateix.

En quan a la resta dels fitxers d'entrada, per simplicitat, la única idea plantejada va ser fitxers de text pla, que després dins del programa es mirarien caràcter a caràcter i transformant-se aquestes paraules a dades.

5.4.6 Creació del text.

Per a crear el text es plantejaven dues alternatives: Implementar-ho de forma iterativa o de forma recursiva. El primer cas suposava que per a cada node s'agafava la informació necessària, el patró que s'utilitzava, s'afegia al text provisional i es mirava el següent node, utilitzant una sola funció. El segon suposava agafar la informació del node propi, crida els fills, els quals facin el mateix fins a que tots tinguin textos, i fusionar-ho amb el patró que sortís, tot i que necessitava dues funcions i més complexitat a l'hora d'implementar.

En aquest cas guanyava clarament la segona opció, no només per qüestió d'eficiència, sinó també per eficàcia, ja que es millor fer el muntatge un cop es té tota la resta fet que no pas abans de saber res. La complexitat és un element molt menys important que els altres dos quan es tracta de crear frases en llenguatge natural.

5.4.7 Paràgrafs

Sobre els paràgrafs hi havia dues coses a plantejar-se. La longitud màxima d'un paràgraf i la forma de presentar els paràgrafs del text.

En la presentació dels paràgrafs hi havia tres plantejaments possibles: Utilitzar-ho de forma molt esquemàtica, amb tot el que es pogués treure en format llista fer-ho així, treure en forma de paràgrafs o bé fer una combinació dels dos. Finalment, es va creure que això podia anar al gust del consumidor, pel qual es van acabar implementant totes tres opcions, podent-se escollir una de les tres en el fitxer d'entrada de paràmetres.

L'altre tema a debatre era la longitud dels paràgrafs. Un paràgraf no té per què tenir una longitud mínima, però una longitud massa gran pot cansar el lector. Les dues primeres opcions plantejades eren que no fossin de més de 100 paraules, el recomanat pels experts, o que no passessin de 75, que és la quantitat utilitzada pel document en el qual ens basem. La solució presa va ser una d'intermèdia. Degut a que la majoria de frases no superen les 20 paraules, es va pensar en separar paràgrafs quan aquests ja superin les 75 paraules. Així, si a un paràgraf de 74 paraules se li afegeix una frase de 20, quedaria en 94, per sota del recomanat.

5.4.8 Textos de sortida

En quan al fitxer de sortida s'havia plantejat fer-ho en XML o en text pla. Finalment, crèiem que el passar a llenguatge XML després d'haver fet tot un text normal i corrent no ens aportava res de nou, pel que s'ha optat per fer-ho només en text pla.

5.5 Recursos i tecnologia

Els recursos software que s'han utilitzat en aquest projecte són el sistema operatiu Linux Ubuntu 14.10, per que és un sistema operatiu fiable i que he utilitzat durant molt temps, l'editor Eclipse en versió 4.7, degut a que era el més compatible amb el sistema operatiu i oferia la possibilitat de provar els resultats,

la llibreria Activiti, que permetia treballar amb BPMNs i els llenguatges de programació Java i XML, en el primer cas per la seva familiaritat i ús per part de molts usuaris i el segon per necessitats de carregar els models, també per la popularitat. Com a element hardware un ordinador Acer E5-571, que és portàtil per a poder ensenyar la feina feta als directors. Tot això s'utilitzarà en totes les fases del projecte.

Al mes d'octubre es va decidir utilitzar un altre software gratuït, Camunda. Aquest software permet crear models BPMN amb més facilitat, estalviant temps en el procés, degut a que no cal escriure codi. El 15 de desembre es va decidir utilitzar una nova eina gratuïta, el Petrify. Aquesta eina permet crear uns RPSTs simplificats a partir d'un autòmat per als models complexos. El mateix dia es va plantejar una altra, per tal de fer una hipotètica millora de la cohesió, Freeling, que és un analitzador semàntic de software lliure per tal de separar nom i verb dins dels diferents elements, sobre el qual es va fer un seminari el dia 27 de novembre de 2015. Finalment, però, es va decidir no utilitzar-ho.

6. Planificació

6.1 Descripció de les tasques

El projecte es dividia en 6 fases. Aquestes fases eren la fita inicial, el coneixement de l'entorn, la creació del RPST i de la xarxa de Petri, la creació del text, la creació del fitxer de sortida i les proves i finalment la fita final. Els detalls es trobaran més endavant dins del subapartats següents.

El projecte s'havia de finalitzar inicialment en un temps de 15 setmanes, que eren de 5 dies cadascuna, amb 8 hores cada dia. Això feia un total de 600 hores. Els càlculs estimats eren com a data d'inici el dia 7 de setembre de 2015 i el final el 23 de desembre del mateix any.

En realitat, però els plans han estat diferents. El projecte es va començar el 2 de juliol de 2015, i s'ha acabat finalment el 18 de gener del 2016. Es van fer vacances durant el mes d'agost. Al juliol van ser 4 setmanes i dos dies, de feina, unes 5 hores al dia de dilluns a divendres, un total de 110 hores. A partir del 31 d'agost es van fer 30 hores setmanals durant 16 setmanes, fins al 18 de desembre, totalitzant 480 hores.

A partir del 21 de desembre es tenia pensat a mig projecte treballar de dilluns a divendres excloent vigílies i festius, 8 hores al dia, essent 112 hores més. Al final, però, es van fer 8 hores els dies 23 i del 27 al 30 de desembre, així com el 9 i del 11 al 18 de gener. Se'n van fer 5 el 10 de gener. Els dies 21, 22, 24 i 31 de desembre i del 2 al 8 de gener, excepte el dia 6, se'n van fer 4 hores. En total el treball ha durat 751 hores de feina, 151 hores més de la planificació inicial prevista i 49 més que a la intermèdia. Això es deu a que es va decidir començar el projecte abans per tal de tenir més temps per a fer un projecte més complet i poder introduir-hi noves millores en el primer cas, a més a més d'haver tingut complicacions en alguns punts del projecte que no s'esperaven respecte la segona.

6.1.1 Fita inicial

La fita inicial va incloure dues parts: la cerca d'informació relacionada amb el projecte i la realització del curs de GEP. Aquesta fase estava estimada en unes 90 hores en total entre les dues parts, essent finalment d'unes 100. La informació relacionada amb el projecte significava trobar les diferents fonts d'informació amb les quals poder tirar endavant aquest projecte per a poder comparar les diferents opcions que hi ha, a més a més de buscar la informació dels BPMNs, els RPSTs i les xarxes de Petri. Això ha portat 20 hores de feina, en front de les 10 previstes, degut a que es va haver de buscar més informació per a tenir més fonts variades.

Posteriorment en el curs de Gestió de Projectes teníem 6 parts diferenciades, i es van complir segons el previst:

- La definició de l'Abast i del Context, el qual ens va dur unes 25 hores.
- La planificació inicial del projecte, unes 8 hores.
- Els càlculs de Gestió Econòmica i sostenibilitat, de 9 hores.
- Una petita presentació oral prèvia, que va portar 6 hores.
- L'entrega de la fita inicial de gestió de projectes i la presentació oral associada a ella, PowerPoint inclòs, acumulant un total de 18 hores.
- La part d'objectius de l'especialitat de computació, que va comportar les 14 hores restants.

El total del curs va ocupar les 80 hores previstes, ja que va acabar sent tal com es preveia, les quals es van portar a terme entre el 14 de setembre i el 26 d'octubre, simultàniament amb el projecte en sí mateix. Aquesta simultaneïtat es deu a que s'havia de fer una feina setmanal tant del projecte com del curs, ja que en aquest últim s'havien de fer les entregues en una data determinada.

6.1.2 Coneixement de l'entorn

Per a començar el projecte calia primer conèixer el sistema Operatiu (Ubuntu), l'editor (Eclipse), els llenguatges (Java i XML) i les llibreries (Activiti, entre d'altres). Això era estrictament necessari fer-ho al principi del tot abans de

començar el projecte en sí, essent aquesta la tasca inicial a fer. El temps era de 40 hores, amb 10 hores per a cadascuna de les 4 parts inicialment previstes de l'entorn, ja que tant Freeling com Petrify com Camunda es van incorporar més tard. Degut al coneixement previ d'Eclipse, Ubuntu i Java només en va portar 12, havent-ho après ja el matí del 4 de juliol. El Camunda no ha necessitat gaire feina, i el Petrify només ha portat 4 hores. El Freeling no ha calgut aprendre-ho, ja que finalment no s'ha utilitzat. El total, doncs són 16 hores, 24 menys de les previstes inicialment.

6.1.3 Creació del RPST i de la xarxa de Petri

En aquest apartat s'incloïen diverses tasques seguides una darrera de l'altra, un cop ja es conegués bé l'entorn en el qual es treballava. La primera va ser crear uns quants fitxers de treball amb els quals es pogués anar provant els diferents afegits del projecte. Va portar les 5 hores esperades.

Després es va crear una entrada/sortida bàsica, la qual ens permetia agafar el fitxer d'entrada i també imprimir per la pantalla allò que es necessités. Això va ser més difícil del previst, trigant finalment 10 hores, més de les 5 previstes.

Posteriorment, el que ens calia fer era crear el RPST a partir del BPMN que teníem d'entrada. Per a això calia crear estructures de dades complexes que continguessin els diferents nodes de l'arbre. Va ser una tasca molt més senzilla, ja que vaig entendre de seguida què calia agafar, trigant en la primera iteració 24 hores, i 15 més en cadascuna de les dues ampliacions de tipus d'elements llegits que s'han fet. En total s'ha trigat 54 hores, que són 6 menys de les 60 previstes. La primera iteració va estar llesta a mitjans de juliol. La segona es va completar al mes d'octubre, i la tercera a principis de desembre.

Quan la primera iteració dels RPST va estar acabada, calia fer la xarxa de Petri. La versió original va ser costosa de fer, però tot i així va costar 30 hores, finalitzant-se la primera setmana de setembre. Aquest, però, era un sistema millorable, i la posterior millora d'eficiència, pel qual se li demana ajut a un programa extern, Petrify, el qual dona un BPMN més llegible i eficient, ens va

portar 36 hores, acabant-se a finals de desembre. Finalment, tota la xarxa de Petri va trigar unes 66 hores, 16 més de les 50 previstes inicialment.

Tot i així, va haver certs problemes inesperats, ja que no es detectava bé quan hi havia rígids que acabaven amb esdeveniments finals. Va caldre modificar la creació dels RPSTs per a fer-ho similar als rígids, el qual va portar unes 34 hores més, acabant-se els primers dies de gener.

En total, aquesta part ha portat 169 hores. Tenia una previsió inicial de 120 hores, i 15 possibles de retard, pel que s'han fet 34 més que el possible retard previst inicialment, justament el temps dels rígids amb finals.

6.1.4 Creació del text

Aquesta fase es divideix en 4 tasques, que s'han anat fent un cop va acabar-se de construir la xarxa de Petri antiga. La primera tasca va ser la creació de les plantilles de treball en les quals hi ha les frases que serveixen per a crear textos. En les diferents iteracions de millora s'han afegit casos de multiplicitat particular (casos de 2, 3 i 4), patrons multi línia i "flags". En totes les iteracions s'ha estat unes 9 hores, i es van fer entre principis d'octubre i finals de novembre, fent últimes millores a finals de desembre. L'afegit del patró català va portar tres hores més, un total de 12, 7 més que les 5 inicials.

El següent procés ha estat, de llarg, el més complex i llarg de tots. Relacionar la plantilla d'entrada amb el RPST i la xarxa de Petri. S'han de fer moltes operacions i relacions entre les diferents parts, buscar el node del RPST, buscar la seva informació, agafar un patró d'acord amb el tipus de node RPST i fusionar-ho tot. Per a aquest procés s'esperaven 120 hores de feina, però finalment s'han tingut moltes complicacions i han calgut moltes més iteracions de les esperades, per tal de millorar els resultats. Degut a les complicacions d'aquesta part i a les diferents iteracions, la última de les quals va ser a inicis de gener, ha estat de 245, més del doble del previst.

La tercera tasca és la separació del text en paràgrafs. Aquesta tasca depenia en part de la generació de text i s'han tingut al final més problemes dels esperats. S'ha trigat, finalment, 50 hores, el doble de les 25 hores previstes,

degut a que hi havia moltes més coses a quadrar que les esperades, acabant-ho a mitjans de gener.

L'última tasca d'aquesta fase tornava a ser una de més complexa: la cohesió del text. Això volia dir que tot encaixés millor i crear un text el màxim comprensible possible. Aquesta part es va acabar el 14 de gener, ja que és on s'incorporaven els "flags", que estava començat des de mitjans de desembre, la qual em va portar 24 hores. La finalització d'aquestes millores n'ha portat 45, totalitzant-ne, doncs 69 hores, 1 menys de les 70 previstes inicialment.

Amb això feia que la fase de creació del text fos la més llarga, un total de 376 hores finals entre totes les tasques, per contra de les 220 previstes inicialment. Això es deu principalment a l'increment d'hores que ha necessitat la relació de RPST i la xarxa de Petri amb els patrons, i també la creació de paràgrafs, tractant-se també de la part més important del projecte, i és la que més s'ha desviat de totes elles.

6.1.5 Creació del fitxer de sortida i proves

Aquesta fase s'ha acabat aquests últims dies. En aquesta fase s'hi encabirien sis tasques diferents a fer, ordenades per l'ordre cronològic que es tenia pensat originalment un cop es tingués un text amb certa cohesió. Finalment, s'han fet un parell de tasques abans d'hora per petició expressa dels directors. La fase es va reduir primer de 70 a 60, i finalment a 48, i contindria aquestes fases:

- La creació del fitxer de sortida en text pla al nou format. Degut a que s'ha decidit no passar a XML, el fitxer de sortida en text pla ha estat molt fàcil i va portar 1 hora en comptes de les 20 inicials de tots dos.
- S'havia pensat en fer una entrada ampliada un cop acabat el fitxer de sortida, però es va implementar finalment a mitjans de novembre. Aquest procés va portar 5 hores amb totes les iteracions, degut a la facilitat d'ella. A l'últim moment, però, es va decidir fer una petita millora que permetia agafar els paràmetres de l'exterior, la qual només va portar una hora més, un total de 6, per les 10 previstes.

- La sortida per pantalla bàsica també s'ha millorat després de la primera actualització d'entrada, fent una sortida més visual de l'arbre. Això ha estat molt més fàcil que l'esperat i s'ha trigat 5 hores en comptes de 10.
- Un cop acabades totes les fases anteriors i els punts anteriors d'aquesta fase, es van crear a principis de gener els fitxers de proves els quals s'han utilitzat per a fer l'estudi de la fita final. Finalment ha portat 24 hores, 14 més de les previstes.
- Després s'han executat les proves, però només s'ha trigat 2 hores, per les 5 previstes inicialment.
- Finalment s'han analitzat els resultats de les proves. Això ens ha portat 10 hores en comptes de les 15 hores inicials.

6.1.6 Fita final

En aquesta fase, que s'està concretant actualment, es va tancar el projecte el dia 15, amb unes 8 hores de duració, donant-lo per acabat. Posteriorment s'ha redactat el que quedava de la memòria per a preparar l'entrega final del projecte i s'ha corregit i actualitzat tot allò que fos diferent respecte a les fites anteriors. Això ha acabat portant 24 hores, en front de les 35 inicials. Finalment la preparació per a la presentació oral final i la presentació en sí mateixa, que ens portarà 10 hores més. La presentació es farà el dia 28 de gener de 2016 a les 10 del matí. El total d'hores s'ha reduït de les 60 de previsions anteriors a 42.

6.1.7 Comprovació i Corregit d'errors

Això no es pot considerar com una fase en sí mateixa, sinó més bé com a quelcom necessari en cadascuna de les tasques de programació del projecte. Cada cop que es treballava en una fase s'havia de comprovar de forma molt regular el correcte funcionament d'aquesta.

6.1.8 Parts eliminades

Finalment, com es va preveure a l'informe de seguiment, s'ha obviat el pas a XML, i només s'ha fet el fitxer de sortida en text pla, ja que no era vital treure-ho en dos formats diferents i amb un n'hi havia prou. A més a més, el traduir a un altre llenguatge s'allunyava massa del propòsit principal d'aquest projecte. Com la creació d'un fitxer de text pla era fàcil amb una hora n'hi havia prou, pel que se'n van descartar 19. Finalment es van fer 40 proves, però igualment ens va portar la meitat de temps. Finalment, també es va descartar posar l'idioma castellà per tal de reduir hores. Tot i això, es va fer un canvi d'última hora per a que es pogués crear un patró en espanyol sense tenir problemes a la lectura, estalviant 10 hores més. Inicialment s'havia descartat les millores d'entrada i sortida en cas d'anar malament de temps, però finalment es va creure que milloraria substancialment els resultats, pel que finalment es va decidir implementar-ho.

6.1.9 Afectacions del projecte

Les afectacions són dues. Per una banda l'ampliació de temps, que potser ha estat conseqüència de voler augmentar el nivell d'aquest projecte. D'una altra banda que finalment només s'imprimeixi en text pla descartant XML, considerat massa allunyat del propòsit inicial d'aquest projecte, i la renúncia al patró castellà, una de les conseqüències del retard patit a l'hora d'ajuntar patró amb informació, RPST i la xarxa de Petri. Aquest últim també es deu a l'inconvenient dels finals que acaben en rígids. Pel costat contrari les millores visuals s'han fet abans del previst per a saber l'estructura real de l'arbre i també la millora de l'entrada, per a que el programa s'adaptés als models i no a l'inrevés. Això podia haver arribat a descartar-se.

A l'informe de seguiment també s'hi van posar unes hores extres de marge, mentre que abans estava tot molt més acotat al temps exacte. Tot això es feia per a que els objectius quedessin el menys trastocats possibles, ja que allò important era que tragués un bon text en un format de text pla que tothom

entengués millor. La resta d'objectius, els més importants, però, han quedat intactes.

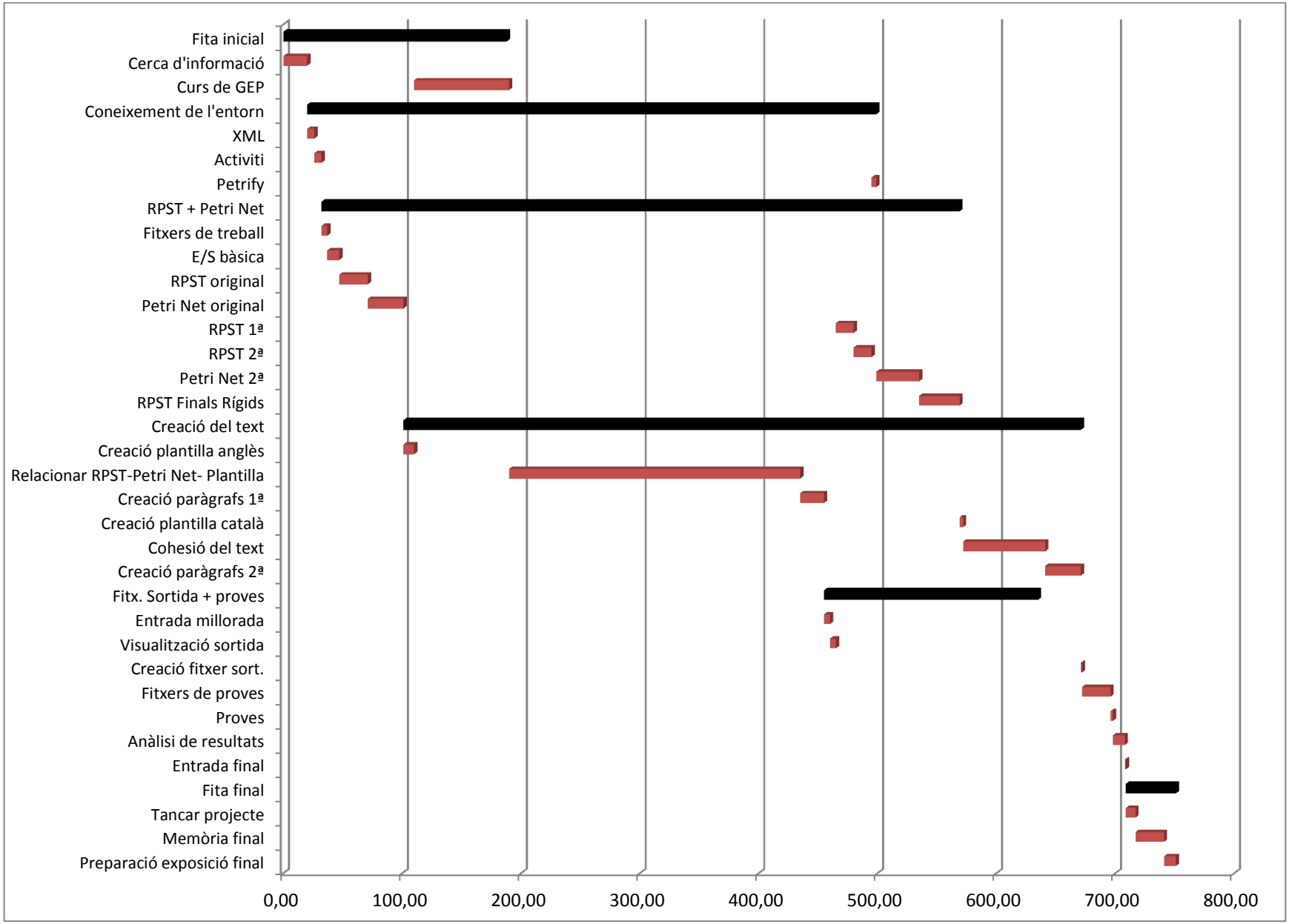
6.1.10 Resum de les hores per tasca

Fase	Hores finals	Hores previsió intermèdia	Hores previsió original
Fita inicial	100	100	90
Coneixement entorn	16	12	40
RPST i xarxa de Petri	169	119	120
Creació text	376	337	240
Fitxer sortida i proves	48	60	70
Fita final	42	60	60
Total	751	688 (702)	600

6.2 Anàlisi del diagrama de Gantt

El que es pot extreure com a conclusió del següent diagrama de Gantt és que la planificació final demostra la màxima de la programació, que els processos no són seqüencials, si no més bé iteratius, ja que es comença amb les parts més senzilles, fent-ne l'esquelet, i a partir d'aquí anar ampliant i provant fins a solucionar tots els errors i arribar a un punt on s'hagin fet totes les ampliacions necessàries. Les barres negres simbolitzen la fase completa, tot i que no s'implementés durant tot el temps, sinó durant part d'ell. Tot i això parts del fitxer de proves i les proves mateixes s'han anat fent durant tot el projecte, i la memòria final també es redactava alhora que els informes inicial i de seguiment, ja que aprofita algunes parts d'ells.

6.3 Diagrama de Gantt de tasques



7. Implementació i problemes

Durant el projecte m'he anat trobant amb diverses dificultats que han fet un endarreriment a l'hora d'acabar una part del projecte. Aquestes s'expliquen en aquest apartat, juntament amb com s'han implementat les diferents parts. Estan separades per blocs: lectura del fitxer, creació del RPST, creació del text i creació del fitxer de sortida.

7.1 Lectura d'un fitxer d'entrada

El primer problema detectat va ser a l'hora de llegir un fitxer d'entrada. El primer fitxer d'entrada que vaig utilitzar no estava adaptat a l'Activiti. Això feia que quan intentava llegir-lo es llegia el fitxer, però en canvi no llegia cap procés. Degut a això no es llegien fluxos, portes, activitats, etc. I la sortida no retornava res. Finalment es van haver de programar manualment els primers models de proves per a que fossin compatibles amb Activiti, un problema que no s'ha trobat més durant la resta del projecte. Un exemple d'un model d'entrada es pot trobar en l'annex 4.

Finalment es fa amb un ordre establert. Primer per a cada procés es posen i re anomenen els diferents elements que hi ha, per tal que després sigui més fàcil trobar-los per tipus. Després es fa el mateix els fluxos de seqüència, actualitzant-hi els noms d'entrada i sortida. El mateix procés es fa posteriorment per a tots els subprocessos que es troben. Finalment es processen els fluxos de missatge.

7.2 Creació dels RPSTs

En aquesta secció s'indiquen els diferents passos de creació d'un arbre RPST a partir del model BPMN. Per tal de fer més completa aquesta explicació s'il·lustrarà amb un exemple que ajudi a clarificar el que succeeix. Això és només un exemple, però no té un text original ja que ha estat simplement un model BPMN de proves. Tracta sobre un cercle de científics que vol incorporar científics joves al seu cercle.

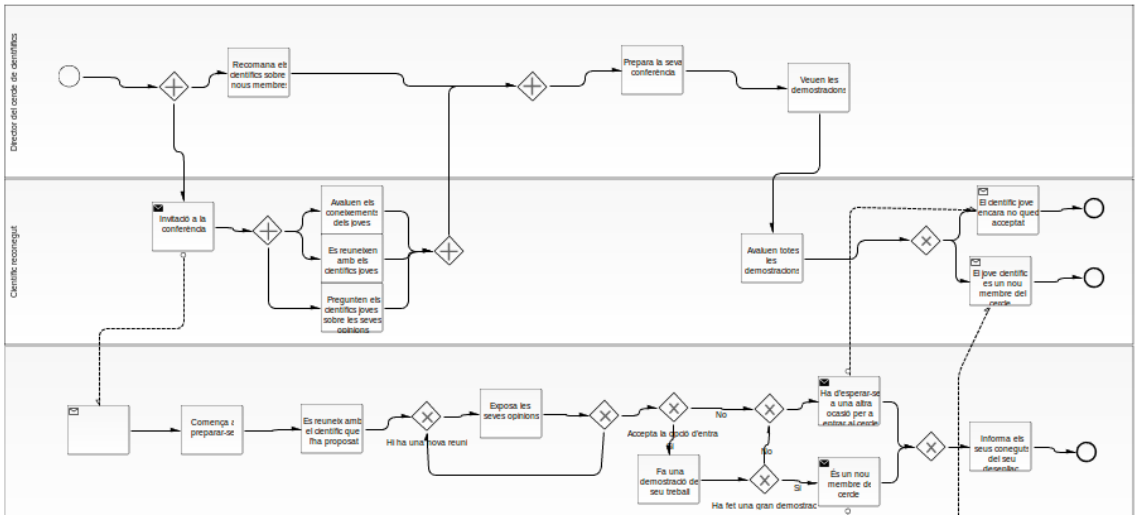


Figura 8. El model BPMN sencer de l'exemple

7.2.1 RPST bàsic

Primer calia crear seqüències en el BPMN. Es feia d'una manera iterativa, començant per la sortida, buscant-ne els fluxos que en sortissin del node actual. Si al final hi havia un esdeveniment final posava el flux i acabava. Si al final del flux ens trobàvem una tasca, s'afegia el flux que les connectava i es continuava a partir d'aquesta. Si hi havia una porta es creava el vincle o el rígid i es continuava des de la porta de sortida d'aquest. Un cas de seqüències són les dues primeres fletxes de l'usuari de baix, que representa el científic jove.

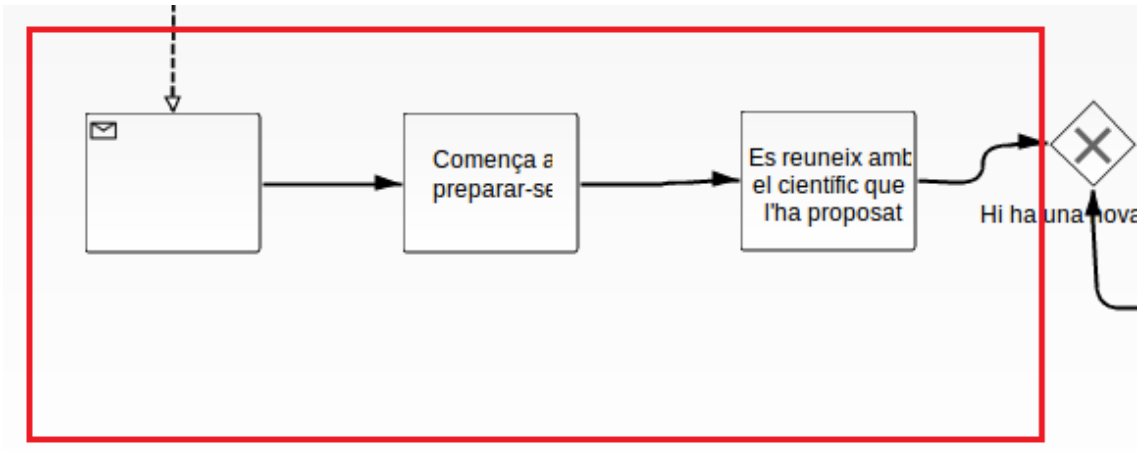


Figura 9. Exemple d'una seqüència

7.2.2 Distinció de vincles i rígids

Els primers problemes de la creació del RPST van ser que les portes creaven indistintament vincles i rígids, i no trobaven pas la diferència entre uns i els altres. Això causava que el programa no posés totes les portes i no busqués totes les alternatives en el cas dels rígids, o bé es quedava en un bucle infinit. El concepte pel qual passava és que per a tots dos tipus de blocs buscava la següent porta en ésser trobada.

Per tal de solucionar-ho es va crear una funció que separés segons el tipus. A partir dels fluxos de sortida de la porta que es trobava, cadascun amb un número diferent, anant a parar des d'aquí a altres nodes. Si era una activitat seguia buscant, però si era una porta mirava les entrades que tenia. Se li posava el nombre d'entrades menys 1, i si li quedaven al menys una per investigar bloquejava la cerca de la porta.

A mesura que es trobaven nodes s'eliminaven de la llista de cerca i es posaven a la de ja trobats, i se li posava el número del flux d'entrada si no en tenia. En cas de les portes, podia ser que tingués el mateix número en totes les entrades (vincle dins de vincle o bé porta inicial), o bé un número diferent en alguna. Si aquest era el cas se li posava un número no assignat a cap sortida. En aquest cas si passava només a la porta de sortida no es tenia en compte i és un vincle, en el qual es crearan els nodes fill. En canvi, si passava en més d'una porta o en una d'intermèdia és un rígid, pel que es crea un rígid amb tots els fluxos interns com a fills.

En la figura 10 veiem els casos de vincle, pintat de color vermell, vincle dins de vincle amb color blau i rígid, de color verd. Com es pot veure en el color blau, la porta paral·lela P2 té tres branques de sortida, a la qual li ha assignat un número diferent, de l'1 al 3. Veiem que segueixen essent independents fins a la porta de sortida, P3. Explicarem amb una mica més de detall, però, el de color vermell. En aquest cas de la porta paral·lela P1 surten dos fluxos, un cap a una tasca normal, T2 i l'altre cap a una d'enviament, T1. Aquestes dues queden a la cua de cerca sense estar congelades. Primer es mira, per exemple, la tasca

d'enviament. Aquesta té una sortida, que és la que va a la porta que obre la part blava, P2. Com només té una entrada s'afegeix a la cua sense congelar.

La cosa canvia quan toca mirar T2. El flux de sortida d'aquesta tasca el du a la porta de tancament del vincle, P4. Aquesta porta paral·lela té dues entrades, pel que quedarà congelat amb el valor de 1. Mentre aquesta porta estigui congelada no es podrà mirar després d'ella. Mentrestant es va mirant el que passa per la part blava. Passarà amb la porta P3 el mateix que amb P4, però a mesura que es passa per T3, T4 i T5 començarà amb un valor de congelació de 2, baixarà a 1, passant després a descongelar-se. A la cua ara només es troben les dues portes de tancament. P3 és la única que està descongelada, pel qual es mira el flux de sortida, que va a parar a la porta P4. Com P4 és l'últim element que queda de la cua es considerarà que és el tancament del vincle. De nou veiem com les branques que surten d'inici són independents des del principi fins al final.

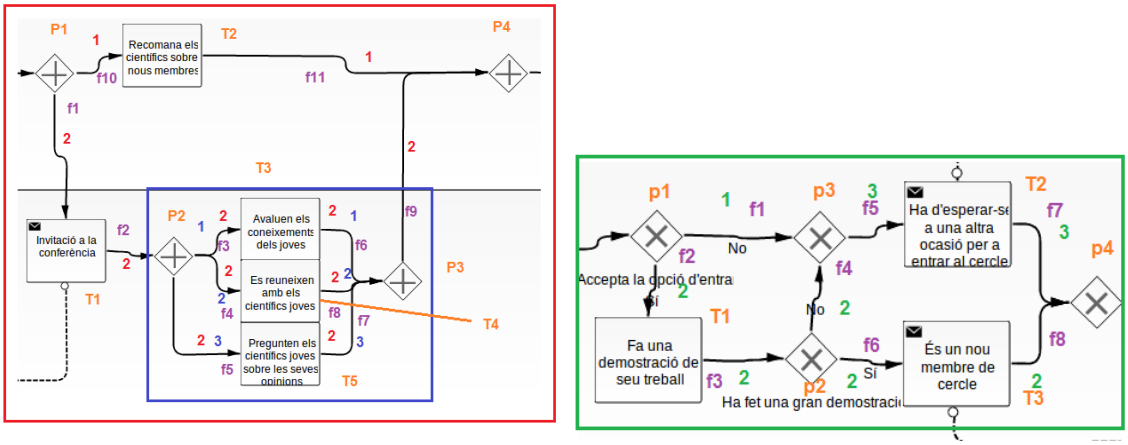


Figura 10. Dos vincles, un dins de l'altre i un rígid

Veiem, d'altra banda, el que passa en el cas verd. De la porta exclusiva P1 surten dos fluxos, un que va cap a una tasca, T1 al qual se li assigna la branca 2 i un altre cap a una altra porta exclusiva, P3, que tindrà la branca 1. T1 estarà descongelat per ser tasca, però com P3 té una altra entrada es quedarà congelat a 1. La tasca T1 passarà la branca 2 a una porta d'entrada, P2, que també estarà descongelada. Degut a que P3 encara ho està, es mira a P2 i li assigna la branca 2 als dos fluxos de sortida que té, cap a T3 i cap a P3. Aquest flux també descongela P3, però veiem una cosa: els dos fluxos que entren són de les branques 1 i 2. El problema és que no és l'únic element a la

cua, ja que també hi és T3. Això significa que s'ha de qualificar això com a rígid. A la figura 11 es pot veure com queden aquestes dues parts en el RPST, amb els valors en lila essent el nom que els hi correspondria als fluxos. El vincle vermell seria B1, i el blau B2, on el rígid verd és R1.

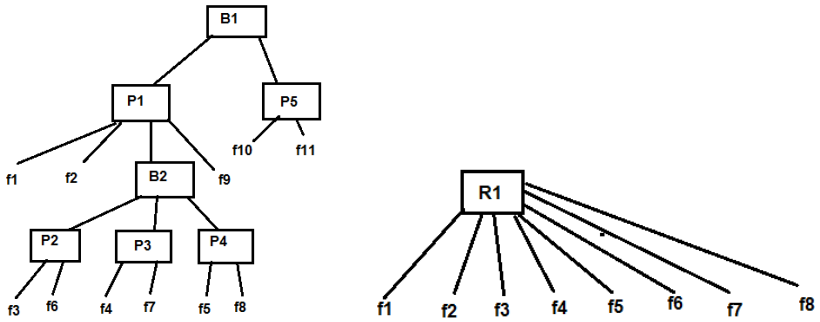


Figura 11. El resultat en el RPST dels dos vincles (dreta) i del rígid (esquerre)

7.2.3 Bucles

El següent problema era la detecció de bucles dins dels models. Quan s'arribava al final d'un bucle tenia dues sortides, i una d'elles portava a l'inici del bucle. Si això passava el programa mateix es quedava en un bucle infinit que creava un munt de nodes del RPST iguals.

Per a solucionar-ho es va decidir mirar els fluxos d'entrada de la porta que es trobava primera, és a dir, la d'entrada. Si se'n trobava 1 no era bucle, però si n'hi havia dos i no era porta de sortida calia mirar si venia d'una altra porta i si a partir d'aquí s'arriba de l'entrada a la sortida volia dir que era un bucle. En aquest cas es feia una funció especial, on es creava un vincle que tenia com a fills un polígon, que era tot el contingut entre les dues portes i un flux, el de tornada a l'inici del bucle.

Com es pot observar a la figura 12, La porta d'entrada PE té dos fluxos d'entrada i només un de sortida. Anant a través de f1 s'arriba a una tasca, de la qual surt F2, que du a la porta de sortida del bucle, PS, la qual s'ha detectat com a anterior a PE pel flux que les connecta. Això denota que és un bucle. Aquest flux, f3, serà el flux que quedarà apartat de la resta a l'hora de fer el RPST.

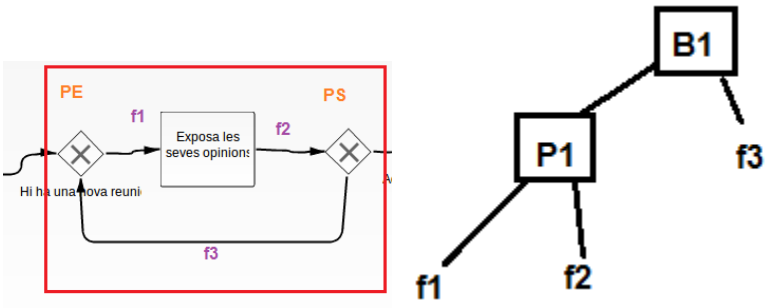


Figura 12. Bucle i RPST resultant

7.2.4 Vincles i bucles barrejats

La solució aportada a l'apartat anterior però, es va descobrir més tard que no servia per a tots els bucles, si no que només per a aquells que només tenien fragments trivials (fluxos), com el de la figura 12. Degut a això un dels exemples que es van utilitzar al principi es quedava en un bucle infinit, o bé fallava. El problema, però, no era en com es buscaven els bucles, si no la seva implementació interior. Es van passar a tractar com a un interior d'un vincle, però en comptes de ser un vincle era un polígon que podia tenir alhora altres vincles.

Això, però, encara era incomplet, ja que els bucles dins de vincles també fallaven, perquè no se'ls tenia en compte. Es van afegir un parell de restriccions que van ajudar a que els bucles dins de vincles o els bucles dins de bucles funcionessin de manera correcta.

7.2.5 Portes amb més de 2 entrades o sortides

Inicialment només es tenien en compte dues sortides o entrades com a màxim per a una porta, com el cas de P1 i P4 a la imatge de la dreta de la figura 10, però canviant lleugerament la implementació per a permetre més de 2 fluxos per porta es va solucionar ràpid, com el cas de P2 i P3 de la mateixa figura.

7.2.6 Més d'un esdeveniment final

Un problema que es va trobar a mig projecte és que no es distingia bé el que era una porta exclusiva que acabava en una altra porta exclusiva de la que acabava en dos finals diferents. Això feia que el programa només anés per una branca o bé directament fallava. Per a solucionar això es va adoptar una solució en cas que hi hagués més d'un final per a un procés o subprocés.

En aquests casos es busca la porta en la qual es divideixen els finals, quedant-ne un llistat. Si totes es separen en la mateixa porta es fa comú fins a aquesta porta, i després per a cada final una branca. Hi ha casos amb més de 2 finals on són dues o més les portes que separen els finals, havent de fer blocs comuns separant les branques una a una. La detecció de les portes de separació es fa d'una forma similar a com es fa en vincles, però a l'inrevés, des del final cap a endavant fins que tots els elements que quedin a la cua estiguin congelats.

Això, però, no sempre servia si hi havia més d'una porta de separació i hi havia més de dos finals. A partir d'aquí el que es va decidir és buscar tots els punts de separació, ja que podia haver-hi més d'un que separés més de dos finals. Es va decidir l'algorisme, tal que un cop trobats els separadors principals es busca si n'hi ha alguna porta de separació abans. A partir dels separadors coneguts es segueix anant enrere, fins que es trobi una d'aquestes opcions. Que la porta ja hagi estat declarada de separació, que s'arribi al principi o que es trobi una nova porta de separació. Es va fent per a totes les portes de separació fins a arribar al principi de tot per a totes elles. Amb això no només es poden buscar les últimes separacions, si no que també es podrà fer per a casos on primer es separen dues i dues i després ho fan les parelles. A partir de totes aquestes portes es crea un vincle amb diferents polígons, un per a cada final o separació trobats

Posteriorment va sorgir gairebé al final del projecte un altre problema. Els rígids que acaben en finals diferents. Per a això va caler part de l'algorisme de creació dels finals alternatius, ajuntat amb els de la creació d'autòmats. Aquí, però, simplement es feia d'una forma que en comptes de crear un autòmat el

que es crea és un arbre RPST, considerant el que abans era un rígid en un vincle, el qual fa que sigui més fàcil treballar amb aquest cas concret. De fet, es podia reconvertir també aprofitant l'algorisme de més de 2 portes de separació. Els resultats han quedat similars a si hagués estat un rígid que tenia els finals inclosos. Això s'ha hagut de fer així per que els rígids són exclusius per a les parts internes i no per a dividir finals.

Veiem a la figura 13 que per a E1 si es retrocedeix s'arriba a T1, que en ser tasca no queda congelada. De T1 s'arriba a G1, que té un flux més de sortida, pel que es queda amb 1 de congelació. En ser l'únic element i estar congelat es troba que G1 és una porta de separació de finals. Es pot comprovar fàcilment que passa el mateix per a E2, ja que són finals simètrics. Es crea un vincle, pel qual es creen dos fills. A aquests fills se'ls tracta com si fossin processos per separat.

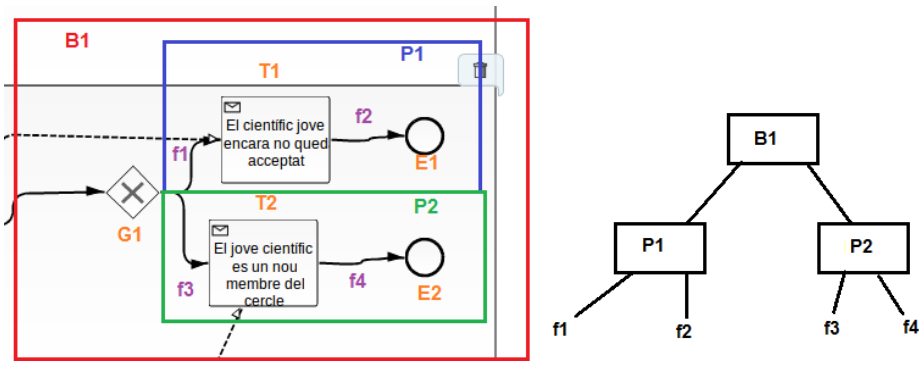


Figura 13. Un exemple de dos finals amb el RPST resultant

7.2.7 Creació de subprocessos

Els subprocessos bàsics han significat algun problema lleuger. La solució però no ha estat difícil. Quan se'n troba un es crea una funció especial que crea un vincle, que té com a entrada i sortida el mateix subprocés i un únic fill polígon que té la mateixa entrada-sortida. A partir d'aquí es tracta com un altre procés més. Quan torna de la funció, però, es busca a partir del final del subprocés com si es tractés d'un vincle normal, mirant quin flux en surt.

7.2.8 Tasques de recepció com a inici

Quan hi ha més d'un procés és possible que més d'un no comenci per un esdeveniment inicial, sinó que esperi un missatge d'un altre usuari. Ha calgut adaptar els RPSTs per si no troben un inici dins del procés (excloent els que es trobin en subprocessos), busquin tasques de recepció com a inici. Aquelles que no tinguin flux de seqüència d'entrada són les que comencen un procés. Un exemple és el que es mostra en la figura 14. La tasca marcada en vermell té un sobre blanc, pel que és de recepció. La fletxa que li arriba, però, no és sencera si no que és intermitent, essent aquest un flux de missatge. Observi's que no hi ha cap flux de seqüència que entri a aquesta tasca, pel que se la tractarà com a inici del procés.

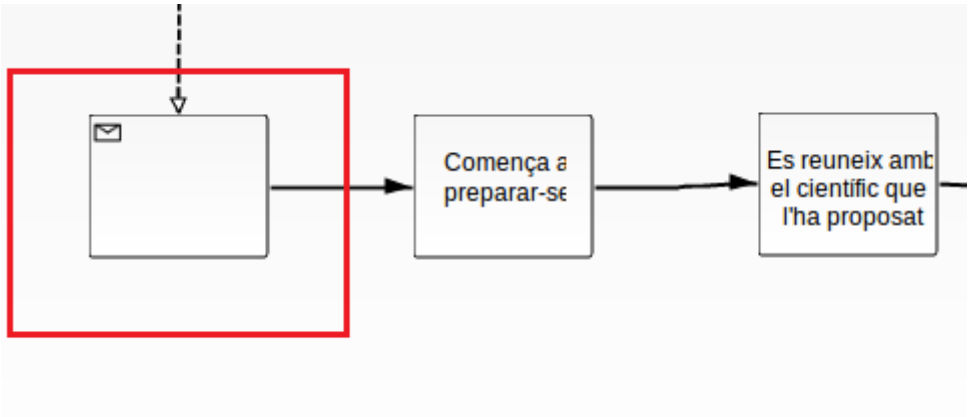


Figura 14. Procés que s'inicia amb una tasca de recepció

7.2.9 Subprocessos dins de finals diferents i amb més d'un final

A vegades pot donar-se el cas que un subprocés tingui més d'un final també, o bé que un subprocés està entre dues separacions del procés principal. Ha calgut adaptar per a que només busqui els finals del mateix procés ignorant els subprocessos interns. En alguns casos hi ha un subprocés dins d'un altre, pel que també s'ha de distingir a quin subprocés correspon cada element. Això s'ha fet amb un conjunt d'elements que ens diu a quin procés o subprocés pertany cada final i inici, amb el qual no buscarà res intermedi, si no simplement la caixa del subprocés per a seguir endavant.

7.3 Creació del text

7.3.1 Funció de creació per a processos

És una de les funcions bàsiques de la creació del text, i és necessari per a tenir tota la informació que caldrà per a poder-la imprimir posteriorment. El primer que fa és mirar si és node inicial i si aquest té una tasca de recepció prèvia. Si és el cas mira primer aquesta, l'afegeix al conjunt de textos a passar a la funció de fusió. Després comprova que el fragment no sigui un rígid, i en cas de ser-ho ho envia a la creació de text per a rígids. Si no continuarà.

Després mira si té fills o no. Si no en té agafa la informació de la tasca que hi acaba, si n'hi ha i la posa tota. Si té fills explorarà recursivament per ells, sempre i quan puguin aportar informació útil. Cada fill li retornarà un text únic que s'hi afegirà al conjunt. Un cop s'han buscat tots els fills útils es passen a la funció de fusió. Si és l'inici d'un vincle exclusiu agafa la condició. El pseudocodi d'aquesta funció es pot veure en el primer annex.

7.3.2 Funció de fusió

Com la funció de creació i la de fusió juntes es van demostrar ineficients i difícils de treballar, es va decidir separar en diferents funcions. A aquesta funció li arriben un conjunt de textos, el node actual, els germans que són, condició de la branca, nombre de fill amb text útil, a més a més d'un indicador de primera o segona iteració. A partir d'aquí s'exploren els diferents casos, comparant tots els casos possibles, mirant inicis i finals del node, nombre de textos, nombre de germans, per a agafar el patró més adient. A vegades passa que ens trobem amb només un text, que retorna un patró simple de $1, com per exemple el cas de la figura 15, on només la tasca dona text. Un cop agafat aquest patró busca caràcter per caràcter, i els posa tots, substituint les marques d'informació per aquesta informació, però deixant les marques d'indentació i paràgraf.

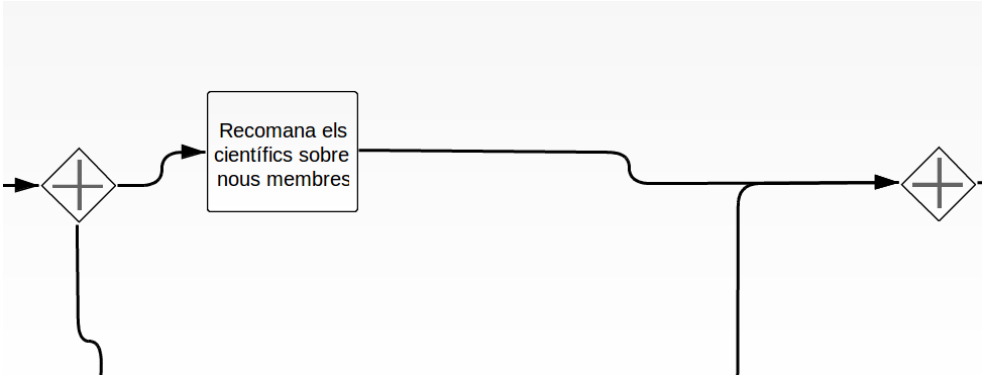


Figura 15. Un clar exemple on el patró seleccionat és $1

7.3.3 Patrons

Primer el programa llegeix un nom de fitxer de patrons, i posteriorment llegeix el fitxer corresponent, que serà l'idioma del patró. Segons quatre paraules fixes, PATTERN x, ENDPATTERN, BEGPT i ENDPT, afegirà patrons, essent la x un nom amb el qual s'identificarà. Un cop s'acaba el patró quedarà guardat com a clau el nom i una llista de frases com a valors. Un PATTERN significa que es comença amb una sèrie de patrons, seguit del seu nom clau, pel qual posteriorment el programa buscarà en cas que el necessiti. ENDPATTERN tanca aquesta sèrie de patrons. BEGPT comença un patró concret, i si a la mateixa línia es troben més paraules clau vol dir que aquest patró ha de complir aquesta condició.

També llegirà les restriccions d'un patró en concret, estant a la línia del BEGPT. Pot ser ACTY, si té usuari o ACTN si no, LISTY si es vol fer en llista i LISTN si es vol tabulador, i N2 si serveix per a 2 textos, N3 per a 3, N4 per a 4 i NN per a qualsevol nombre de textos. Inicialment, però, només hi havia un patró, pel qual el nom no canviava, i després llegia d'un fitxer segons l'idioma.

En quan al tipus de patrons hi ha per a tot tipus de casos. Per a les seqüències existeixen SEQUENCE, per a definir un nombre i INTERNSEQ si són múltiples textos. Per a les tasques normals i d'usuari s'utilitza ACTIONS. Per a les tasques de recepció, si aquesta té text s'utilitza ACTIONR, si no en té però qui envia sí, ACTIONRO, i si tampoc és el cas ACTIONRNONAME. El simètric per a les d'enviament, si té text és SENDACTION, si no, SENDACTIONNAME.

Les portes tenen diversos casos. BOND, INCLUSIVE i ENDING introdueixen vincles exclusius, inclusius i exclusius amb finals diferents, respectivament. PARALLEL i PARALLELN ho fan per als paral·lels, sense nom en el primer cas i amb nom en el segon. LOOPB introdueix els bucles. SPARALLELW s'utilitza per a cada branca paral·lela. Per a les exclusives, inclusives i de diferents finals es posen BONDCOND, INCLUSIVECOND i ENDINGINSIDECOND si tenen condició i BONDNOCOND, INCLUSIVENOCOND i ENDINGINSIDE si no en tenen. Les portes basades en esdeveniments es consideren exclusives.

Els subprocessos s'introdueixen amb el patró SUBPROCESSBEG. MESSAGES introdueix els missatges del final, essent MESSAGES4 quan tots dos tenen text, MESSAGESORIG quan en té l'emissor, MESSAGESTARG quan el té el receptor i MESSAGESNO en cas que cap dels dos tingui. Per últim, TIME s'utilitza per als esdeveniments intermedis i USER per a introduir un usuari de pool.

Es va descobrir que posar tots els patrons separats per casos uniformitzava massa el text resultant. Per això es va decidir posar "flags" a alguns d'ells. En conseqüència, segons el tipus de patró que es necessiti cridarà a una funció o una altra. Si la sèrie de patrons no té restriccions n'agafarà un d'aleatori d'entre tots ells. En canvi si en té buscarà un patró qualsevol. Si alguna opció del patró trobat no encaixa amb les requerides es torna a buscar fins a trobar una que serveixi, limitant així els errors. En els patrons situats a l'annex 2 per a les portes paral·leles es poden veure alguns dels "flags" utilitzats.

7.3.4 Activitats, Tasques i esdeveniments intermedis

Primer era saber com es troben les activitats, i es va descobrir les activitats i tasques només es troben en nodes fulla. En aquest cas es fa segons el tipus i les condicions. Si és una tasca d'usuari s'agafarà sempre el text, i l'usuari, si en té, i si no es busca a la "lane" o a la pool, segons el que estigui més a prop. La tasca normal obviarà l'usuari propi i anirà directe a la "lane". En cas que cap de les dues tingui usuari no es posa. Per als esdeveniments intermedis, que en aquest projecte només es fan per al temps, s'utilitzarà directament el text lligat a ell. En el cas que es mostra a la figura 16 posarà dos textos en el cas de la

tasca encerclada. Per un costat un text és *Avaluen els coneixements dels joves*. En ser una tasca mirarà la "lane". En aquest cas té text, *Científics reconeguts*. Així doncs, aquests són els dos textos que passen a la fusió.

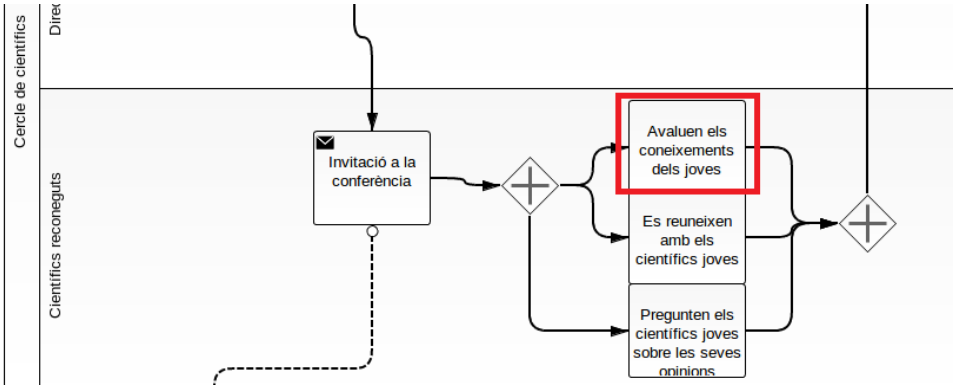


Figura 16. Exemple d'una tasca amb pool.

El cas de les tasques de recepció o enviament és diferent, ja que no sempre tenen text. Si és d'enviament i té text es posa text i usuari, i si no en té només l'usuari. La recepció si té text posa usuari i text. Si no, busca a la tasca de la qual ve el flux de missatge. En aquest cas posa usuari, l'emissor i el contingut d'aquest últim, si en té. Aquest és el cas de la figura 17, on la tasca en blau de *Científic Jove* no té text, però si que en té la verda, *Invitació a la conferència*, que és enviada per *Científics reconeguts.*

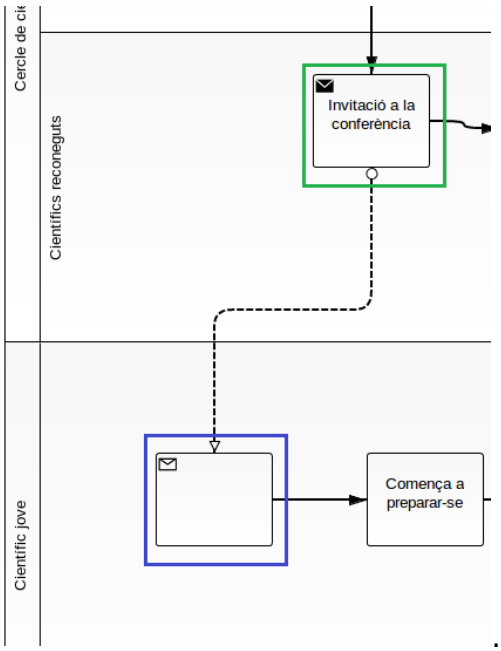


Figura 17. Tasques de recepció i enviament

Segons el tipus es busca un patró diferent. Per a les tasques normals o d'usuari són patrons únics i molt simples. En el cas de la figura 16, el resultat seria *Científics reconeguts avaluen els coneixements dels joves*. En quan a les tasques de recepció i enviament, cadascuna té els seus patrons segons els casos en els que es trobin. En el cas de la tasca de recepció de la figura 17, el text resultant és *científics reconeguts li diu a científic jove que ell invitació a la conferencia*.

7.3.5 Seqüències

Els polígons normalment són seqüències. Un cop té tots els textos agafarà un patró de seqüència per tal de posar per ordre tots els textos que li arriben per part dels seus fills, que són normalment fluxos i vincles. Si té dos o més busca un patró de seqüència. Si en té un directament el retorna com a text propi que li arriba, cosa que s'aconsegueix amb $1..

El problema principal és que moltes d'aquestes seqüències es trobaven dins de vincles. Per tant, cal posar la condició que tenien per a que s'executi la branca i després posar una seqüència. Degut a això no es podia fer en una sola iteració i en calien dues. Una per a agafar el patró de la condició i una altra per a la seqüència mateixa. És el cas, per exemple de la figura 18, on hi ha 6 textos, pel que caldrà agafar els patrons múltiples.

El patró que surt és Primer, $1. $v. Finalment $u. Per aquest $v s'agafen altres quatre introduccions de seqüència, essent aquestes Després d'això, $1., Després, $1., Després, $1. I Després d'això, $1. Per als quatre primers textos el text resultat seria *Primer, "científics reconeguts" li diu a "científic jove" que ell "invitació a la conferència". Després d'això, "científic reconegut" "comença a preparar-se"*.

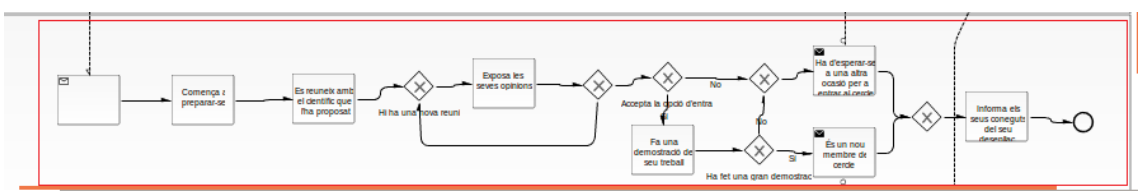


Figura 18. Una de les seqüències de l'exemple, la del procés de Científic jove sencer

També hi ha un problema més afegit. A vegades aquests estan restringits al que facin germans anteriors o pare. Si són més de 4 germans serà un patró adaptat a multiplicitat. Si no qui tria multiplicitat és el primer fill, i els germans l'han d'imitar. En el primer cas cada fill s'ha de posar una marca de llista o tabulació addicional al davant i una de final al darrere. En cas contrari directament el fill no s'afegeix res.

Es va descobrir, però, que hi havia una excepció: els fluxos que van de porta inicial a un final. En aquest cas s'ha d'agafar un patró de final de procés a més del de condició.

7.3.6 Vincles i subprocessos

Els vincles es poden fer de dues maneres, depenent del que ell mateix o el primer fill polígon esculli. Si hi ha més de 4 fills o bé el primer fill vol que sigui adaptable a multiplicitat s'agafarà un patró adaptable a multiplicitat, tenint només la introducció i via lliure per als seus fills. En cas contrari el pare també introdueix els fills, tot i que no diu condició. Els finals són com els vincles, només canviant les paraules. Aquí, a la figura 19, es mostra un vincle, on hi ha tres branques. En aquest cas el primer fill podia escollir i ho ha fet a favor d'un de múltiples opcions. El patró exterior escollit és el següent:

Hi ha $s coses a fer al mateix temps:
$c

Mentre que els tres interiors són.

$lAl grup $i, $o. $e
$lA la tasca paral·lela $i, $o. $e"
$lA la tasca paral·lela $i, $o. $e"

El resultat final és aquest:

Hi ha 3 coses a fer al mateix temps:
** Al grup 1, " científics reconeguts" "avaluen els coneixements dels joves".*

** A la tasca paral·lela 2, "científics reconeguts " "es reuneixen amb els científics joves".*

** A la tasca paral·lela 3, "científics reconeguts" "pregunten els científics joves sobre les seves opinions".*

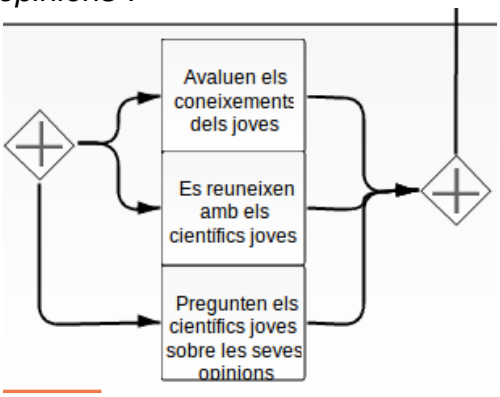


Figura 19. Les branques paral·leles impreses

Hi ha un cas particular, però, que és quan el vincle té dos fills, un polígon i un flux. Aquest és flux únic és un dels pocs casos que acaben en porta però donen informació útil, agafant directament un patró sense informació, el de continuïtat.

Per als subprocessos és similar, amb la principal diferència que sempre introdueix un subprocés i li deixa a aquest fer la feina bruta.

7.3.7 Bucles

En el cas dels bucles només ens interessa la informació interior i la condició exterior. És per això que aquests actuen com a una seqüència tot i que l'arrel sigui un vincle, ja que el flux de tornada no aporta informació rellevant. Degut a això, el vincle només torna el que dona el fill polígon. En el cas de la figura 20 el patró és *Mentre estigui en vigor $a, $o.* El resultat és *Mentre estigui en vigor "hi ha una nova reunió", "jove científic" "exposa les seves opinions".*

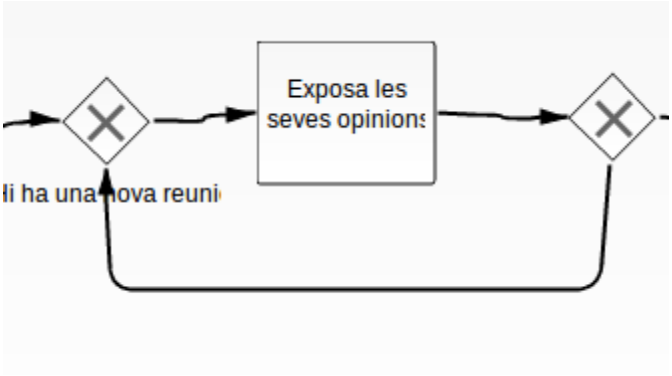


Figura 20. Un exemple de bucle on el flux de tornada no dona informació

7.3.8 Rígids

Els rígids sempre s'han tractat per separat de la resta de tipus de fragments. Això es deu per que hi ha un problema, ja que no s'ha pogut concretar prèviament les separacions. Tenim, però, una pista: l'inici i el final del rígid. Això permetrà que es pugui descompondre de nou només aquesta part.

Antigament, es creava una xarxa de Petri sencera. La primera iteració era, a partir de l'inici es creaven fluxos que anaven a parar a tasques fictícies, que continuaven en portes fictícies i en tasques. Després es mirava si alguna de les portes reals transformades a la xarxa de Petri ja existia amb un altre nom, connectant-hi un flux. Un cop muntat la xarxa de Petri sencera s'havia de posar almenys via per a cada flux d'ell, per tal de tenir totes les alternatives. Una tercera iteració permetia treure tot el text, però posant totes les condicions alhora seguit de les activitats per a elles. Aquesta tècnica, a més de ser ineficient, donava resultats pocs clars, ja que es deia tot com si fos una condició. Això s'explica amb més detall a l'apartat 2.3.1.

Actualment s'ha canviat la forma de fer-ho, una forma que s'explica en els dos subapartats següents.

7.3.9 Creació dels autòmats

Veient que hi havia masses nodes que no aportaven res, va arribar l'hora de canviar a un sistema que tingués més informació útil. Així va començar a fer-se

la opció del Petrify. Aquesta opció requeria, però tenir un autòmat a passar al Petrify i que donés la opció més eficient possible dels rígids.

Ja es sabia on començava i acabava el rígid, per tant havíem de fer una cerca recursiva per a totes les sortides de la porta inicial, i així amb totes fins a arribar a la porta final. Si ens trobàvem una porta es creava un node i se li posava informació, si n'hi havia. Cada node pot tenir 0 a moltes entrades i 0 a moltes sortides, pel que cal guardar-se totes. En quan a arestes, representaven les tasques i se li afegia informació, tant la condició per la qual hi arriba, si en té, com usuari, si en té, com text. Aquestes tenen entrada única i sortida única. En cas que dues tasques vagin seguides cal crear un node fictici sense informació, que tindrà entrada i sortida única. En els nodes també es guarda el tipus de porta.

Aquesta solució s'ha aplicat amb dues funcions que es criden l'una a l'altra de forma recursiva si es troba un flux de sortida d'un node no final.

En acabat, es crea un fitxer de sortida amb les dades necessàries de l'autòmat, inclosos les tasques i els fluxos entre elles o entre dues portes com a arestes. Després l'autòmat retorna una xarxa de Petri eficient i simple. Degut a que a vegades trigava massa en acabar, podia acabar agafant una xarxa de Petri Net antiga, pel que ha calgut afegir un comptador de fins a 10 milions abans de llegir el nou fitxer. A partir d'aquí ignora la informació irrellevant que ens retorni el fitxer generat, buscant només inici i successions, les quals es passaran a la fase de creació.

Si es vol saber més sobre els autòmats es pot veure l'apartat 2.3.2.

7.3.10 Creació del text rígid

El text del rígid es fa a partir tant de la informació de les estructures pròpies com de les adquirides pel Petrify. L'arrel del nostre arbre d'autòmat coincideix amb l'inici que dóna el Petrify. Aquí, però, a diferència d'altres casos, la informació ve dels fills separada per fills directes. Com només fan fusió de textos els nodes, caldrà 4 iteracions. Dels fills, però, també ens ve la quantitat d'informació que ens arriba separada per nodes. Si només ens arriba d'un fill,

agafa un patró que passarà ja directe a la tercera iteració, ja que és part final. Si no per a cada cosa que arriba s'agafa un patró per a la segona iteració i cada valor. Per a la tercera, però, es farà sempre.

La segona iteració més que res hi afegeix la condició com si fos una d'interior de vincle per a cada fill, traient a cada grup de textos el primer element. La tercera iteració hi posa la seqüència, si té més d'una unitat, per a cadascuna. La quarta separa aquestes unitats, que més que res son les tasques que es fan, o bé resultats de portes més interiors i les retorna a la tercera per a que les pugui posar. Quan cada iteració acaba torna a l'anterior. Amb això es crea un text únic per a tot el node, que anirà a parar a nodes superiors.

La cerca de la informació d'altra banda es fa així. Un node sempre ens portarà a arestes. A vegades aquesta no es filla directe del node de l'arbre autòmat en el que ens trobem, però si se sap que s'hi arriba amb el mateix ordre de fill directe. Per tant, es va buscant pels fills del fill fins que s'hi troba. També pot passar que una aresta en segueixi una altra. En aquest cas és més fàcil, ja que sempre es cercarà el fill únic del fill node.

Per a demostrar-ho, tenim aquí l'únic rígid de tot el model, el que surt a la figura 21.

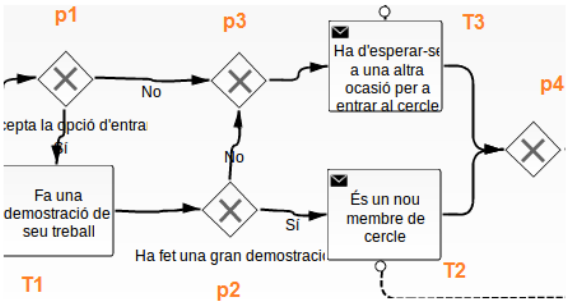


Figura 21. El rígid de l'exemple

Es pot comparar com l'estructura jeràrquica dels dos resultats a la figura 22 és gairebé idèntica. La porta 1 és el node arrel tant a l'autòmat com al nou BPMN. Tots dos tenen dues arestes de sortida. La petita diferència és que en l'autòmat l'aresta és la tasca, mentre que en el BPMN millorat l'aresta és el conjunt de fluxos que van d'una porta a la tasca. També es pot veure que la porta 2 té

dues sortides, que tenen T2 i T3 com a sortida. Un altre punt a destacar és que la tasca T3 apareix fins a dues vegades, degut a que hi ha dos camins que passen per ell.

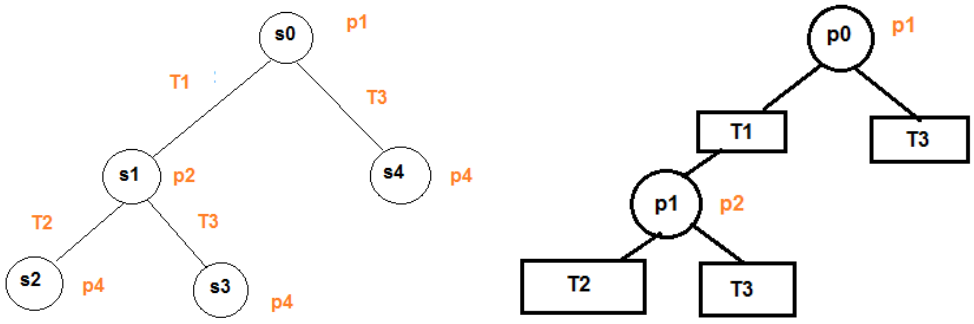


Figura 22. L'autòmat i el BPMN millorat

Això dóna un resultat similar al que donaria un vincle normal i corrent. En cas d'aquest exemple, aquest és el resultat:

Només una cosa es farà d'entre aquestes 2 opcions:
- En cas que la condició "accepta la opció d'entrar" sigui "sí", primer, "científic jove" "fa una demostració del seu treball". Finalment aquí es poden escollir 2 opcions:
    * Si el cas és "ha fet una gran demostració" amb valor "no", "científic jove" "ha d'esperar-se a una altra ocasió per a entrar al cercle".
    * En cas que la condició "ha fet una gran demostració" sigui "sí", "científic jove" "És un nou membre del cercle".
- En cas que la condició "accepta la opció d'entrar" sigui "no", "científic jove" "ha d'esperar-se a una altra ocasió per a entrar al cercle".

Finalment "científic reconegut" "informa els seus coneguts del seu desenllaç".

7.3.11 Missatges

La creació de missatges és una funció apartada totalment de les lligades als processos. Primer es posa una introducció, i després es busquen tots els fluxos de missatge. Per a cada missatge es mira de quina tasca surt i a quina arriba. Si les dues tenen text, es busca un patró de missatge que tingui per a dir les tasques prèvia i següent a aquest missatge. Si només en té el d'entrada diu el que fa aquest, i ídem si és només el de sortida. En cas que cap tingui text, simplement s'ha decidit que digui qui són emissor i receptor. Aprofitant el de la figura 17 de l'apartat 8.3.5, posarem una entrada de missatge i un patró de missatge per a aquest cas. Els dos patrons serien aquests:

Hi ha un nombre de missatges passats entre els participants:
$l Després que $1 $2, passa un missatge a $3. $e

Es pot veure que el patró d'introducció a missatge no té forats, mentre que el segon, el que li dona la informació en té tres, $1 per a l'emissor, $3 per al receptor i $2 la tasca que fa l'emissor. Un hipotètic $4 hagués donat la tasca del receptor. El resultat final serà aquest:

Hi ha un nombre de missatges passats entre els participants:
- Després que "científics reconeguts" "invitació a la conferència", passa un missatge a "científic jove".

7.3.12 Filtrat

Un cop creat un primer text provisional sortien molts elements extres i inútils, la majoria dels quals eren signes de puntuació o salts de línia repetits que s'acumulaven de fills a pares i que no representen més que un destorb. Per a això es crea una operació de filtrat, que quan detecta un signe de puntuació prohibeix que n'hi hagi més fins que no hi hagi una lletra normal. Quan detecta també un salt de línia també filtra els punts i altres salts de línia per a que no quedin massa separats. Passa al revés, però, amb els tabuladors, que només es filtraran si es troba un $e, o be les $l es filtren si no hi ha un salt de línia anterior.

Una altra millora que es fa sobre el text és canviar minúscules per majúscules o a l'inrevés quan calgui. Si la lletra és després d'un punt o de dos punts es posa una majúscula, i si no una minúscula. Així s'evita que quedin massa majúscules al text degut a les majúscules que hi hagués al model. Un exemple possible podria ser el següent:

$l A la tasca paral·lela 2, "Científics reconeguts " "Es reuneixen amb els científics joves". $e.

$l A la tasca paral·lela 3, "Científics reconeguts" "Pregunten els científics joves sobre les seves opinions". $e.

Com es pot veure, entre una frase i l'altre hi ha dos espais. També hi ha dos punt separats, un davant de $e i l'altre darrere en cada frase. També es pot

observar que el subjecte i el predicat també comencen amb majúscula, quan ho haurien de fer en minúscula en ser en mig d'una oració. Els canvis de majúscula per minúscula i l'eliminació de duplicitats quedaria així:

$l A la tasca paral·lela 2, "científics reconeguts " "es reuneixen amb els científics joves". $e
$l A la tasca paral·lela 3, "científics reconeguts" "pregunten els científics joves sobre les seves opinions". $e

7.3.13 Paràgrafs

Les marques que queden s'utilitzen per tal que el text quedi estructurat de la manera que es vulgui. En cas que sigui llista el programa sempre agafarà un patró que tingui un $l a l'inici de la línia per al vincle de menys de 5 opcions, o li afegeix en cas que sigui fill de múltiple. El mateix es farà amb $t en cas del paràgraf. Si és un mixt hi ha un valor aleatori, el qual decidirà si surt com a llista o com a paràgraf. Tot i així, la decisió del pare, o en el seu defecte del primer fill trobat, obliga a la resta a fer-ho d'una o altre forma per a que quedi bé. Per això es creen dues variables en forma de llista, on ens diu que ha d'utilitzar les llistes o que no en pot utilitzar. Si no és en cap de les dues tindrà llibertat d'escollir. Posem, per exemple, una frase de l'apartat 8.3.7:

$l A la tasca paral·lela 2, "científics reconeguts " "es reuneixen amb els científics joves". $e

Ja tenia una tabulació de 1, a la qual se li suma la nova de $l. Estant en nombre parell de tabulació, 2, se li posarà una estrelleta. El $e li traurà aquesta tabulació addicional, quedant-se després en 1. El resultat final és aquest:

$l A la tasca paral·lela 2, "científics reconeguts " "es reuneixen amb els científics joves". $e

7.3.14 Cometes

A vegades es veia que el text quedava poc clar. Per a solucionar el problema es va decidir a posar una variable per a ajudar a la comprensió del text. Si aquesta variable resulta activada un parell de cometes serà posat a l'inici i al final de l'usuari, la tasca, la condició de la porta o el seu valor. Amb això ajuda a trobar amb més facilitat el més important del model. Per exemple, *científic*

jove, al ser el valor d'una pool es posaria entre cometes, quedant "*científic jove"*.

7.3.15 Resultat final de la generació del text

Aquest és el resultat final que queda després de la generació de text d'aquest exemple:

L'usuari "cercle de científics" fa les següents coses. Primer, aquestes 2 coses s'haurien de fer simultàniament:
- A la branca 1, "director del cercle de científics" "recomana els científics sobre nous membres".
- A la tasca paral·lela 2, començant per "científics reconeguts" envia un missatge dient "invitació a la conferència". Acaba amb hi ha 3 coses a fer al mateix temps:
** Al grup 1, " científics reconeguts" "avaluen els coneixements dels joves".*
** A la tasca paral·lela 2, "científics reconeguts " "es reuneixen amb els científics joves".*
** A la tasca paral·lela 3, "científics reconeguts" "pregunten els científics joves sobre les seves opinions".*
Després d'això, "director del cercle de científics" "prepara la seva conferència".
Després d'això, "director del cercle de científics" "veuen les demostracions".
Després d'això, "científics reconeguts" "avaluen totes les demostracions".
Finalment aquesta decisió acabarà amb diferents finals per a cada valor d'aquestes 2 opcions:
- El final 1 té "científics reconeguts" "el científic jove encara no queda acceptat".
- El final 2 té "científics reconeguts" "el jove científic es un nou membre del cercle".
L'usuari "científic jove" fa les següents coses. Primer, "científics reconeguts" li diu a "científic jove" que ell "invitació a la conferència". Després d'això, "científic jove" "comença a preparar-se". Després, "científic jove" "es reuneix amb el científic que l'ha proposat". Després, mentre estigui en vigor "hi ha una nova reunió", "director del cercle de científics" "exposa les seves opinions".
Després d'això, només una cosa es farà d'entre aquestes 2 opcions:
- En cas que la condició "accepta la opció d'entrar" sigui "sí", primer, "científic jove" "fa una demostració del seu treball". Finalment aquí es poden escollir 2 opcions:
** Si el cas és "ha fet una gran demostració" amb valor "no", "científic jove" "ha d'esperar-se a una altra ocasió per a entrar al cercle".*
** En cas que la condició "ha fet una gran demostració" sigui "sí", "científic jove" "És un nou membre del cercle".*
- En cas que la condició "accepta la opció d'entrar" sigui "no", "científic jove" "ha d'esperar-se a una altra ocasió per a entrar al cercle".
Finalment "científic reconegut" "informa els seus coneguts del seu desenllaç".

Hi ha un nombre de missatges passats entre els participants:

- Després que "científics reconeguts" "invitació a la conferència", passa un missatge a " científic jove".

- Després que "científic jove" "ha d'esperar-se a una altra ocasió per a entrar al cercle", passa un missatge a "científics reconeguts", que fa "el científic jove encara no queda acceptat".

- Després que "científic jove" "És un nou membre del cercle", passa un missatge a "científics reconeguts", que fa "el jove científic es un nou membre del cercle".

7.3.16 Marques dels patrons

Aquesta no és més que una breu descripció de totes les marques que s'han anat posant durant el projecte en els patrons, incloent-ne algunes de desaparegudes.

- $1, $2, $3 i $4 posen els textos d'entrada a la funció de fusió que es troben a la posició 0, 1, 2 i 3 respectivament
- $p creava salts de línia sempre, desaparegut amb els patrons multi línia.
- $d posa la condició de la porta i $a el seu valor.
- $c Imprimeix tots els textos d'entrada un darrera l'altre, amb salts de línia entre ells.
- $o Crida recursiva a la funció de fusió per a posar els textos en forma de seqüència.
- $u Posa l'últim text d'entrada.
- $s Diu el nombre de textos d'entrada, normalment per a dir les opcions de sortida.
- $i L'ordre del text d'entrada + 1, és a dir el 0 és 1, el 1 és 2, per a enumerar branques.
- $l Crea un caràcter * o – depenent de la indentació, afegint-hi un nivell més.
- $t Indenta al nivell actual si la tabulació està activada. En cas contrari no fa res.
- $e Resta un nivell a la indentació actual si l'últim $l o $t n'ha incrementat un.
- $b Indenta al nivell actual si la tabulació està activada. Si no és el cas no fa res.
- $v Crida per a tots els fills no últims, per a seqüències.

- $n Crea salts de línia si el paràgraf porta 75 paraules o més en el punt on es troba aquesta marca.

7.4 Fitxer de sortida

Ha estat potser el més fàcil de resoldre, ja que abans s'havien fet els fitxers de sortida de l'autòmat. És tan senzill com copiar el text resultant al fitxer de sortida que s'acaba de crear.

8. Anàlisi dels resultats

8.1 Resultats obtinguts i experiments

En aquest apartat es mostraran diversos resultats sobre alguns dels models, comparant sortides del mateix model aplicant variacions sobre tenir o no cometes, tenir paràgrafs, llistes o barrejar i, finalment, tabular o no els paràgrafs. De pas, es vol també mostrar la variabilitat dels resultats sortints. Posteriorment s'oferirà un exemple de dos models en anglès i els que s'han creat traduïts al català, per demostrar que existeix un patró de qualitat similar en les dues llengües.

8.1.1 Variabilitat dels resultats

Per començar, però, oferirem un parell d'exemples del mateix model amb els mateixos paràmetres. Així doncs, es podrà comprovar amb més facilitat que els resultats entre dues execucions són similars i tenen el mateix contingut aproximadament, però no són idèntiques. Per a aquesta demostració utilitzarem el model número 8, amb llistes activades i cometes activades.

La primera execució dona el següent resultat:

Starting with "examine patient", it continues with there are two endings:
- The first ending, when "additional example required?" with value "yes", after starting in "order examination & follow-up treatment", it goes on with "inform about risks", and then this choice has two possible endings:
** The first ending, the branch "patient agrees?" under condition "no", "release patient".*
- The second ending, the option "patient agrees?" as "yes" has as a consequence starting with "make appointment". Subsequently, "prepare examination". After that, all the time the condition "sample is not ok" is accomplished, first thing to do is "take sample", then two groups of activities are done in parallel:
** On the first group "perform follow-up treatment".*
** On the second group "send sample".*
And finally "validate sample state". Subsequently, "analyze sample". After that, "validate results". Subsequently, "make diagnosis". It ends with "describe therapy".
- The second possible ending, when "additional example required?" with value "no", "release patient".

La segona, aquest:

This sequence starts with "examine patient". At the end, this decision will end up with a different ending for each value of these 2 options:
- The option "additional example required?" as "yes" has as a consequence first, "order examination & follow-up treatment". Then, "inform about risks". Finally different decision in each one of the 2 branches will give different endings:
** The option "patient agrees?" as "no" has as a consequence "release patient".*
- The option "patient agrees?" as "yes" has as a consequence starting with "make appointment". Then, "prepare examination". After that, all the time the condition "sample is not ok" is accomplished, first, "take sample". Then, these 2 things should be done simultaneously:
** On the branch 1, "perform follow-up treatment".*
** In the parallel task 2, "send sample".*
Finally "validate sample state".
Subsequently, "analyses sample". Subsequently, "validate results". After that, "make diagnosis". It ends with "describe therapy".
- The branch "additional example required?" under condition "no", "release patient".

Ja en la primera frase es pot veure la diferència entre dues execucions seguides, demostrant que tot i ser el mateix model i agafar patrons similars donen resultats lleugerament diferents. La primera diu que *començant per l'examinació del pacient segueix cap a dos finals diferents*. La segona diu el mateix amb diferents paraules i separant-ho en dues frases*. Diu aquesta seqüència comença per examinar el pacient. Al final, una decisió portarà a un final diferent per a cada valor de 2 opcions*.

Aquests dos exemples també ens serveixen per a veure les dues formes diferents de fer per a 4 o menys finals, en tots els casos del primer text, on diu els casos i el seu ordre amb lletres. Compari's amb el cas de les dues branques paral·leles, on l'ordre és descrit amb un nombre. També es pot veure en el cas del segon text després de la sisena línia, on no diu l'ordinal, sinó que només la causa per la qual s'ha arribat. En el segon text també es pot observar que hi ha més d'un patró per a seqüències, ja que usa "Subsequently" dues vegades seguides, però després diu "After that". Això, però, només s'aplica en casos de seqüències per a més d'un ús.

8.1.2 Cometes

En aquest apartat es faran dues comparacions entre els mateixos models, escollint en aquest cas els models 11 i 9. Les llistes estan activades, però compararem textos amb cometes i sense cometes.

Comencem amb el model 11 amb cometes:

There are two parallel activities:
- First parallel group it starts with it's "7 days". Then, "approval in progress email". Subsequently, the current time is "23 days". It finishes with "advise to employee to start again".
- Second parallel group starting with 2 options can be chosen here:
** In case the condition "does account exist?" is "yes", go on.*
When "does account exist?" "no" is accomplished, "create account".
It continues with "review for approval", finishing with 2 options can be chosen here:
** If the case is "pre-approve" under value "less than 200 ,starting with supervisor review. It ends with the two different exclusive alternatives that follow are these:*
- If the case is "authorize payment" under value "yes", after send for payment goes on with transfer to employee account.
- When "authorize payment" "no" is accomplished, notify employee.
** If the case is "pre-approve" under value "more than 200 ,starting with send for payment. It ends with transfer to employee account.*

A continuació el mateix model sense cometes:

There are two parallel activities:
- First parallel group the first to do is it's 7 days. After that, approval in progress email. Then, it's 23 days. At the end, advise to employee to start again.
- Second parallel group after starting in one of the next 2 things has to be done:
** If the case is does account exist? under value yes, it's not necessary to do anything.*
When does account exist? no is accomplished, create account.
It goes on with review for approval, and then one of the next 2 things has to be done:
** If the case is pre-approve under value less than 200 this sequence starts with supervisor review. At the end, one of these two options will be selected:*
- The first alternative, if the case is authorize payment under value yes, first, send for payment, and then transfer to employee account.
- The second alternative, if the case is authorize payment under value no, notify employee.
** When pre-approve more than 200 is accomplished, starting with send for payment. It ends with transfer to employee account.*

Ara, primer amb cometes i després sense, els dos textos del model 9:

For user "town authority planning", first, "coordination unit" "draft dates". Then, "coordination unit" "enter into next years calendar". After that, "support officer" "checks & suggests updates". Then, "coordination unit" "finalize calendar schedule". After that, "coordination unit" passes a message with content "send schedule to members". Then, "support officer" "receive schedule conflicts". Then, "support officer" "updates group calendars". Finally "support officer" gives a message which says "send final schedule to members".
For user "committee members", this sequence starts with after receiving a message, "committee members" does "receive meeting schedule". Then, "committee members" "check for conflicts". After that, "committee members" gives a message which says "advise of schedule conflicts". At the end, "committee members" "receive final schedule".

The messages passed from one participant to another are the following:
- After "coordination unit" performs "send schedule to members", he passes a message to "committee members", who does "receive meeting schedule".
- After "committee members" performs "advise of schedule conflicts", he passes a message to "support officer", who does "receive schedule conflicts".
- After "support officer" performs "send final schedule to members", he passes a message to "committee members", who does "receive final schedule".

The user town authority planning does the next things. This sequence starts with coordination unit draft dates. Subsequently, coordination unit enter into next years calendar. Subsequently, support officer checks & suggests updates. Subsequently, coordination unit finalize calendar schedule. Subsequently, coordination unit passes a message with content send schedule to members. After that, support officer receive schedule conflicts. After that, support officer updates group calendars. At the end, support officer sends a message saying send final schedule to members.
The user committee members does the next things. The first to do is committee members receives a message and performs receive meeting schedule. After that, committee members check for conflicts. Then, committee members passes a message with content advise of schedule conflicts. At the end, committee members receive final schedule.

The messages passed from one participant to another are the following:
- After coordination unit performs send schedule to members, he passes a message to committee members, who does receive meeting schedule.
- After committee members performs advise of schedule conflicts, he passes a message to support officer, who does receive schedule conflicts.

- After support officer performs send final schedule to members, he passes a message to committee members, who does receive final schedule.

Es pot comprovar en ambdós casos que és més fàcil trobar la informació important quan el text es troba entre cometes que quan es barreja tot. En tots dos casos és més còmode i ràpid, ja que es pot extreure millor el gra de la palla, pel que permetria ignorar allò que no és vital per a l'usuari, com per exemple quantes opcions hi ha per cada porta. En cas contrari, però, cal llegir tot el text per a no perdre's res important de les seqüències o alternatives. Per tant, el tema d'utilitzar cometes ha estat una aportació que millora la comprensió del text.

D'altra banda en el model 9 es veu com funciona el tema dels missatges. Es posen a part dels processos principals, introduint-los primer, i després posant d'on a on van de la manera més textual possible.

8.1.3 Llistes, paràgrafs i tabuladors

De nou, aquí es posaran dos models, però aquí es volen comparar quatre casos: Quan s'imprimeix tot en format llista, tot en format paràgraf, mixt amb tabulacions i mixt sense tabulacions, mostrades per aquest ordre per a tots dos models:

Començarem amb el model 1:

Starting with "room-service manager" "take down order". Then, one of these two options will be selected:
- The first alternative, in case the condition "premium customer" is "no", starting with "room-service manager" "check customer solvency", it continues with one of the next 2 things has to be done:
** When "ok" "yes" is accomplished, "room-service manager" "approve order without debit".*
** If the case is "ok" under value "no", "room-service manager" "debit guest's account".*
- The second alternative, if the case is "premium customer" under value "yes", "room-service manager" "approve order without debit".
After that, these 2 things will be done at the same time:
- On the branch 1, first, "room-service manager" "submit order to kitchen". Finally "kitchen" "prepare meal".
- In the parallel task 2, one of these two options will be selected:
** The first alternative, when "alcoholic beverages ordered" "yes" is accomplished, starting with "room-service manager" "give order to sommelier", it*

continues with despite the two options are possible, at least one of them has to be done:

- Either the first option, the group of things to do in group 1 under the condition "wine is ordered" "yes", "sommelier" "fetch wine from cellar".

- Or the second option, the group of things to do in group 2, "sommelier" "prepare alcoholic beverages".

** The second alternative, if the case is "alcoholic beverages ordered" under value "no", it's not necessary to do anything.*

It ends with "waiter" "deliver to guest's room".

First, "room-service manager" "take down order". After that, two alternatives are possible, and only one will be done:

When "premium customer" "no" is accomplished, after "room-service manager" "check customer solvency" goes on with one of the next 2 things has to be done:

When "ok" "yes" is accomplished, "room-service manager" "approve order without debit".

If the case is "ok" under value "no", "room-service manager" "debit guest's account".

If the case is "premium customer" under value "yes", "room-service manager" "approve order without debit".

After that, these two groups are all done at the same time:

The first, starting with "room-service manager" "submit order to kitchen", it continues with "kitchen" "prepare meal".

The second, one of these two options will be selected:

The first alternative, if the case is "alcoholic beverages ordered" under value "yes", starting with "room-service manager" "give order to sommelier", it continues with it's necessary to pick at least one of these 2 options:

The option 1 with condition "wine is ordered" "yes", "sommelier" "fetch wine from cellar".

The branch 2 has no condition, and the things done are "sommelier" "prepare alcoholic beverages".

The second alternative, if the case is "alcoholic beverages ordered" under value "no", go on.

Finally "waiter" "deliver to guest's room".

First, "room-service manager" "take down order". Then, one of these two options will be selected:

The first alternative, when "premium customer" "no" is accomplished, first, "room-service manager" "check customer solvency", and then only one thing will be done from these 2 options:

- In case the condition "ok" is "yes", "room-service manager" "approve order without debit".

- If the case is "ok" under value "no", "room-service manager" "debit guest's account".

The second alternative, in case the condition "premium customer" is "yes", "room-service manager" "approve order without debit".

Subsequently, two groups of activities are done in parallel:

On the first group after "room-service manager" "submit order to kitchen" goes on with "kitchen" "prepare meal".

On the second group 2 options can be chosen here:
When "alcoholic beverages ordered" "yes" is accomplished, starting with "room-service manager" "give order to sommelier". It ends with it's necessary to pick at least one of these 2 options:
- The group of things to do in group 1 under the condition "wine is ordered" "yes", "sommelier" "fetch wine from cellar".
- The option 2, "sommelier" "prepare alcoholic beverages".
When "alcoholic beverages ordered" "no" is accomplished, go on.
Finally "waiter" "deliver to guest's room".

First, "room-service manager" "take down order". Then, one of the next 2 things has to be done:
- If the case is "premium customer" under value "no", first, "room-service manager" "check customer solvency". Finally one of the next 2 things has to be done:
If the case is "ok" under value "yes", "room-service manager" "approve order without debit".
If the case is "ok" under value "no", "room-service manager" "debit guest's account".
- In case the condition "premium customer" is "yes", "room-service manager" "approve order without debit".
After that, these 2 things will be done at the same time:
In the parallel task 1, this sequence starts with "room-service manager" "submit order to kitchen". At the end, "kitchen" "prepare meal".
In the parallel task 2, one of these two options will be selected:
- The first alternative, when "alcoholic beverages ordered" "yes" is accomplished, starting with "room-service manager" "give order to sommelier", it continues with one or both options have to be performed:
** The branch 1, has a condition, which is "wine is ordered" "yes". If this occurs "sommelier" "fetch wine from cellar".*
** The branch 2 has no condition, and the things done are "sommelier" "prepare alcoholic beverages".*
- The second alternative, when "alcoholic beverages ordered" "no" is accomplished, it's not necessary to do anything.
Finally "waiter" "deliver to guest's room".

Es pot veure com en el cas de les llistes surt tot de forma més esquemàtica. Això és alhora un avantatge i un desavantatge. Per un cantó el que fa és estructurar d'una manera més clara el text per a que l'usuari trobi més ràpides les alternatives diferents. D'altra banda, però, pot arribar a revelar una mica l'estructura de l'arbre intern a partir del qual s'ha acabat generant el text. El format paràgraf té justament les característiques contràries. Amaga per complet l'estructura interna, però fa més difícil de llegir el text imprès, o com a mínim, més lent, ja que no es tan fàcil trobar les alternatives.

Per això es va decidir fer també una solució intermèdia. La impressió mixta. El tercer text demostra el seu correcte funcionament. En aquest cas ha donat la coincidència que són les parts més profundes de l'arbre les que han sortit en format llista, tot i que no sempre passa d'aquesta manera, com es demostra en el quart text, on la primera separació només fa llista la part exterior i la interior es fa com a paràgraf. Fixi's que el quart, a més a més, els paràgrafs estan indentats al mateix nivell que els pares, demostrant que la tabulació activada funciona. En tots dos també s'ha de veure que els que són germans estan o tots en format llista o tots en format paràgraf, per tal que sigui més fàcil distingir uns dels altres.

Tot seguit es mostrarà que passa amb els missatges en el model 5:

For user "customer", this sequence starts with "customer" "bring in defective computer". After that, "customer" "receive cost calculation". At the end, this decision will end up with a different ending for each value of these 2 options:
- The branch "acceptable" under condition "no", "customer" "take computer home".
- The branch "acceptable" under condition "yes", "customer" "continue process".
About "crs", after starting in "customer" tells "crs" that he "bring in defective computer", it goes on with "crs" "prepare cost calculation", and then the two possible endings from here are these:
- First ending, the option 1 has as a consequence "crs" receives the message from "customer" saying "take computer home".
- Second ending, ending 2, first thing to do is "customer" tells "crs" that he "continue process", then continues with a subprocess: These 2 things should be done simultaneously:
** In the parallel task 1, "crs" "check and repair hardware".*
** In the parallel task 2, "crs" "check and configure software".*
And finally "crs" "test system functionality".

There is a number of messages passed between the participants:
- After "customer" performs "bring in defective computer", he passes a message to "crs".
- After "crs" performs "prepare cost calculation", he passes a message to "customer", who does "receive cost calculation".
- After "customer" performs "take computer home", he passes a message to "crs".
- After "customer" performs "continue process", he passes a message to "crs".

The user “customer” does the next things. First thing to do is “customer” “bring in defective computer”, then “customer” “receive cost calculation” and finally this choice has two possible endings:
The first ending, the branch “acceptable” under condition “no”, “customer” “take computer home”.
The second ending, when “acceptable” with value “yes”, “customer” “continue process”.
For user “crs”, first thing to do is “customer” tells “crs” that he “bring in defective computer”, then “crs” “prepare cost calculation” and finally this decision will end up with a different ending for each value of these 2 options:
The option 1, “customer” tells “crs” that he “take computer home”.
The ending 2, first, “customer” transmits a message to “crs” to tell him that “continue process”. Then, continues with a subprocess: There are two parallel activities:
First parallel group “crs” “check and repair hardware”.
Second parallel group “crs” “check and configure software”.
Finally “crs” “test system functionality”.

The messages passed from one participant to another are the following:
After “customer” performs “bring in defective computer”, he passes a message to “crs”.
After “crs” performs “prepare cost calculation”, he passes a message to “customer”, who does “receive cost calculation”.
After “customer” performs “take computer home”, he passes a message to “crs”.
After “customer” performs “continue process”, he passes a message to “crs”.

About “customer”, first thing to do is “customer” “bring in defective computer”, then “customer” “receive cost calculation” and finally this choice has two possible endings:
The first ending, the option “acceptable” as “no” has as a consequence “customer” “take computer home”.
The second ending, the branch “acceptable” under condition “yes”, “customer” “continue process”.
About “crs”, first, “customer” tells “crs” that he “bring in defective computer”. After that, “crs” “prepare cost calculation”. Finally different decision in each one of the 2 branches will give different endings:
- Ending 1, “customer” transmits a message to “crs” to tell him that “take computer home”.
- The option 2 has as a consequence starting with “crs” receives the message from “customer” saying “continue process”. After that, it follows a subprocess: These 2 things should be done simultaneously:
In the parallel task 1, “crs” “check and repair hardware”.
In the parallel task 2, “crs” “check and configure software”.
It ends with “crs” “test system functionality”.

The messages passed from one participant to another are the following:
After “customer” performs “bring in defective computer”, he passes a message to “crs”.
- After “crs” performs “prepare cost calculation”, he passes a message to “customer”, who does “receive cost calculation”.

After “customer” performs “take computer home”, he passes a message to “crs”.
After “customer” performs “continue process”, he passes a message to “crs”.

For user “customer”, first thing to do is “customer” “bring in defective computer”, then “customer” “receive cost calculation” and finally this decision will end up with a different ending for each value of these 2 options:
- When “acceptable” with value “no”, “customer” “take computer home”.
- The branch “acceptable” under condition “yes”, “customer” “continue process”.
For user “crs”, after starting in “customer” tells “crs” that he “bring in defective computer”, it goes on with “crs” “prepare cost calculation”, and then this decision will end up with a different ending for each value of these 2 options:
- The option 1 has as a consequence “customer” tells “crs” that he “take computer home”.
- The option 2 has as a consequence first, “customer” transmits a message to “crs” to tell him that “continue process”. Subsequently, it follows a subprocess: These 2 things will be done at the same time:
On the group 1, “crs” “check and repair hardware”.
In the parallel task 2, “crs” “check and configure software”.
Finally “crs” “test system functionality”.

These are the messages passed:
- After “customer” performs “bring in defective computer”, he passes a message to “crs”.
- After “crs” performs “prepare cost calculation”, he passes a message to “customer”, who does “receive cost calculation”.
After “customer” performs “take computer home”, he passes a message to “crs”.
- After “customer” performs “continue process”, he passes a message to “crs”.

Per als missatges es pot veure com en el cas de que es posi en format llista (1er text) surten tots ells en format llista amb un guió al davant i quan és en estil paràgraf (2on text) surten tots en format paràgraf. En el cas dels mixtos no hi ha diferència entre tabulat i sense tabular, però surten en format llista o paràgraf de forma totalment aleatòria. En el 3r text veiem que l’únic que surt en format llista és el segon missatge, mentre que el 4t text l’únic que surt en format paràgraf és el tercer missatge.

En el tercer cas es pot veure clarament la tabulació del paràgraf just abans de la última frase de l’últim procés. Aquesta és demostració que la tabulació també funciona bé en aquests casos.

8.1.4 Català/Anglès

Les últimes comparacions que es faran són de dos models per veure que els patrons són similars tant per al català com per a l'anglès. Són models breus per a poder comparar que realment aquests models diuen coses similars:

Començarem amb el model 2, el més breu d'ells:

Starting with "john" "wakes up". After that, while "john is at the office" is true, "john" "works".
Subsequently, all the time the condition "john is returning home" is accomplished, "john" "stops at the shops".
Then, "john" "has dinner". It ends with "john" "goes to bed".

Aquesta seqüència comença amb "en joan" "s'aixeca". Després, mentre estigui en vigor "en joan és a la oficina", "en joan" "treballa".
Després d'això, tot el temps que la condició "en joan torna a casa" es compleix, "s'atura a les tendes".
Posteriorment, "en joan" "sopa". Al final, "en joan" "se'n va al llit".

Es pot veure que les úniques variacions que hi ha entre els dos idiomes es deuen a la pura aleatorietat dels patrons demostrada al primer subapartat. Es pot veure que les condicions i l'ordre dels bucles són el mateix. L'inici tampoc és massa diferent, de "començant" a "Aquesta seqüència comença". Les continuacions tampoc donen gaires diferències. De "Subsequently" a "Després d'això".

Anem ara a veure què passa amb un model una mica més llarg, el 6.

This sequence starts with "student" "does an exam". After that, "professor" "corrects the exam". Subsequently, there are 4 possible exclusive options:
- If the case is "mark of the exam" under value "lower than 5", "student" "fails the exam".
- When "mark of the exam" "between 5 and 7" is accomplished, "student" "just passes the exam with a c".
- If the case is "mark of the exam" under value "between 7 and 9", "passes well the exam with a b".
- If the case is "mark of the exam" under value "9 or higher", "student" "passes easily the exam with an a".
At the end, "student" "receives mark".

Primer, "estudiant" "fa un examen". Després, "professor" "corregeix l'examen". Posteriorment, hi ha 4 possibles opcions exclusives:

- En cas que la condició "nota de l'examen" sigui "entre 0 i 5", "l'estudiant" "suspèn l'examen".
- Si el cas és "nota de l'examen" amb valor "entre 5 i 7", "només aprova l'examen".
- Quan es compleix "nota de l'examen" "entre 7 i 9", "l'estudiant" "aprova l'examen amb notable".
- Quan es compleix "nota de l'examen" "9 o superior", "aprova l'examen amb excel·lent".
Finalment "l'estudiant" "rep la nota".

En aquest cas veiem que fins i tot hi ha un cas on la traducció és literal en quan a patró, l'entrada als quatre casos. De "there are 4 possible exclusive options" a "hi ha 4 possibles opcions exclusives". Es veu també que l'estructura en tots dos casos és comuna i que segueix el mateix ordre. També es pot comprovar gràcies a les cometes que el fet d'imprimir subjecte i predicat seguits serveix per als dos idiomes. Això demostra l'adaptabilitat del programa per a diferents idiomes, fent-ho fàcilment ampliable al castellà, al francès, l'alemany o altres.

8.2 Anàlisi dels resultats

Per a cadascun dels models en anglès s'han fet dues execucions, totes elles amb els paràmetres tabulació i cometes activats i la llista aplicada com a llistes sempre. En total són 20 models, dels quals 5 són els models que es van fer de proves, un és obtingut de l'hospital de Ulm, un altre un exemple del document en el qual ens basem per a fer aquest tipus d'anàlisi[7] i els 13 restants obtinguts d'un repositori públic que contenia diversos models, intentant agafar-ne la major varietat possible d'aquests.

Per a cadascun d'aquests models farem dues execucions a partir dels quals calcularem les mitjanes de frases, paraules per frase, clàusules per frase, t-unitats per frase, t-unitats complexes per frase i frase per node, comparant els valor amb els textos originals. Les t-unitats són clàusules que contenen altres clàusules simples en el seu interior, i les complexes contenen altres t-unitats.

D'altra banda, es farà una comparació de quantes activitats, esdeveniments, portes, fluxos es fan, quantes frases no tenen contingut de les anteriors i frases d'introducció, tot i que aquesta part només s'aplicarà per a els textos generats.

En la primera taula la llegenda és la següent: La O significa original i la G generat, i la resta F és frases, P/F paraules per frase, C/F clàusules per frase, T/F t-unitats per frase (clàusula amb clàusules subordinades), TC/F t-unitats complexes per frase i F/N frases per node.

En quan als models escollits, el 1 és el que fan els del document com a exemple. Del 2 al 7, excepte el 5, són els propis, el 8 és el de l'hospital i del 9 al 20, juntament amb el 5, els del repositori públic.

Model	F O	P/F O	C/F O	T/F O	TC/F O	F/N O	F G	P/F G	C/F G	T/F G	TC/F G	F/N G
1	10	10,4	1,5	0,5	0,1	0,45	14,5	14,89	1,73	1,08	0,35	0,66
2	5	5,2	1,4	0,4	0	0,45	5	8,2	1,7	0,6	0	0,45
3	23	10,78	1,74	0,39	0,09	0,46	30	13,16	1,75	0,45	0,20	0,6
4	5	5,8	1	0	0	0,56	6	8,75	1,25	0,25	0	0,67
5	10	10	1,7	0,8	0	0,67	18,5	13,3	1,92	0,57	0,25	1,23
6	6	12,5	1,83	0,17	0	0,54	8	13,44	1,63	0,5	0	0,73
7	23	10,83	1,74	0,39	0,09	0,49	106,5	9,34	1,36	0,25	0,09	2,27
8	14	19,29	1,92	0,64	0,21	0,64	15,5	9,83	1,81	0,35	0,1	0,7
9	8	10,13	1,13	0,13	0	0,25	16	11,91	1,88	1	0,31	0,8
10	8	18,5	2,63	0,75	0,25	0,31	20	11,5	1,23	0,3	0,03	0,77
11	9	13	1,22	0,11	0	0,47	14,5	12,24	1,52	0,59	0,13	0,76
12	13	17,46	1,62	0,23	0,08	0,59	10,5	9,4	1,43	0,48	0,1	0,48
13	4	7,5	1	0	0	0,33	13,5	12,39	1,48	0,48	0	1,13
14	5	8,6	1	0	0	0,56	9,5	14,13	1,9	0,56	0	1,06
15	16	15	1,25	0,19	0	0,29	61	14,15	1,55	1,33	0,23	1,11
16	5	16,4	1,4	0,6	0	0,56	6	8,25	1,25	0,33	0	0,67
17	8	12,25	1,38	0,63	0,13	0,44	15,5	11,31	1,9	0,87	0,34	0,86
18	13	12,23	1,54	0,31	0,08	0,5	24	9,65	1,29	0,31	0	0,92
19	12	13,33	1,75	0,33	0,08	0,39	26	9,02	1,31	0,35	0	0,84
20	14	15,43	1,5	0,86	0,07	0,56	21,5	8,87	1,33	0,33	0,09	0,86

Un tret que identifica els resultats són que la majoria de models tenen un nombre de frases per node comprès entre el 0,7 i l'1. L'ideal seria, però, l'1. En alguns casos, però, veiem que el nombre es passa d'1, més notòriament en el cas del 7. Això es deu a que gairebé el model sencer és un rígid, el que fa que per a alguns nodes l'algorisme passi 4 vegades, el que fa augmentar molt el nombre de frases. També és justificable l'excés en els casos on hi ha pas de missatges el que estigui per sobre d'1, ja que no es toca cap node però s'imprimeixen noves frases. La variació és segons el nombre de nodes i missatges que hi hagi.

De fet, veiem que els nodes estan millor separats, per que en la majoria de casos (exceptuant-ne dos), aquest nombre augmenta. El primer dels dos, el 2, veiem que té el mateix nombre de frases que en l'original, i es deu a ser un model bastant simple. En el cas del model 12, però, es deu a hi ha moltes portes, les quals compten com a nodes. En haver tantes portes que tanquen i finals i inicis que per les condicions del model no apareixen el ràtio és menor. Per contra, s'explica aquest model en moltes menys paraules, ja que cada frase en aquest cas té la meitat de paraules, tenint alhora menys frases.

Es veu que la tendència de paraules per frase original és de 12,23 com a mitjana. Els nostres models utilitzen, de mitjana, una paraula menys per frase, 11,19. Tot i així, no es pot veure una tendència clara. Hi ha models, com el 8, el 12 o el 16 on s'utilitzen la meitat de paraules per frase, mentre que en altres models com el 13 o el 14 les paraules per frase són gairebé el doble. Sembla, però, que depèn de la simplicitat dels models. Els models on augmenten les paraules per frase són seqüencials gairebé o bé són models molt curts. En canvi, els models 12 i 16 presenten moltes portes per flux, i el 8 divideix molt les parts en el text original.

Es pot constatar, però, que el nombre de paraules per frase és menys variable en els generats, estant entre 8 i 15 els generats, mentre que en l'original es troben entre 5 i 19 i mig. Es pot veure clarament en la variància, 4,53 dels generats per 14,88 dels originals.

En la majoria de casos, però, hi ha més frases en els generats que en els originals. En alguns casos gairebé es dobla, però degut a que les frases per node s'apropen a 1 venint de valors al voltant del 0,5, com és el cas dels tres últims models. En d'altres casos l'augment és petit degut a que no es pot separar gaire més degut a un nombre reduït de portes (2, 4 i 6) o per que té rígids de mida petita (1 i 8).

Hi ha, però, dos casos excepcionals en quan a major nombre: el model 7 multiplica per 4 i mig les frases originals i el 15 ho fa per 4. En el primer cas és interesant fer la comparativa amb el model 3. Tots dos models tenen les mateixes tasques i són gairebé idèntics, però tenen una petita diferència. El 3 té dues portes exclusives que porten a tres finals, mentre que per al 7 aquestes

dues portes són a l'inici d'un rígid complex que ocupa gairebé tot el model, el que fa que es passi 4 vegades per algunes tasques. En el cas del model 15 es deu al gran nombre de missatges que n'hi ha i també al gran nombre d'usuaris, 13 i 5 respectivament. Sense això el nombre es reduiria a 42 frases (descomptem també la introducció als missatges). També es deu a tenir un alt nombre de portes en ell.

Els rígids, doncs, poden ser exponencials. Si ens trobem de petits millora l'eficiència, però si són grans el nombre és molt més alt, demostrant doncs l'exponencialitat de tal tipus de blocs. Per tant, els rígids simples o els models amb poques portes tendiran a crear el mateix nombre de frases, mentre que rígids complexos, moltes portes o missatges augmenten molt el nombre. En cas dels models lineals la proporció d'augment és la de frases per node.

En quan a clàusules per frase no hi ha una tendència clara. En 9 models hi ha més clàusules per frase en el text original, mentre que en 11 generats hi ha més clàusules per frase que en els originals. Tot i així es pot observar que estan tots els valors entre 1,2 i 2 en la nostra. Els valors més alts corresponen als models 5, 9, 14 i 17, mentre que els més baixos són el 4, el 10 o el 16. La mitjana original és de 1,51 i la generada és superior de forma molt lleugera, 1.56. Això només pot demostrar una cosa. Que els models generats són aproximats en aquest aspecte al llenguatge natural.

El fet de voler acostar-se al llenguatge natural, però, també té contrapartides negatives. Aquestes són les clàusules que contenen altres, és a dir, les t-unitats. Es pot veure que en gran part dels models aquest nombre és superior en els generats que en els originals, i que a més a més la mitjana és superior, 0,55 els generats per 0,35 dels originals. Hi ha, però, tres casos a destacar, el 1, el 9 i el 15. En aquest últim cas l'explicació és molt senzilla: la gran quantitat de missatges. A l'hora d'escriure els missatges pot passar tres coses: que el receptor i l'emissor no tinguin nom a les tasques, que tots dos la tinguin o que només la tingui un d'ells. En el primer cas no hi haurà subordinació, en el segon n'hi haurà una de complexa i en el tercer una de simple. En 10 dels 13 missatges només té text l'emissor, i d'aquí ve també la gran diferència amb les t-unitat complexes, on el resultat és més d'1 per sota de les t-clàusules simples.

Els altres dos casos són, però, diferents d'aquest i entre ells. En el cas de l'1 es deu probablement a la complexitat del mateix model que conté molts vincles dins d'altres de forma directa, pel que després de dir-se una condició introdueix una altra part. En cas del 9, o bé del 17, que també s'acosta a 1, es deu a que el model té un nombre prou gran de missatges amb emissor i receptor respecte la mida de tasques. Degut a això, aquests tres són els que tenen també més t-unitats complexes. En aquest apartat de les t-unitats complexes, però, veiem que en alguns casos es queden a 0. Tots aquests són models que són molt lineals(18 i 19) o simples (2, 4, 6 i 16) i que, en cas que tinguin missatges, ni emissor ni receptor tenen text (13 i 14). El 10, un model que conté una alta densitat de subprocessos, també s'hi acosta molt, tenint només una t-unitat en tot el text. S'ha de dir, però, que la proporció de t-clàusules complexes respecte el total és similar en generats que en originals, 20% a 16%.

En la segona taula la llegenda A és activitats, E esdeveniments, P portes, F fluxos, N frases sense aquests continguts i I, frases d'introducció. Els models són els mateixos. S'utilitzen els mateixos models que en la taula anterior.

Model	A	E	P	F	N	I
1	100%	100%	50%	69,23%	10,48%	10,48%
2	100%	100%	50%	58,33%	0%	0%
3	100%	100%	55,56%	66,67 %	8,40%	8,40%
4	100%	100%	50%	66,67%	16,67%	16,67%
5	72,73%	71,43%	75%	61,90%	43,24%	27,19%
6	100%	100%	50%	61,54%	12,5%	12,5%
7	100%	100%	50%	65,52%	12,5%	12,5%
8	100%	100%	57,14%	73,08%	16,04%	16,04%
9	100%	100%	N/A	84,62%	31.25%	12,5%
10	100%	66.67%	100%	66,67%	40,63%	40,63%
11	100%	50%	50%	69,57%	17,38%	17,38%
12	100%	0%	50%	44%	23,64%	23,64%
13	71,43%	100%	100%	70%	46,15%	23,08%
14	57,43%	100%	N/A	71,43%	47,73%	15,34%
15	58,33%	70%	60%	72,73%	32,79%	11,48%
16	100%	100%	50%	66,67%	16,67%	16,67%
17	100%	100%	50%	76,47%	22,5%	9,58%
18	100%	100%	50%	74,19%	20,83%	20,83%
19	100%	66,67%	57,14%	76,47%	15,38%	15,38%
20	100%	100%	50%	77,78%	11,58%	11,58%

La segona taula és per fer una comparativa i extreure característiques dels models que afecten el rendiment final segons les activitats, esdeveniments, portes i fluxos de seqüència que surten. També s'han posat dos indicadors extres. Frases completes que no aporten informació directa dels processos i frase que introdueixen dins dels processos.

Comencem analitzant les activitats impreses. Es pot veure com en 16 dels 20 models s'imprimeixen el 100% de les activitats impreses, usuari inclòs en el cas que en tinguin. Tot i això hi ha quatre models (5, 13, 14 i 15) que no estan al 100%. Si ens fixem en la taula tots quatre tenen una característica comú: el valor de la penúltima columna és major que el de la última. Això significa que aquests models tenen pas de missatges.

Tot i així, no tots els models que tenen pas de missatges estan per sota del 100% (com per exemple el 9). En aquests casos es deu a que els models són tasques d'enviament o recepció que no tenen text. En el 9 totes aquestes tasques tenen text. En els dos primers casos són models relativament petits que tenen un parell de passos de missatges que no tenen text. El model 14 només té 7 activitats, de les quals en 3 no hi ha text. En el model 15, però, hi ha moltes activitats, un total de 36, de les quals 25 són enviament i recepció de missatges, de les quals només 10, totes elles d'enviament, tenen text. Això fa que els textos "perduts" siguin molts. Tot i així, no es pot considerar perdut del tot, ja que ens indica en cas que no es tingui text a qui va a parar el missatge o de qui el rep, a més a més d'ampliar-se la informació al final del text, amb tots els passos de missatge detallats.

En quan a esdeveniments, la mitjana és d'un 86.24% d'efectivitat. La idea va ser que en una seqüència s'indiqués l'inici i el final d'aquesta, a més a més de tots els esdeveniments intermedis. Hi ha casos, però, on l'inici i el final això no són aplicable, ja que és una seqüència d'un sol element. Això és el que passa en tots els casos que estan per sota del 100%.

Especialment cridanera és la situació del model 12. Aquest model té 6 esdeveniments, un d'inicial i un de final per al procés i un altre inicial i un altre final per a dos subprocessos. En tots tres casos l'únic element de la seqüència més externa és un bucle. Això fa que no es citi ni l'inici ni el fi de cap de les tres

parts. Quelcom similar passa amb els esdeveniments del model 5, però en aquest cas només afecta els dos esdeveniments del subprocés. El cas de l'11 és similar. En aquest cas l'únic que hi ha més a fora és un paral·lelisme, pel que també es perden l'esdeveniment inicial i el final. En canvi, té dos esdeveniments intermedis que sí es diuen.

El model 19 té una característica que comparteix amb la majoria de la resta. En aquest cas, té un inici i dos finals. Un d'aquests surt directe d'una porta, pel que sí que es diu. L'altre però, passa per una sola activitat abans d'arribar a la porta, pel que torna a ser una seqüència d'un element. El model 10 té característiques dels anteriors. Hi ha 15 esdeveniments, dels quals dos finals pateixen el mateix que el del model 19, mentre que tres inicials pateixen el mateix que el model 12, salvant-se només el del segon subprocés. En quan al model 15, 3 dels esdeveniments finals cauen. En dos d'ells pateixen el del model 19, mentre que el tercer es troba en una seqüència d'un sol flux.

Sobre portes només s'ha pogut fer en 18 dels models, degut a que en els models 9 i 14 no hi havia cap porta, pel que no es pot aplicar cap quantitat. En la resta de casos la mitjana és d'un 58,6%. El més ideal és apropar-se al 50%, anomenant només les portes d'entrada a un vincle, o bé només aquelles que tenen informació. De fet, hi ha diversos models on això efectivament es dona. Aquest són els que tenen la mateixa quantitat d'esdeveniments finals i inicials, en tots els casos aquí és un inici i un final, però també es podria donar en dos processos paral·lels.

Es pot veure una proporció també. Quantes més portes i menys finals més s'aproparà al 50%, i amb poques portes i molts finals s'aproparà o arribarà al 100% de portes esmentades. Hi ha, de fet, dos casos on el 100% de les portes s'esmenten. En el cas del procés 13 es deu a que hi ha una única porta, i aquesta porta a dos finals diferents. El cas del 10 és l'altra banda del raonament. Es un model que té 9 finals alternatius, amb el problema que només te 8 portes i totes elles acaben en finals diferents. L'altre cas cridaner és el 5, on les dues úniques portes dels processos principals separen finals. Baixa aquesta mitjana el fet que el subprocés té un inici i un final. En canvi, veiem

que tot i que el model 3 té tres finals alternatius és un procés que té 18 portes, pel que aquest percentatge es redueix.

Per a la majoria de models es veu com la proporció de fluxos impresos és d'un 68,68% de mitjana i que la majoria es troben al voltant del 70%. Això es deu principalment a que s'ha considerat inútil haver d'imprimir fluxos que tanquen vincles o seqüències i que no aportin cap tipus d'informació rellevant. Hi ha, però, dos casos extrems.

El model 9 imprimeix gairebé tots els fluxos, un 84%. Aquest cas extrem es deu a una causa: és un model de dos processos lineals, pel que de tots els fluxos que hi ha només s'han de descartar els dels final de la seqüència. Per al procés 14 s'aplicaria el mateix, però degut a tenir només 7 fluxos totals pels 13 del procés 9 el percentatge de fluxos no impresos és gairebé el doble. Els altres casos que superen el 75% es deuen a causes similars. Poques portes, dues o tres, comparat amb un nombre d'unes 20 tasques.

El cas clarament diferent és el 12, amb menys de la meitat dels fluxos impresos. Aquest cas reuneix moltes condicions. Té tres bucles, que alhora es troben directament després de l'inici i abans del final dels seus processos. En aquest cas tenim 8 tasques per a 8 portes. A això cal afegir que el flux de tornada de cada bucle també es perd. Els altres casos pitjors són el 2 i 6. En el primer cas hi ha molt poques tasques, algunes d'elles dins de bucles. En el cas 6 ens trobem que la meitat de les tasques es troben en branques separades, pel que els fluxos de sortida d'aquestes es perden.

Això indica que quantes més tasques seguides hi hagi més fluxos s'imprimiran i quantes més branques, bucles o finals pitjor anirà.

En quant a frases d'introducció, la mitjana es situa entorn al 16%. Com es pot veure només hi ha un cas on no hi ha frases que no aporten informació. Aquest és el model 2. En aquest cas però, ens fixarem més en la de introducció, ja que en molts casos totes les frases que no aporten informació directa són de introducció a un paràgraf de diferents alternatives. El model 2, però, segueix una única línia, tenint només dos bucles d'una sola tasca en cadascun, fent que els camins no es separin. Els altres que es troben per sota del 10% són el 3 i el

17. El cas del model 3 es deu probablement al fet que hi ha moltes frases i que en una part del model hi ha 7 tasques seguides una darrere l'altra. També hi contribueix a això l'existència de 4 bucles en tot el model. En el segon cas es tracta d'un model que és gairebé lineal. Només té un bucle. El que fa pujar el percentatge és l'existència de dues pools diferents. Com s'introdueixen també les dues pools per a separar el més possible els diferents usuaris. També s'introdueixen els passos de missatges entre usuaris.

En la banda contrària crida l'atenció el 40% de frases introductòries del model número 10. Aquest model té tres subprocessos, continguts un dins de l'altre. Aquests subprocessos, a més a més, estan just després de l'inici dels altres processos o just després d'una porta, el qual fa que el resultat sigui realment molt pitjor que en molts altres casos. També s'hi troben subprocessos en el model 5 i el 12, que són el segon i el tercer amb més frases d'introducció, tot i que aquests només en tenen un. En el cas del 5 és un model que té una proporció alta de vincles respecte les tasques, característica que al 10 passa també de forma exagerada. En el 12, en canvi, hi ha dos subprocessos just darrere de la mateixa porta, però el fet que hi hagi bucles sembla reduir el problema de masses frases introductòries. L'últim model que supera el 21% és el 13. En aquest cas té els problemes del 17 maximitzats. Té dues pools que s'introdueixen i pas de missatges. A més a més també té una porta exclusiva que fa reduir l'acumulació d'informació.

Per tant, els models tindran menys frases introductòries quan menys usuaris, portes separadores (no comptant bucles) i subprocessos tinguin, essent un nombre major en cas contrari.

A la columna del costat, en la de frases que no tenen activitat, esdeveniment, porta o flux de seqüència es pot veure que en molts casos tenen el mateix nombre que a la última. En d'altres, però, com els models 13, el 14 o el 15 es veu un increment significatiu en la primera columna respecte la segona. Això es deu al pas de missatges entre participants, que es posa al final de tot. Tot i que pot contenir textos d'activitats es posa com si no en tingués, ja que part de la informació ja era aportada pel procés en el que es troba la tasca. La única

informació que dona extra és ordinal entre la tasca d'enviament d'un usuari i la recepció de l'altre.

Tot i que aquí està posat en percentatges, es pot veure que quants més missatges respecte la mida del model més creix. El 14 és un model petit, de només 9 nodes, però conté tres passos de missatge, el que fa augmentar un 30% el nombre de frases sense informació nova respecte a les de introducció. En canvi, el 15 té moltes frases i més de 50 nodes, però té 13 intercanvis de missatges, pel que aquest increment del 20% és també significatiu, tot i que menor. El model 17 només passa dos missatges, però també es veu afectat en que és un model petit. Com es pot veure el pas de missatges és entorn a un 20% de les frases totals en els casos que s'hi passen, sense comptar la que els introdueix.

9. Possibles millores per a treballs futurs

Probablement la millora més evident de cara a un futur seria una cohesió del text encara millor. Hi ha casos on el nombre del subjecte i el predicat no encaixen. Això es deu a que aquests ja no encaixen en el model. Per exemple, si l'usuari és "l'actor" i el text "treballen al cinema", el resultat acabarà sent "l'actor" "treballen al cinema". Això es pot millorar posant-hi un programa que millori la semàntica i la sintaxi.

A vegades, fins i tot, segons el model hi ha coses que no encaixen. Hi ha un cas on la solució és "el cap de serveis de taula" "el sommelier està fent coses". Com es pot comprovar aquesta frase pot resultar confosa, i fins i tot, perd cert sentit. Això, però no és un problema tampoc lligat a la generació del text, si no a que el model buscarà sempre l'usuari a la mateixa tasca, sinó a la "lane" i sinó a la pool per a que quedi un model més complet. En aquest cas "el sommelier està fent coses" és el text de l'activitat, que es troba dins d'una "lane" que és "el cap de serveis de taula".

Un altre problema possible és a l'hora de fer textos en portes exclusives o d'esdeveniment que no tenen text. Això es deu principalment, de nou, a que el programa s'espera que tant la porta com els seus fluxos de seqüència sortints tinguin text. A vegades, però, el model no té aquesta informació. En aquest cas el que fa és posar l'ordre d'alternativa. El generador potser ja havia posat l'ordre d'alternativa per al pare, quedant dues vegades seguides el nombre de l'alternativa. Això, però, tampoc és quelcom que el programa s'espera, ja que el fill no sap quin text tindran els germans ni el pare.

La última millora futura possible seria poder ajuntar els missatges dins dels processos una mica més per a que no es posi al final de tot tots els missatges passats entre els diversos participants.

10. Execució del programa

Aquest programa només es pot executar des de un terminal Linux, posant l'ordre "java –jar Text_Generator" (sense cometes). Per a fer-ho només cal descarregar-se la carpeta comprimida que conté tots els fitxers necessaris. En ser un executable no caldrà instal·lar-ho. Es necessitarà, però, instal·lar el Petrify per a que pugui fer-se la conversió d'autòmat a BPMN.

El Petrify tampoc és un arxiu instal·lable. Simplement es descarrega l'aplicació des d'Internet (es recomana buscar per Petrify application), es descomprimeix en una carpeta a part i se li crea un PATH addicional a la consola per tal que es pugui executar que vagi dirigit a la carpeta.

Un cop instal·lat aquest programa per a executar-lo no només farà falta això, sinó que també caldrà tenir els patrons dels idiomes a seleccionar. De moment ja n'hi ha dos, en català i anglès (dataent_cat i dataent_eng, respectivament). Aquests dos arxius contenen els textos i les marques que utilitzen tots els patrons que es fan servir per a aquest programa. Es necessitarà també, òbviament, el model de processos que es vol passar a text. Aquest, però, ha d'estar en llenguatge XML. Tots ells han d'estar inclosos en la carpeta del programa. Aquest fitxer XML contindrà totes les pool, "lanes", activitats, portes, fluxos, etc. Que té el model. Un exemple de tal model es pot trobar en el quart annex.

L'últim fitxer imprescindible és el fitxer input. Si aquest fitxer no hi és hi haurà un error, ja que aquest és el fitxer on es posen els sis paràmetres d'entrada, inclosos el nom del fitxer del model a llegir. El primer paràmetre és l'idioma (Language). Per a seleccionar un idioma dels patrons cal posar una combinació de lletres que coincideixi amb els noms dels fitxers per al patró, és a dir, "eng" per a anglès i "cat" per a català. El segon paràmetre, "commas", és l'activació o desactivació de les cometes. Per a activar-les cal posar "true" i per a desactivar-les, "false". El tercer paràmetre és com es vol que surtin les llistes (list). Si es vol que surti tot amb llista cal posar "list", si es vol tot paràgrafs cal posar "paragraph", i si es prefereix una barreja, "mix". El quart paràmetre és "tabulate", per a activar o desactivar la tabulació en els textos amb llistes i

paràgrafs mixtos. Amb “true” s’activa la característica i amb “false” es desactiva. El paràmetre Petrify és bàsicament la ubicació de l’executable Petrify. Per últim el paràmetre “name” és el nom del fitxer complet que es vol executar.

Ara s’explicarà breument un exemple d’execució. El model que entra en l’aplicació és, precisament el que apareix a l’annex 4. Els fitxer de patrons, degut a la seva extensió no es mostraran aquí. Sí que es mostrarà, però, el fitxer d’entrada input, el qual és el que marcarà aquests paràmetres.

```
language eng
commas true
list list
tabulate true
petrify /home/genis/petrify/bin/petrify
name bpmn_6.xml
```

Figura 23. El fitxer d’entrada per a aquest exemple

Posteriorment s’executa el programa amb l’ordre esmentada, java –jar Text_Generator. No cal posar cap fitxer d’entrada, el programa té per defecte que llegeixi input.

```
genis@genis-Aspire-E5-571:~/text Generator$ java -jar Text_Generator.jar
```

Figura 24. Com executar el programa

Això generarà dues coses. D’una banda imprimirà per consola dues coses. D’una banda l’arbre RPST que ha generat l’aplicació. De l’altre, el text que ha acabat generant. A més a més, generarà un nou fitxer de text, anomenat output.txt, el qual tindrà només el text, ja que és el que l’usuari realment busca. Es poden veure els exemples d’això en les figures 25 i 26

```
First, "student" "does an exam". Then, "professor" "corrects the exam". Afterwards only one thing will be done from these 4 options:
    - If the case is "mark of the exam" under value "lower than 5", "student" "fails the exam".
    - If the case is "mark of the exam" under value "between 5 and 7", "student" "just passes the exam with a c".
    - When "mark of the exam" "between 7 and 9" is accomplished, "passes well the exam with a b".
    - In case the condition "mark of the exam" is "9 or higher", "student" "passes easily the exam with an a".
Finally "student" "receives mark".
```

Figura 25. El resultat en el fitxer generat output.txt

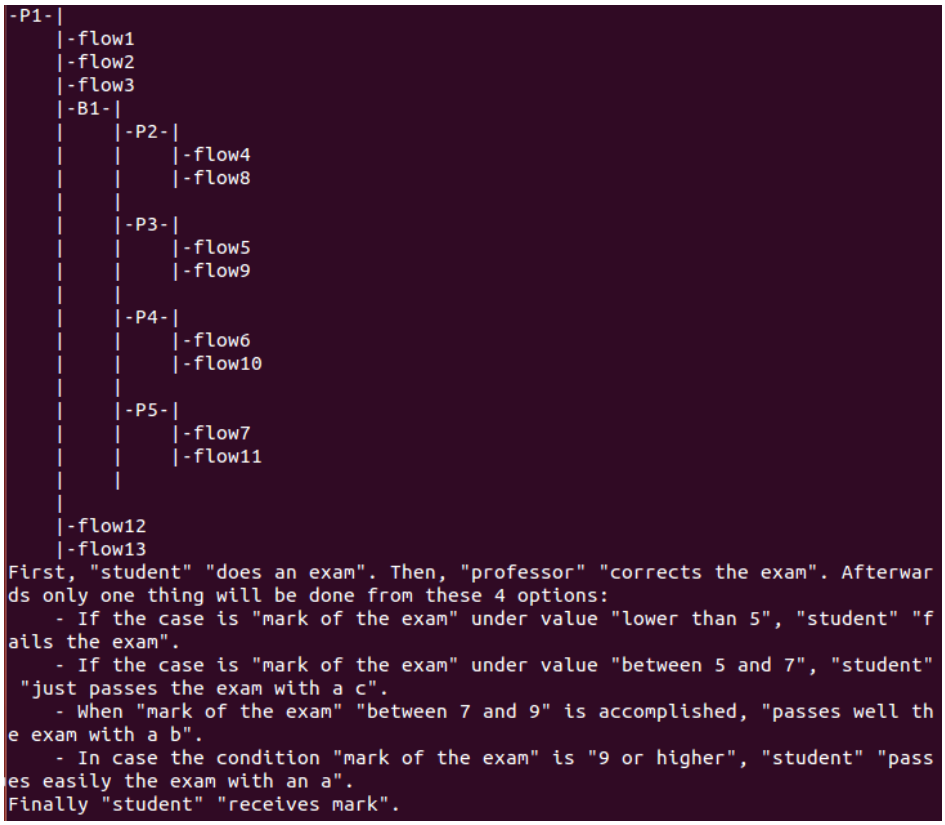


Figura 26. El que surt per la consola, observi's com l'estructura de l'arbre queda plasmada en el text.

És possible que en el primer ús després de canviar el nom del model a l'entrada el programa falli degut a que no s'hagi generat correctament la traducció de l'autòmat. La segona execució sempre funcionarà.

11. Sostenibilitat

11.1 Pressupost i sostenibilitat econòmica

11.1.1 Introducció

Degut a que es tracta d'un projecte informàtic s'havien de tenir quatre tipus de costos en compte. Aquests eren els recursos humans, hardware, software i costos indirectes. Els costos s'han tingut en compte segons el diagrama de Gantt de la planificació. El control s'ha tingut en compte tasca a tasca i per a les 751 hores que ha durat el projecte.

11.1.2 Recursos humans

Degut a que aquest projecte l'ha realitzat una sola persona, aquesta ha hagut de fer les tasques de tothom, és a dir, els 5 rols: cap de projecte, analista, dissenyador, programador i tester.

Com s'ha dit en l'apartat anterior, aquest treball ha durat 751 hores. Tota la part de la fita inicial, és a dir, tant la cerca d'informació com el curs de GEP les ha fet el cap de projecte. El cap de projecte també ha fet la fita final. Això fa un total de 142 hores com a cap de projecte, 18 menys que en l'informe anterior. L'únic que necessita conèixer l'entorn (a l'Eclipse, el XML, el Java, l'Activiti i també el Petrify) és el programador, a qui atribuirem aquestes 16 hores, 4 més que les 12 anteriors.

La part de la creació de RPST i xarxa de Petri es repartiria d'una forma similar a la que es fa en un treball de programació real, on se li atribuiria un 10% al dissenyador, un 10% a l'analista, un 60% al programador i un 20% al Beta Tester. Aquest repartiment es basa en un càlcul aproximat, ja que en la majoria de casos el programador és qui més estona treballa, seguit de lluny pel Beta Tester. Això, sobre 169 hores, són 16,9 per al dissenyador i per a l'analista, 101,4 per al programador i 33,8 per al Beta Tester.

En la creació del text el dissenyador i el programador es repartirien a parts iguals la plantilla de treball, 6 hores. En la resta de les tasques, les quals duren en total 364 hores es repartiria de la mateixa manera que en les tasques de

RPST i xarxa de Petri (36,4 per l'analista i el dissenyador, 218,4 el programador i 72,8 el Beta Tester). En total l'analista faria 36,4 hores, el dissenyador 42,4, el programador 224,4 i el Beta Tester 72,8.

En la fase final la creació del fitxer en XML, l'entrada i sortida definitives es repartirien com en els casos anteriors (10% analista, 10% dissenyador, 60% programador i 20% Beta Tester). Això sobre 12 hores són 1,2 per a l'analista i per al dissenyador, 2,4 pel Beta Tester i les 7,2 restants per al programador.

Els fitxers de proves se'ls reparteixen a parts iguals programador i dissenyador, quedant en 12 hores cadascú. Les proves (2 hores), les durà a terme el Beta Tester tot sol. L'anàlisi dels resultats el farà en solitari l'analista (10 hores).

El total final seria de 142 hores per al cap de projecte, 64,5 per a l'analista, 72,5 el dissenyador, 111 el Beta Tester i les 361 restants el programador. El salari per hora d'un cap de projecte és d'uns 50 €/hora, el d'un analista, el de programador i el de dissenyador de 35 i finalment el de Beta Tester d'uns 30 aproximadament. Tenint tot això en compte, el cap de projecte costaria 7100 €, l'analista 2257,5, el dissenyador 2537,5, el programador 12635 i el tester 3330 en el global de les 751 hores. Això és un total de 27860 € en recursos humans.

Rol	Hores	Preu/hora	Preu total
Cap de projecte	142 h	50€/h	7100 €
Analista	64,5 h	35€/h	2257,5 €
Dissenyador	72,5 h	35€/h	2537,5 €
Programador	361 h	35€/h	12635 €
Beta Tester	111 h	30€/h	3330 €
Total	751 h	37,10€/h	27860 €

11.1.3 Recursos de hardware

En hardware només s'ha tingut en compte l'equip amb el qual es treballa, ja que tot el projecte es farà amb aquest. Aquest equip és un Acer E5-571 que ha costat 399 €. Tindria un temps útil d'uns 4 anys aproximadament. Del cost de l'ordinador se n'amortitzen 80,43 €, quedant el cost final en 318,57 €. Per tant, per al temps del projecte, 25 setmanes, el cost seria de 38,16 €.

Les reparacions de hardware, en cas que es necessitessin, costarien de mitjana uns 25 €, tot i que això es tindrà en compte a imprevistos.

Component	Preu sense IVA	Preu final	Vida útil	Amortització	Cost total	Cost per 25 setmanes
Acer E5-571	329,75 €	399 €	4 anys	80,43 €	318,57 €	38,16 €
Total	329,75 €	399 €	4 anys	80,43 €	318,57 €	38,16 €

11.1.4 Recursos de software

Tot el software que s'utilitza en aquest projecte és gratuït. Això és la versió 14.10 del sistema operatiu Linux Ubuntu i del programa Eclipse en versió 4.7, que ja incorpora els llenguatges Java i XML. El plug-in Activiti s'aconsegueix també mitjançant una descàrrega gratuïta, de la mateixa manera que el Petrify i que el Camunda. Totes les actualitzacions de software, alhora, també són gratuïtes, ja que tot es tracta de software lliure. Per tant, el cost dels recursos software és negligible o 0.

11.1.5 Costos indirectes

Només hi ha dos factors a tenir en compte en quant a costos indirectes: llum i paper. El paper s'utilitzarà per a imprimir la memòria. Un paquet de 500 costarà 5 €. La memòria n'ocuparà uns 100, pel que el cost real seria de 1 €.

En quant a la llum hem de tenir en compte que es tracta d'un ordinador portàtil i, per tant, no consumeix llum durant tota l'estona d'ús. Tenint 6 hores de bateria es pot treballar sense carregar 6 hores i carregant a la llum, 2. Per tant, són 187,75 hores de consum a 0,36 kW/h. El preu d'un Kw/h és de 0,15 €, i això dóna un total de 8,10 €. Degut a que es treballa des de casa pròpia no s'ha de pagar cap tipus de lloguer.

En total en costos indirectes ens surt a 10,98 €.

	Preu/Unitat	Usos	Cost
Electricitat	0,15 €/(kW/h)	187,75 hores, 63,18kW	10,14 €
Fulls	0,01 €/full	100 fulls	1 €
Total			11,14€

11.1.6 Costos totals

Comptant els 27860 € de recursos humans, els 22,97 de hardware i els 11,14 dels costos indirectes el cost total del projecte no arriba els 28000 €, essent el total exacte de 27894,11 € de despesa.

Hi havia una probabilitat molt petita que calguessin reparacions, un cop per cada any i mig, però per si de cas se li posava el que calgués, uns 8,01 € addicionals per a una eventual reparació, que era l'eventual ~32% de probabilitats d'avaria de l'ordinador. Aquest seria el pressupost total d'imprevistos, un pressupost que al final no s'ha hagut d'utilitzar.

Finalment el temps ha estat de 751 hores, que ha superat les 702 hores de l'informe de seguiment i les 600 inicials. El pressupost total ha estat finalment de 27894,11, uns 40 € superior al pressupost de l'informe de seguiment i uns 3500 € superior a l'inicial.

Recursos gastats	Cost
Humans	27860 €
Hardware	38,16 €
Software	0 €
Indirectes	11,14 €
Imprevistos	0 €
Total	27894,11 €

El cost del projecte ha estat superior a l'inicial, degut a que es va decidir començar abans aquest projecte, tenint 88 hores més i haver hagut de fer 63 hores més al final. Això ha fet augmentar una mica el cost, dels 24500 aproximats inicials, els costos han sobrepassat el pressupost inicial en un 13,9% major, sobretot degut a les hores addicionals. Respecte al pressupost de la fita de seguiment, però, la diferència és minsa, d'uns 40 €, essent el cost final només un 0,14 % superior, demostrant la precisió de l'anterior pressupost.

11.1.7 Control de gestió

Tots els problemes han estat temporals, i com bé es va suposar, no hi ha hagut problemes de hardware ni de software. El paper tampoc ha representat cap

problema, perquè només ha calgut fer la memòria un sol cop, pel que ens hem estalviat 4 € en paper.

Tot i això, el temps ha estat més llarg del previst i es surt lleugerament de l'últim pressupost. Tot i així, com ja eren casos extrems, la diferència ha estat molt petita i per tant ha estat assumible.

Donats els reduïts costos en hardware i indirectes i el nul cost en software, aquest projecte no podia ser menys costós, perquè els recursos humans són el 99,88% de la despesa. Es pogués haver fet en menys temps en cas que ho hagués fet més d'una persona i ser menys costós degut a que es gastaria el mateix en recursos humans per hora, ja que es treballaria menys temps amb la mateixa despesa per persona, reduint-ne doncs els costos. Però com això no s'ha donat degut a que només ho feia una persona, ha estat impossible reduir més aquests costos i temps.

11.2. Sostenibilitat social

El sector de la informàtica viu en ple auge degut al creixement exponencial de les tecnologies informatitzades i de la telefonia mòbil. Aquest projecte pot, a més a més, ajudar una mica més a aquesta situació. Hi ha certa necessitat d'un projecte com aquest, ja que en aquest moment no hi ha massa eines que una persona sense coneixements matemàtics pugui utilitzar per a transformar un BPMN en text.

Això pot millorar la qualitat de vida dels que ho utilitzin sense tenir coneixements matemàtics, i també ajudarà a gastar menys paper ja que s'imprimiria en un format text llegible i a facilitar la feina dels departaments de recursos humans. Hi hauria un únic col·lectiu que es veuria lleugerament perjudicat: la indústria paperera, ja que no caldria utilitzar paper per a organitzar l'empresa que utilitzi aquesta eina. Tot i això, estaria limitat a l'interès de les empreses, ja que seria poc útil per a particulars.

11.3. Sostenibilitat ambiental

La incidència ambiental d'aquest projecte és realment mínima, ja que l'únic que utilitzem en tractar-se d'un projecte de software són els costos indirectes. Aquests són la llum i el paper. Tots dos deixen poca petjada ecològica, tot i que no inexistent.

Es necessita, però el hardware com a element manufacturat previ. Tot i així, l'ordinador és d'una de les companyies més respectuoses amb el medi ambient i, alhora, una de les més ètiques. Aquest, però, seria l'únic problema mediambiental, ja que s'utilitzen diversos recursos, alguns en gran quantitat. Té un avantatge, però, i és que es pot reciclar el hardware en la seva pràctica totalitat.

L'impacte i consum serien gairebé els mateixos sense el TFG. A posteriori el consum serà 0, ja que no caldrà gastar cap recurs nou i, a més a més, els usuaris estalviaran paper, pel que el paper utilitzat per a la memòria quedaria compensat per la no utilització futura. L'ús de CO2 per a tot el projecte seria el de la creació de la llum, del paper i la fabricació de l'ordinador.

Degut a que és software només es genera la contaminació de la generació de llum i de la fabricació de paper en el desenvolupament del projecte i no tindrà cap impacte mediambiental després de la seva vida útil per la mateixa causa, i tampoc tindria efecte en la petjada ecològica. A més a més, d'aquest projecte se'n podrà reutilitzar o reciclar en altres projectes la totalitat, i el mateix passa amb aquest projecte respecte altres projectes anteriors.

11.4. Puntuacions de sostenibilitat

	Econòmica	Social	Ambiental
Sostenibilitat	7	5	8

11.4.1 Raonament de les puntuacions

L'alta sostenibilitat econòmica es deu al baix cost que suposa un projecte de software. Només és millorable en quant a mà d'obra, ja que la millor opció són tres empleats, amb un menor cost laboral que compensa l'increment del preu

de la llum. La puntuació és lleugerament inferior en relació a les fites anteriors degut a que finalment el projecte ha sobrepassat els costos inicials degut al temps un 20% superior que ha tingut. Tot i això només ha empitjorat lleugerament la previsió, ja que no arriba als 28000€.

En quant a social, si bé és cert que seria una eina que ajudaria molt a aquelles empreses en les quals la gent de Recursos Humans no té coneixements matemàtics ni tècnics, és un projecte que potser està una mica fora de l'abast dels particulars o persones fora del món empresarial. Per tant, dependria de l'interès de les empreses, les quals sí que veurien una millora en la seva eficiència.

En quant a l'ambiental, tret de la fabricació de l'ordinador, els recursos exigits són mínims. Però la fabricació de l'ordinador ja fa baixar una mica la seva puntuació degut al gran nombre de recursos que es necessiten i l'afegit de la contaminació que se'n genera. Tret d'això, és un projecte que fins i tot reduiria el consum de paper, pel que tindria una contrapartida positiva.

12. Identificació de lleis i regulacions

Degut a que és un software que és sense intenció de comercialitzar, sinó més bé amb un caire acadèmic i amb una intenció d'investigació, no hi ha cap regulació o llei que reguli els projectes acadèmics en quant a competència amb empreses, i els únics problemes que podria tenir aquest projecte podrien ser el plagi i la corrupció de software exterior, el qual pot ser un tema legal d'importància.

En l'aspecte del plagi, no existeix la possibilitat de plagi per tres motius principals. El primer, sempre es cita les fonts de les quals n'he extret la informació necessària, sobre la qual s'ha estudiat per a veure quina era la millor alternativa, a més a més de proposar variacions sobre aquests estudis anteriors. En segon lloc, tot i agafar els conceptes bàsics d'aquestes publicacions i posar-ne alguns en pràctica, el plantejament de les solucions finals i la implementació del codi és íntegrament pròpia, pel que no existiria la possibilitat de plagi. Per últim, els exemples són una barreja entre exemples fets per mi mateix i els extrets d'una font pública, però fets al Camunda a partir dels originals, per tal de poder-los entrar com a XML.

En el cas de la corrupció de software, aquest projecte no hauria d'afectar de cap manera el software extern si té la seva pròpia carpeta. Això es deu a que aquest programa agafarà dos fitxers amb un nom concret i en crearà tres dins de la carpeta del programa, també amb un nom concret. Només hi hauria problemes en cas que un fitxer a la mateixa carpeta tingués el mateix nom que l'autòmat creat, el del BPMN sorgit a partir de l'autòmat o el nom del fitxer de sortida, on es podria arribar a esborrar l'arxiu original. Això, però, passaria en cas d'ús indegut, pel que no em responsabilitzo dels problemes que aquest programa pogués causar.

Si finalment es decidís que aquest projecte acabés convertint-se en un producte comercial en un futur s'haurien de tenir en compte les empreses que es veurien perjudicades. En aquest cas només seria la indústria paperera. Per tant, caldria veure quin seria el prejudici que tindrien aquestes empreses. En principi, però, l'impacte no deuria de ser gaire gran, però igualment s'hauria de

mirar de no fer competència deslleial. També s'hauria d'aconseguir una llicència per a poder comercialitzar-ho. Probablement, com aquest producte no seria software lliure, s'hauria de donar una petita compensació a aquells dels quals se'n han extret algunes idees, tot i que el preu d'aquesta eina seria relativament baix. El producte estaria així, doncs, dins de la legalitat.

Tot i així, la probabilitat de convertir-se en un producte comercial és més aviat minsa, pel que probablement el paràgraf anterior no s'arribi a aplicar. De fet, la idea és també que s'hi puguin fer millores a partir d'aquest projecte, o bé extreure'n informació.

13. Bibliografia

[1] https://en.wikipedia.org/wiki/Natural_language_processing

[2] https://en.wikipedia.org/wiki/Program_structure_tree

[3] https://en.wikipedia.org/wiki/Petri_net

[4] Lavoie B., Rambow O., Reiter E.: The ModelExplainer (1996)

[5] https://es.wikipedia.org/wiki/Modelo_de_objeto

[6] Meziane F., Athanasakis N., Ananiadou S.: Generating Natural specifications from UML class diagrams (2011)

[7] Leopold H., Mendling J., Polyvyanyy A.: Supporting Process Model Validation through Natural Language Generation (2014)

[8] Friedrich F., Mendling J., Puhlmann F.: Process Model Generation from Natural Language Text (2011)

[9] https://en.wikipedia.org/wiki/Finite-state_machine

Annex 1. Pseudocodi de la funció de creació del text

```
printChildren (node actual, condicions anteriors, ordre de node amb text respecte el pare, germans totals) {
  m = 0; //ordre dels fills amb text
  ArrayList<String> t = []; // textos propis i/o dels fills
  Seps = []; //separació de les frases d'un inici amb tasca de recepció
  Act = false; //és activitat
  Si és un vincle i té més de 4 fills programa per a més de 4 fills, si no aleatori entre 4 i més de 4.
  Si el primer flux surt d'una tasca receptora{
     Funció de primera tasca receptora, que canvia el valor seps[]
     Afegeix el text que surt a t.
  }
  Si (el node és rígid) {
    r = valor que torna de la funció de rígids
    afegix r al vector t.
  }
  En cas contrari {
     Si (és node fulla (o flux)) {
       Si (el final del flux és una tasca d'usuari) {
         Afegeix el text de la tasca a t.
         Si la tasca té usuari, aquest s'afegeix a t.
         Si la tasca no té usuari però té line, s'afegeix el nom de la line a t.
         Si tampoc té line però té pool, s'afegeix el nom de la pool a t.
       }
       Si (el final del flux és una tasca) {
         Afegeix el text de la tasca a t.
         Si la tasca té line, s'afegeix el nom de la line a t.
         Si no té line però té pool, s'afegeix el nom de la pool a t.
       }
       Si (el final del flux és una tasca de recepció) {
         Si la tasca té text, l'afegeix a t.
         Si no, busca les tasques d'enviament que li envien un missatge a l'actual i les afegeix totes a t, modificant seps durant el procés
         Si la tasca té line, s'afegeix el nom de la line a t.
         Si tampoc té line però té pool, s'afegeix el nom de la pool a t.
       }
       Si (el final del flux és una tasca d'enviament) {
         Si la tasca té text, l'afegeix a t.
         Si la tasca té line, s'afegeix el nom de la line a t.
         Si tampoc té line però té pool, s'afegeix el nom de la pool a t.
       }
       Si (el final del flux és un esdeveniment intermedi) {
         Afegeix el text de l'esdeveniment a t.
       }
     }
     Si (no és node fulla) {
       Per a cada fill {
         Si (el fill no és un flux de tornada a bucle) {
```

```
          Si (el fill és un polígon d'una porta paral·lela) {
            l'assigna la porta com a més o menys de 4 per als patrons si no ho
esta
            Si (és flux no final o bloc) {
              Si (el fill és un vincle exclusiu, inclusiu o d'esdeveniment) {
                String Cond = la condició de la porta;
                S = printChildren (node fill, cond, m, germans del node fill);
                Afegeix s a t
              }
              Si no {
                S = printChildren (node fill, "", m, germans del node fill);
                Afegeix s a t
              }
              m++;
            }
            Si (el fill és flux, té un germà i el pare és vincle) {
                String Cond = la condició de la porta;
                S = printChildren (node fill, cond, m, germans del node fill);
            }
           Si (el fill és fill únic, flux, comença en porta exclusiva o d'esdeveniment
i acaba en final) {
                S = printChildren (node fill, "", m, germans del node fill);
                Afegeix s a t
          }
        }
      }
    }
  Fusion ()
 }
```

Annex 2. Patrons per a les portes paral·leles

```
PATTERN PARALLEL
BEGPT NN LISTN LISTY
$bHi ha $s coses a fer al mateix temps:
 $c

ENDPT
BEGPT NN LISTN LISTY
$bAquestes $s coses es faran alhora:
 $c

ENDPT
BEGPT NN LISTN LISTY
$bAquestes $s coses s'haurien de fer simultàniament:
 $c

ENDPT
BEGPT N2 LISTY
$bDos grups d'activitats es fan en paral·lel:
$l El primer grup, $1. $e
$l El segon grup, $2. $e

ENDPT
BEGPT N2 LISTY
$bHi ha dues activitats paral·leles:
$l Al primer grup $1. $e
$l Al segon grup $2. $e

ENDPT
BEGPT N2 LISTY
$bAquests dos grups es fan alhora:
$l El primer grup de tasques, $1. $e
$l El segon grup de tasques, $2. $e

ENDPT
BEGPT N2 LISTN
$bDos grups d'activitats es fan en paral·lel:
$t El primer grup, $1. $e
$t El segon grup, $2. $e

ENDPT
BEGPT N2 LISTN
$bHi ha dues activitats paral·leles:
$t Al primer grup $1. $e
$t Al segon grup $2. $e

ENDPT
BEGPT N2 LISTN
```

$bAquests dos grups es fan alhora:
$l El primer grup de tasques, $1. $e
$l El segon grup de tasques, $2. $e

ENDPT
BEGPT N3 LISTY
$bLes 3 branques paral·leles a fer sÃ³n:
$l La primera branca, $1. $e
$l La segona branca, $2. $e
$l La tercera branca, $3. $e

ENDPT
BEGPT N3 LISTY
$bTres grups d'activitats es fan al mateix temps:
$l El primer, $1. $e
$l El segon, $2. $e
$l El tercer, $3. $e

ENDPT
BEGPT N3 LISTY
$bAquests tres grups es fan en paral·lel:
$l El primer grup, $1. $e
$l El segon grup, $2. $e
$l El tercer i Últim grup, $3. $e

ENDPT
BEGPT N3 LISTN
$bLes 3 branques paral·leles a fer sÃ³n:
$t La primera branca, $1. $e
$t La segona branca, $2. $e
$t La tercera branca, $3. $e

ENDPT
BEGPT N3 LISTN
$bTres grups d'activitats es fan al mateix temps:
$t El primer, $1. $e
$t El segon, $2. $e
$t El tercer, $3. $e

ENDPT
BEGPT N3 LISTN
$bAquests tres grups es fan en paral·lel:
$t El primer grup, $1. $e
$t El segon grup, $2. $e
$t El tercer i Últim grup, $3. $e

ENDPT
BEGPT N4 LISTY
$bHi ha 4 grups paral·lels de tasques:
$l El primer grup, $1. $e

$l El segon grup, $2. $e
$l El tercer, $3. $e
$l L'Últim, $4. $e

ENDPT
BEGPT N4 LISTY
$bLes 4 branques paral·leles a fer sÃ³n:
$l La primera branca, $1. $e
$l La segona branca, $2. $e
$l La tercera branca, $3. $e
$l La quarta branca, $4. $e

ENDPT
BEGPT N4 LISTY
$bEs fan quatre grups paral·lels:
$l El grup 1 de tasques paral·leles, $1. $e
$l El segon grup paral·lel, $2. $e
$l El tercer paral·lel, $3. $e
$l La Última conjunciÃ³ paral·lela, $4. $e

ENDPT
BEGPT N4 LISTN
$bHi ha 4 grups paral·lels de tasques:
$t El primer grup, $1. $e
$t El segon grup, $2. $e
$t El tercer, $3. $e
$t L'Últim, $4. $e

ENDPT
BEGPT N4 LISTN
$bLes 4 branques paral·leles a fer sÃ³n:
$t La primera branca, $1. $e
$t La segona branca, $2. $e
$t La tercera branca, $3. $e
$t La quarta branca, $4. $e

ENDPT
BEGPT N4 LISTN
$bEs fan quatre grups paral·lels:
$t El grup 1 de tasques paral·leles, $1. $e
$t El segon grup paral·lel, $2. $e
$t El tercer paral·lel, $3. $e
$t La Última conjunciÃ³ paral·lela, $4. $e

ENDPT
ENDPATTERN

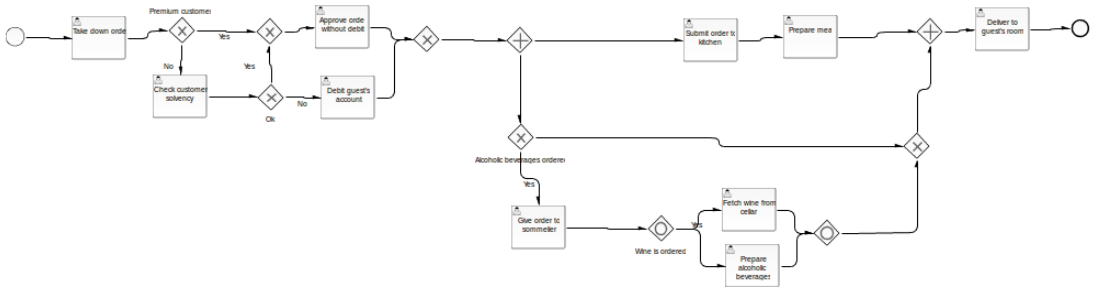
Annex 3. Models utilitzats

Aquí es poden veure els models que s'han utilitzat en els exemples de l'apartat 9.1.

Text similar a l'original del model 1:

At a restaurant, the room-service manager takes down order. If the guest is a premium customer, the room-service manager approves order without debit. If not, he checks the customer's solvency. If the solvency is Ok, the room-service manager approves order without debit. If not, he debits guest's account. Then, the room-service manager submits the order to the kitchen, who prepares meal. At the same time, if there are alcoholic beverages ordered, he gives an order to sommelier. The sommelier, then, prepares the alcoholic beverages, and, if wine is ordered, fetches wine from cellar. When both kitchen and sommelier have finished, the waiter delivers to guest's room.

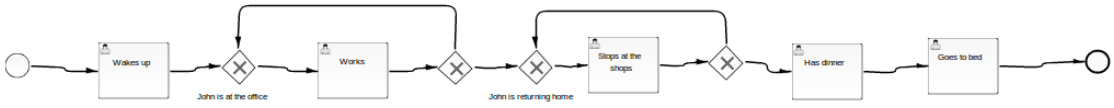
Model 1:



Text similar a l'original en anglès del model 2:

At the morning, John wakes up. He works while he is at the office. John stops at the shops while he is returning home. Then, he has dinner and after that, he goes to the bed.

Model 2:

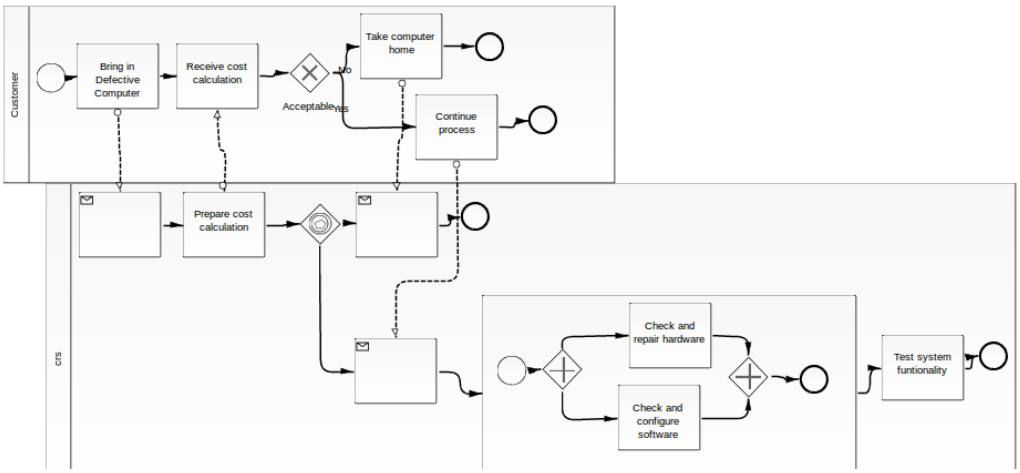


Text original del model 5:

A customer brings in a defective computer and the CRS checks the defect and hands out a repair cost calculation back. If the customer decides that the costs are acceptable, the process continues, otherwise she takes her computer home unrepaired. The ongoing repair consists of two activities, which are executed, in an arbitrary order. The first activity is to check and repair the hardware, whereas the second activity checks and configures the software. After each of these

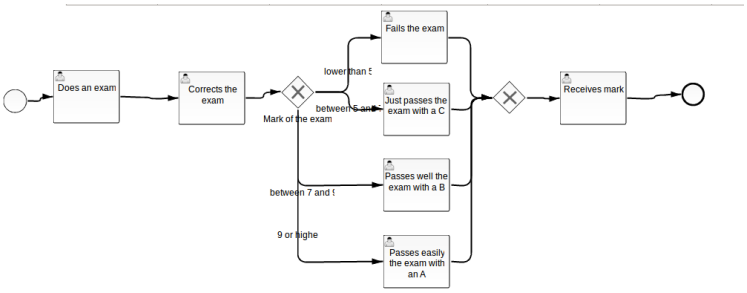
activities, the proper system functionality is tested. If an error is detected another arbitrary repair activity is executed, otherwise the repair is finished.

Model 5:



Text similar a l'original en anglès del model 6:

When a student has done an exam, his teacher corrects it. If the mark is lower than 5 the student has failed the exam. If the mark is between 5 and 7, the student just passes the exam with a C. If the mark is between 7 and 9 the student passes well the exam with a B. In case the mark is 9 or higher, the student passes easily the exam with an A.
Model 6:

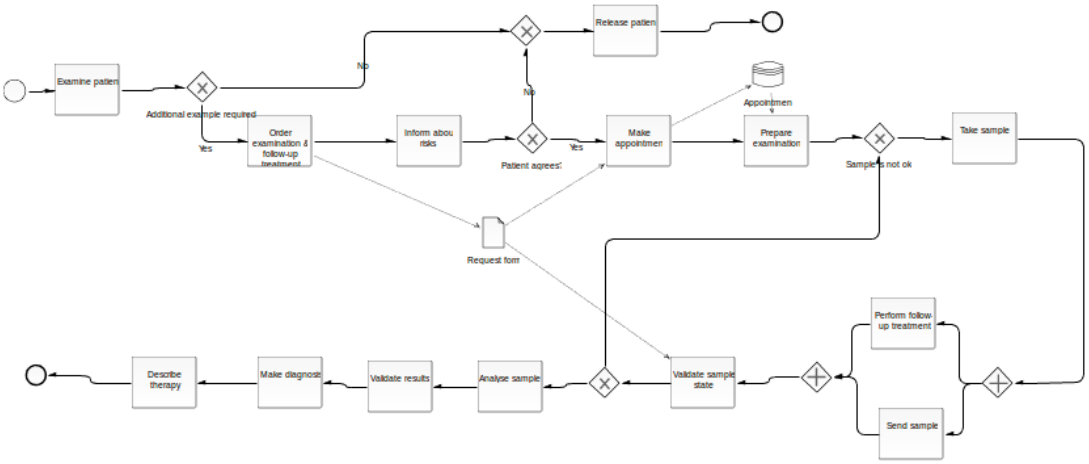


Text original del model 8:

The examination process can be sumarised as follows. The process starts when the female patient is examined by an outpatient physician, who decides whether she is healthy or needs to undertake an additional examination. In the former case, the physician fills out the examination form and the patient can leave. In the latter case, an examination and follow-up treatment order is placed by the physician who additionally fills out a request form. Beyond information about the patient, the request form includes details about the examination requested and refers to a suitable lab. Furthermore, the outpatient physician informs the patient about potential risks. If the patient signs an informed consent and agrees to continue with the procedure, a delegate of the physician arranges an appointment of the patient with one of the wards. The latter is then responsible for taking a sample to be analyzed in the lab later. Before the appointment, the required examination and sampling is prepared by a nurse of the ward based on the information provided by the outpatient section. Then, a

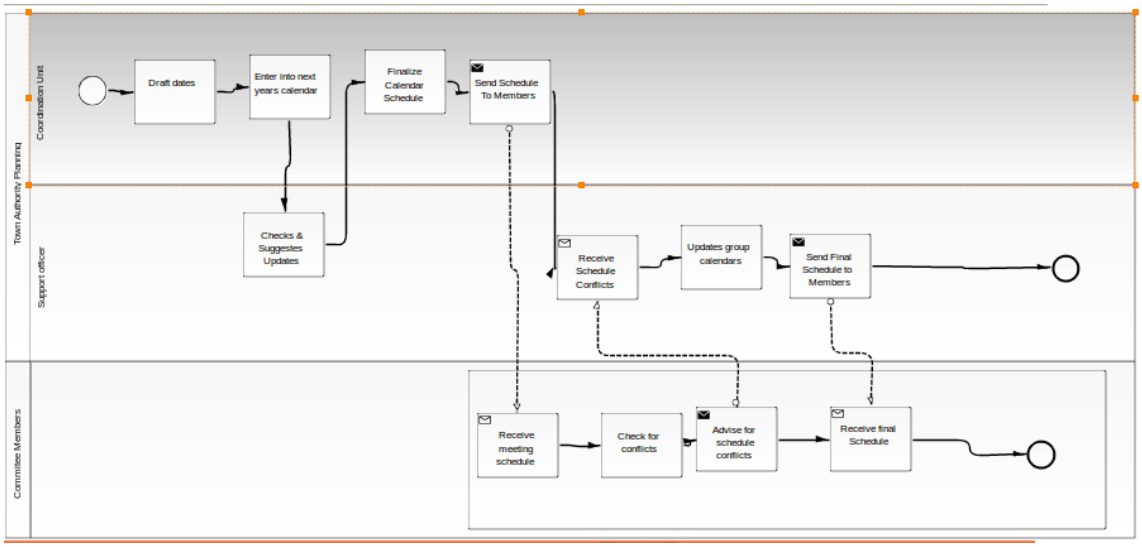
ward physician takes the sample requested. He further sends it to the lab indicated
in the request form and conducts the follow-up treatment of the patient.
After receiving the sample, a physician of the lab validates its state and decides whether the sample can be used for analysis or whether it is contaminated and a new sample is required. After the analysis is performed by a medical technical assistant of the lab, a lab physician validates the results. Finally, a physician from the outpatient department makes the diagnosis and prescribes the therapy for the patient.

Model 8:



Text original del model 9:

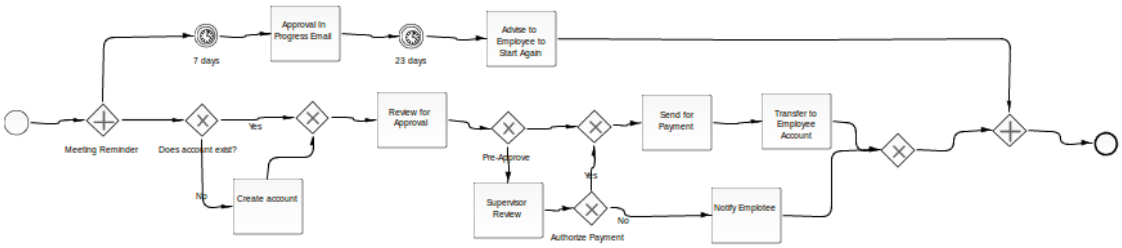
In November of each year, the Coordination Unit at the Town Planning authority draft a Schedule of meetings for the next calendar year and adds draft dates to all calendars. The Support Officer then checks the dates and suggests modifications. The Coordination Unit then rechecks all dates and looks for potential conflicts. The final schedule of meeting dates is sent to all the independent Committee Members by email, who then check their diaries and advise the Coordination Unit of any conflicts
Model 9:



Text original del model 11:

After the Expense Report is received, a new account must be created if the employee does not already have one. The report is then reviewed for automatic approval. Amounts under $200 are automatically approved, whereas amounts equal to or over $200 require approval of the supervisor. In case of rejection, the employee must receive a rejection notice by email. Otherwise, the reimbursement goes to the employee's direct deposit bank account. If the request is not completed in 7 days, then the employee must receive an "approval in progress" email. If the request is not finished within 30 days, then the process is stopped and the employee receives an email cancellation notice and must re-submit the expense report.

Model 11:



Annex 4. Exemple d'entrada d'un model

```
<?xml version="1.0" encoding="UTF-8"?>
<bpmn2:definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:camunda="http://activiti.org/bpmn"
xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
BPMN20.xsd" id="_h8aaYIhMEeWdgsG1wkHdeg" exporter="camunda
modeler" exporterVersion="2.7.0" targetNamespace="http://activiti.org/bpmn">
  <bpmn2:process id="Process_1" isExecutable="false">
    <bpmn2:startEvent id="StartEvent_1">
      <bpmn2:outgoing>flow1</bpmn2:outgoing>
    </bpmn2:startEvent>
    <bpmn2:userTask id="uTask1" camunda:assignee="Student" name="Does
an exam">
      <bpmn2:incoming>flow1</bpmn2:incoming>
      <bpmn2:outgoing>flow2</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:userTask id="uTask2" camunda:assignee="Professor"
name="Corrects the exam">
      <bpmn2:incoming>flow2</bpmn2:incoming>
      <bpmn2:outgoing>flow3</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:exclusiveGateway id="eGate1" name="Mark of the exam">
      <bpmn2:incoming>flow3</bpmn2:incoming>
      <bpmn2:outgoing>flow4</bpmn2:outgoing>
      <bpmn2:outgoing>flow5</bpmn2:outgoing>
      <bpmn2:outgoing>flow6</bpmn2:outgoing>
      <bpmn2:outgoing>flow7</bpmn2:outgoing>
    </bpmn2:exclusiveGateway>
    <bpmn2:userTask id="uTask3" camunda:assignee="Student" name="Fails
the exam">
      <bpmn2:incoming>flow4</bpmn2:incoming>
      <bpmn2:outgoing>flow8</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:userTask id="uTask5" name="Passes well the exam with a B">
      <bpmn2:incoming>flow6</bpmn2:incoming>
      <bpmn2:outgoing>flow10</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:userTask id="uTask4" camunda:assignee="Student" name="Just
passes the exam with a C">
      <bpmn2:incoming>flow5</bpmn2:incoming>
      <bpmn2:outgoing>flow9</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:userTask id="uTask6" camunda:assignee="Student" name="Passes
easily the exam with an A">
      <bpmn2:incoming>flow7</bpmn2:incoming>
```

```
      <bpmn2:outgoing>flow11</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:exclusiveGateway id="ExclusiveGateway_2">
      <bpmn2:incoming>flow8</bpmn2:incoming>
      <bpmn2:incoming>flow9</bpmn2:incoming>
      <bpmn2:incoming>flow10</bpmn2:incoming>
      <bpmn2:incoming>flow11</bpmn2:incoming>
      <bpmn2:outgoing>flow12</bpmn2:outgoing>
    </bpmn2:exclusiveGateway>
    <bpmn2:userTask id="uTask7" camunda:assignee="Student"
name="Receives mark">
      <bpmn2:incoming>flow12</bpmn2:incoming>
      <bpmn2:outgoing>SequenceFlow_13</bpmn2:outgoing>
    </bpmn2:userTask>
    <bpmn2:endEvent id="end1">
      <bpmn2:incoming>SequenceFlow_13</bpmn2:incoming>
    </bpmn2:endEvent>
    <bpmn2:sequenceFlow id="flow1" name="" sourceRef="StartEvent_1"
targetRef="uTask1"/>
    <bpmn2:sequenceFlow id="flow2" name="" sourceRef="uTask1"
targetRef="uTask2"/>
    <bpmn2:sequenceFlow id="flow3" name="" sourceRef="uTask2"
targetRef="eGate1"/>
    <bpmn2:sequenceFlow id="flow4" name="lower than 5" sourceRef="eGate1"
targetRef="uTask3"/>
    <bpmn2:sequenceFlow id="flow5" name="between 5 and 7"
sourceRef="eGate1" targetRef="uTask4"/>
    <bpmn2:sequenceFlow id="flow6" name="between 7 and 9"
sourceRef="eGate1" targetRef="uTask5"/>
    <bpmn2:sequenceFlow id="flow7" name="9 or higher" sourceRef="eGate1"
targetRef="uTask6"/>
    <bpmn2:sequenceFlow id="flow8" name="" sourceRef="uTask3"
targetRef="ExclusiveGateway_2"/>
    <bpmn2:sequenceFlow id="flow9" name="" sourceRef="uTask4"
targetRef="ExclusiveGateway_2"/>
    <bpmn2:sequenceFlow id="flow10" name="" sourceRef="uTask5"
targetRef="ExclusiveGateway_2"/>
    <bpmn2:sequenceFlow id="flow11" name="" sourceRef="uTask6"
targetRef="ExclusiveGateway_2"/>
    <bpmn2:sequenceFlow id="flow12" name=""
sourceRef="ExclusiveGateway_2" targetRef="uTask7"/>
    <bpmn2:sequenceFlow id="SequenceFlow_13" name=""
sourceRef="uTask7" targetRef="end1"/>
  </bpmn2:process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_1">
    <bpmndi:BPMNPlane id="BPMNPlane_1" bpmnElement="Process_1">
      <bpmndi:BPMNShape id="_BPMNShape_StartEvent_7"
bpmnElement="StartEvent_1">
        <dc:Bounds height="36.0" width="36.0" x="30.0" y="118.0"/>
      </bpmndi:BPMNShape>
```

```
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_61"
bpmnElement="uTask1">
      <dc:Bounds height="80.0" width="100.0" x="105.0" y="89.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_62"
bpmnElement="uTask2">
      <dc:Bounds height="80.0" width="100.0" x="293.0" y="92.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_ExclusiveGateway_2"
bpmnElement="eGate1" isMarkerVisible="true">
      <dc:Bounds height="50.0" width="50.0" x="446.0" y="97.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_63"
bpmnElement="uTask3">
      <dc:Bounds height="80.0" width="100.0" x="596.0" y="0.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_64"
bpmnElement="uTask4">
      <dc:Bounds height="80.0" width="100.0" x="599.0" y="108.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_65"
bpmnElement="uTask5">
      <dc:Bounds height="80.0" width="100.0" x="599.0" y="222.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_66"
bpmnElement="uTask6">
      <dc:Bounds height="80.0" width="100.0" x="599.0" y="336.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_ExclusiveGateway_3"
bpmnElement="ExclusiveGateway_2" isMarkerVisible="true">
      <dc:Bounds height="50.0" width="50.0" x="763.0" y="105.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_UserTask_67"
bpmnElement="uTask7">
      <dc:Bounds height="80.0" width="100.0" x="864.0" y="92.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNShape id="_BPMNShape_EndEvent_7"
bpmnElement="end1">
      <dc:Bounds height="36.0" width="36.0" x="1046.0" y="108.0"/>
    </bpmndi:BPMNShape>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_1"
bpmnElement="flow1" sourceElement="_BPMNShape_StartEvent_7"
targetElement="_BPMNShape_UserTask_61">
      <di:waypoint xsi:type="dc:Point" x="66.0" y="136.0"/>
      <di:waypoint xsi:type="dc:Point" x="85.0" y="136.0"/>
      <di:waypoint xsi:type="dc:Point" x="85.0" y="129.0"/>
      <di:waypoint xsi:type="dc:Point" x="105.0" y="129.0"/>
    </bpmndi:BPMNEdge>
```

```
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_2"
bpmnElement="flow2" sourceElement="_BPMNShape_UserTask_61"
targetElement="_BPMNShape_UserTask_62">
      <di:waypoint xsi:type="dc:Point" x="205.0" y="129.0"/>
      <di:waypoint xsi:type="dc:Point" x="249.0" y="129.0"/>
      <di:waypoint xsi:type="dc:Point" x="249.0" y="132.0"/>
      <di:waypoint xsi:type="dc:Point" x="293.0" y="132.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_3"
bpmnElement="flow3" sourceElement="_BPMNShape_UserTask_62"
targetElement="_BPMNShape_ExclusiveGateway_2">
      <di:waypoint xsi:type="dc:Point" x="393.0" y="132.0"/>
      <di:waypoint xsi:type="dc:Point" x="419.0" y="132.0"/>
      <di:waypoint xsi:type="dc:Point" x="419.0" y="122.0"/>
      <di:waypoint xsi:type="dc:Point" x="446.0" y="122.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_4"
bpmnElement="flow4" sourceElement="_BPMNShape_ExclusiveGateway_2"
targetElement="_BPMNShape_UserTask_63">
      <di:waypoint xsi:type="dc:Point" x="496.0" y="122.0"/>
      <di:waypoint xsi:type="dc:Point" x="546.0" y="122.0"/>
      <di:waypoint xsi:type="dc:Point" x="546.0" y="40.0"/>
      <di:waypoint xsi:type="dc:Point" x="596.0" y="40.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_5"
bpmnElement="flow5" sourceElement="_BPMNShape_ExclusiveGateway_2"
targetElement="_BPMNShape_UserTask_64">
      <di:waypoint xsi:type="dc:Point" x="496.0" y="122.0"/>
      <di:waypoint xsi:type="dc:Point" x="547.0" y="122.0"/>
      <di:waypoint xsi:type="dc:Point" x="547.0" y="148.0"/>
      <di:waypoint xsi:type="dc:Point" x="599.0" y="148.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_6"
bpmnElement="flow6" sourceElement="_BPMNShape_ExclusiveGateway_2"
targetElement="_BPMNShape_UserTask_65">
      <di:waypoint xsi:type="dc:Point" x="471.0" y="147.0"/>
      <di:waypoint xsi:type="dc:Point" x="471.0" y="262.0"/>
      <di:waypoint xsi:type="dc:Point" x="599.0" y="262.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_7"
bpmnElement="flow7" sourceElement="_BPMNShape_ExclusiveGateway_2"
targetElement="_BPMNShape_UserTask_66">
      <di:waypoint xsi:type="dc:Point" x="471.0" y="147.0"/>
      <di:waypoint xsi:type="dc:Point" x="471.0" y="376.0"/>
      <di:waypoint xsi:type="dc:Point" x="599.0" y="376.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_8"
bpmnElement="flow8" sourceElement="_BPMNShape_UserTask_63"
targetElement="_BPMNShape_ExclusiveGateway_3">
      <di:waypoint xsi:type="dc:Point" x="696.0" y="40.0"/>
```

```
      <di:waypoint xsi:type="dc:Point" x="729.0" y="40.0"/>
      <di:waypoint xsi:type="dc:Point" x="729.0" y="130.0"/>
      <di:waypoint xsi:type="dc:Point" x="763.0" y="130.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_9"
bpmnElement="flow9" sourceElement="_BPMNShape_UserTask_64"
targetElement="_BPMNShape_ExclusiveGateway_3">
      <di:waypoint xsi:type="dc:Point" x="699.0" y="148.0"/>
      <di:waypoint xsi:type="dc:Point" x="731.0" y="148.0"/>
      <di:waypoint xsi:type="dc:Point" x="731.0" y="130.0"/>
      <di:waypoint xsi:type="dc:Point" x="763.0" y="130.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_10"
bpmnElement="flow10" sourceElement="_BPMNShape_UserTask_65"
targetElement="_BPMNShape_ExclusiveGateway_3">
      <di:waypoint xsi:type="dc:Point" x="699.0" y="262.0"/>
      <di:waypoint xsi:type="dc:Point" x="731.0" y="262.0"/>
      <di:waypoint xsi:type="dc:Point" x="731.0" y="130.0"/>
      <di:waypoint xsi:type="dc:Point" x="763.0" y="130.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_11"
bpmnElement="flow11" sourceElement="_BPMNShape_UserTask_66"
targetElement="_BPMNShape_ExclusiveGateway_3">
      <di:waypoint xsi:type="dc:Point" x="699.0" y="376.0"/>
      <di:waypoint xsi:type="dc:Point" x="731.0" y="376.0"/>
      <di:waypoint xsi:type="dc:Point" x="731.0" y="130.0"/>
      <di:waypoint xsi:type="dc:Point" x="763.0" y="130.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_12"
bpmnElement="flow12"  sourceElement="_BPMNShape_ExclusiveGateway_3"
targetElement="_BPMNShape_UserTask_67">
      <di:waypoint xsi:type="dc:Point" x="813.0" y="130.0"/>
      <di:waypoint xsi:type="dc:Point" x="838.0" y="130.0"/>
      <di:waypoint xsi:type="dc:Point" x="838.0" y="132.0"/>
      <di:waypoint xsi:type="dc:Point" x="864.0" y="132.0"/>
    </bpmndi:BPMNEdge>
    <bpmndi:BPMNEdge id="BPMNEdge_SequenceFlow_13"
bpmnElement="SequenceFlow_13"
sourceElement="_BPMNShape_UserTask_67"
targetElement="_BPMNShape_EndEvent_7">
      <di:waypoint xsi:type="dc:Point" x="964.0" y="132.0"/>
      <di:waypoint xsi:type="dc:Point" x="1005.0" y="132.0"/>
      <di:waypoint xsi:type="dc:Point" x="1005.0" y="126.0"/>
      <di:waypoint xsi:type="dc:Point" x="1046.0" y="126.0"/>
    </bpmndi:BPMNEdge>
  </bpmndi:BPMNPlane>
 </bpmndi:BPMNDiagram>
</bpmn2:definitions>
```