

·
·
·
·
·
·

Alumne: Carles Ferrer Parera
DNI: 53126315-H
Tutor: Ferran Prados Carrasco
Esteve del Acebo Peña
Carrera: EINF



Projecte Final de Carrera

Corrector d'autòmats per la plataforma ACME



Mòdul pertanyent al Projecte ACME

· · · · · · · · · ·



Índex

1	Descripció del projecte.....	5
1.1	Introducció	5
1.2	Objectius.....	6
1.3	Contingut de la memòria	7
2	Metodologia	9
3	Àmbit i abast del projecte	10
3.1	Àmbit del projecte	10
3.2	Abast del projecte.....	11
3.3	Mòdul d'autòmats finits	12
3.4	Eines utilitzades.....	13
3.4.1	Llenguatges de programació	13
3.4.2	Entorns de desenvolupament.....	14
3.4.3	Eines de comunicacions.....	15
3.4.4	Eines de Modelatge	15
3.4.5	Altres eines emprades.....	15
4	Especificació dels requeriments	16
4.1	Requeriments no funcionals	16
4.1.1	Rendiment	16
4.1.2	Seguretat	16
4.1.3	Disseny.....	17
4.1.4	Configuració.....	17
4.2	Requeriments funcionals	17
4.2.1	Diagrames de casos d'ús	18
4.2.2	Fitxes de casos d'ús	25
5	Anàlisi i disseny.....	34
5.1	Jerarquia de paquets usats	34
5.2	Diagrama de classes	36
5.3	Explicació classes implicades	39
5.4	Diagrames d'activitat	49
5.4.1	Diagrama d'activitat correcció AFD	51
5.4.2	Diagrama d'activitat correcció AFND.....	52
5.4.3	Diagrama d'activitat correcció APD	53
5.4.4	Diagrama d'activitat correcció APND	54
5.4.5	Diagrama d'activitat per obtenir la cadena que defineix l'editor	55
5.4.6	Diagrama d'activitat per restaurar els paràmetres de l'editor	56
5.4.7	Diagrama d'activitat per validar un fitxer de problemes.....	58
5.5	Diagrames de seqüència.....	59
5.5.1	Refresc de la pissarra i zona de zoom	60
5.5.2	Afegir estats.....	62
5.5.3	Afegir transicions	64
5.5.4	Seleccionar un objecte de la pissarra.....	68
5.5.5	Moure un objecte de la pissarra	70
5.5.6	Eliminar objecte seleccionat	72

5.5.7	Modificar objecte seleccionat	73
5.5.8	Augmentar el zoom.....	78
5.5.9	Disminuir el zoom	79
5.5.10	Modificar la vista de la pissarra	80
5.5.11	Obtenir la cadena que defineix el contingut de l'editor	83
5.5.12	Restaurar contingut de l'editor.....	85
5.5.13	Configurar l'idioma de l'editor	86
6	Implementació	87
6.1	Interfície	87
6.1.1	Característiques de l'editor d'autòmats	96
6.2	Transformació, traducció i reconstrucció d'una solució a través de l'editor	105
6.2.1	Representació del contingut de l'editor.....	105
6.2.2	Representació que rep el nucli corrector.....	107
6.2.3	Exemple de transformació.....	110
6.3	Configuració de l'idioma de l'editor.....	111
6.4	Mòduls correctors	111
6.4.1	Corrector d'AFD's	114
6.4.2	Corrector d'AFND's.....	115
6.4.3	Corrector d'APD's.....	117
6.4.4	Corrector d'APND's	119
7	Tipus 27 – Autòmats finits	121
7.1	Definició dels enunciats dels problemes	121
7.2	Definició dels problemes	121
7.3	Implementació dels problemes.....	121
7.3.1	Capçalera del fitxer.....	121
7.3.2	Enunciat del problema.....	122
7.3.3	Definició dels paràmetres per a la correcció	122
7.4	Exemples complets de problemes	123
7.4.1	Exemple complet d'un problema d'un AFD	123
7.4.2	Exemple complet d'un problema d'un AFND.....	124
7.4.3	Exemple complet d'un problema d'un APD	125
7.4.4	Exemple complet d'un problema d'un APND	126
8	Integració a la plataforma ACME	127
8.1	Integració del tipus de problema 27.....	128
9	Posta en marxa.....	130
9.1	Incorporació de la nova aplicació	130
9.2	Com fer proves	131
10	Conclusions	132
11	Possibles millores i treballs futurs.....	133
12	Índex de figures	134
13	Bibliografia	137

1 Descripció del projecte

1.1 Introducció

Tothom coneix la importància que tenen les TIC (Tecnologies de la Informació i la Comunicació) en l'ensenyament, per l'ajuda que signifiquen a la millora del model educatiu. És per això que fa uns anys un grup de professors del departament d'Informàtica i Matemàtica Aplicada de la Universitat de Girona decideix endinsar-se al món de l'ensenyament a través d'Internet (*e-learning*). D'aquí va néixer el projecte ACME (Avaluació Continuada i Millora de l'Ensenyament).

L'ACME és una plataforma d'*e-learning* orientada a la correcció automàtica d'exercicis, que permet als professors assignar als alumnes dossiers personalitzats de problemes per tal que aquests puguin aprendre els continguts de l'assignatura de forma progressiva. D'aquesta manera els professors poden anar seguint l'evolució de l'alumne en la matèria i detectar així les possibles mancances en la seva formació.

Inicialment el projecte ACME anava dirigit a reduir l'elevat grau de fracàs dels alumnes en les assignatures de matemàtiques. Un cop el projecte va estar en funcionament el resultat va ser tan bo que es va ampliar a assignatures d'altres camps d'estudi, com Química o Informàtica. Amb tot i això, encara hi ha moltes matèries a les quals el sistema no dona suport. És per això que cada any es proposen nous tipus de problemes en matèries on es pot generar un corrector automàtic.

Aquest projecte final de carrera neix per donar suport a un nou tipus de problemes dins de la plataforma ACME, els autòmats finits. Aquest nou mòdul inclourà les eines necessàries per poder generar diferents tipus de problemes sobre autòmats finits i la seva posterior correcció.

1.2 Objectius

El que es pretén amb aquest Projecte Final de Carrera és incorporar un nou mòdul que permeti als alumnes poder desenvolupar els exercicis d'autòmats finits d'assignatures com LGA (Llenguatges, Gramàtiques i Autòmats) o TALLF (Teoria d'Autòmats i Llenguatges Formals).

La correcció d'exercicis d'autòmats finits a l'ACME suposa la incorporació d'un nou tipus de problemes. És per això que per integrar aquest mòdul al sistema actual caldrà desenvolupar:

- L'especificació del problema segons els criteris de l'ACME.
- El verificador de l'estructura del problema per tal de facilitar l'edició d'aquests.
- El sistema de generació d'enunciats.
- Les interfícies d'introducció de solucions, amb les quals es permetrà a l'alumne reflectir els diferents passos que ha fet per arribar a la solució del problema.
- El nucli corrector, que corregirà la solució enviada, indicarà en quin pas s'ha equivocat l'alumne i, si es creu convenient, se l'ajudarà a través d'algun missatge de *feedback*.

1.3 *Contingut de la memòria*

En aquest punt es pot veure una breu explicació del contingut dels diferents capítols que componen aquesta memòria.

Capítol 1. Hi trobem la introducció i els objectius d'aquest Projecte final de Carrera.

Capítol 2. Hi trobem una explicació sobre la metodologia usada per implementar el projecte.

Capítol 3. S'hi defineix l'àmbit i abast del projecte. Hi trobem l'anàlisi de les característiques generals dels diferents blocs que componen aquest mòdul corrector d'autòmats. També es parla sobre els llenguatges i altres eines utilitzades per implementar el projecte.

Capítol 4. Hi trobem l'especificació de requeriments que ha de satisfer aquest projecte. Per una banda tenim els requeriments no funcionals i per l'altre els requeriments funcionals expressats en diagrames i fitxes de casos d'ús.

Capítol 5. Hi trobem els diagrames d'anàlisi i disseny de les interfícies i correctors a implementar, concretament els diagrames de classes, d'activitat i de seqüència.

Capítol 6. Hi trobem explicades les característiques més importants de les interfícies i correctors implementats. També s'hi explica el procés de transformació i reconstrucció d'una solució a través de l'editor d'autòmats.

Capítol 7. Hi trobem explicades les pautes a seguir a l'hora de crear nous problemes d'autòmats.

Capítol 8. Hi trobem explicada la integració d'aquest mòdul d'autòmats dins de la plataforma ACME.

Capítol 9. Hi trobem explicades les tasques que s'han dut a terme per la posta en marxa d'aquest nou mòdul dins de la plataforma ACME. També es dona la informació necessària per tal de poder provar la nova aplicació des de la plataforma ACME de proves.

Capítol 10. S'hi recullen les conclusions finals d'aquest Projecte Final de Carrera.

Capítol 11. Hi trobem exposades algunes de les possibles millores a fer en aquest nou mòdul de l'ACME en el futur.

Capítol 12. Hi trobem l'índex de figures que apareixen al llarg d'aquesta memòria.

Capítol 13. Hi trobem la llista de referències bibliogràfiques consultades.

2 Metodologia

La metodologia que s'utilitzarà per desenvolupar aquest Projecte Final de Carrera serà l'XP (*Extreme Programming*), ja que ens trobem davant d'un projecte difícil de dissenyar amb antelació a la implementació, i l'XP, a diferència dels mètodes tradicionals, es centralitza més en l'adaptabilitat que no pas en la previsibilitat.

Les característiques més importants de l'XP són:

- És una metodologia lleugera pensada per projectes curts.
- Elevada interacció entre l'equip de programació i el client o usuari.
- Desenvolupament iteratiu i incremental. El projecte es divideix en una sèrie de fases, al final de cadascuna de les quals es lliura una versió del projecte. De mica en mica, en cadascuna de les fases, es van afegint noves funcionalitats que pot provar l'usuari.
- Usa estàndards de codificació.
- Tot es centra en el resultat, és a dir, complir amb el que es va dissenyar i res més.
- Simplicitat en el codi. És la millor manera que les coses funcionin. Un cop tot funcioni ja s'aniran afegint noves funcionalitats. La filosofia de l'XP és que és més fàcil fer una cosa simple i després tenir un mica de feina en afegir noves funcionalitats, que no pas fer una cosa complexa que potser no es farà servir mai.

Les passes que es seguiran en aquest projecte seran les següents:

- Definició del requeriment a implementar.
- Anàlisi del requeriment.
- Programació de la funcionalitat del requeriment definit al punt 1.
- Proves de validació de la implementació. Si es detecten errors seran corregits i es tornaran a fer proves de validació.
- S'integrarà el nou mòdul a l'ACME per poder mostrar el seu funcionament a l'usuari. Un cop finalitzat aquest punt tornarem al primer punt.

3 Àmbit i abast del projecte

3.1 Àmbit del projecte

Com ja s'ha comentat anteriorment, aquest projecte té com objectiu la implementació d'un mòdul d'autòmats finits i de pila per a la plataforma ACME, amb la idea que s'utilitzi com a eina de suport a la docència. Els professors podran proposar diferents problemes relacionats amb autòmats finits i de pila i els alumnes els podran resoldre des de qualsevol ordinador amb connexió a Internet.

La plataforma ACME estar dissenyada des d'un punt de vista modular, de manera que permet anar afegint nous tipus de problemes sense cap tipus de dificultat. Per aquest motiu, tot i que inicialment va ser pensada per problemes de matemàtiques, trobem un gran nombre i variat de temàtiques com poden ser les bases de dades relacionals, la programació, la química inorgànica, etc. Cadascuna d'aquestes temàtiques tindrà una manera diferent d'interactuar amb l'usuari, de representar la resposta i de corregir-la, però totes elles tindran un punt en comú: s'ha de poder especificar problemes amb un corrector automàtic.

Actualment l'ACME dona suport a més de 50 assignatures de diferents carreres en les que hi ha matriculats aproximadament 2500 alumnes. Algunes d'aquestes assignatures són:

- Metodologia i Tecnologia de la Programació, Introducció als Fitxers i les Bases de dades, Bases de dades i Introducció a les Estructures de Dades de les Enginyeries Tècniques Informàtiques.
- Càlcul i Organització de la Informació d' Enginyeria Industrial.
- Fonaments Matemàtics de l'Enginyeria i Anàlisi Químic d'Enginyeria Tècnica Industrial especialitzada en Química Industrial.
- Càlcul d'Arquitectura Tècnica.
- Matemàtiques de les Enginyeries Tècniques Alimentàries.
- Matemàtiques Bàsiques de l'Escola Politècnica Superior i de la Facultat de Ciències.

Amb aquest nou mòdul es vol donar suport a l'assignatura de LGA d'Enginyeria Informàtica (EINF) i a l'assignatura de TALLF d'Enginyeria Tècnica en Informàtica de Sistemes (ETIS).

3.2 Abast del projecte

Els autòmats finits i els autòmats de pila només són una petita part del que podríem anomenar Informàtica Teòrica o, més concretament, Teoria d'Autòmats i Llenguatges Formals, matèria que constitueix part de la base i dels fonaments matemàtics de la computació. La Informàtica Teòrica contempla dues vessants principals, la primera, que podríem considerar més pràctica, que estudia conceptes com els llenguatges regulars, els autòmats finits, les expressions regulars les gramàtiques no contextuals o els autòmats de pila, té una aplicació molt directe a l'hora de construir compiladors, dissenyar llenguatges de programació, analitzar i processar cadenes de caràcters o fins i tot crear jocs d'ordinador. La segona vessant estudia conceptes més teòrics, com la complexitat dels algorismes i la computabilitat dels problemes. D'aquest estudi s'han extret conseqüències de gran importància per a la Informàtica com, per exemple, que hi ha problemes que no es poden resoldre mitjançant algorismes i d'altres que si que es poden resoldre però que no resulta factible fer-ho.

Com podem veure, el tema és molt ampli i seria impossible fer un mòdul corrector que ho englobés tot. És per això que en aquest mòdul només ens dedicarem a la correcció dels autòmats finits i dels autòmats amb pila, siguin deterministes o no, i deixarem la resta de conceptes per més endavant. La tria d'aquests tipus de problemes l'ha fet el professor Esteve del Acebo, professor del departament d'IMA (Informàtica i Matemàtica Aplicada) de l'Escola Politècnica Superior de la UdG, després d'analitzar les assignatures de LGA i TALLF de les Enginyeries Informàtiques.

En la nostra opinió, aquestes assignatures es compten entre les més importants de la carrera. Potser la majoria dels alumnes mai dissenyaran cap compilador ni cap llenguatge de programació, ni potser tampoc cap joc, però què passa amb el tractament de cadenes de caràcters? Imaginem-nos que necessitem tractar la informació que es troba en un fitxer XML, cosa ben normal avui en dia i que cada vegada serà més freqüent, com ho podem fer si no disposem de cap *parser*? Ens podem crear el nostre propi *parser*, però per fer-ho haurem d'aplicar els coneixements apresos en assignatures com aquestes. Sense aquests coneixements la creació d'un *parser* pot arribar a ser molt complicada o, fins i tot, impossible.

Tot i ser una de les assignatures més importants, el grau de fracàs dels estudiants acostuma a no ser baix. Això ho podríem solucionar donat més problemes als alumnes per fer a casa, però cal tenir en compte els temps que necessita el professor per corregir-los. Posem pel cas que a l'assignatura tenim 60 alumnes, a on cada alumne té assignats 20 problemes per resoldre, això fa un total de 1200 problemes. El professor es pot passar unes tres setmanes per corregir-los tots. Això és temps que perd el professor i sobretot els alumnes.

És d'aquí d'on neix la idea per a aquest projecte final de carrera. Amb aquest nou mòdul per l'ACME es pretén que els alumnes d'aquestes assignatures consolidin els coneixements adquirits a classe a partir de una gran varietat de problemes pràctics. El professor podrà seguir de ben a prop l'evolució de cadascun dels alumnes i veure en quin punt de la matèria l'alumne té dificultats, tenint més temps per donar suport a aquells alumnes que ho necessitin. L'alumne per una altre banda tindrà el resultat de la correcció en el mateix moment i tampoc s'haurà d'esperar.

3.3 Mòdul d'autòmats finits

Actualment a l'ACME no tenim cap mòdul que tracti sobre alguna matèria que parli d'autòmats, per tant en aquest mòdul tractarem una matèria totalment nova, això no vol dir que no ens puguem basar en alguns dels mòduls ja existents.

El desenvolupament d'aquest mòdul el dividirem en quatre parts:

- Caldrà desenvolupar una interfície que permeti tant als alumnes com al professor la introducció de les solucions dels autòmats en forma de diagrama de transicions. Per dur a terme aquesta part ens basarem en mòdul del model entitat-relació per dues raons, la primera per que necessitem una àrea de dibuix on l'usuari pugui dibuixar diagrames i això el corrector del model entitat-relació ja ho fa, i la segona perquè volem fer una interfície semblant que sigui familiar als alumnes.
- Caldrà també desenvolupar un corrector que corregeixi les solucions enviades pels alumnes de forma instantània. Per desenvolupar aquesta part també ens basarem amb alguna mòduls ja existents, com són el del model entitat-relació i el del model relacional. La manera que tenen de comparar la solució enviada pels alumnes amb la del professor és semblant a la que necessitaré per corregir problemes d'autòmats.
- Caldrà trobar la forma d'escriure i especificar l'enunciat i solució del problemes, i també implementar el codi necessari que verifiqui l'estructura del problema proposat pel professor.
- Per últim caldrà integrà aquest nou mòdul dins del sistema ACME i un cop més ens basarem en problemes ja existents, com són els del model entitat-relació o els del model relacional. Algunes funcions, com per exemple el sorteig de problemes, es podran aprofitar del tot ja que funcionaran exactament igual. No obstant hi haurà

funcions que caldrà modificar-les per adaptar-les a les noves interfícies, definicions de problemes i correctors utilitzats.

3.4 *Eines utilitzades*

Per desenvolupar aquest projecte utilitzarem diferents tipus d'eines com són llenguatges de programació, entorns de desenvolupament, navegadors web, transferències de fitxers i altres eines d'edició com el Microsoft Word o el Microsoft Office Visio.

3.4.1 Lenguatges de programació

A l'hora d'implementar aquest mòdul corrector d'autòmats tenim dos blocs ben diferenciats, per una banda tenim la interfície on els alumnes i el professor dibuixaran els autòmats, i per l'altre tenim els nuclis correctors.

3.4.1.1 La interfície

Per desenvolupar la interfície ens trobem davant d'una aplicació que s'ha d'integrar dins d'una pàgina web. Es podria desenvolupar utilitzant *JavaScript* i *HTML*, però tractant-se d'una aplicació gran seria difícil de mantenir i d'implementar, per això aquests dos llenguatges queden descartats. Tenim encara dos llenguatges que ens poden fer servei ja que es poden integrar fàcilment dins d'una aplicació web, són el *Java* i el *Flash*.

Escollir entre *Java* i *Flash* no és senzill. Per una banda tenim que *Java* ens proporciona l'orientació a objectes, amb classes i herència, dotant-lo d'una gran potència. Per l'altre banda tenim que *Flash* usa el llenguatge de programació *ActionScript*, un llenguatge que també és orientat a objectes, el qual proporciona a *Flash* una potència similar a la de *Java*, tot i que la de *Java* encara és superior.

Finalment per desenvolupar la interfície s'ha optat per fer un *applet* amb *Java* utilitzant els paquets *Swing* i *AWT*. Per una banda *Java* ofereix més potència que *ActionScript* i per l'altra aquesta interfície ha de ser semblant a la del mòdul del model entitat-relació que ja es troba implementada en *Java*.

Un altre dels motius que m'ha fet decantar pel *Java* és que amb aquest ja hi he treballat mentre que amb l'*ActionScript* no hi he treballat mai. A més a més, mai he realitzat interfícies gràfiques amb *Java* i serà una bona ocasió per aprendre'n així com també aprofundir en l'aprenentatge d'aquest llenguatge.

Així doncs la interfície del corrector d'autòmats serà un *applet* en *Java* que s'executarà sobre la màquina local.

3.4.1.2 El nucli corrector

El corrector s'implementarà amb el llenguatge PHP (*Hypertext Processor*). La raó d'escollir aquest llenguatge és que la plataforma ACME es troba implementada en PHP i per tant la integració es fa més fàcil. Aquest llenguatge es distribueix sota llicència d'*Open Source* i és usat principalment per la creació de pàgines web dinàmiques. A diferència d'altres llenguatges, el PHP s'executa a la màquina remota, és a dir, en el propi servidor web.

3.4.2 Entorns de desenvolupament

El primer que necessitem per poder programar amb *Java* és instal·lar el paquet JDK (*Java Development Kit*), concretament treballarem amb la versió 1.5.0. El paquet JDK es pot descarregar gratuïtament des de la web.

La segona cosa que necessitem per poder programar amb *Java* d'una forma més amigable és un IDE (*Integrated Development Environment*). L'IDE que usarem serà l'*Eclipse*, concretament la versió 3.2. *Eclipse* és multiplataforma i també es pot descarregar gratuïtament des de la web. Des d'*Eclipse* podem editar el codi, compilar-lo, depurar-lo i executar-lo. *Eclipse* també dóna suport a l'eina *Javadoc*, que ens permet generar documentació de forma automàtica a partir dels comentaris del codi font.

Una de les característiques més importants d'*Eclipse* és el seu disseny modular, que ens permet augmentar la seves funcionalitats afegint nous mòduls. Un dels mòduls que s'ha afegit és el *Visual Editor 1.2.0*. Gràcies al *Visual Editor* podem dissenyar la interfície de l'*applet* de forma visual.

Eclipse també ens permet la creació de fitxers JAR (*Java ARchive*) de format comprimit. D'aquesta manera millorem el temps de descàrrega de les classes de l'*applet* des del servidor web cap a l'ordinador on s'està executant el navegador web.

Per la programació en PHP dels nuclis correctors ens ha fet falta instal·lar un intèrpret de PHP en el nostre PC per poder fer proves locals abans de penjar-ho en el servidor web. El paquet que hem instal·lat és XAMPP que, entre moltes d'altres funcionalitats, porta un servidor web, l'*Apache*, compatible amb PHP. Aquest paquet també es pot descarregar gratuïtament des de la web.

L'editor usat per treballar amb PHP serà l'*UltraEdit-32* versió 12.00. Una des les avantatges d'aquest editor és que es poden editar fitxers remotament via FTP. D'aquesta manera podem treballar amb els fitxers del servidor web amb tota comoditat, com si de fitxers d'una unitat local es tractessin.

3.4.3 Eines de comunicacions

Per poder integrar el nou mòdul a l'ACME primer de tot caldrà pujar l'*applet* cap al servidor web, per fer-ho necessitarem un programa de FTP client, el que farem servir serà el *SSH Secure Shell 3.2*, que ens permetrà transferir dades a les màquines remotes de forma segura ja que utilitza un canal segur (*Secure Shell*)

Per poder provar el nou mòdul un cop pujat al servidor web ens caldrà un navegador web per connectar-nos a l'ACME. Els navegadors que utilitzarem seran l'*Internet Explorer* Versió 6 i el *Mozilla Firefox* versió 2.0. El *Mozilla* el provarem des de Windows XP i des de Linux.

3.4.4 Eines de Modelatge

Per tal d'especificar d'una manera entenedora les funcionalitats de la interfície d'aquest projecte farem servir UML (*Unified Modeling Language*) o Llenguatge Unificat de Modelat. L'UML és un llenguatge que serveix per especificar, construir i documentar els elements d'un sistema basat en software des del punt de vista de l'orientació a objectes.

3.4.5 Altres eines emprades

Per l'elaboració dels diferents diagrames UML usarem el Microsoft Office Visio 2003.

Per l'elaboració d'aquesta memòria usarem el Microsoft Office Word 2003.

4 Especificació dels requeriments

Entenem com a requeriment el conjunt d'idees que té el client sobre què ha de ser i fer el software a desenvolupar. L'objectiu d'aquest punt és expressar d'una forma no ambigua i amb pocs tecnicismes allò que ha de fer el sistema que es vol desenvolupar. En el nostre cas especificarem els requeriments de la interfície a desenvolupar en aquest projecte.

Tenim dos tipus de requeriments, els no funcionals i els funcionals, que veurem en els següents dos punts.

4.1 *Requeriments no funcionals*

Són aquelles restriccions imposades pel client o pel mateix problema que acaben afectant al disseny de l'aplicació i que no tenen res a veure amb les funcionalitats que ha de tenir el sistema. Són requeriments no funcionals temes de seguretat a tenir en compte, configuració de la màquina mínima per poder executar l'aplicació, característiques de les interfícies, etc.

A continuació analitzem alguns dels requeriments que afecten en el disseny d'aquest mòdul.

4.1.1 Rendiment

Tal com ja s'ha comentat en anteriors punts d'aquesta memòria, la interfície a desenvolupar serà integrada a la plataforma ACME. Actualment l'ACME té donats d'alta uns 2500 alumnes, això fa que el servidor web pugui estar servint moltes peticions a l'hora i per tant és necessari que les pàgines webs dels diferents mòduls no estiguin gaire carregades.

Aquest aspecte de rendiment no ens afectarà gaire ja que la nostra única interfície a implementar serà un applet. Al tractar-se d'un applet, aquest es descarregarà el primer cop que algú es connecti a alguna de les pàgines web de l'ACME que continguin l'*applet*, a les posteriors connexions ja no farà falta descarregar-se l'*applet* per trobar-se a la memòria cau del navegador. Un cop l'*applet* estigui descarregat s'executarà a la màquina local, oferint la mateixa rapidesa que qualsevol altre aplicació local.

4.1.2 Seguretat

La plataforma ACME ja ens proporciona les següents mesures de seguretat:

- Per poder accedir al nostre mòdul corrector o a qualsevol altre mòdul de l'ACME l'usuari necessita identificar-se prèviament.
- Es treballa sobre una connexió segura a través del protocol HTTPS. D'aquesta manera la informació que circula a través d'internet entre el servidor i el client sempre estarà encriptada.

4.1.3 Disseny

La interfície que hem d'implementar ha de satisfer tota una sèrie de característiques.

- Ha de ser una interfície intuïtiva. Tant l'alumne com el professor han de ser capaços d'utilitzar-la sense cap apartat d'ajuda específic.
- Ha de ser flexible i fàcil d'utilitzar pels usuaris. S'ha de permetre que moltes de les accions que es podran fer amb el ratolí també es puguin fer des del teclat.
- Ha de ser ràpida. Ha de permetre a l'usuari una visualització àgil de les nostres pàgines, i els processos de correcció han de retornar un resultat amb el menor temps possible.
- Coherència en l'estil. Al dissenyar les interfícies cal mantenir una coherència amb l'estil de la resta de mòduls que hi pugui haver a la plataforma ACME.

4.1.4 Configuració

Per tal que els usuaris puguin utilitzar sense cap problema la nostra aplicació cal que tinguin instal·lat l'entorn d'execució de *Java* (JRE) versió 1.5 o superior. També necessitaran una connexió a Internet, a poder ser una connexió de banda ample.



4.2 *Requeriments funcionals*

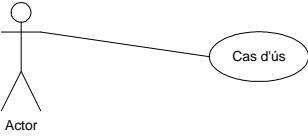
Els requeriments funcionals venen a descriure com s'haurà de comportar l'aplicació que anem a desenvolupar davant d'estímuls externs, és a dir, especifica les sortides que s'han de generar davant d'unes determinades entrades i les operacions necessàries per aconseguir això. Una bona manera de dur a terme l'anàlisi de requeriments funcionals és a partir dels diagrames de casos d'ús i de les fitxes de casos d'ús, ja que ens permeten fer una descripció tan detallada com es vulgui sobre el comportament de l'aplicació.

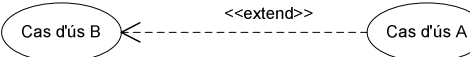
4.2.1 Diagrames de casos d'ús

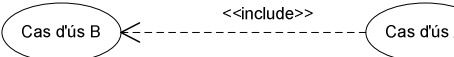
Els diagrames de casos d'ús ens permeten fer una descripció funcional d'un sistema i dels seus principals processos, delimitant el problema a tractar, de forma visual.

A continuació es mostren els diagrames realitzats per l'especificació del mòdul corrector d'autòmats. La simbologia usada pels casos d'ús serà la següent:

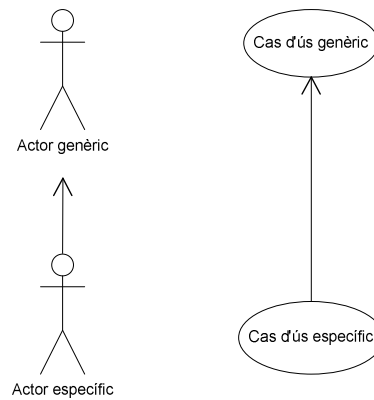
- Per representar els actors, entitat externa que interactua amb el sistema, farem servir una figura humana, tot i que no ha de ser forçosament una persona. 
- Per representar els casos d'ús, seqüència de transaccions relacionades i executades per un actor i el sistema, farem servir ovals. 
- Per representar les relacions ho farem de diferents formes, depenent de l'estereotip de la relació.

- Relació de comunicació, es genera entre un actor i un cas d'ús, i la representarem amb una línia entre l'actor i el cas d'us, pot portar informació cap als dos sentits. 

- Relació d'extensió, es genera entre dos casos d'ús A i B i vol dir que el cas A pot estendre el comportament especificat en el cas B. La representarem amb una fletxa puntejada des del cas A al cas B amb l'etiqueta <<extend>>. 

- Relació d'utilització, es genera entre dos casos d'ús A i B i vol dir que el cas A també inclou el comportament especificat en el cas B. La representarem amb una fletxa puntejada des del cas A al cas B amb l'etiqueta <<include>>. 

- Relació de generalització, relació d'herència entre actors o casos d'ús, i la representarem amb una fletxa que va de l'actor/cas d'ús més específic al més genèric.



4.2.1.1 Diagrama de casos d'ús de context

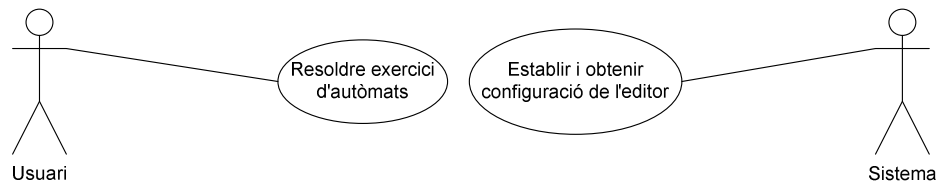


Fig. 4-1 Diagrama de casos d'ús de context.

Tal com s'ha dit anteriorment aquest corrector consta de dues parts, d'una interfície, on entrar i visualitzar la solució, i d'un nucli corrector. El diagrama anterior es centre en la interfície, s'hi pot veure els actors que amb les seves accions corresponents. Per una banda tenim l'usuari que només pot fer una sola acció, resoldre exercicis. Per l'altre banda tenim el sistema que serà l'encarregat d'establir i obtenir la configuració de l'editor.

L'últim diagrama es pot descompondre en altres diagrames més refinats. Això és el que veurem en els següents punts.

4.2.1.2 Diagrama de casos d'ús de resoldre exercici d'autòmats

En aquest punt refinarem un mica més el cas d'ús resoldre exercici d'autòmats.

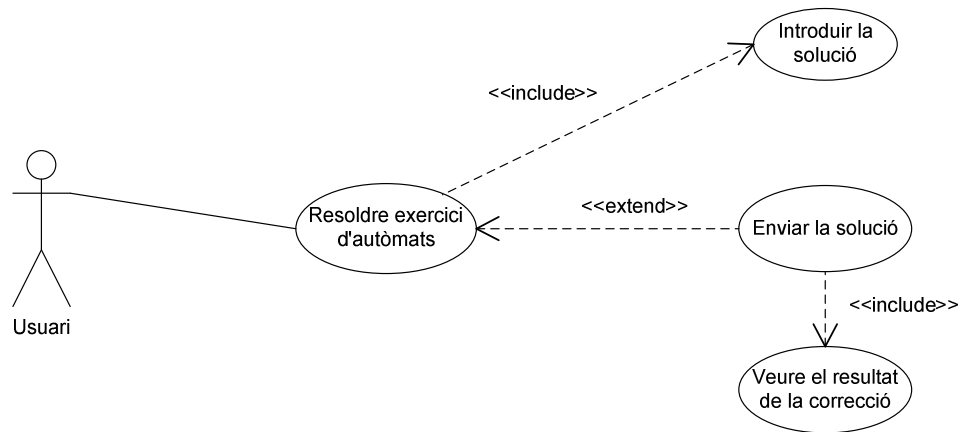


Fig. 4-2 Diagrama de casos d'ús per resoldre exercicis d'autòmats.

Es pot observar com en la resolució d'un problema d'autòmats hi intervenen varies accions:

- S'ha d'introduir la solució del problema.
- Es pot enviar la solució i per tant veure el resultat de la correcció.

Alguns casos d'ús del diagrama anterior tenen associats altres casos d'ús més detallats.

4.2.1.2.1 Diagrama de casos d'ús d'introduir solució

L'usuari a l'hora d'introduir la solució ho podrà fer visualment a través d'una pissarra o àrea de dibuix, sobre la qual podrà fer tota una sèrie d'accions, per tant el cas d'ús introduir solució té associats altres casos d'ús que podem veure en el següent diagrama.



Fig. 4-3 Diagrama de casos d'ús per introduir la solució d'un problema.

Com es pot observar en el diagrama anterior en el cas d'ús introduir solució hi intervenen sis accions diferents:

- Afegir un objecte a la pissarra
- Seleccionar un objecte de la pissarra
- Moure objectes de la pissarra
- Eliminar objecte seleccionat
- Modificar objecte seleccionat
- Modificar la vista de la pissarra

En algunes d'aquestes accions hi ha implicats altres casos d'ús que es detallen en els següents punts.

4.2.1.2.1.1 Diagrama de casos d'ús d'afegir un objecte a la pissarra

En el cas d'ús afegir objecte a la pissarra hi ha implicats altres casos d'ús que es reflecteixen en el següent diagrama.

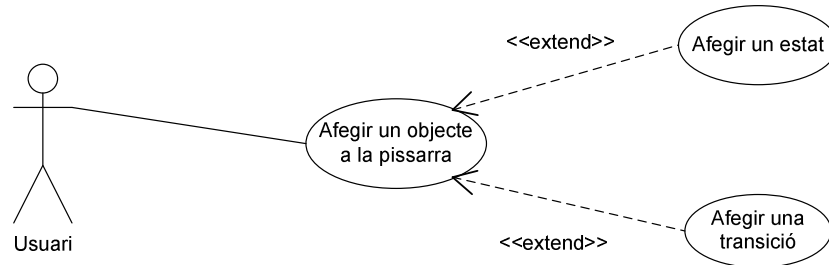


Fig. 4-4 Diagrama de casos d'ús per afegir un objecte a la pissarra.

En aquest cas d'ús hi intervenen dues accions:

- Afegir un estat
- Afegir una transició

Aquestes dues accions encara es poden refinar un nivell més, obtenint així més casos d'ús.

4.2.1.2.1.1.1 Diagrama de casos d'ús d'afegir un estat

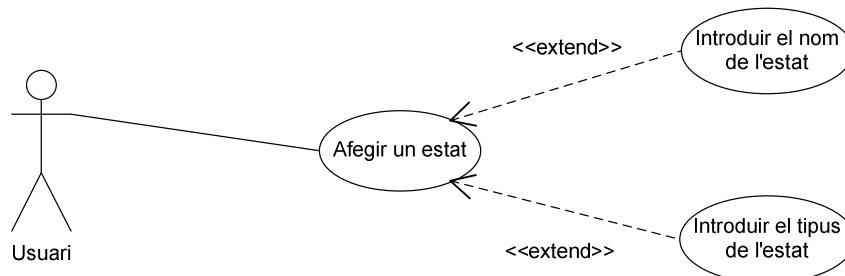


Fig. 4-5 Diagrama de casos d'ús per afegir un estat.

Dur a terme l'acció afegir estat implica dues accions:

- Introduir el nom de l'estat
- Introduir el tipus de l'estat

4.2.1.2.1.2 Diagrama de casos d'ús d'afegir una transició

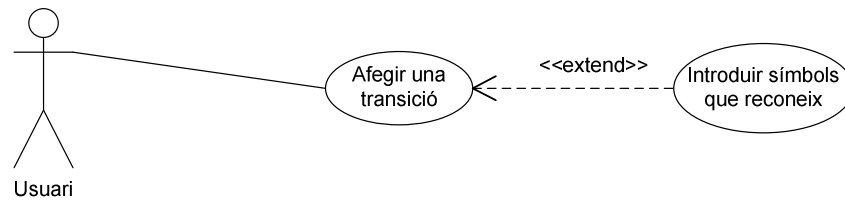


Fig. 4-6 Diagrama de casos d'ús per afegir una transició.

Dur a terme l'acció d'afegir una transició pot implicar dur a terme també l'acció introduir símbols que reconeix.

4.2.1.2.1.2 Diagrama de casos d'ús modificar la vista de la pissarra

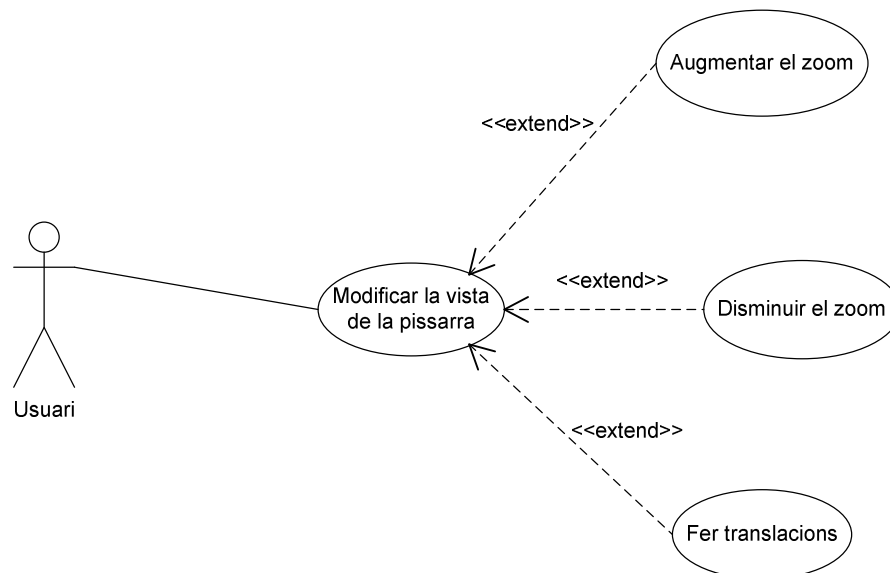


Fig. 4-7 Diagrama de casos d'ús per modificar la vista de la pissarra.

L'usuari podrà fer tota una sèrie d'operacions sobre la vista de la pissarra, podrà augmentar o disminuir el zoom segons les seves necessitats, així com també fer translacions, poden dibuixar autòmats més grans que l'àrea de dibuix.

4.2.1.3 Diagrama de casos d'ús d'establir i obtenir configuració de l'editor

Tal com em vist en el diagrama de context, l'usuari no és l'únic actor que actua amb l'editor, també tenim el sistema, que ha de poder fer les següents accions:

- Establir els paràmetres de l'editor
- Obtenir el contingut de l'editor

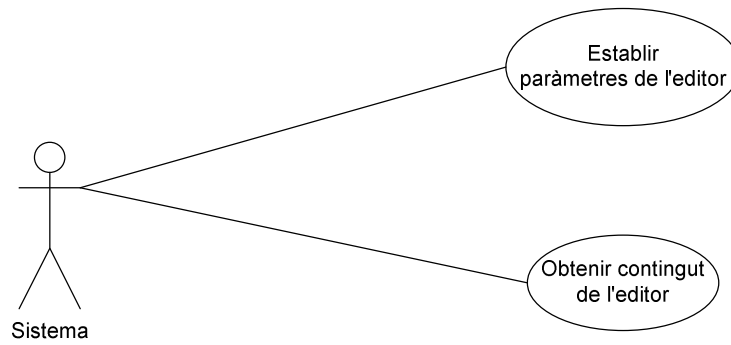


Fig. 4-8 Diagrama de casos d'ús per obtenir i establir els paràmetres de l'editor.

4.2.1.3.1 Diagrama de casos d'ús establir paràmetres de l'editor

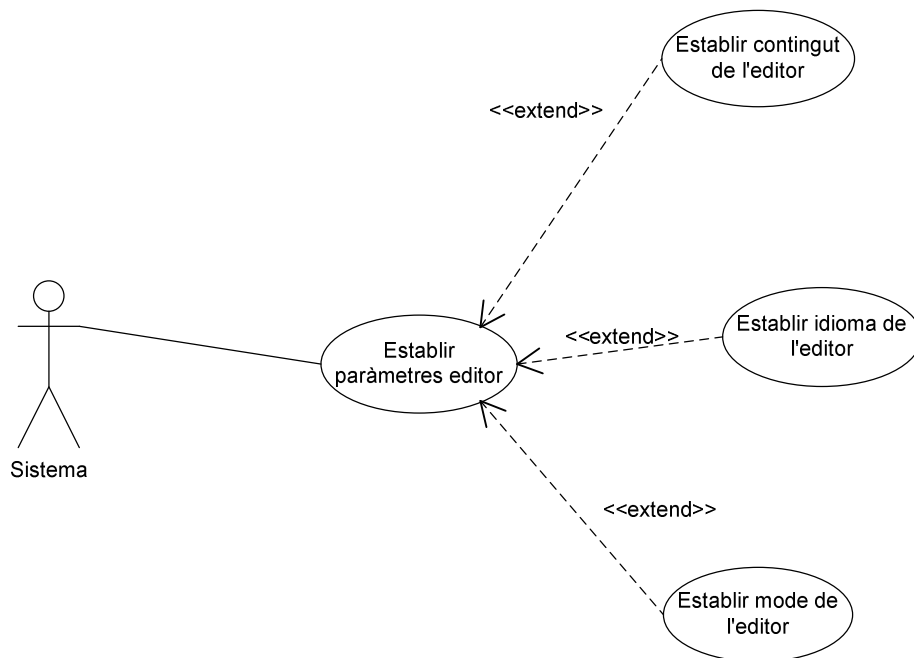


Fig. 4-9 Diagrama de casos d'ús per establir els paràmetres de l'editor.

Podem observar com en l'acció establir paràmetres de l'editor si poden veure implicades fins a tres accions més:

- Establir el contingut de l'editor
- Establir idioma de l'editor
- Establir mode de l'editor

4.2.2 Fitxes de casos d'ús

Les fitxes de casos d'ús ens permeten fer una descripció amb més detall d'allò que fa cada cas d'ús. A cada fitxa hi consta l'actor que pot dur a terme el cas d'ús, una breu explicació informal del que fa el cas d'ús, les precondicions que s'han de complir abans d'executar-se, les postcondicions que s'han de complir un cop executat, el flux principal d'esdeveniments, els subfluxos o alternatives dins del flux principal i finalment comentaris a tenir en compte.

4.2.2.1 Fitxa de casos d'ús de context

CAS D'ÚS	Mòdul corrector autòmats
Descripció	Visió global del mòdul corrector d'autòmats finits i de pila. Per una banda tenim l'usuari que pot resoldre problemes, i per l'altre tenim el sistema que es comunica amb el nostre mòdul.
Actors	Usuari i Sistema.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura. El sistema envia al nostre mòdul una cadena de text amb un format preestablert per configurar-lo.
Flux principal	1.- L'usuari accedeix a l'enunciat del problema d'autòmats, introdueix i visualitza la solució i la resposta a la correcció. 2.- El sistema envia una cadena de text amb la configuració. Rep una cadena de text amb el contingut de l'editor.
Postcondició	Es marca el problema amb el resultat de la correcció. S'ha carregat la configuració o s'ha guardat aquesta al sistema.
Comentaris	

4.2.2.2 Fitxa de casos d'ús de resoldre exercici d'autòmats

CAS D'ÚS	Resoldre exercici d'autòmats
Descripció	Ens permet introduir i enviar la solució d'un enunciat d'un problema d'autòmats.
Actors	Usuari.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura.
Flux principal	1.- Accedir a l'enunciat del problema d'autòmats. Introduir a través de l'àrea de dibuix o pissarra la solució al problema de l'enunciat. 2.- Enviar la solució del problema proposat. 3.- Es mostra el resultat de la correcció.
Postcondició	Es marca el problema com a resolt amb el resultat que hagi donat el procés de correcció.
Comentaris	L'enunciat que es mostra a l'usuari s'ha assignat de manera aleatòria. Més endavant podrem veure els diagrames d'activitat que descriuen el procés de correcció. Veure diagrama de casos d'ús.

4.2.2.2.1 Fitxa de casos d'ús d'introduir solució

CAS D'ÚS	Introduir solució
Descripció	Ens facilita un editor amb una àrea de dibuix on dibuixar l'autòmat de la solució al problema exposat a l'enunciat.
Actors	Usuari.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura.
Flux principal	Les accions que es podran dur a terme amb l'editor d'autòmats seran: <ul style="list-style-type: none"> • Afegir un objecte a la pissarra. • Seleccionar un objecte de la pissarra. • Moure els objectes de la pissarra. • Eliminar objecte seleccionat. • Modificar objecte seleccionat. • Modificar la vista de la pissarra.
Subfluxos	Per més comoditat es mostren a continuació.
Postcondició	L'usuari ha dibuixat la solució de l'exercici proposat.
Comentaris	L'editor constarà d'una àrea de dibuix o pissarra on l'alumne

	podrà dibuixar l'autòmat que demani l'enunciat del problema. Es podran dibuixar cercles per representar els estats de l'autòmat i fletxes per representar les transicions entre dos estats. Veure diagrama de casos d'ús.
--	---

1. Subflux afegir un objecte a la pissarra

Permet a l'usuari afegir els estats i les transicions necessàries per representar un autòmat, i introduir els atributs dels estats i les transicions.

2. Subflux seleccionar un objecte de la pissarra

L'usuari podrà desplaçar el cursor per sobre de la pissarra amb l'ajuda del ratolí i podrà seleccionar un estat o una transició qualsevol de la pissarra. Al seleccionar un estat o una transició l'editor mostrarà els seus atributs, aquesta informació es detalla més en les següents fitxes de casos d'ús.

3. Subflux moure objectes de la pissarra

L'usuari podrà desplaçar els objectes que hi hagi dibuixats a la pissarra a qualsevol altre punt de la pissarra amb l'ajuda del ratolí polsant i arrossegant amb el botó esquerra.

4. Subflux eliminar objecte seleccionat

L'usuari podrà eliminar l'objecte seleccionat de la pissarra polsant sobre un botó de l'editor o la tecla *suprimir* del teclat.

5. Subflux modificar objecte seleccionat

L'usuari podrà modificar els atributs de l'objecte seleccionat introduint-los des del teclat.

6. Subflux modificar la vista de la pissarra

L'editor disposarà d'un parell de botons que permetrà a l'usuari fer zooms del contingut de la pissarra. Com que l'autòmat dibuixat podrà ser més gran que l'àrea de dibuix també s'han de poder fer translacions.

4.2.2.2.1.1 Fitxa de casos d'ús d'afegir un objecte a la pissarra

CAS D'ÚS	Afegir un objecte a la pissarra
Descripció	S'identifiquen els diferents objectes a dibuixar en l'editor.
Actors	Usuari.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura.
Flux principal	A l'àrea de dibuix de l'editor s'hi podran afegir: <ul style="list-style-type: none"> • Estats. • Transicions.
Subfluxos	Per més comoditat es mostren a continuació.
Postcondició	L'usuari ha dibuixat un estat o una transició que formaran part de la solució.
Comentaris	L'editor haurà de tenir dos botons excloents entre si, en funció de quin estigui activat l'usuari podrà dibuixar un estat o una transició. Els estats es representaran amb cercles i les transicions amb fletxes que aniran d'un estat a un altre. Veure diagrama de casos d'ús.

1. Subflux afegir estat

Polsant sobre el botó d'afegir estat que tindrà l'editor, aquest es quedarà activat i es desactivarà el botó d'afegir transició. Amb el botó activat, l'usuari polsarà sobre el punt de l'àrea de dibuix on vulgui afegir l'estat.

2. Subflux afegir transició

Polsant sobre el botó d'afegir transició que tindrà l'editor, aquest es quedarà activat i es desactivarà el botó d'afegir estat. Amb el botó activat, l'usuari polsarà sobre l'estat origen de la transició i després sobre l'estat destí.

4.2.2.2.1.1.1 Fitxa de casos d'ús d'afegir un estat

CAS D'ÚS	Afegir un estat a la pissarra
Descripció	Un cop dibuixat un estat a la pissarra permet introduir els seus atributs.
Actors	Usuari.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura.
Flux principal	Es podran dur a terme les següents accions <ul style="list-style-type: none"> • Introduir el nom de l'estat. • Introduir el tipus de l'estat
Subfluxos	Per més comoditat es mostren a continuació.
Postcondició	L'usuari ha dibuixat un estat i ha pogut introduir els seus atributs.
Comentaris	Un estat tindrà com atributs el seu nom i el tipus. El tipus d'un estat podrà ser inicial, final, inicial-final i no final. Veure diagrama de casos d'ús.

1. Subflux introduir el nom de l'estat

L'editor tindrà una casella de text on introduir el nom de l'estat. Per defecte l'estat tindrà el nom *ei*, on *i* és un enter que comença per 0 i es va incrementant a mesura que s'afegeixen estats.

2. Subflux introduir el tipus de l'estat

L'editor tindrà una llista desplegable amb els quatre tipus d'estat: estat inicial, estat final, estat inicial-final i estat no final. L'usuari podrà escollir el tipus de l'estat a partir de la llista. Per defecte un estat serà no final.

4.2.2.2.1.1.2 Fitxa de casos d'ús d'afegir una transició

CAS D'ÚS	Afegir una transició a la pissarra
Descripció	Un cop dibuixada la transició entre dos estats permet afegir els seus atributs.
Actors	Usuari.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura.
Flux principal	Es podran dur a terme les següents accions <ul style="list-style-type: none"> • Introduir símbols que reconeix la transició
Subfluxos	Per més comoditat es mostren a continuació.
Postcondició	L'usuari ha dibuixat una transició entre dos estats i ha introduït els símbols que reconeix.
Comentaris	Una transició com atributs té una llista de símbols que reconeix. Veure diagrama de casos d'ús.

1. Subflux introduir símbols que reconeix

L'editor tindrà una casella de text per introduir la llista de símbols.

Si estem davant d'un problema d'autòmats finits l'usuari introduirà els símbols reconeguts separats per comes, i l'editor validarà que tots ells pertanyin a l'alfabet.

Si estem davant d'un problema d'autòmats de pila els símbols acceptats s'introduiran de forma diferent. L'usuari els haurà d'introduir un a un a través de la casella de text i polsar la tecla *retorn* o el botó *ok*. Per introduir els símbols acceptats s'haurà de seguir el format $a,A/BA$, on a és un símbol de l'alfabet, A i B són símbols de l'alfabet de la pila. L'editor validarà que la llista estigui ben entrada.

L'editor tindrà tota una sèrie de botons per introduir aquells símbols que no es troben al teclat, aquests símbols són:

- Lambda λ .
- Símbol de pila buida \perp .
- Barra vertical $|$. La barra vertical també es podrà introduir des del teclat.

4.2.2.2.1.2 Fitxa de casos d'ús de modificar la vista de la pissarra

CAS D'ÚS	Modificar la vista de la pissarra
Descripció	Ens permet fer zooms del contingut de la pissarra i translacions ja que al fer un zoom perdrem àrea de dibuix.
Actors	Usuari.
Precondició	L'usuari s'ha identificat podent ser: <ul style="list-style-type: none"> • Un alumne que pertany al grup d'alumnes de l'assignatura del problema d'autòmats. • Un professor responsable de l'assignatura.
Flux principal	Es podran dur a terme les següents accions <ul style="list-style-type: none"> • Augmentar el zoom • Disminuir el zoom • Fer translacions
Subfluxos	Per més comoditat es mostren a continuació.
Postcondició	Es mostra el contingut a l'editor amb el zoom escollit i els desplaçaments aplicats.
Comentaris	Veure diagrama de casos d'ús.

1. Subflux augmentar zoom

L'editor presentarà un botó que ens permetrà augmentar el zoom de l'àrea de dibuix. Tindrem fins a set nivells diferents de zoom. Per defecte sempre hi haurà aplicats dos nivells de zoom.

2. Subflux disminuir zoom

L'editor també presentarà un botó que ens permetrà disminuir el zoom de l'àrea de dibuix.

3. Subflux fer translacions

L'editor ha de tenir una eina per fer translacions sobre l'àrea de dibuix, ja que al fer un augment de zoom podem perdre part de l'autòmat dibuixat o al disminuir-lo ens pot quedar descentrada.

Aquesta eina serà una pissarra petita on es visualitzarà el contingut íntegre de la pissarra gran. Sobre aquesta pissarra tindrem un requadre vermell que ens assenyalarà el contingut que s'està mostrant a la pissarra gran. Aquesta pissarra petita serà només de lectura.

4.2.2.3 Fitxa de casos d'ús d'establir i obtenir configuració de l'editor

CAS D'ÚS	Modificar la vista de la pissarra
Descripció	Ens permet obtenir i establir la cadena de caràcters que defineix el contingut i configuració de l'editor en un moment determinat.
Actors	Sistema.
Precondició	El sistema ha de conèixer la configuració a enviar a l'editor.
Flux principal	1.- El sistema envia una cadena de caràcters a l'editor amb la configuració d'aquest i el seu contingut. 2.- El sistema captura una cadena de caràcters amb el contingut de l'editor.
Postcondició	Es configura els paràmetres de l'editor o s'obté el contingut d'aquest.
Comentaris	Veure diagrama de casos d'ús.

4.2.2.3.1 Fitxa de casos d'ús establir paràmetres de l'editor

CAS D'ÚS	Modificar la vista de la pissarra
Descripció	Ens permet establir el contingut i altres aspectes sobre la configuració de l'editor.
Actors	Sistema.
Precondició	El sistema ha de conèixer la configuració a enviar a l'editor.
Flux principal	Es podran dur a terme les següents accions: <ul style="list-style-type: none"> • Establir el contingut de l'editor • Establir l'idioma de l'editor • Establir el mode de l'editor
Subfluxos	Per més comoditat es mostren a continuació.
Postcondició	Tenim l'editor carregat a memòria amb el contingut, idioma i mode que li correspongui.
Comentaris	En el contingut hi haurà la solució enviada per l'alumne o res si encara no l'ha solucionat. L'idioma el determina l'usuari quan entra al sistema. El mode de l'editor pot ser de dos tipus, o bé només visualitzador i no deixa modificar el contingut, o bé deixa modificar el contingut.

1. Suflux establir el contingut de l'editor

El sistema envia a l'editor una cadena de caràcters amb un format preestablert. A partir del contingut d'aquesta cadena l'editor sap crear els objectes necessaris que acabaran formant part del contingut de la pissarra. D'aquesta cadena s'obindrà el tipus d'autòmat, els alfabetes, els estats i les transicions.

2. Subflux establir l'idioma de l'editor

El sistema enviarà a l'editor una cadena de caràcters on hi haurà una llista amb tots els textos que pugui mostrar l'editor, des de títols fins als noms dels botons o els missatges d'error.

3. Subflux establir el mode de l'editor

El sistema enviarà a l'editor un enter que li indicarà a l'editor en quin mode s'ha de mostrar a l'usuari. Si l'editor rep un 0 funcionarà en mode visualitzador, no permetrà alterar el contingut de la pissarra. Si l'editor rep un 1 permetrà manipular el contingut de la pissarra.

5 Anàlisi i disseny

En aquest capítol veurem els diagrames d'activitat, de seqüència i de classes que ens permetran representar el comportament dinàmic descrit en l'anàlisi de requeriments funcionals exposat en el capítol anterior. Es podrà observar que els diagrames tenen en compte el software amb que s'implementarà l'aplicació, això es degut a que la metodologia de treball usada ha estat l'XP.

S'haurà d'implementar una interfície que permeti dibuixar els autòmats i un nucli corrector per corregir-los.

La interfície simularà una pissarra on s'hi podran dibuixar els autòmats i serà implementada amb el llenguatge *Java* a través d'un *applet*, aprofitant la potència de la programació orientada a objectes, amb classes i herència, que ens ofereix.

El nucli corrector haurà de ser capaç de reconèixer el tipus de l'autòmat, ja que en funció del tipus tindrà un comportament o un altre.

5.1 Jerarquia de paquets usats

Les classes usades per implementar la interfície de l'editor es troben organitzades en paquets tal com es mostra en la següent figura.

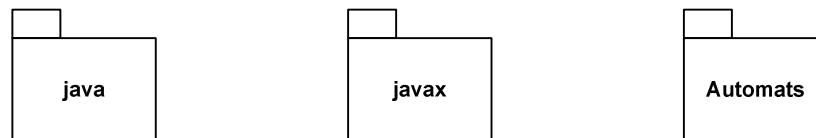


Fig. 5-1 Diagrama de paquets implementació editor.

Els dos primers paquets són paquets de la plataforma *Java*, mentre que l'últim conté les classes implementades.

Com que els paquets de *Java* són molt extensos en els pròxims diagrames veurem quins paquets específics s'han usat dels paquet *java* i quins del *javax*.

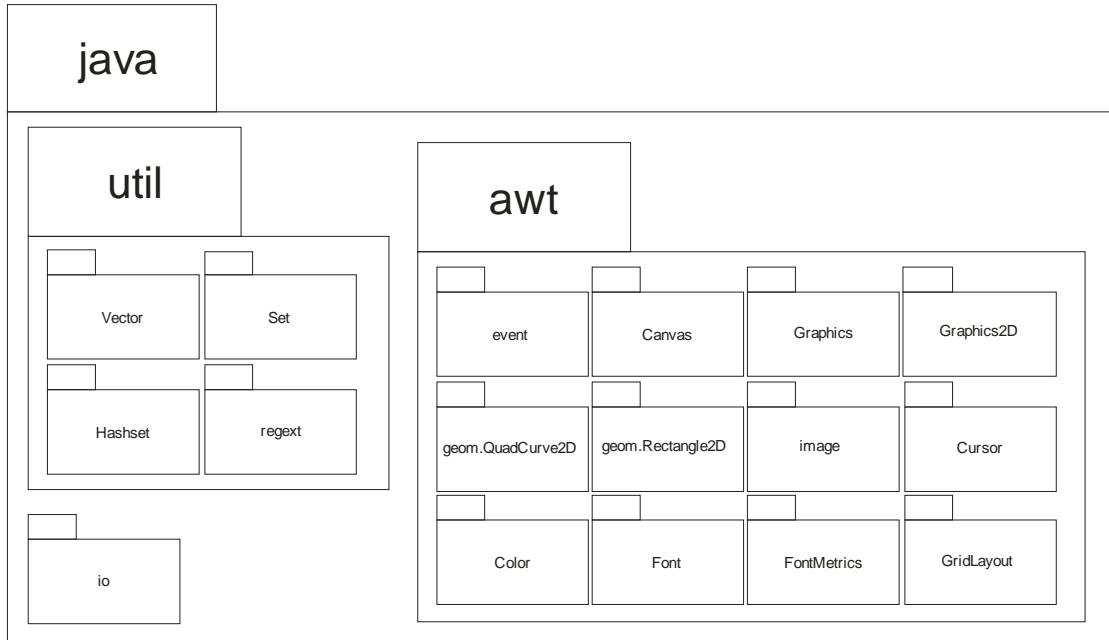


Fig. 5-2 Paquets utilitzats del paquet java.

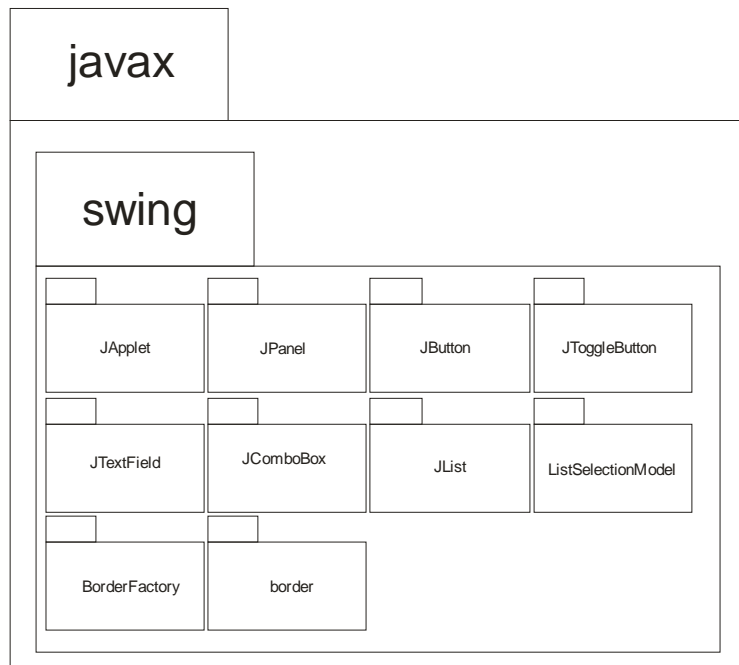


Fig. 5-3 Paquets utilitzats del paquet javax.

5.2 Diagrama de classes

En aquest punt podem veure el diagrama de classes implicades per aconseguir el funcionament desitjat de l'editor. La simbologia usada pel diagrama de classes serà la següent:

- Per representar les classes ho farem amb un rectangle dividit en tres parts, a la part superior hi posarem el nom de la classe, a la part central els atributs de la classe i a la part inferior les operacions de la classe.

nomClasse
-atribut1 : Integer +atribut2 : String #atribut3 : Byte = 1
+operacio1() : Integer +operacio2(entrada parametre1 : Integer)

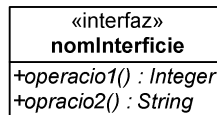
- El nom de la classe s'escriurà en cursiva si la classe és abstracta.

nomClasseAbstracta
-atribut1 : Integer +atribut2 : String #atribut3 : Byte = 1
+operacio1() : Integer +operacio2(entrada parametre1 : Integer)

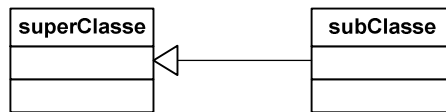
- La sintaxi usada pels atributs serà *visibilitat nomAtribut:tipus = valorInicial*. La visibilitat s'indicarà posant al davant de l'atribut un dels símbols de la següent taula:

Símbol	Accés	Accessible des de...
+	Públic	Tots els objectes de l'aplicació.
-	Privat	Instàncies de la pròpia classe.
#	Protegit	Instàncies de la pròpia classe i subclasses.
~	Paquet	Tots els objectes del mateix paquet.

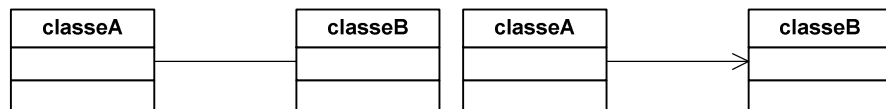
- La sintaxi usada per les operacions serà *visibilitat nomOperació(nomParàmetre:tipusParàmetre,...):tipusRetorn*. Per indicar la visibilitat s'usaran els mateixos símbols que pel cas dels atributs.
- Per representar les interfícies ho farem amb un rectangle dividit en dues parts, a la part superior hi posarem el nom de la interfície i a la part inferior les operacions.



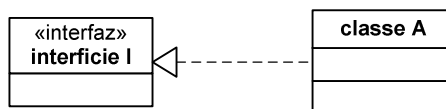
- Per representar les relacions entre les classes ho farem amb segments que aniran d'una classe A a una classe B. En funció del tipus de relació el segment tindrà un format o un altre.
 - Per representar les relacions de generalització/especificació farem servir un segment amb un extrem triangular apuntant cap a la superclasse.



- Per representar les associacions farem servir un segment sense fletxa si la comunicació és en ambdós sentits o amb una fletxa si només hi ha comunicació en un sentit.



- Per representar les realitzacions, que una classe A implementa la interfície I, ho farem amb un segment en línia discontinua amb un extrem triangular apuntant cap a la interfície.



En la següent figura es mostra el diagrama de classes de l'editor d'autòmats. No s'hi mostren ni els atributs ni les operacions associades a cadascuna de les classes ja que quedaria un diagrama molt espès i poc entenedor. Tant els atributs com les operacions es detallen en el següent punt d'aquest capítol.

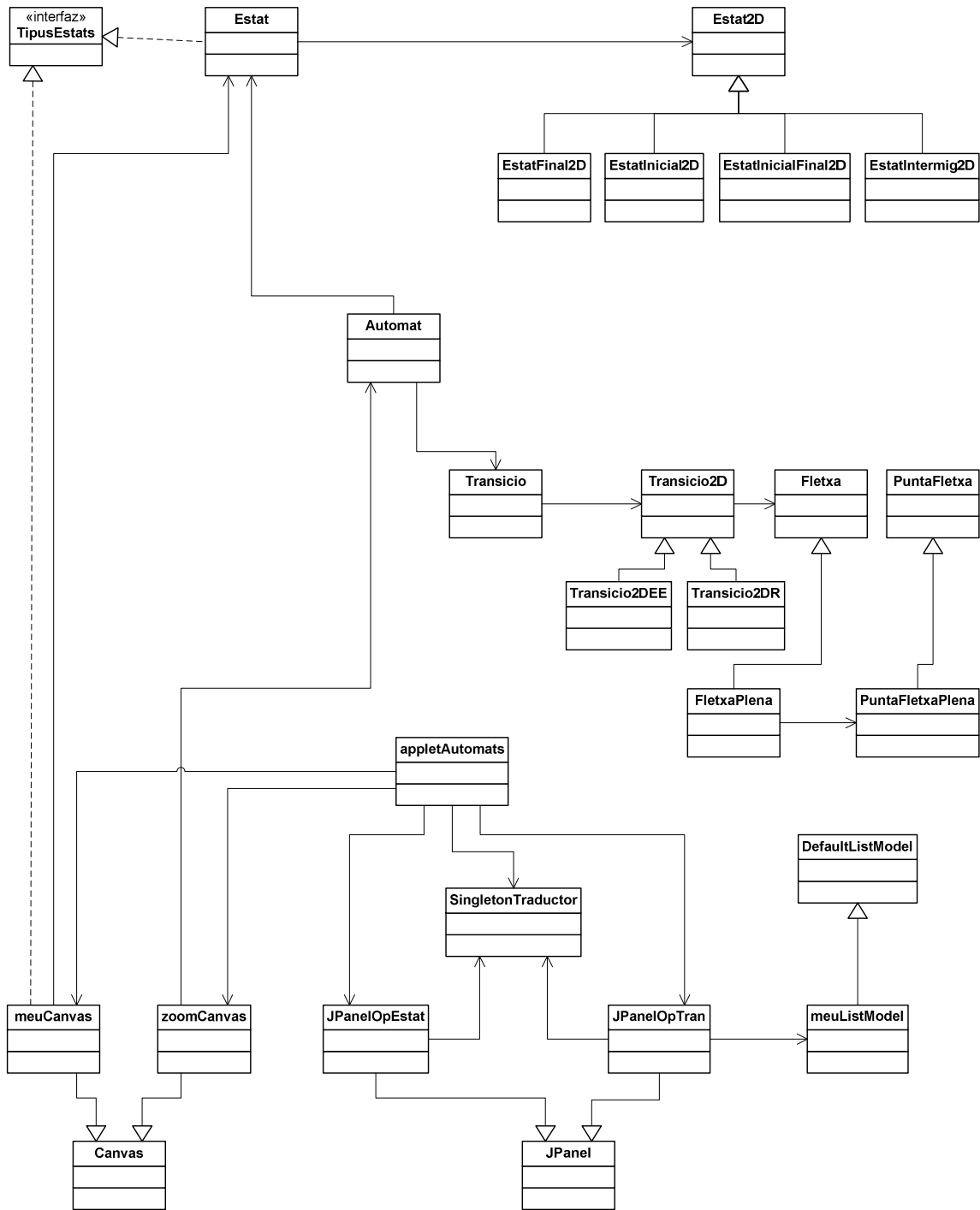


Fig. 5-4 Diagrama de classes de l'editor.

5.3 *Explicació classes implicades*

En aquest punt veurem una breu explicació de cadascuna de les classes a implementar per dur a terme el desenvolupament de l'editor d'autòmats.

appletAutomats

Aquesta classe hereta de la classe *JApplet* que ens proporciona el paquet *javax.swing* de *Java*. És l'encarregada de crear la interfície de l'editor d'autòmats utilitzant els paquets *Swing* i *AWT* de *Java*. Gràcies aquests paquets podrem afegir els controls necessaris (botons, caselles de text, llistes, àrees de dibuix, etc) que ens permetran dur a terme les accions exposades en l'especificació de requeriments.

Aquesta classe s'implementarà amb el *Visual Editor*, eina ja explicada anteriorment i que ens proporciona l'*Eclipse*, de manera que la majoria d'atributs i mètodes d'aquesta classe seran generats automàticament, és per això que no les explicarem.

Un dels mètodes que no es generarà automàticament és el mètode *toString2()*. Aquest mètode és diu així per que no es vol sobrecarregar el mètode *toString()* de la classe *JApplet*, i la seva funció és retornar un *String* amb el contingut de l'editor en un moment determinat. Aquest mètode el cridarem des de la plana web per tal d'informar al sistema del contingut de l'editor.

JPanelOpEstat

Aquesta classe hereta de la classe *JPanel* que ens proporciona el paquet *javax.swing* de *Java*. Aquesta classe també forma part de la interfície de l'editor. Aquí hi tindrem els controls necessaris per operar amb els estats a l'hora de dibuixar un autòmat.

Aquesta classe també s'implementarà amb el *Visual Editor*.

JPanelOpTran

Aquesta classe hereta de la classe *JPanel* que ens proporciona el paquet *javax.swing* de *Java*. Aquesta classe també forma part de la interfície de l'editor. Aquí hi tindrem els controls necessaris per operar amb les transicions a l'hora de dibuixar un autòmat.

Aquesta classe també s'implementarà amb el *Visual Editor*.

meuListModel

Aquesta classe hereta de la classe *DefaultListModel* que ens proporciona el paquet *javax.swing* de *Java*.

Si treballem amb autòmats de pila, al seleccionar una transició entre dos estats es mostrarà una llista amb els símbols que accepta aquesta transició, aquest tema es troba explicat en més detall al capítol d'anàlisi de requeriments, la classe responsable de mantenir aquesta llista és *meuListModel*.

Els atributs i mètodes d'aquesta classe són:

meuListModel
+addElement(entradas : String[])
+toArrayString() : String[]

meuCanvas

Aquesta classe hereta de la classe *Canvas* que ens proporciona el paquet *javax.awt* de *Java*. La classe *Canvas* ens proporciona una àrea que permet dibuixar figures simples com ara línies o polígons, ideal per representar els diagrames d'estats dels nostres autòmats.

En la següent figura es mostren els atributs i mètodes d'aquesta classe.

meuCanvas
-a : Automat -zoomX : float = 1 -zoomY : float = 1 -offsetX : Integer = 0 -offsetY : Integer = 0 -x : Integer -y : Integer -estatOrigen : Estat -estatDesti : Estat -numTransicions : Integer = -1 -transicioNova : Boolean = False -estatNou : Boolean = False -mouEstat : Boolean = False -numEstats : Integer = -1
+meuCanvas() +meuCanvas(entrada sAutomat : String) +getAutomat() : Automat +setXY(entrada x : Integer, entrada y : Integer) +setCursor() +setXOffset(entrada x : Integer) +setYOffset(entrada y : Integer) +setXYOffset(entrada x : Integer, entrada y : Integer) +setZoom(entrada x : float, entrada y : float) +addEstat(entrada x : Integer, entrada y : Integer) +addTransicio() +removeEstat() +removeTransicio() +moureEstat(entrada x : Integer, entrada y : Integer) +selElement(entrada x : Integer, entrada y : Integer) : Object

zoomCanvas

Aquesta classe també hereta de la classe *Canvas* i serà una vista en miniatura de la pissarra de l'editor. Servirà per augmentar i disminuir el zoom i fer translacions del contingut de l'editor.

Els atributs i mètodes de la classe *zoomCanvas* són els següents:

zoomCanvas
-vista : Rectangle2D.Float -a : Automat -xMouse : Integer -yMouse : Integer -zoom : Integer -t_zoom : float[] = {0.5f,0.75f,1,1.5f,2,2.5f,3,3.5f}
+zoomCanvas(entrada a : float[]) +setX(entrada x : Integer) +setY(entrada y : Integer) +setXY(entrada x : Integer, entrada y : Integer) +setXYMouse(entrada x : Integer, entrada y : Integer) +arrestarVista(entrada x : Integer, entrada y : Integer) +getXOffset() : Integer +getYOffset() : Integer +getZoom() : float +posarZoom() +treureZoom()

SingletonTraductor

Com que la plataforma ACME és multilingüe el nostre editor també ho ha de ser. Aquesta classe serà una estructura de dades que tindrà la funció de diccionari traductor, com a clau tindrem la paraula en l'idioma original i com element la paraula amb l'idioma que l'usuari esculli en el moment d'entrar a l'ACME.

Aquesta estructura s'inicialitzarà al carregar l'editor a partir d'una cadena de text el format de la qual s'explicarà en pròxims capítols.

Els mètodes i atributs d'aquesta classe són:

SingletonTraductor
-singleton : SingletonTraductor -hm : HashMap
-SingletonTraductor() +getSingleton() : SingletonTraductor +setTraductor(entrada s : String) +getTraduccio(entrada clau : String)

Per dissenyar aquesta classe hem fet ús del patró *singleton*, aquest patró és molt útil quan ens volem assegurar que d'una determinada classe només hi hagi una sola instància, com és el cas del nostre diccionari.

Fixem-nos que el constructor el tenim com a un mètode privat, de manera que la única forma d'obtenir una instància de la classe serà usant el mètode *getSingleton()* que sempre ens retornarà la mateixa instància. Al carregar el nostre editor farem ús del mètode *setTraductor()* per inicialitzar-lo, un cop inicialitzat farem ús del mètode *getTraduccio()* cada vegada que vulguem posar un text en un botó, en una etiqueta, etc.

Estat

Aquesta classe ens permet guardar la informació de cadascun dels estats de què estar format un autòmat. Fixem-nos que d'un estat només ens interessa guardar el nom i el tipus, a part d'un identificador únic que ens servirà per reconèixer l'estat. Un estat podrà ser de quatre tipus: inicial, final, inicial i final al mateix temps i no final. Aquests quatre tipus estan definits dins de la interfície *TipusEstats*.

De la representació gràfica de l'estat se'n encarregarà la classe *Estat2D*, quedant desacobrada la representació gràfica de la representació a nivell lògic.

Els atributs i mètodes són els següents:

Estat
-Id : Integer -nom : String -tipus : Integer -estat : Estat2D
+Estat(entrada Id : Integer) +Estat(entrada Id : Integer, entrada nom : String) +Estat(entrada sEstat : String) +getId() : Integer +getNom() : String +getTipus() : Integer +setId(entrada Id : Integer) +setNom(entrada nom : String) +setTipus(entrada tipus : Integer) +toString() : String +toString(entrada c : Char)

Podem veure que la classe *Estat* disposa de diferents constructors, el tercer serveix per crear un estat a partir d'un *String*, que ens servirà a l'hora de restaurar el contingut de l'editor. Pel mateix motiu em sobrecarregat el mètode *toString()*, que pertany a la classe *Object* de *Java* de la qual hereten la resta de classes, i que ens servirà per crear la cadena de text a l'hora d'enviar el contingut de l'editor a les bases de dades de l'ACME.

Estat2D

Aquesta classe ens servirà per representar gràficament els estats de manera que si algun dia es vol canviar la forma en què es dibuixen els estats en l'editor només caldrà tocar aquesta classe. Ens trobem davant d'una classe

abstracta, ja que un estat es dibuixarà diferent segons els tipus que tingui. No es representarà igual un estat inicial que un estat final, però tindran atributs i mètodes comuns.

A continuació podem veure els atributs i mètodes d'aquesta classe.

Estat2D
-x : Integer -y : Integer
+getX() : Integer +getY() : Integer +setX(entrada x : Integer) +setY(entrada y : Integer) +setXY(entrada x : Integer, entrada y : Integer) +contexY(entrada x : Integer, entrada y : Integer) : Boolean +getRadi() : Integer +dibuixa(entrada g : Graphics, entrada c : Color, entrada nom : String) +pinta(entrada g : Graphics, entrada c : Color, entrada pc : Color, entrada nom : String)

Els mètodes *dibuixa()* i *pinta()* són abstractes, ja que en funció del tipus d'estat la manera de dibuixar-lo i pintar-lo pot canviar.

Per últim destacar que el mètode *dibuixa()* serveix per representar un estat dibuixant només el contorn, mentre que el mètode *pinta()* el dibuixa omplint el seu interior amb el color del paràmetre *pc*.

EstatFinal2D

Aquesta classe ens servirà per representar gràficament un estat final, de manera que si algun dia es vol canviar la manera en què es mostra un estat final només caldrà tocar aquesta classe.

En la següent figura veiem que només hem afegit un constructor amb paràmetres i que la resta d'atributs i mètodes seran els mateixos que els de la superclasse.

EstatFinal2D
+EstatFinal2D(entrada x : Integer, entrada y : Integer)

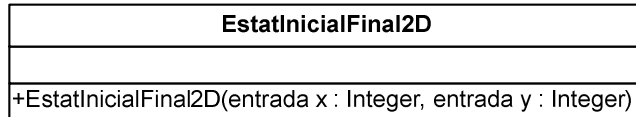
EstatInicial2D

Aquesta classe és igual a la anterior però per representar estats inicials.

EstatInicial2D
+EstatInicial2D(entrada x : Integer, entrada y : Integer)

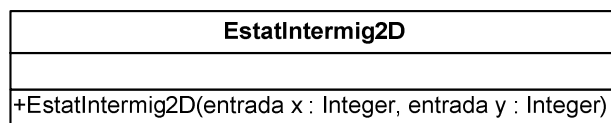
EstatInicialFinal2D

Aquesta classe també és igual a les dues anterior però per representar estats que són inicials i finals al mateix temps.



EstatIntermig2D

Aquesta classe també és igual a les tres anteriors però servirà per representar els estats no finals.

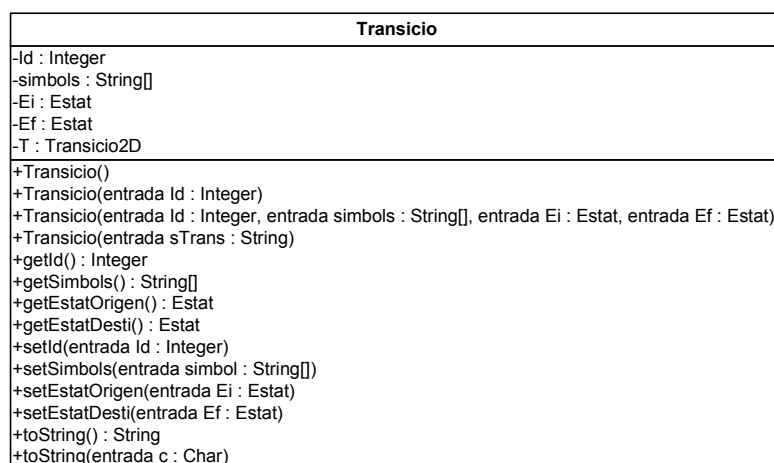


Transicio

La classe *Transicio* ens servirà per guardar les dades de cadascuna transicions que puguin formar part de l'autòmat. D'una transició ens interessa guardar els símbols que reconeix, l'estat d'origen i l'estat de destí. També tindrem un enter que farà d'identificador únic.

De la representació gràfica de la transició se n'encarregarà la classe *Transicio2D*, d'aquesta manera un cop més la representació gràfica quedarà desacoblada de la representació lògica.

Els atributs i mètodes d'aquesta classe són els següents:



Podem veure que la classe *Transicio* disposa de diferents constructors, el quart serveix per crear una transició a partir d'un *String*, que ens servirà a l'hora de restaurar el contingut de l'editor. Pel mateix motiu hem

sobrecarregat el mètode *toString()*, que pertany a la classe *Object* de *Java* de la qual hereten la resta de classes, i que ens servirà per crear la cadena a l'hora d'enviar el contingut de l'editor a les bases de dades de l'ACME.

Transicio2D

Aquesta classe ens servirà per representar gràficament les transicions, de manera que si algun dia es vol canviar la forma en què es dibuixen les transicions a l'editor només caldrà tocar aquesta classe. Ens trobem davant d'una classe abstracta, ja que una transició es dibuixarà diferent segons si el seu estat d'origen i destí són o no el mateix estat.

Els atributs i mètodes d'aquesta classe són:

Transicio2D
-fletxa : Fletxa
+getXOrigen() : Integer +getYOrigen() : Integer +getXDesti() : Integer +getYDesti() : Integer +getXControl() : Integer +getYControl() : Integer +setXOrigen(entrada x : Integer) +setYOrigen(entrada y : Integer) +setXDesti(entrada x : Integer) +setYDesti(entrada y : Integer) +setXControl(entrada x : Integer) +setYControl(entrada y : Integer) +setFletxa(entrada x1 : Integer, entrada y1 : Integer, entrada xc : Integer, entrada yc : Integer, entrada x2 : Integer, entrada y2 : Integer) +setOrigenDestiFletxa(entrada x1 : Integer, entrada y1 : Integer, entrada x2 : Integer, entrada y2 : Integer) +conteXY(entrada x : Integer, entrada x : Integer) +dibuixa(entrada g : Graphics, entrada c : Color, entrada simbols : String) +pinta(entrada g : Graphics, entrada c : Color, entrada simbols : String)

Tal com passava amb la classe *Estat2D* els mètodes *dibuixa()* i *pinta()* són abstractes, ja que en funció del tipus de transició la manera de dibuixar-la i pintar-la pot canviar.

Amb el mètode *dibuixa()* mostrarem un arc que unirà dos estats, o un estat amb ell mateix. Amb el mètode *pinta()* farem el mateix però amb negreta, aquest mètode el podem usar per exemple per remarcar la transició que l'usuari tingui seleccionada.

El mètode *setOrigenDestiFletxa* també és abstracte ja que varia segons el tipus de transició. Amb aquest mètode dibuixem una transició entre dos estats amb un cert grau de curvatura. La forma de calcular la curvatura varia segons si la transició va d'un estat A a un estat B o si va d'un estat A al mateix estat A.

Transicio2DEE

Aquesta classe ens servirà per dibuixar una transició que vagi d'un estat A cap a un estat B essent A i B diferents estats. Si algun dia volguéssim canviar la forma de representar una transició entre dos estats només hauríem de tocar aquesta classe.

A continuació es mostren els atributs i mètodes de la classe.

Transicio2DEE
+Transicio2DEE() +Transicio2DEE(entrada x1 : Integer, entrada y1 : Integer, entrada x2 : Integer, entrada y2 : Integer)

Podem observar que no aporta cap atribut ni cap mètode nou respecte la superclasse.

Transicio2DR

Aquesta classe ens servirà per representar visualment una transició que vagi d'un estat A a ell mateix. Si algun dia volguéssim canviar la forma de representar aquest tipus de transició només hauríem de tocar aquesta classe.

A continuació es mostren els atributs i mètodes de la classe.

Transicio2DR
+Transicio2DR() +Transicio2DR(entrada x1 : Integer, entrada y1 : Integer, entrada x2 : Integer, entrada y2 : Integer)

Fletxa

Aquesta classe ens servirà per dibuixar la fletxa que representa una transició. Aquesta classe és abstracta ja que hi ha una infinitat de formes de dibuixar una fletxa, i un dels requisits que ha de complir l'editor és que en el futur pugui ser adaptat fàcilment per ser usat en altres matèries que no siguin les d'autòmats finits o de pila.

Fletxa
-quad : QuadCurve2D.Double -punta : PuntaFletxa +getXOrigen() : Integer +getYOrigen() : Integer +getXDesti() : Integer +getYDesti() : Integer +getXControl() : Integer +getYControl() : Integer +getAlfa() : Double +getTheta() : Double +setXOrigen(entrada x : Integer) +setYOrigen(entrada y : Integer) +setXDesti(entrada x : Integer) +setYDesti(entrada y : Integer) +setXControl(entrada x : Integer) +setYControl(entrada y : Integer) +setFletxa(entrada x1 : Integer, entrada y1 : Integer, entrada xc : Integer, entrada yc : Integer, entrada x2 : Integer, entrada y2 : Integer) +contexY(entrada x : Integer, entrada y : Integer) +dibuixa(entrada g : Graphics, entrada c : Color) +pinta(entrada g : Graphics, entrada c : Color)

Un cop més tenim els mètodes abstractes *dibuixa()* i *pinta()*. La majoria de mètodes d'aquesta classe són els mateixos que els de la classe *Transicio2D*. Ens apareixen dos mètodes nous: *getAlfa* i *getTheta*. L'angle alfa és l'angle d'inclinació de la línia recta que uneix els punts d'inici i de

fi de la fletxa respecte l'eix x. L'angle theta és l'angle d'inclinació de la punta de la fletxa també respecte l'eix x.

PuntaFletxa

Aquesta classe ens servirà per dibuixar les puntes de les fletxes, també és abstracta ja que podem tenir una gran varietat de puntes. En aquesta classe hi tindrem els atributs i mètodes comuns als diferents tipus de puntes possibles, els mètodes per *dibuixar()* o *pintar()* cadascun dels tipus de punta els trobarem a les subclasses corresponents.

Vegem els mètodes i atributs de la classe *PuntaFletxa*.

PuntaFletxa
-x : Integer -y : Integer -theta : Double
+getX() : Integer +getY() : Integer +getTheta() : Double +setX(entrada x : Integer) +setY(entrada y : Integer) +setTheta(entrada theta : Double) +setXY(entrada x : Integer, entrada y : Integer) +conteXY(entrada x : Integer, entrada y : Integer) : Boolean +dibuixa(entrada g : Graphics, entrada c : Color)

Com a mètodes abstractes té el mètode *dibuixa()*.

FletxaPlena

Aquesta classe ens servirà per crear una fletxa amb la punta plena. Si algun dia es vol modificar el format de la fletxa n'hi haurà prou en modificar aquesta classe o crear una nova classe que també hereti de la classe Fletxa.

Els mètodes i atributs d'aquesta classe són:

FletxaPlena
+FletxaPlena() +FletxaPlena(entrada x1 : Integer, entrada y1 : Integer, entrada x2 : Integer, entrada y2 : Integer) +FletxaPlena(entrada x1 : Integer, entrada y1 : Integer, entrada xc : Integer, entrada yc : Integer, entrada x2 : Integer, entrada y2 : Integer)

El constructor amb quatre paràmetres servirà per dibuixar una fletxa recta entre els punts 1 i 2, mentre que amb el constructor que té sis paràmetres podrem dibuixar fletxes corbes, la curvatura la podrem assignar usant el punt de control (paràmetres xc i yc).

PuntaFletxaPlena

Amb aquesta classe dibuixarem puntes de fletxa plenes. Si més endavant és té la necessitat de modificar el format de la punta de la fletxa només caldrà

modificar aquesta classe o crear-ne una de nova que també hereti de la classe *PuntaFletxa*.

Els mètodes i atributs d'aquesta classe són:

PuntaFletxaPlena
+PuntaFletxaPlena(entrada x : Integer, entrada y : Integer, entrada theta : Double)

Automat

Amb aquesta classe guardarem l'autòmat, per fer-ho necessitarem un vector amb els estats i un altre vector per les transicions, un punter que apunti a l'estat actual i un punter que apunti a la transició actual, també necessitarem guardar l'alfabet de l'autòmat i l'alfabet de la pila i el tipus de l'autòmat, i per últim tindrem un punter a l'estat amb identificador més gran i un altre punter a la transició amb identificador més gran.

Vegem els atributs i els mètodes d'aquesta classe.

Automat
-vE : Vector -vT : Vector -vA : Vector -vAP : Vector -eActual : Estat -tActual : Transicio -eMax : Estat -tMax : Transicio -sTipus : String
+Automat() +Automat(entrada sAutomat : String) +getTipus() : String +getEstatActual() : Estat +getTransicioActual() : Transicio +getElementXY() : Object +getTransicions(entrada e : Estat) : Set +getTransicions(entrada eo : Estat, entrada ed : Estat) : Set +getNumeroEstats() : Integer +getMaxEstat() : Integer +getNumeroTransicions() : Integer +getMaxTransicio() : Integer +getAlfabet() : String +getAlfabetPila() : String +getSimbolsUsats() : String +getSimbolsUsatsPila() : String +setEstatActualXY(entrada x : Integer, entrada y : Integer) +setTransicioActualXY(entrada x : Integer, entrada y : Integer) +setTransicioActual(entrada t : Transicio) +addEstat(entrada e : Estat) +addTransicio(entrada t : Transicio) +cercaEstat(entrada id : Integer) : Estat +removeEstat() +removeTransicio() +dibuixa(entrada g : Graphics) +alfabetConte(entrada sSimbol : String) : Boolean +toString() : String +toString(entrada c : Char) : String +validaTran(entrada sLlista : String) : Integer

Amb em mètode *getElementXY()* podrem obtenir l'estat o la transició que es torba en el punt x i y o retornarà nul si no hi ha cap element. Amb els mètodes *getTransicions(...)* podrem obtenir les transicions que surten o entren a l'estat *e*, o les transicions que surten de l'estat *eo* i van cap l'estat *ed*. Amb el mètode *setEstatActualXY(...)* posarem l'estat del punt x i y com estat actual, tenim un mètode semblant per les transicions.



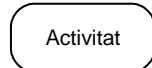

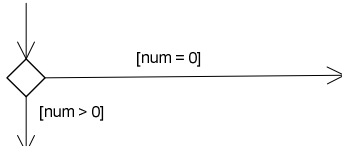
En aquesta classe també necessitarem alguns mètodes per transformar el contingut de l'editor a una cadena. Amb els mètodes *toString* i *toString(c: char)* transformem l'autòmat a una cadena de text el format de la qual s'explica en propers capítols. Amb els mètodes *getAlfabet()* i *getAlfabetPila()* obtindrem una cadena de text amb els símbols que componen l'alfabet de l'autòmat i l'alfabet de la pila, el format d'aquestes dues cadenes també s'explica en propers capítols.

5.4 Diagrames d'activitat


Els diagrames d'activitat són una bona eina que ens ofereix l'UML per tal de descriure de forma senzilla el flux d'alguns processos més o menys complexes del sistema a desenvolupar.

En aquest punt veurem els diagrames d'activitat dels processos del nostre sistema que ens han semblat més complexos. La majoria d'aquests diagrames faran referència als processos de correcció i validació de problemes que s'executaran sobre el servidor, però també veurem algun procés de l'editor com ara el procés encarregat de definir el contingut de l'editor o el procés encarregat de restaurar el contingut de l'editor. També veurem el procés que valida els enunciats del problema.

La nomenclatura que seguirem per fer aquests diagrames serà la següent:

- El punt d'inici del diagrama el representarem amb un cercle de color negre. 
- El punt de fi del diagrama el representarem amb un cercle blanc i un de més petit al seu interior de color negre. 
- Les activitats les representarem amb rectangles amb els vèrtexs arrodonits. 
- Les transicions entre les activitats les representarem amb fletxes. 
- Quan una transició derivi cap a dues o més transicions excloents ho representarem amb un rombe, i les condicions que controlen la transició a 

seguir es posaran entre claudàtors al costat de la transició.

- Les derivacions i unions d'activitats es representaran amb una barra sòlida horitzontal. 

5.4.1 Diagrama d'activitat correcció AFD

En aquest punt es mostra el procés que es segueix per corregir un AFD.

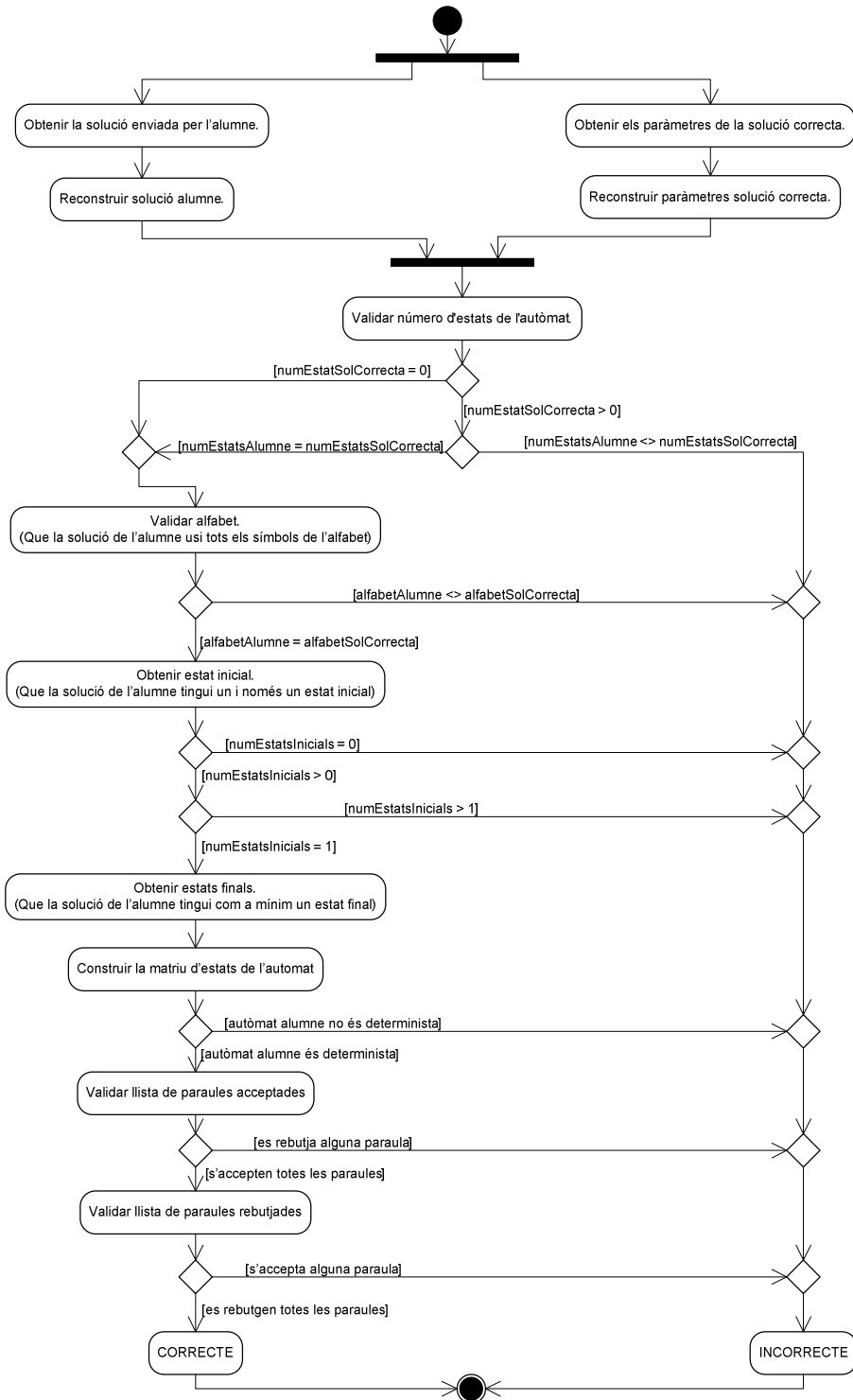


Fig. 5-5 Diagrama d'activitat procés de correcció d'un AFD

5.4.2 Diagrama d'activitat correcció AFND

En el següent diagrama es mostra el procés que es segueix per corregir un AFND.



Fig. 5-6 Diagrama d'activitat procés de correcció d'un AFND

5.4.3 Diagrama d'activitat correcció APD

En el següent diagrama es mostra el procés que es segueix per corregir un APD.

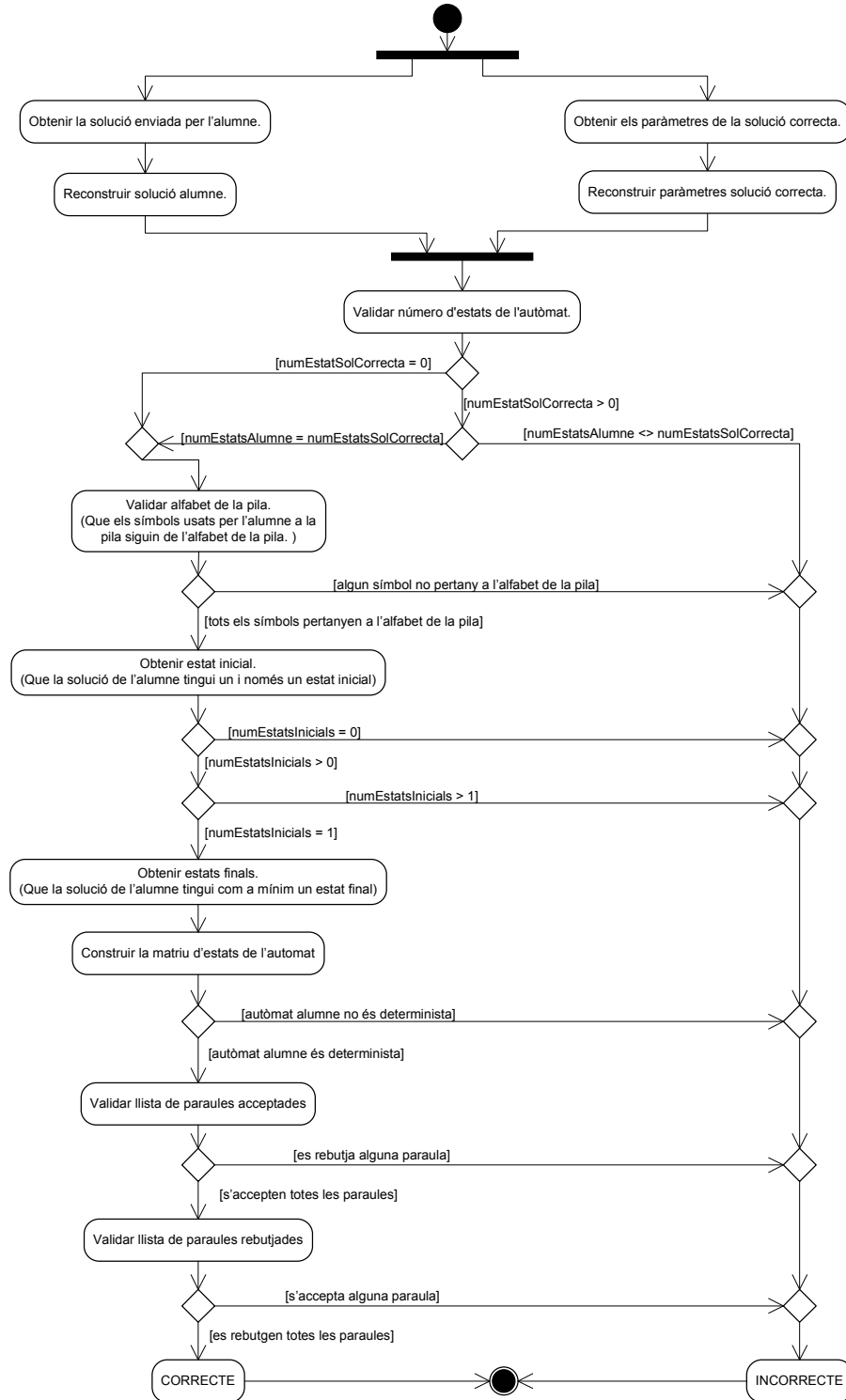


Fig. 5-7 Diagrama d'activitat procés de correcció d'un APD

5.4.4 Diagrama d'activitat correcció APND

En el següent diagrama es mostra el procés que es segueix per corregir un APND.

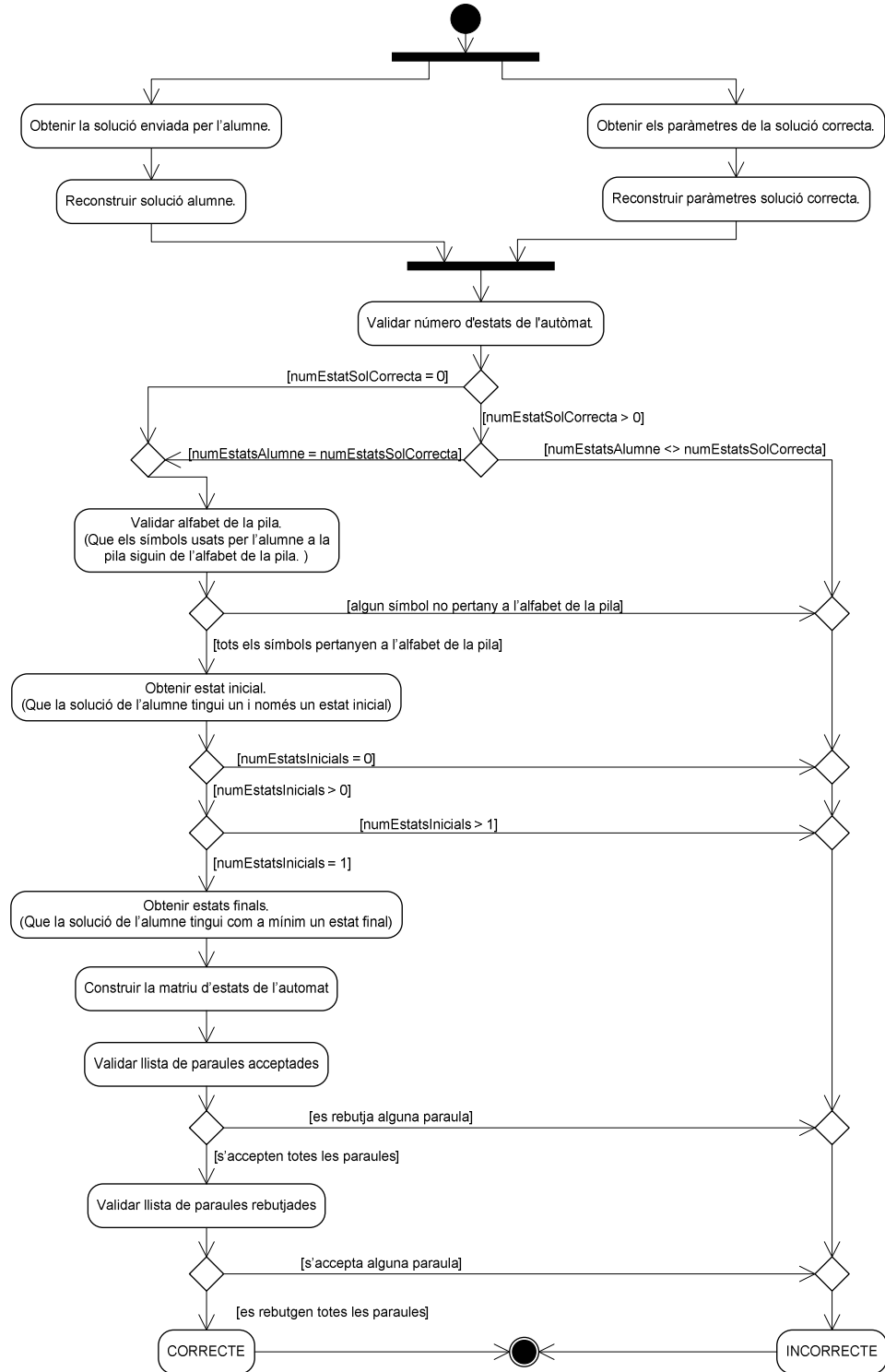


Fig. 5-8 Diagrama d'activitat procés de correcció d'un APND

5.4.5 Diagrama d'activitat per obtenir la cadena que defineix l'editor

El següent diagrama mostra el procés que es segueix per obtenir la cadena que defineix el contingut de l'editor. En el capítol 7 s'explica la nomenclatura que segueix aquesta cadena.

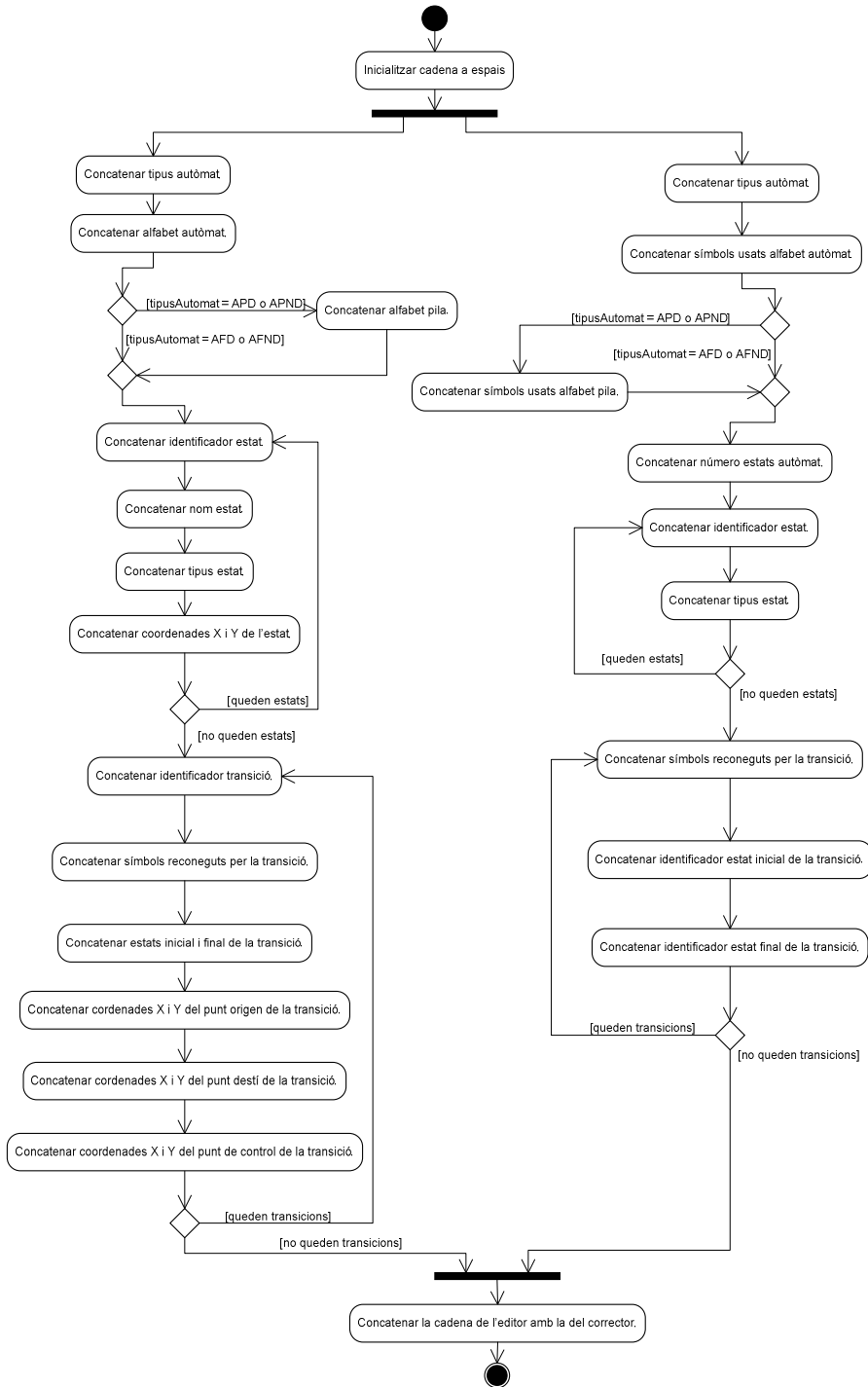


Fig. 5-9 Diagrama d'activitat procés d'obtenció de la cadena que defineix el contingut de l'editor.

5.4.6 Diagrama d'activitat per restaurar els paràmetres de l'editor

El següent diagrama mostra el procés que es segueix per restaurar el contingut de l'editor a partir de la cadena obtinguda en el procés del diagrama anterior, i que l'ACME s'haurà guardat a la seva base de dades. En el capítol 7 s'explica la nomenclatura que segueix aquesta cadena.

Aquest mateix procés també captura el mode de l'editor, lectura o lectura/escriptura, i l'idioma de l'editor. El format de la cadena que defineix l'idioma de l'editor també s'explica en el capítol 7.

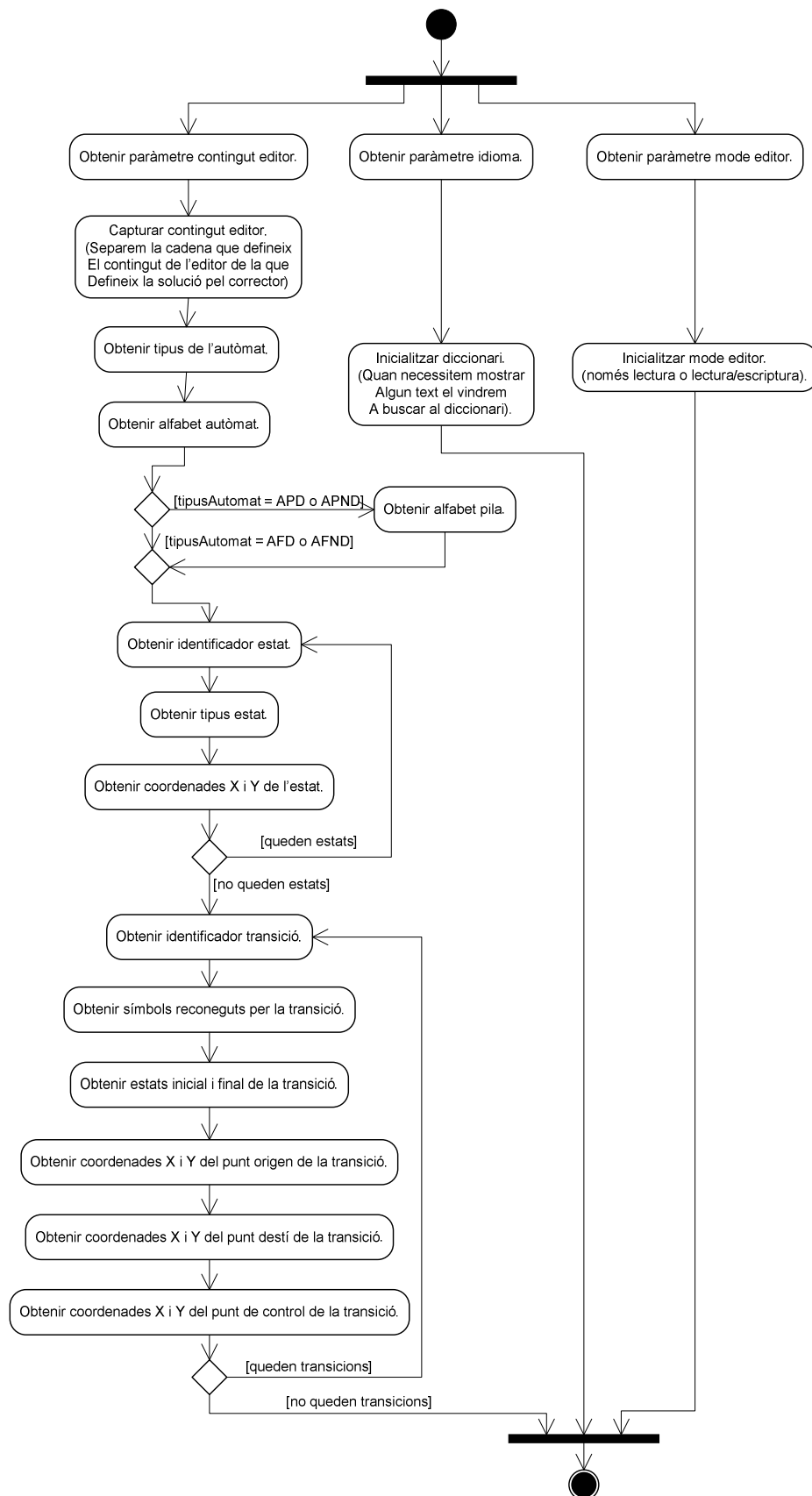


Fig. 5-10 Diagrama d'activitat procés de restauració dels paràmetres de l'editor.

5.4.7 Diagrama d'activitat per validar un fitxer de problemes

El següent diagrama mostra el procés que es segueix per validar un fitxer de problemes. Aquest procés s'executarà quan un professor o l'administrador pugin un problema a l'ACME. En el capítol 7 s'aprofundeix més sobre aquest tema.

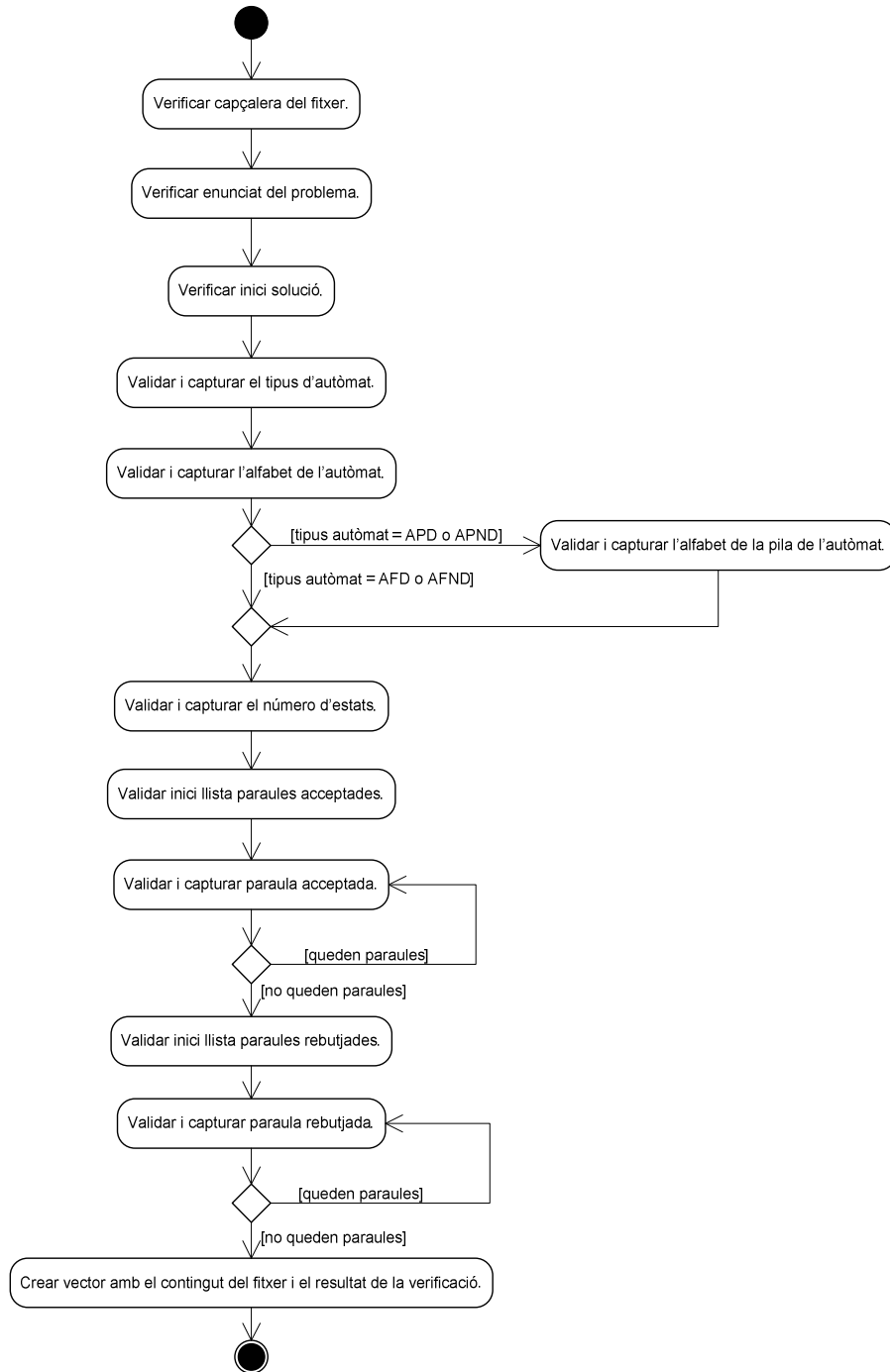


Fig. 5-11 Diagrama d'activitat procés de validació d'un fitxer de problemes.

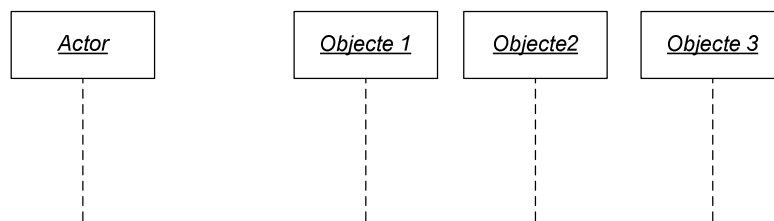
5.5 Diagrames de seqüència

I per acabar de representar el comportament dinàmic descrit a l'anàlisi de requeriments funcionals exposat en el capítol anterior, veurem els diagrames de seqüència.

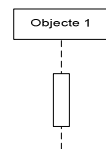
Amb els diagrames de seqüència podrem detallar com s'executen les operacions al llarg del temps, podrem veure quin missatge envia un objecte, cap a on l'envia i quan l'envia.

Per representar el diagrames de seqüència hem seguit la següent nomenclatura:

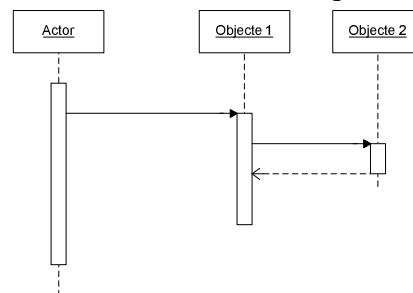
- A la part superior i dins un requadre hi tindrem els objectes que generen i reben els missatges. Estaran ordenats d'esquerra a dreta d'acord en el moment en que generen o reben el missatge. L'actor anirà a l'esquerra del tot. De cadascun dels objectes, inclòs l'actor, en sortirà una línia vertical puntejada que representarà el temps.



- El temps d'execució d'un missatge es representarà amb una barra d'activació, representada amb un rectangle situat sobre la línia puntejada que representa el temps.

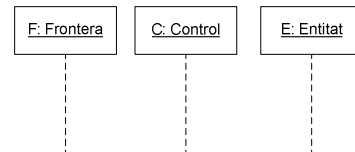


- Per representar un missatge ho farem amb una fletxa que anirà des de l'emissor cap al vèrtex superior esquerra de la barra d'activació situada sobre la línia de vida del receptor.



- Per representar la resposta del missatge, que no serà obligatòria, ho farem amb una fletxa puntejada.

- Per indicar el tipus d'objecte (frontera, control o entitat) ho farem amb les lletres F, C i E posant-les davant del nom de la classe.



5.5.1 Refresc de la pissarra i zona de zoom

Ja s'ha comentat en altres capítols d'aquesta memòria que per crear l'àrea de dibuix o pissarra usarem la classe *canvas* que ens proporciona *Java*. Per crear la zona de zoom també farem servir aquesta classe. Tant la pissarra com la zona de zoom s'hauran de mantenir actualitzades, és a dir, que qualsevol canvi que es faci a l'autòmat i que afecti a la representació gràfica d'aquest, s'haurà de veure reflectida de forma instantània tant a la pissarra com a la zona de zoom. Això ho aconseguirem cridant al mètode *repaint()* de la classe *canvas*.

Com que cada vegada que es faci un acció sobre l'autòmat s'acabarà cridant aquest mètode perquè actualitzi la pissarra, s'ha cregut convenient crear un diagrama de seqüència només per aquest mètode, d'aquesta manera els posteriors diagrames quedaran més simples.

En la següent figura podem veure com cada vegada que es crida el mètode *repaint()* de la classe *meuCanvas*, que recordem que hereta de la classe *canvas*, aquesta crida al mètode *dibuixa()* de la classe *Automat*. La classe *Automat* recorrerà la seva llista d'estats i els dibuixarà cridant al mètode *dibuixa()* de la classe *Estat*. Un cop dibuixats tots els estats cridarà al mètode *pinta()* de la classe *Estat* per pintar l'estat actual en cas que n'hi hagi. La classe *Automat* també recorrerà la seva llista de transicions i les dibuixarà cridant al mètode *dibuixa()* de la classe *Transicio*. Un cop dibuixades totes les transicions es cridarà al mètode *pinta()* de la classe *Transicio* i de la classe *Estat* per tal de remarcar la transició actual i els seus estats d'origen i fi. La classe *Estat* per dibuixar i pintar l'estat cridarà als mètodes *dibuixa()* i *pinta()* de la classe *Estat2D*, i la classe *Transicio* per dibuixar i pintar les transicions cridarà als mètodes *dibuixa()* i *pinta()* de la classe *Transicio2D*.

Fins aquí la seqüència de missatges que desencadena el mètode *repaint()* de la classe *meuCanvas* és exactament la mateixa que per la classe *zoomCanvas*, a partir d'aquí hi ha algunes petites diferències.

La classe *meuCanvas*, si té activat l'indicador de nou estat, dibuixa un estat que va seguint al punter del ratolí, d'aquesta manera l'usuari sap en tot moment la posició on es dibuixarà el nou estat. Veure la part inferior de la figura *Fig. 5-12* Diagrama de seqüència general per refrescar la pissarra i la zona de zoom.

En canvi la classe *zoomCanvas* el que fa és dibuixar el rectangle vermell que ens indica la part de l'autòmat que s'està mostrant a la pissarra. Per fer-ho es crida al mètode *setRect()* de la classe *Rectangle2d.float*. Aquesta última seqüència de missatges no es mostra a cap diagrama degut a la seva simplicitat.

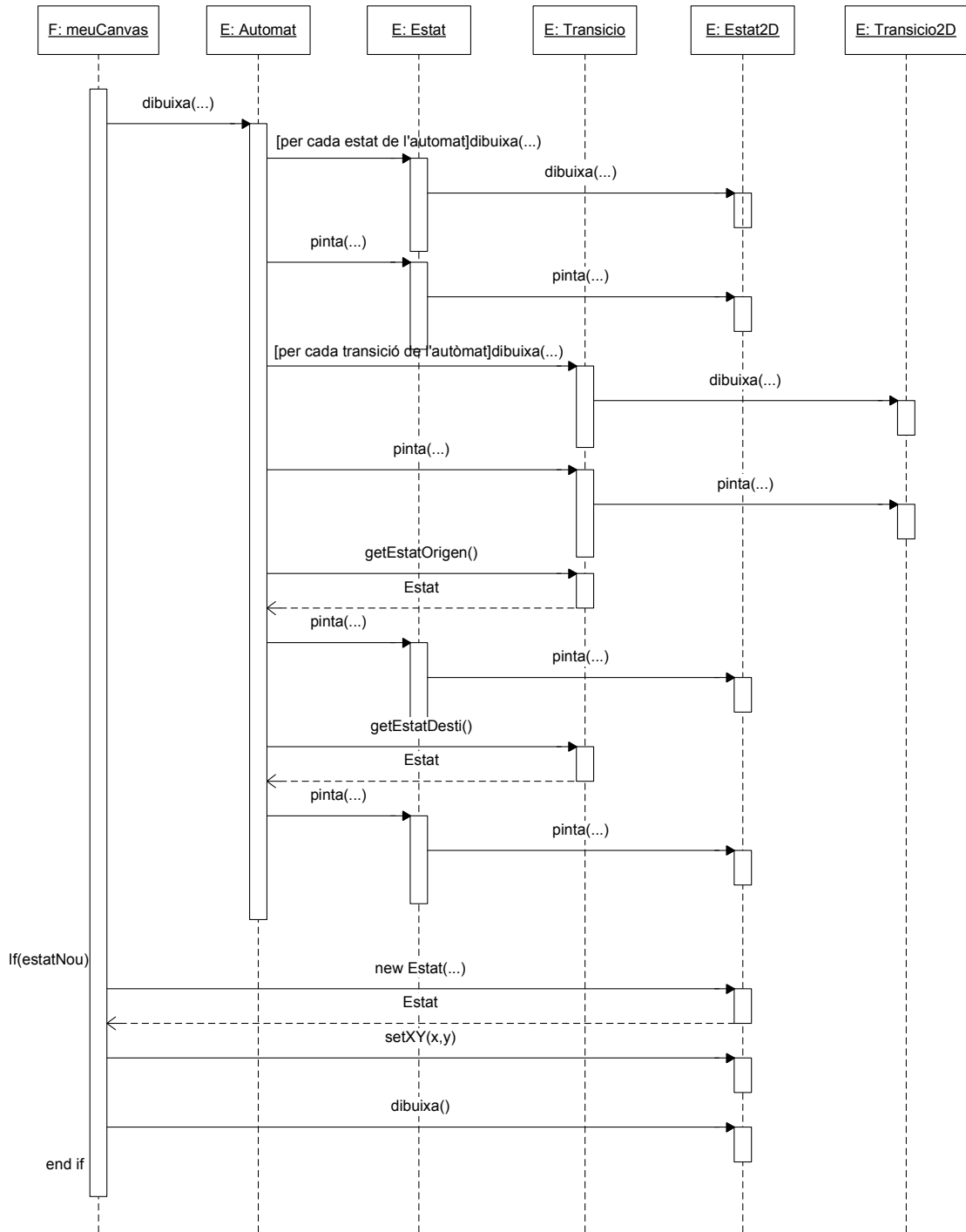


Fig. 5-12 Diagrama de seqüència general per refrescar la pissarra i la zona de zoom.

5.5.2 Afegir estats

En aquest punt veurem quines classes intervenen i com es comuniquen a l'hora d'afegir un estat. Perquè quedi més entenedor ho dividirem en tres diagrames diferents. El procés d'afegir un estat el podem dividir en tres etapes: activació del botó *Estat*, selecció del punt de la pissarra, i inserció de l'estat a la pissarra.

La seqüència de missatges que es desencadena al pulsar sobre el botó *Estat* i les classes que hi intervenen es poden veure en la següent figura. Bàsicament les accions dutes a terme són dues: per un costat amagar la pantalla de dades d'una transició i mostrar la pantalla de dades d'un estat, i per l'altre costat informar a la pissarra que anem a afegir un estat.

Un cop el botó *Estat* es troba activat l'usuari pot tornar a pulsar sobre seu per desactivar-lo i no afegir cap estat.

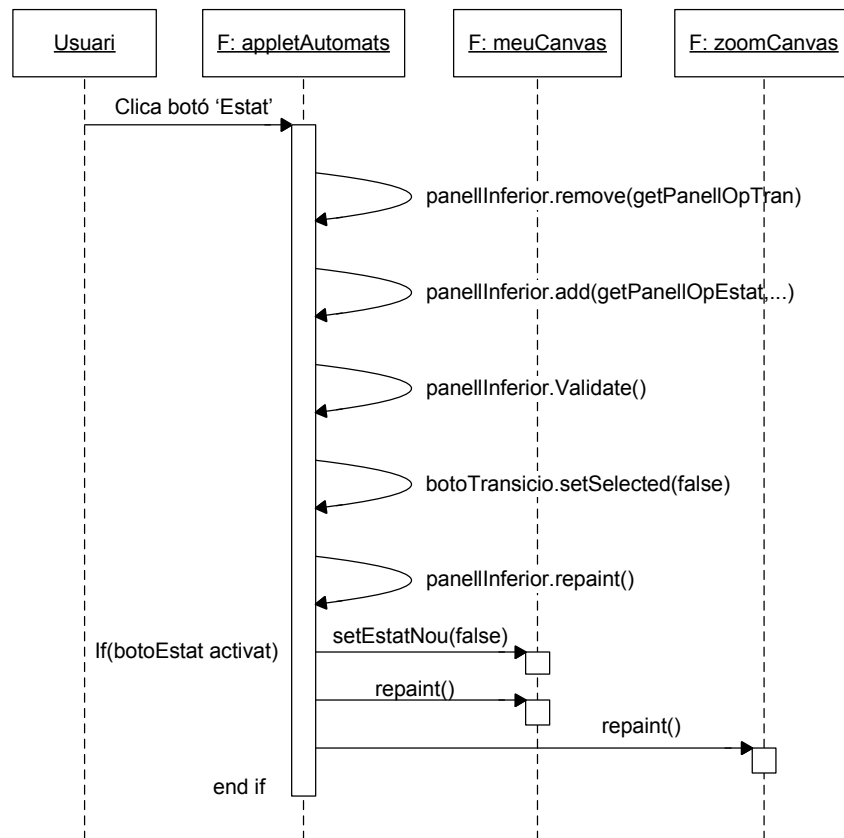


Fig. 5-13 Diagrama de seqüència per afegir un estat. Activació del boto 'Estat'.

El següent diagrama mostra els missatges i classes que intervenen quan, en ple procés d'afegir un estat, l'usuari mou el punter del ratolí per sobre de la pissarra o àrea de dibuix. Bàsicament el que es fa es mantenir actualitzades les coordenades x i y de la pissarra, tenint en compte en tot moment el zoom i la vista aplicades.

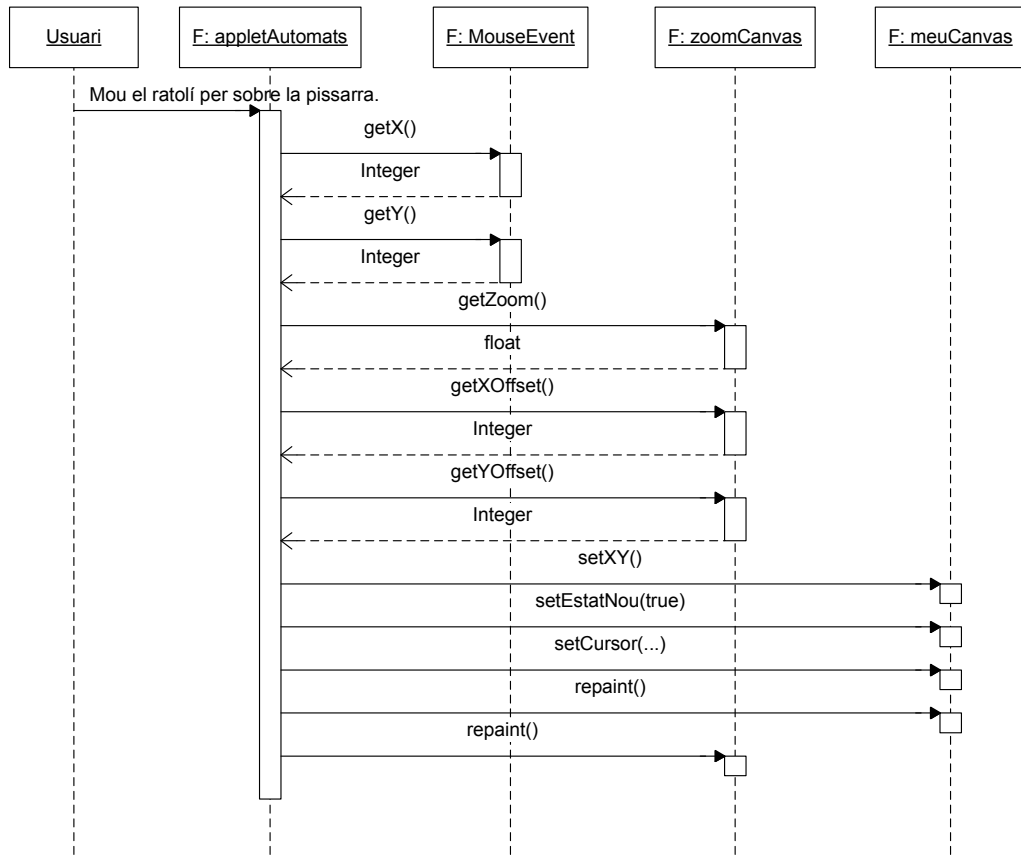


Fig. 5-14 Diagrama de seqüència per afegir un estat. Selecció del punt de la pissarra.

Per acabar amb el procés d’afegir un estat ens queda inserir-lo a la pissarra. El següent diagrama mostra els missatges i classes que intervenen en el moment en que l’usuari polsa sobre la pissarra i insereix un nou estat.

Primer calculem les coordenades x i y a on s’inserirà l’estat, tenint en compte en tot moment el zoom i la vista aplicades. Un cop tenim les coordenades creem un nou estat que per defecte serà no final, i li modifiquem les coordenades posant-li les anteriorment calculades. Afegim el nou estat a l’autòmat.

Per acabar indiquem a la pissarra que ja hem acabat d’afegir l’estat i desactivem el botó *Estat*.

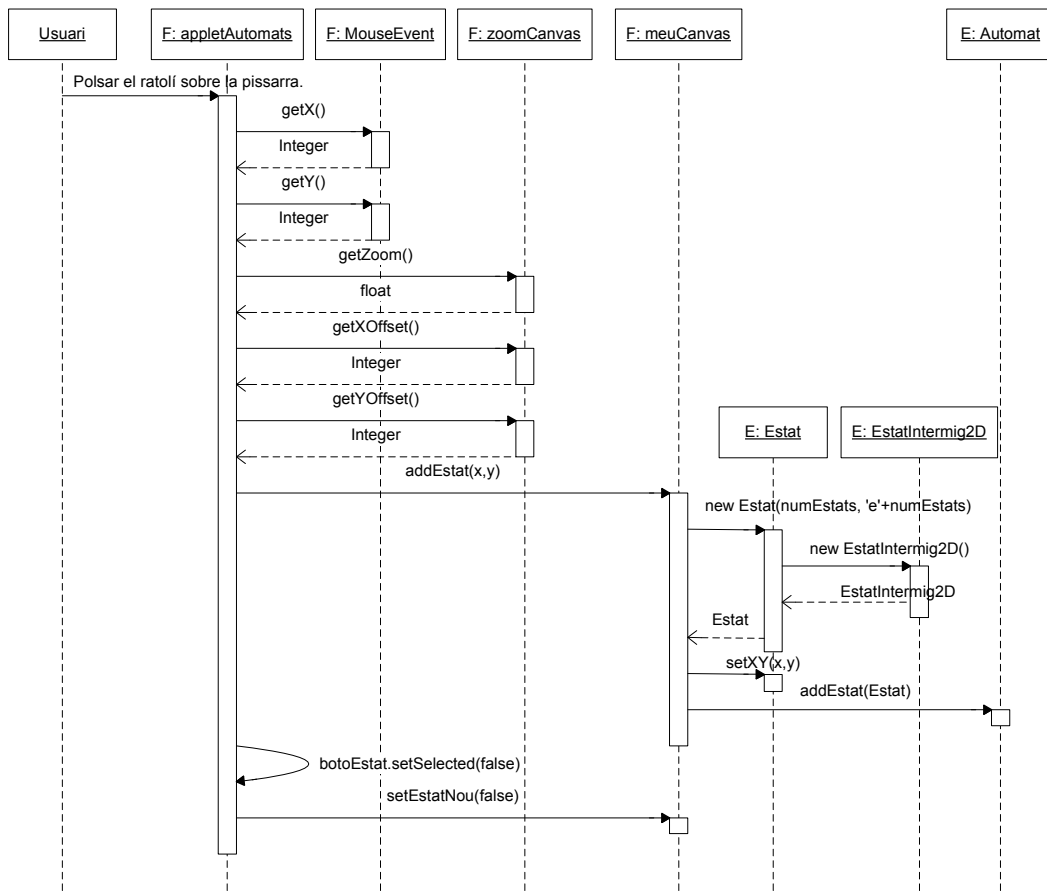


Fig. 5-15 Diagrama de seqüència per afegir un estat. Inserció de l'estat a la pissarra.

5.5.3 Afegir transicions

En aquest punt veurem quines classes intervenen i com es comuniquen a l'hora d'afegir una transició. Perquè quedi més entenedor ho dividirem en tres diagrames diferents. El procés d'afegir una transició el podem dividir en tres etapes: activació del botó *Transició*, selecció de l'estat origen, i selecció de l'estat final i inserció de la transició a la pissarra.

La seqüència de missatges que es desencadena al pulsar sobre el botó *Transició* i les classes que hi intervenen es poden veure en la següent figura. Bàsicament les accions dutes a terme són dues: per un costat amagar la pantalla de dades d'un estat i mostrar la pantalla de dades d'una transició, i per l'altre costat informar a la pissarra que anem a afegir una transició.

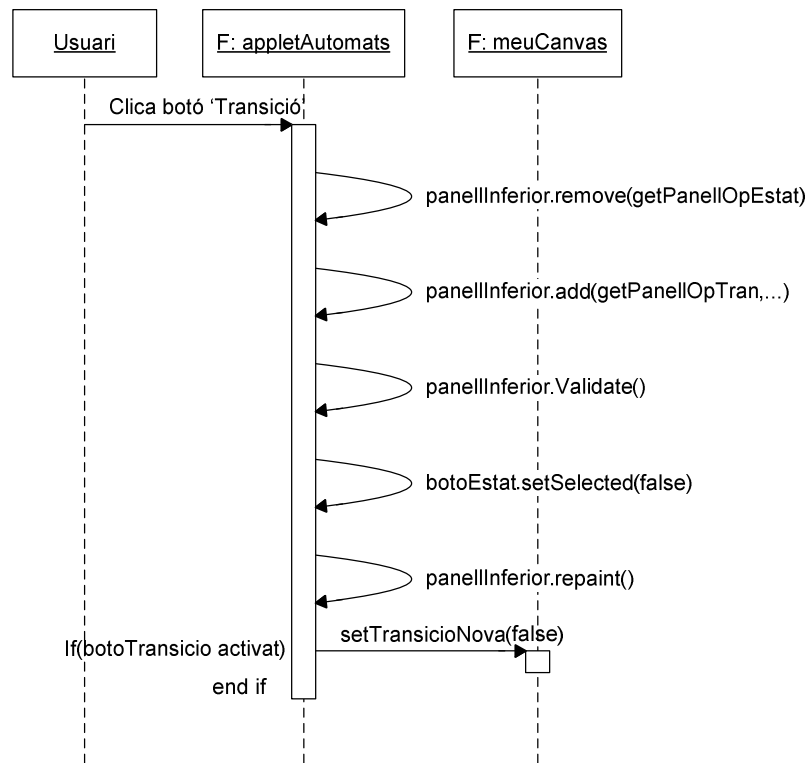


Fig. 5-16 Diagrama de seqüència per afegir una transició. Activació del botó 'Transició'.

En el següent diagrama es mostren els missatges i classes que participen al seleccionar l'estat origen de la transició. Informem a la pissarra que anem a afegir una transició. Busquem si el punt de la pissarra on ha polsat l'usuari hi ha un estat, si no n'hi ha cap avortem l'operació, si n'hi ha algun el marquem com estat actual.

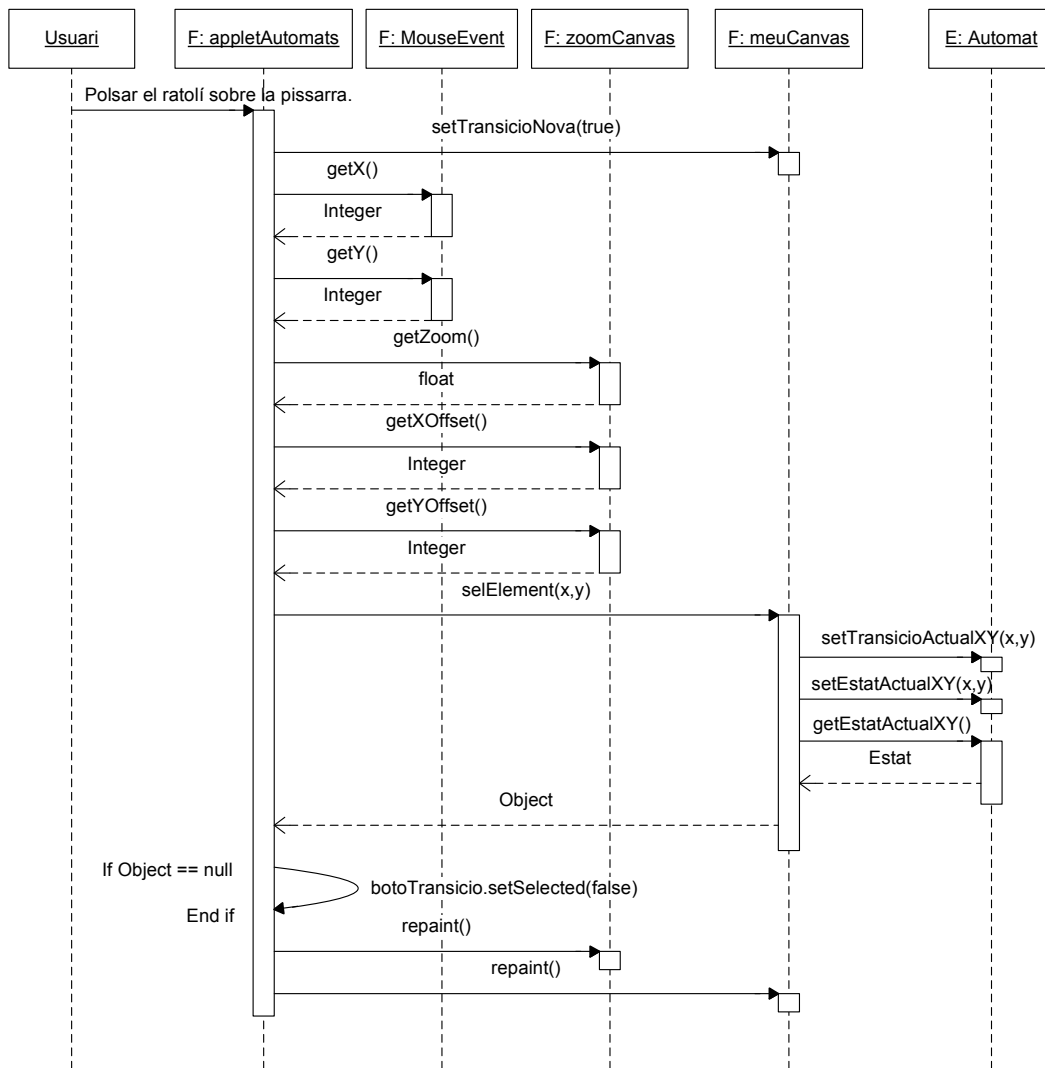


Fig. 5-17 Diagrama de seqüència per afegir una transició. Selecció de l'estat origen.

El següent diagrama mostra la seqüència de missatges que es desencadena quan l'usuari polsa sobre l'estat destí de la transició i les classes que s'hi veuen implicades. Busquem el punt de la pissarra on ha polsat l'usuari i cridem al mètode *selElement(...)*. Aquest mètode mira si a on ha polsat l'usuari hi ha un estat, si n'hi ha un afegeix la transició, i si no n'hi ha cap avorta el procés. Abans d'afegir la transició entre els dos estats mira si ja n'hi ha alguna, en cas que ja existeixi no l'afegeix i retorna la ja existent. Primer es crea una instància de la classe *Transicio*, en funció de si l'origen i el destí són en el mateix estat o estats diferents es crea una instància de la classe *Transicio2DR* o de la classe *Transicio2DD*. S'obtenen les coordenades dels estats d'origen i fi per tal d'establir els punts d'origen i fi de la transició. Finalment s'afegeix la transició a l'autòmat.

Per últim caldrà cridar al mètode *repaint()* per refrescar la pissarra i la zona de zoom.

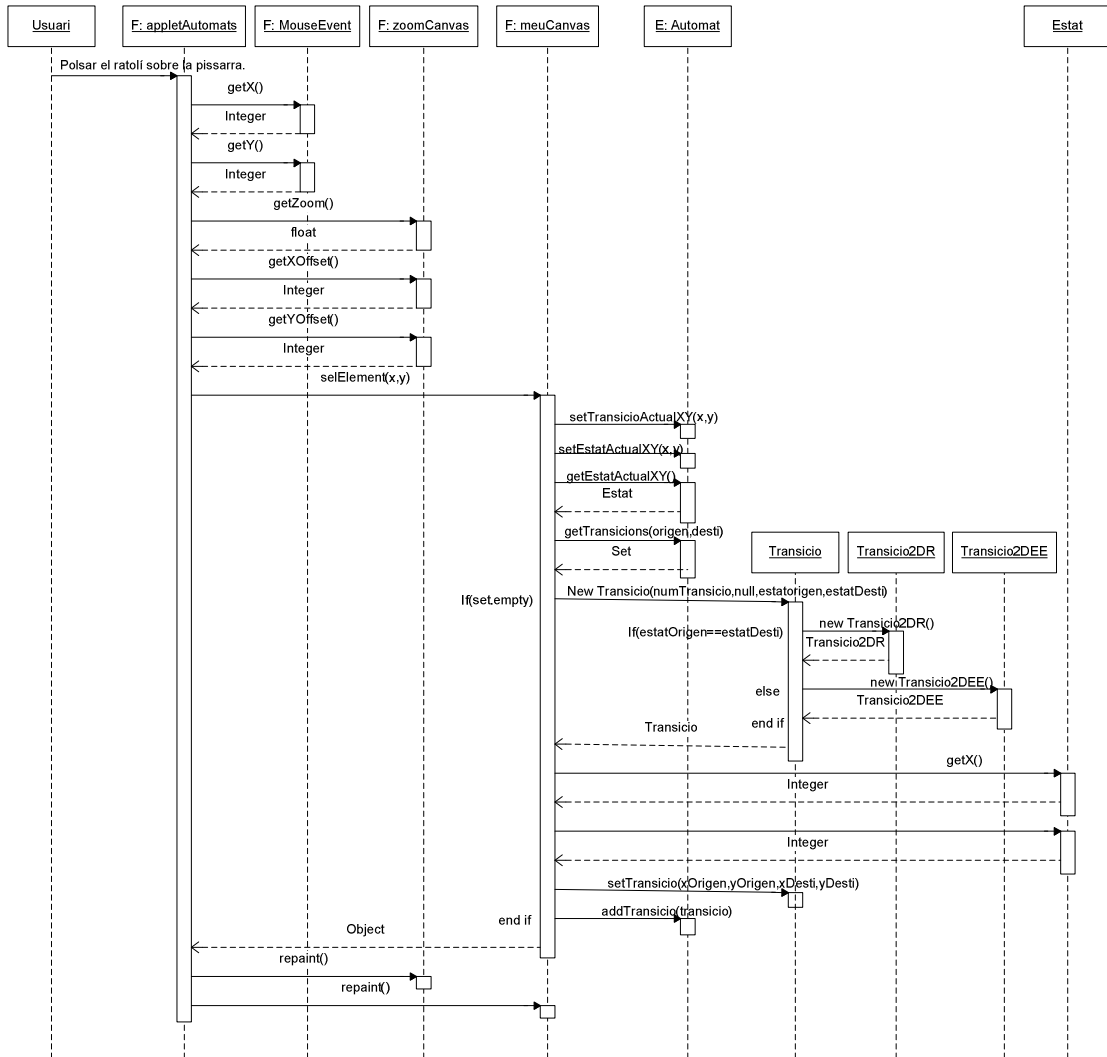


Fig. 5-18 Diagrama de seqüència per afegir una transició. Selecció de l'estat destí i inserció de la transició.

5.5.4 Seleccionar un objecte de la pissarra

Un objecte de la pissarra és qualsevol estat o transició que estigui dibuixada a la pissarra en un moment determinat.

L'usuari podrà seleccionar un element de la pissarra polsant amb el ratolí sobre d'aquest. El següent diagrama mostra les classes que intervenen i els missatges que s'intercanvien al dur l'acció de seleccionar un element de la pissarra.

Busquem el punt de la pissarra on ha polsat l'usuari i cridem al mètode *selElement(...)*. Aquest mètode ens retornarà l'objecte que hi ha sobre el punt on a polsat l'usuari, o nul si no n'hi ha cap. Si l'objecte és un estat haurem de mostrar per pantalla les dades de l'estat, ho farem amb la classe *JpanelOpEstat*. Si l'objecte seleccionat és una transició haurem de mostrar per pantalla les dades de la transició, això ho farem amb la classe *JpanelOpTran*. En cas que l'autòmat sigui de pila mostrarem els símbols acceptats per la transició seleccionada a través de l'objecte *meuListModel*, que no és res més que una llista.

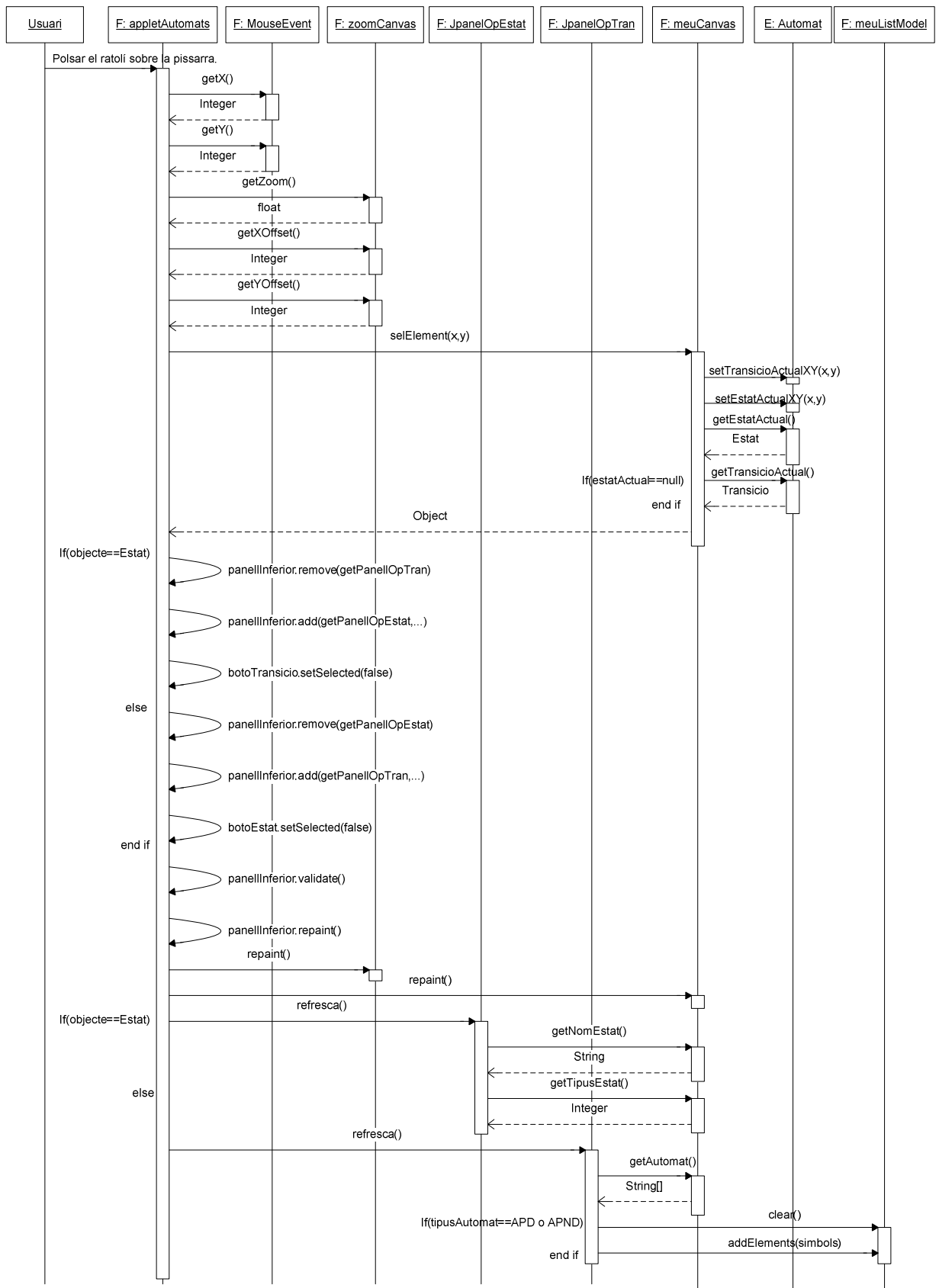


Fig. 5-19 Diagrama de seqüència per seleccionar un objecte de la pissarra.

5.5.5 Moure un objecte de la pissarra

Només es poden moure estats, les transicions aniran allà on vagin els estats. Per moure un estat cal polsar sobre d'ell amb el ratolí i arrossegar-lo fins a la posició desitjada.

Aquesta acció la dividirem en tres parts: polsar sobre l'estat, arrossegar l'estat i deixar el botó del ratolí.

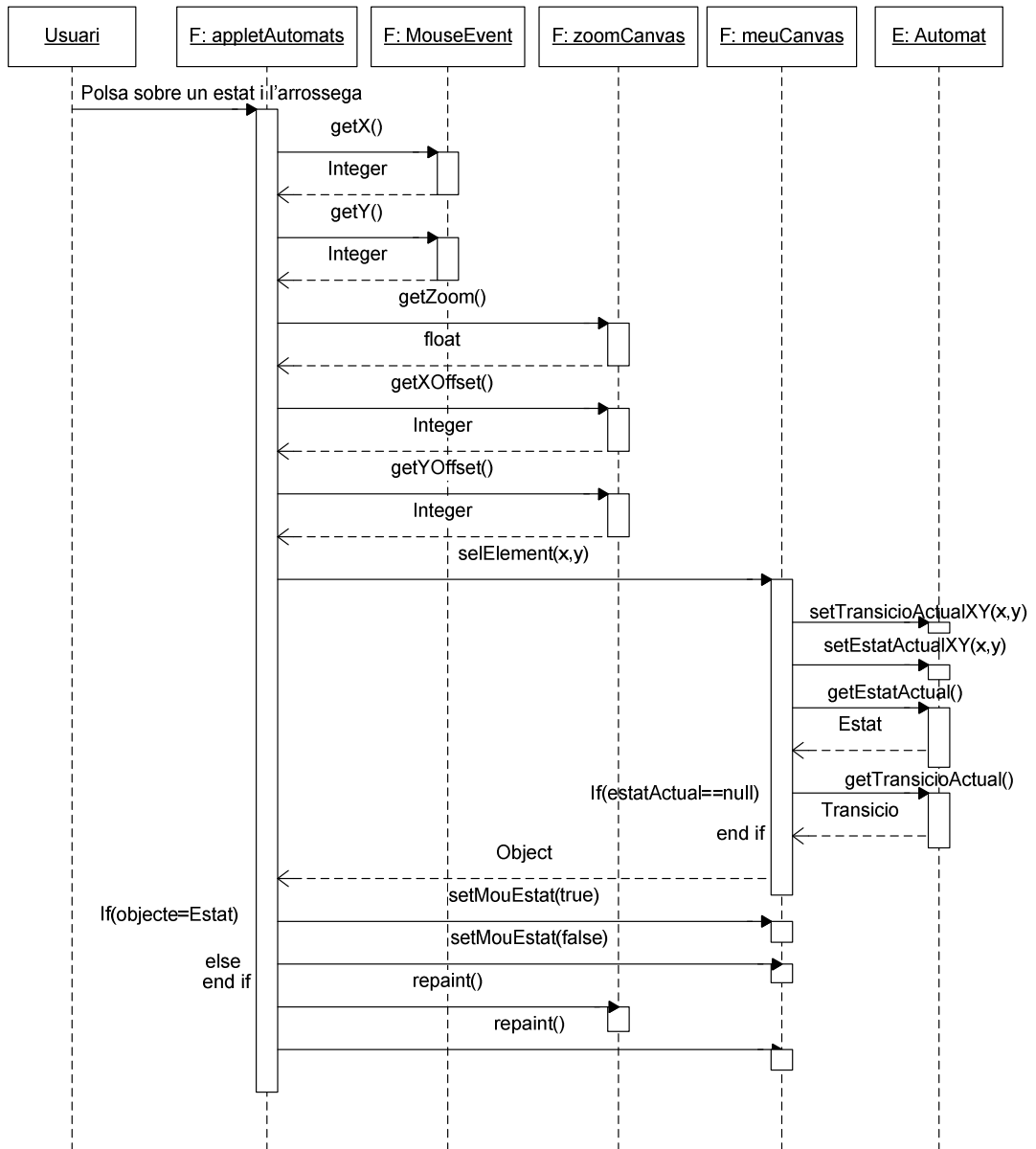


Fig. 5-20 Diagrama de seqüència per moure un objecte de la pissarra. Polsar sobre l'objecte.

L'anterior diagrama mostra els missatges i classes que intervenen quan l'usuari polsa sobre un estat amb la intenció de moure'l de lloc. Podem

observar que és molt semblant a altres diagrames ja vistos, sobretot pel que fa a l'hora de buscar l'estat que hi ha en el punt on ha pulsat l'usuari. Amb el mètode *setMouEstat()* indiquem a la pissarra que anem a moure un estat.

En el següent diagrama es mostren els missatges i classes que intervenen mentre l'usuari arrastra un estat per la pissarra. Anem obtenim les coordenades per on passa el ratolí i actualitzem les coordenades de l'estat amb el mètode *moureEstat(...)*. Al moure l'estat també cal actualitzar les coordenades de les transicions que surten de l'estat i les coordenades de les transicions que entren a l'estat, això ho fem amb el mètode *setTransicio(xo,yo, xd, yd)* de la classe *Transicio*. Per últim cal anar refrescant la pissarra i la zona de zoom amb el mètode *repaint()*.

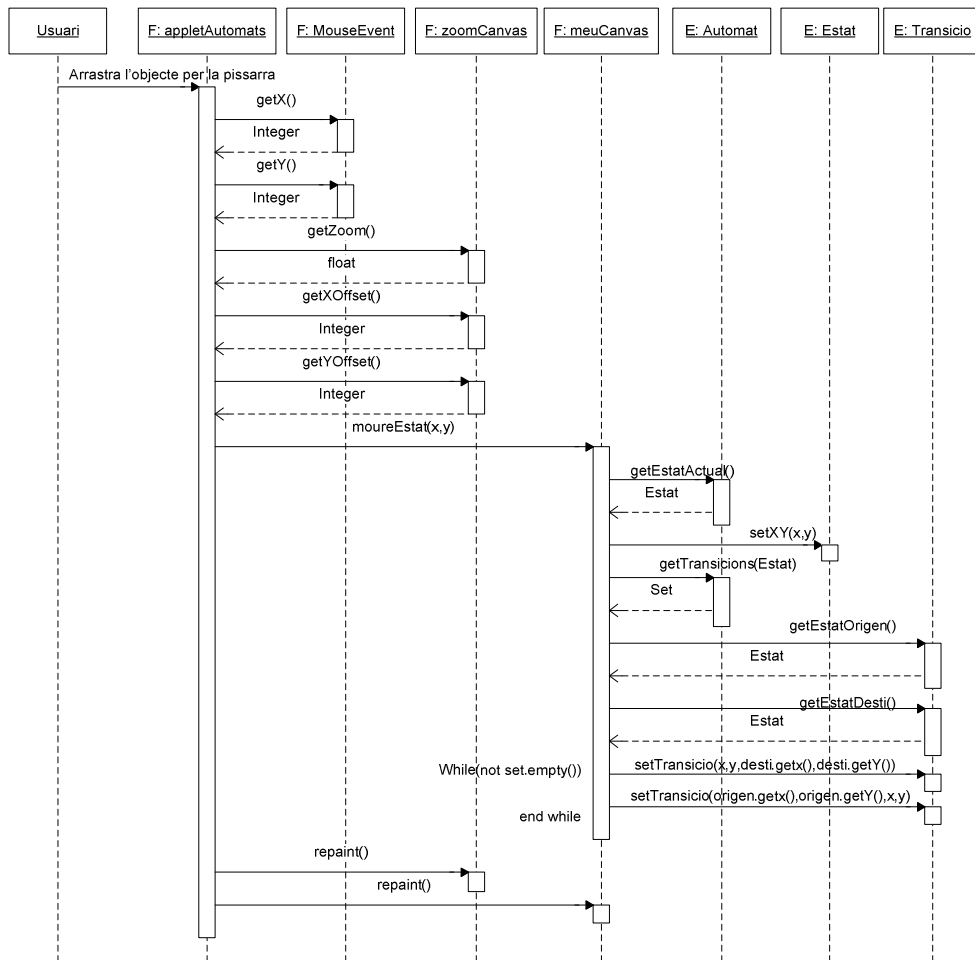


Fig. 5-21 Diagrama de seqüència per moure un objecte de la pissarra. Arrastrar l'objecte.

Per acabar de moure l'estat ens queda deixar anar el botó del ratolí. El següent diagrama se seqüència mostra aquesta acció.

És un diagrama senzill, només hem d'indicar a la pissarra que hem acabat de moure l'estat. Això ho fem cridant al mètode *setMouEstat(false)*.

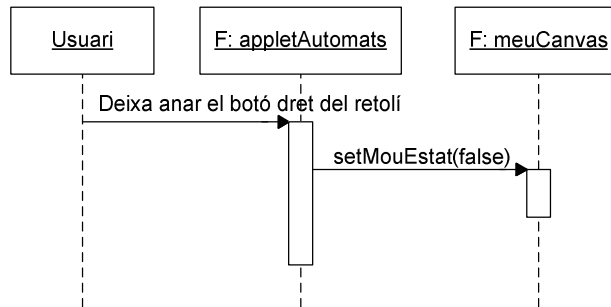


Fig. 5-22 Diagrama de seqüència per moure un objecte de la pissarra. Deixar el botó dret del ratolí.

5.5.6 Eliminar objecte seleccionat

Per eliminar un objecte de la pissarra s'ha d'haver seleccionat prèviament. Dibuixarem dos diagrammes de seqüència, un per eliminar l'estat seleccionat i l'altre per eliminar la transició seleccionada.

En els següent diagrama podem veure les classes i els missatges que s'intercanvien per eliminar un estat. Tot comença quan l'usuari, tenint seleccionat un estat, polsa sobre el botó *Eliminar* o la tecla *Supr* del teclat. Es crida al mètode *removeEstat()* de la classe *meuCanvas*, aquest a la vegada crida al mètode *removeEstat()* de la classe *Automat*. Aquest últim obté l'identificador de l'estat a eliminar, elimina totes les transicions que surten o entren de l'estat i finalment elimina l'estat. Per últim cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

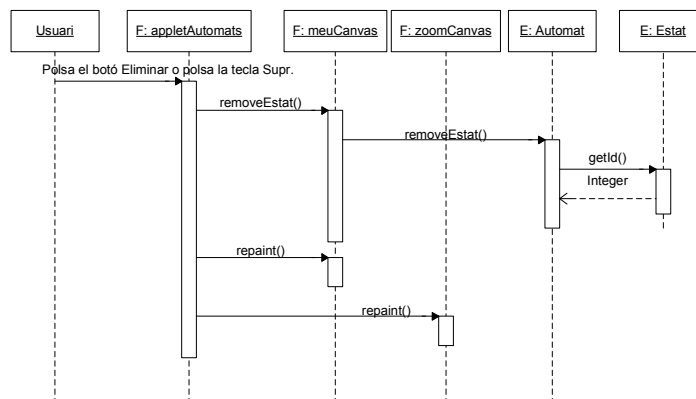


Fig. 5-23 Diagrama de seqüència per eliminar un objecte de la pissarra. Eliminar un estat.

En els següent diagrama podem veure les classes i els missatges que s'intercanvien per eliminar una transició. Els passos són molt semblants als

d'eliminar un estat. Tot comença quan l'usuari, tenint seleccionada una transició, polsa sobre el botó *Eliminar* o la tecla *Supr* del teclat. Es crida al mètode *removeTransicio()* de la classe *meuCanvas*, aquest a la vegada crida al mètode *removeTransicio()* de la classe *Automat*. Aquest últim obté l'identificador de la transició a eliminar i l'elimina. Per últim cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

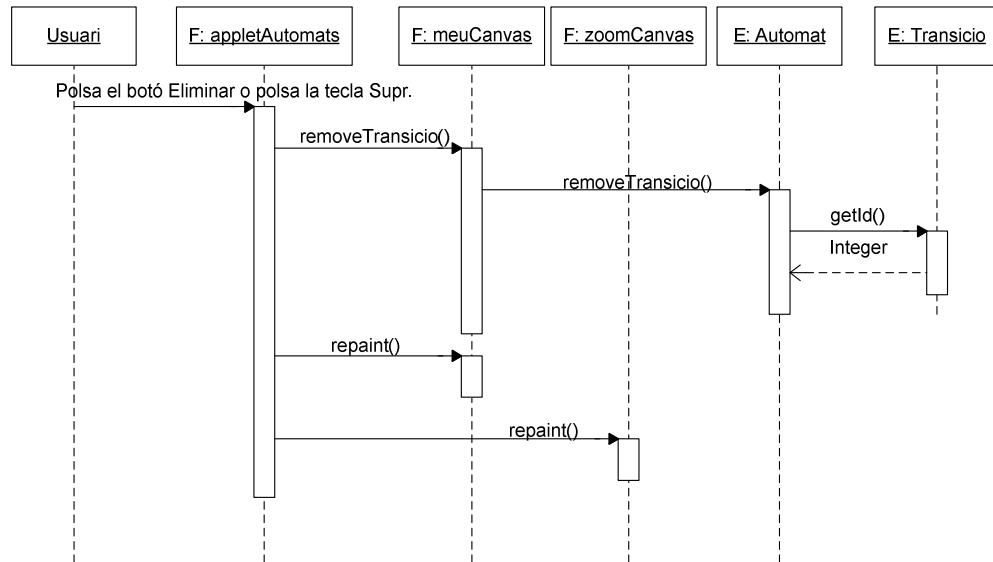


Fig. 5-24 Diagrama de seqüència per eliminar un objecte de la pissarra. Eliminar una transició.

5.5.7 Modificar objecte seleccionat

En aquest punt veurem el diagrama de seqüència que fa referència a l'acció de modificar l'objecte seleccionat. Partim del fet que tenim un estat o una transició seleccionats. Dibuixarem varis diagrames en funció de l'objecte seleccionat sigui un estat o una transició.

En els següent diagrama podem veure les classes i els missatges que s'intercanvien per modificar el nom d'un estat. Un cop l'usuari ha canviat el nom i ha polsat sobre el botó *OK* o la tecla *Return*, es crida al mètode *setNomEstat(nouNom)* de la classe *meuCanvas*, aquest a la vegada també crida al mètode *setNomEstat(nouNom)* de la classe *Automat*. Aquest últim crida al mètode *setNom(nouNom)* de la classe *Estat* que serà qui acabarà modificant el nom de l'estat. Per últim cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

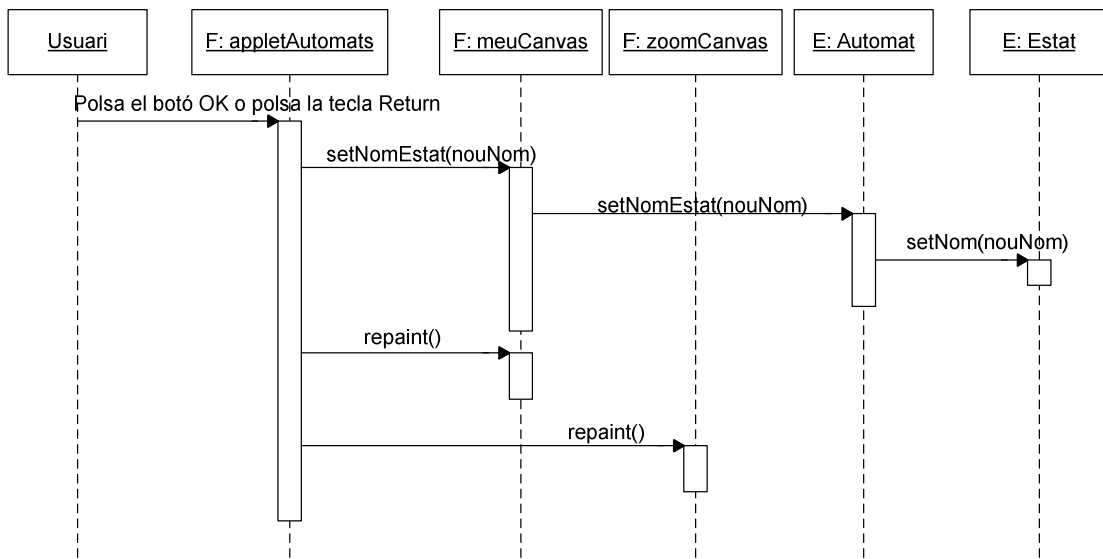


Fig. 5-25 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar el nom d'un estat.

En els següent diagrama podem veure les classes i els missatges que s'intercanvien per modificar el tipus d'un estat. Quan l'usuari selecciona un tipus de la llista *Tipus Estat* es crida al mètode *setTipusEstat(nouTipus)* de la classe *meuCanvas*, aquesta a la vegada també crida al mètode *setTipusEstat(nouTipus)* de la classe *Automat*. Aquest últim crida al mètode *setTipus(nouTipus)* de la classe *Estat* que serà qui acabarà modificant el tipus de l'estat. Per últim cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

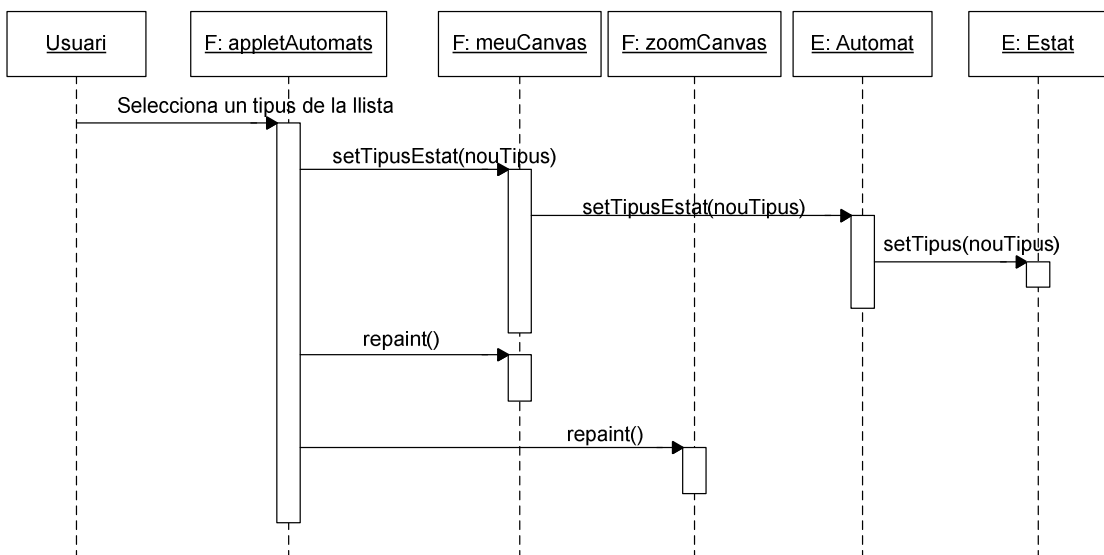


Fig. 5-26 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar el tipus d'un estat.

En el pròxim diagrama podem veure les classes i els missatges que s'intercanvien per modificar els símbols reconeguts per una transició d'un autòmat finit. Quan l'usuari polsa sobre el botó *OK* o la tecla *Return* del teclat es crida al mètode *validaTran(simbols)* de la classe *meuCanvas*, aquest a la vegada crida al mètode *validaTran(simbols)* de la classe *Automat*. Amb aquest mètode validem que la llista de símbols entrada per l'usuari sigui correcta, es a dir, que tots els símbols pertanyin a l'alfabet de l'autòmat i que estiguin separats per comes en cas que n'hi hagi més d'un. En cas que l'anterior mètode no retorni cap error, cridem al mètode *setSimbolsTran(simbols)* de la classe *meuCanvas*, aquest crida a la vegada al mètode *setSimbolsTran(simbols)* de la classe *Automat*. Aquest últim acaba cridant al mètode *setSimbol(simbols)* de la classe *Transicio* que serà qui acabarà modificant els símbols reconeguts per la transició. Per últim cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

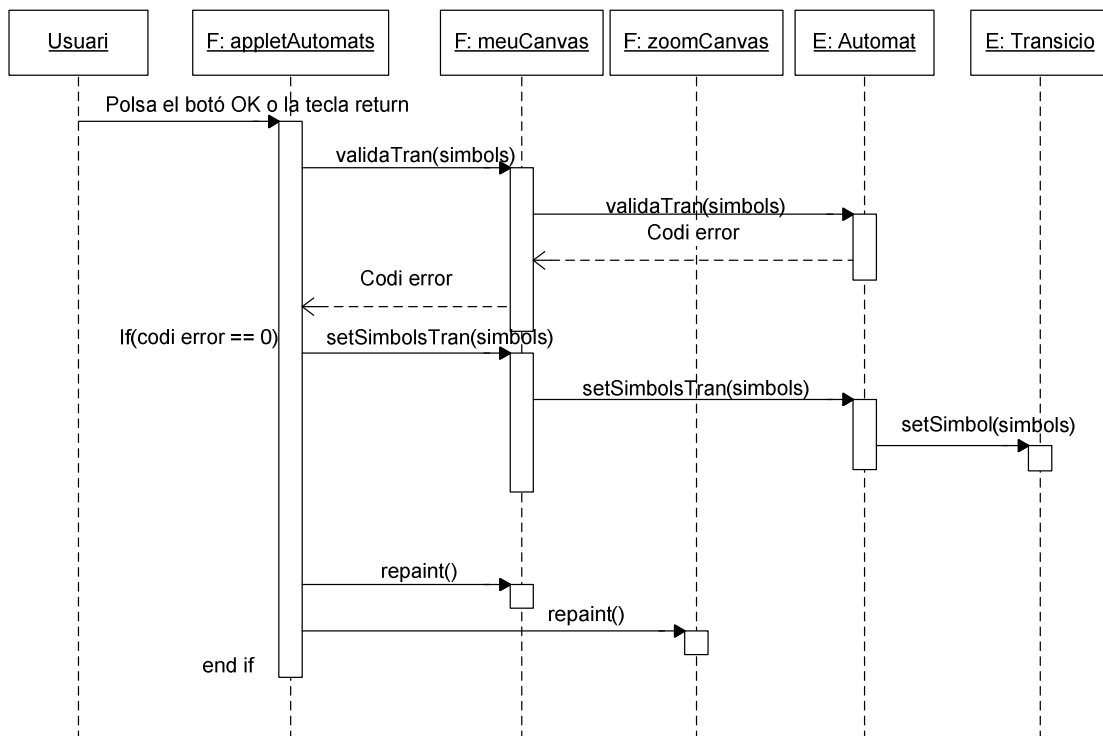


Fig. 5-27 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar els símbols d'una transició.

En els pròxims diagrames podem veure les classes i els missatges que s'intercanvien per modificar els símbols reconeguts per una transició d'un autòmat de pila. Ho hem dividit en tres diagrames: alta d'un símbol, baixa d'un símbol i modificació d'un símbol.

En el pròxim diagrama podem veure les classes i els missatges que s'intervenien a l'hora d'afegir símbols a una transició d'un autòmat de pila.

Quan l'usuari polsa sobre el botó *OK* o la tecla *Return* del teclat es crida al mètode *validaTran(simbols)* de la classe *meuCanvas*, aquest a la vegada crida al mètode *validaTran(simbols)* de la classe *Automat*. Amb aquest mètode validem que la llista de símbols entrada per l'usuari sigui correcta, es a dir, que els símbols pertanyin a l'alfabet de l'autòmat o a l'alfabet de la pila, i que tinguin una sintaxi correcta. En cas que l'anterior mètode no retorni cap error, mirem que el nou símbol no existeixi, i si no existeix l'afegim a la llista cridant al mètode *addElement(nouSimbol)* de la classe *meuListModel*. Un cop actualitzada la llista actualitzem els símbols reconeguts per la transició cridant al mètode *setSimbolsTran(simbols)* de la classe *meuCanvas*, aquest crida a la vegada al mètode *setSimbolsTran(simbols)* de la classe *Automat*. Aquest últim acaba cridant al mètode *setSimbol(simbols)* de la classe *Transicio* que serà qui acabarà modificant els símbols reconeguts per la transició. Desmarquem l'element seleccionat de la llista en cas que n'hi hagués algun cridant al mètode *clearSelection()*. Per últim, cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

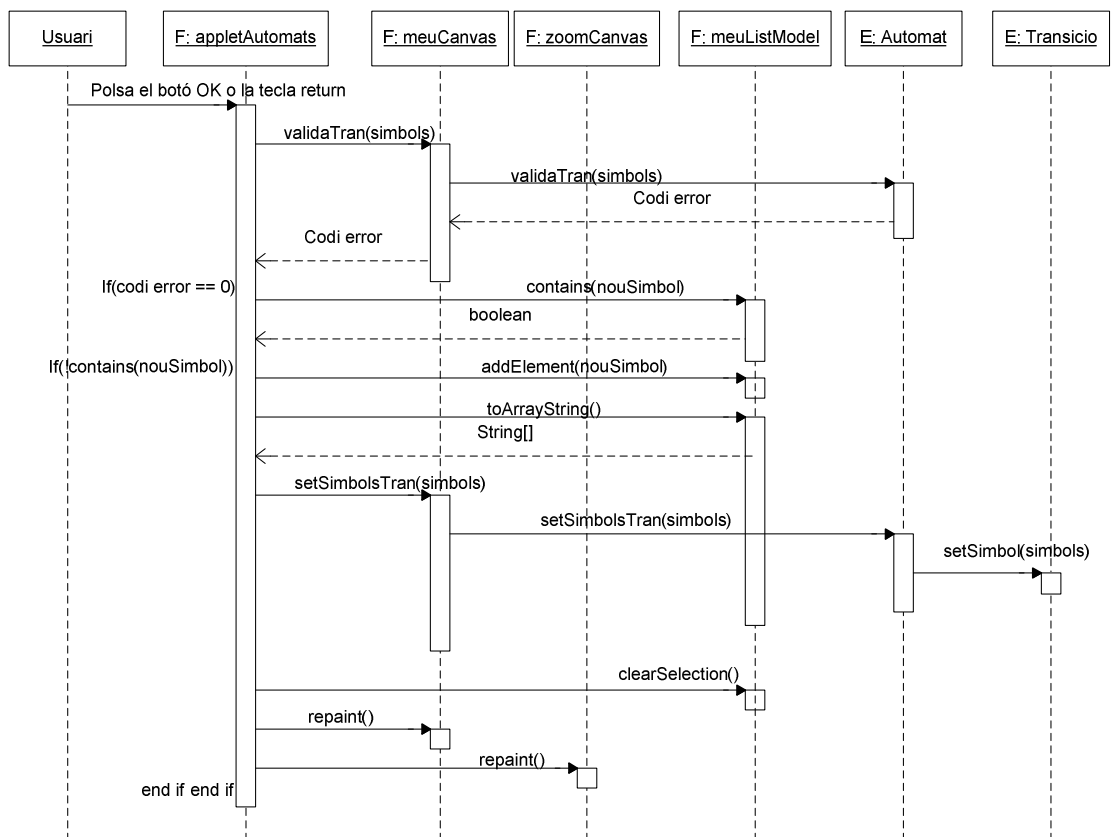


Fig. 5-28 Diagrama de seqüència per modificar un objecte de la pissarra. Afegir un símbol a una transició.

En el pròxim diagrama podem veure les classes i els missatges que s'intervenien a l'hora d'eliminar símbols d'una transició d'un autòmat de

pila. Quan l'usuari polsa sobre el botó *OK* o la tecla *Return* del teclat deixant la casella de text *Símbols* en blanc es procedeix a eliminar el símbol que es trobi seleccionat a la llista *Llista Transicions*. Comencem per obtenir l'índex de l'element seleccionat de la llista amb el mètode *getSelectedIndex()*, un cop el tenim l'eliminem de la llista amb el mètode *remove(index)* de la classe *meuListModel*. Un cop actualitzada la llista actualitzem els símbols reconeguts per la transició cridant al mètode *setSimbolsTran(simbols)* de la classe *meuCanvas*, aquest crida a la vegada al mètode *setSimbolsTran(simbols)* de la classe *Automat*. Aquest últim acaba cridant al mètode *setSimbol(simbols)* de la classe *Transicio* que serà qui acabarà modificant els símbols reconeguts per la transició. Desmarquem l'element seleccionat de la llista en cas que n'hi hagués algun cridant al mètode *clearSelection()*. Per últim, cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

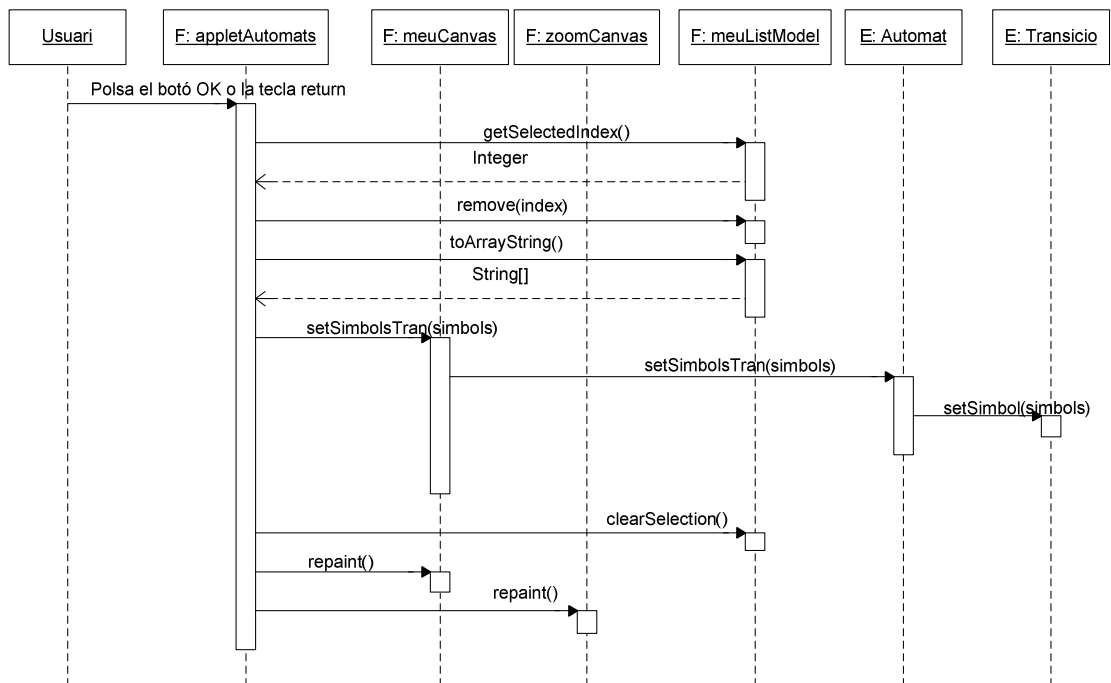


Fig. 5-29 Diagrama de seqüència per modificar un objecte de la pissarra. Eliminar un símbol d'una transició.

Per últim ens queda veure les classes i els missatges que intervenen a l'hora de modificar símbols d'una transició d'un autòmat de pila, això ho podem veure en el següent diagrama de seqüència. Quan l'usuari polsa sobre el botó *OK* o la tecla *Return* del teclat havent introduït un símbol a la casella de text *Símbols* es procedeix a modificar el símbol que es trobi seleccionat a la llista *Llista Transicions*. Comencem per cridar al mètode *validaTran(simbols)* de la classe *meuCanvas*, aquest a la vegada crida al mètode *validaTran(simbols)* de la classe *Automat*. Amb aquest mètode validem que la llista de símbols entrada per l'usuari sigui correcta, es a dir,

que els símbols pertanyin a l'alfabet de l'autòmat o a l'alfabet de la pila, i que tinguin una sintaxi correcta. En cas que l'anterior mètode no retorni cap error, actualitzem l'element seleccionat de la llista cridant al mètode *setElementAt(simbol,index)*. Un cop actualitzada la llista actualitzem els símbols reconeguts per la transició cridant al mètode *setSimbolsTran(simbols)* de la classe *meuCanvas*, aquest crida a la vegada al mètode *setSimbolsTran(simbols)* de la classe *Automat*. Aquest últim acaba cridant al mètode *setSimbol(simbols)* de la classe *Transicio* que serà qui acabarà modificant els símbols reconeguts per la transició. Desmarquem l'element seleccionat de la llista cridant al mètode *clearSelection()*. Per últim cal refrescar la pissarra i la zona de zoom per això es crida al mètode *repaint()*.

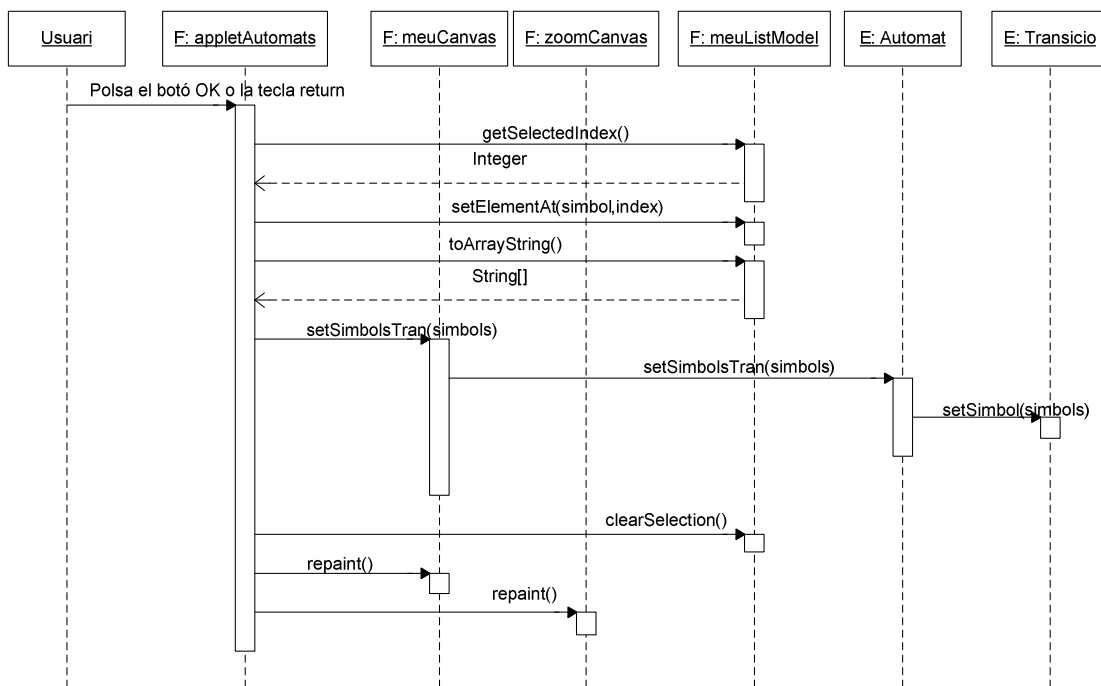


Fig. 5-30 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar un símbol d'una transició.

5.5.8 Augmentar el zoom

En aquest punt veurem el diagrama de seqüència que fa referència a l'acció d'augmentar el zoom de la pissarra. Quan l'usuari polsa sobre el botó + es crida al mètode *posarZoom()* de la classe *zoomCanvas*, amb aquest mètode augmentarem el zoom en un nivell. Cridarem al mètode *setZoom(xZoom, yZoom)* de la classe *meuCanvas* per tal d'actualitzar la pissarra al nou nivell de zoom, prèviament haurem cridat al mètode *getZoom()* de la classe *zoomCanvas* per tal de conèixer el nou nivell de zoom. També caldrà actualitzar el desplaçament del contingut de la pissarra respecte els eixos x i y, ho farem cridant al mètode *setXYOffset(x,y)*. Prèviament haurem cridat als mètodes *getXOffset()* i *getYOffset()* per tal de conèixer el desplaçament.

Per últim ens caldrà refrescar la pissarra i la zona de zoom per això cridarem al mètode *repaint()*.

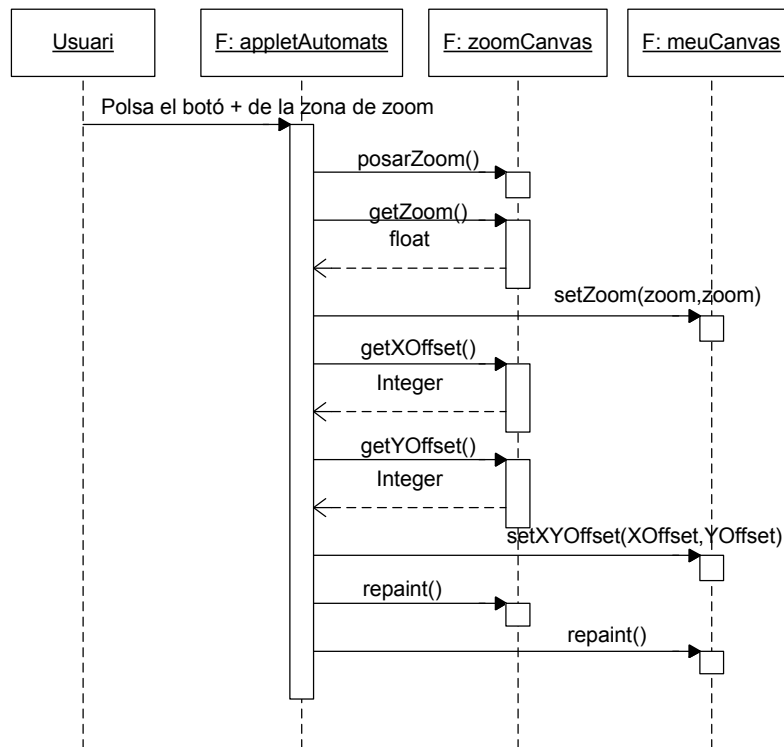


Fig. 5-31 Diagrama de seqüència per augmentar el zoom de la pissarra.

5.5.9 Disminuir el zoom

En aquest punt veurem el diagrama de seqüència que fa referència a l'acció de disminuir el zoom de la pissarra. Quan l'usuari polsa sobre el botó - es crida al mètode *treureZoom()* de la classe *zoomCanvas*, amb aquest mètode disminuïm el zoom en un nivell. La resta de missatges que s'intercanvien les classes són els mateixos que per l'acció de posar zoom.

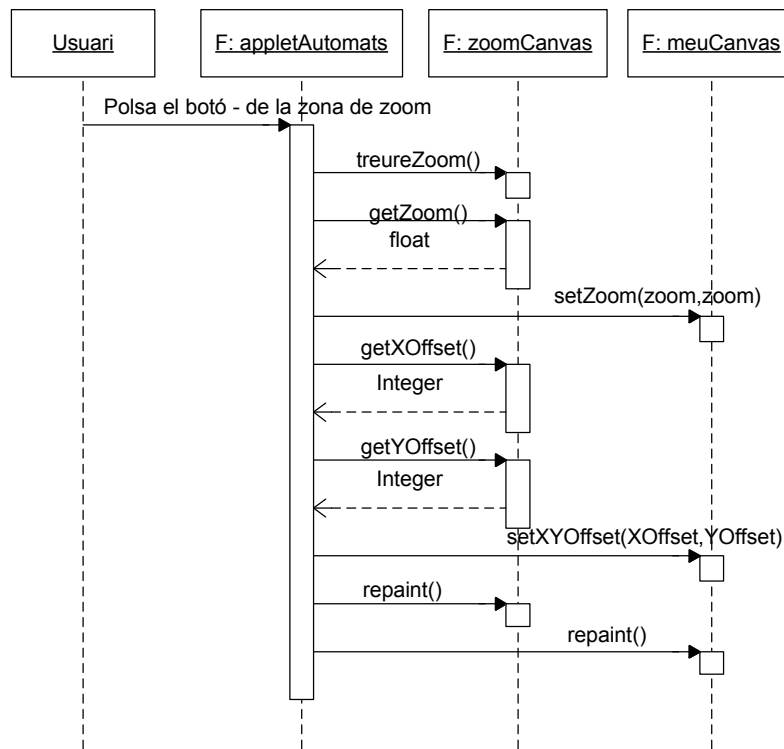


Fig. 5-32 Diagrama de seqüència per disminuir el zoom de la pissarra.

5.5.10 Modificar la vista de la pissarra

En aquest punt veurem els diagrames de seqüència que fan referència a l'acció de modificar la vista de la pissarra. Aquesta acció l'usuari la pot dur a terme de dues maneres diferents. Polsant amb el ratolí sobre qualsevol punt de la zona de zoom, el requadre vermell es mourà allà on s'ha polsat i es modificarà la vista de la pissarra. O bé polsar dins del requadre vermell i arrastrar-lo fins on es vulgui.

En aquest primer diagrama podem observar la seqüència de missatges que es desencadena i les classes que hi intervenen quan l'usuari polsa amb el ratolí sobre de la zona de zoom. Primer de tot hem d'obtenir les coordenades del punt sobre el que s'ha polsat, ho fem cridant als mètodes *getX()* i *getY()* de la classe *MouseEvent*. Un cop tenim les coordenades caldrà actualitzar les coordenades del requadre vermell, ho fem cridant al mètode *setXY()* de la classe *zoomCanvas*. Ara caldrà reflectir el desplaçament del requadre vermell a la pissarra, per fer-ho cridarem al mètode *setXYOffset(x,y)*. Prèviament haurem cridat als mètodes *getXOffset()* i *getYOffset()* per tal de conèixer el desplaçament. Per últim ens caldrà refrescar la pissarra i la zona de zoom per això cridarem al mètode *repaint()*.

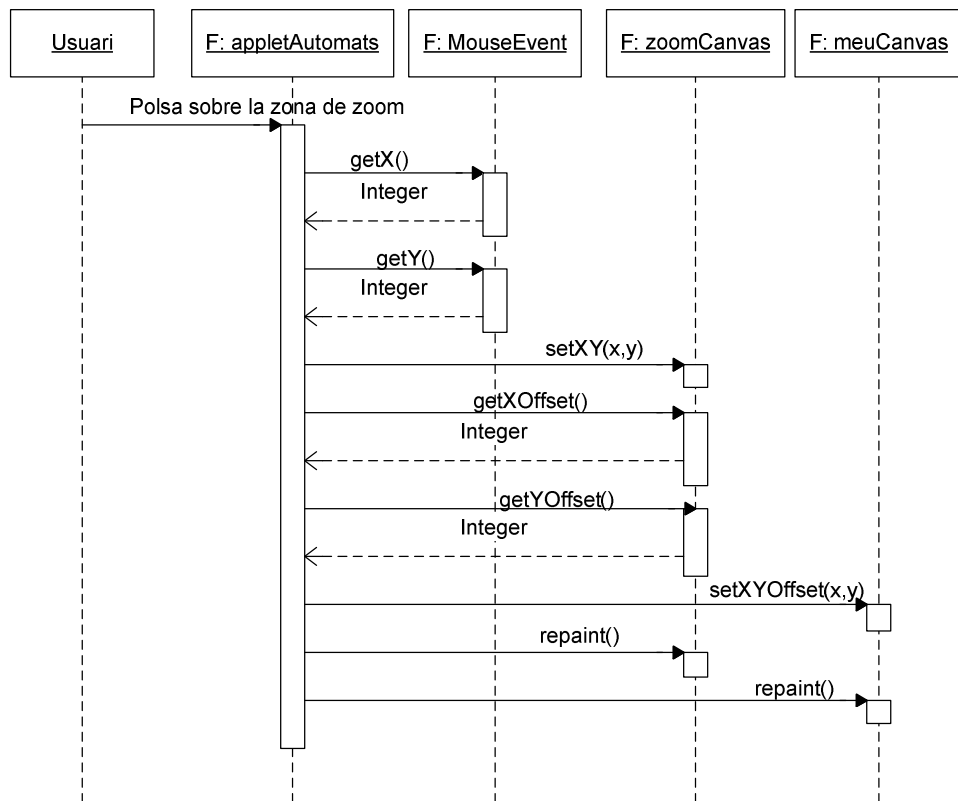


Fig. 5-33 Diagrama de seqüència per modificar la vista de la pissarra.

En els dos següents diagrames podem observar la seqüència de missatges que es desencadena i les classes que hi intervenen quan l'usuari arrossega el requadre vermell amb el ratolí.

En aquest primer diagrama podem veure les classes i els missatges que s'intercanvien quan l'usuari polsa el botó dret del ratolí. Obtenim les coordenades del punter del ratolí cridant als mètodes *getX()* i *getY()* de la classe *MouseEvent* i les informem a la zona de zoom cridant al mètode *setXYMouse(x,y)* de la classe *zoomCanvas*.

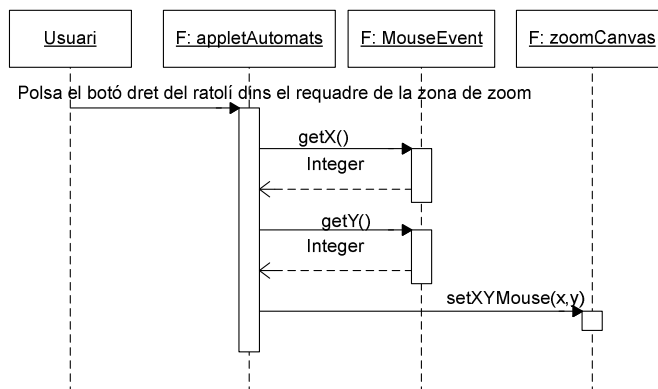


Fig. 5-34 Diagrama de seqüència per modificar la vista de la pissarra. Polsar botó dret del ratolí.

En aquest segon diagrama podem veure les classes i els missatges que s'intercanvien quan l'usuari arrossega el ratolí per sobre de la zona de zoom. Primer obtenim les coordenades per on va passant el punter del ratolí cridant als mètodes *getX()* i *getY()* de la classe *MouseEvent*. Amb aquestes coordenades anem movent el requadre vermell, això ho fem cridant al mètode *arrestarVista(x,y)* de la classe *zoomCanvas*. Aquest mètode acaba cridant als mètodes *setXY(x,y)* i *setXYMouse(x,y)* d'aquesta mateixa classe. Caldrà reflectir el desplaçament del requadre vermell a la pissarra, per fer-ho cridarem al mètode *setXYOffset(x,y)*. Prèviament haurem cridat als mètodes *getXOffset()* i *getYOffset()* per tal de conèixer el desplaçament. Per últim ens caldrà refrescar la pissarra i la zona de zoom per això cridarem al mètode *repaint()*.

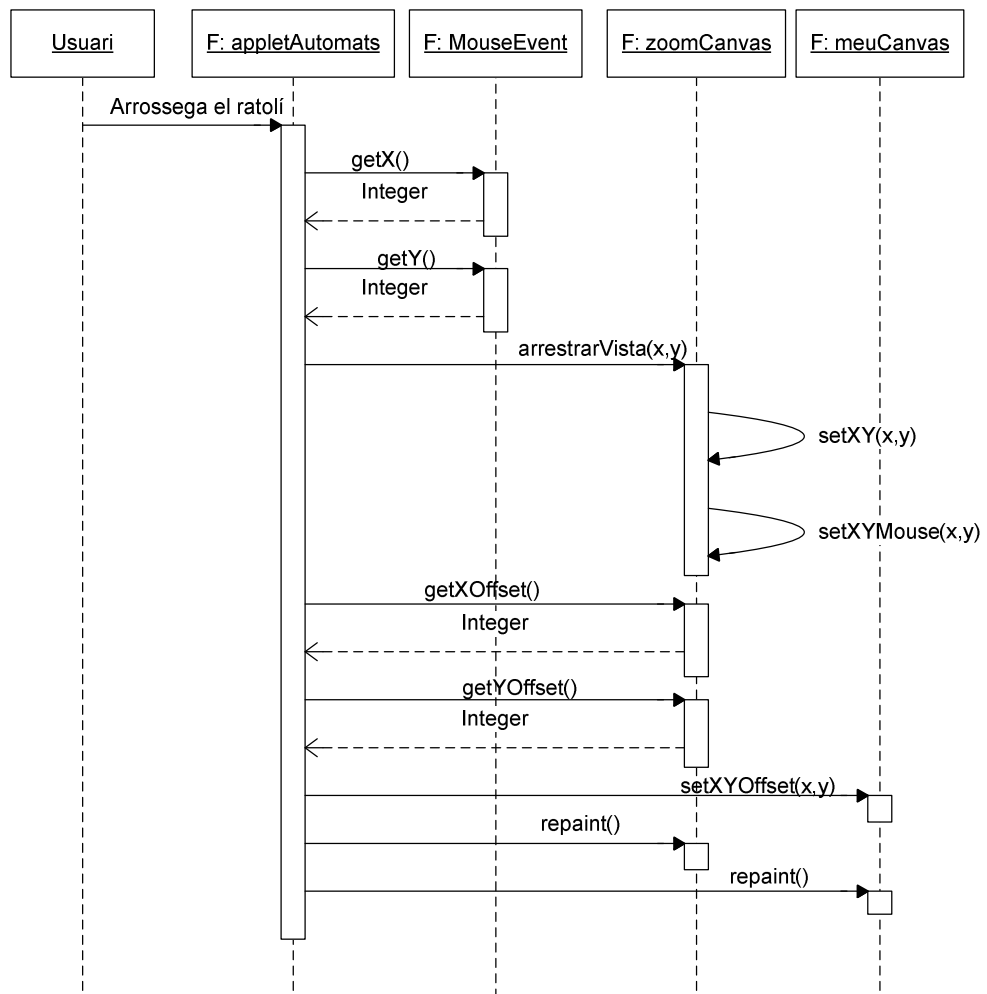


Fig. 5-35 Diagrama de seqüència per modificar la vista de la pissarra. Arrossegar el requadre de la vista.

5.5.11 Obtener la cadena que defineix el contingut de l'editor

En el següent diagrama es mostren les classes que intervenen a l'hora d'obtenir la cadena que defineix el contingut de l'editor i els missatges que s'intercanvien.

Per obtenir la cadena, el sistema, a través d'una funció de *JavaScript*, crida al mètode *toString2()* del nostre applet. La cadena estarà formada per dues parts, una part amb informació per l'editor i l'altra part amb informació pel corrector. El format de la cadena s'explicarà en més detall al pròxim capítol.

Per obtenir la informació necessària per l'editor, l'*applet* crida al mètode *toString()* de la classe *meuCanvas* que a la vegada crida al mètode *toString()* de la classe *Automat*. Aquest últim crida al mètode *toString('G')* de la mateixa classe. El caràcter 'G' indica que volem la informació referent a l'editor. Per cada estat cridem al mètode *toString('G')* obtenint la informació dels estats. Per cada transició també cridem el seu mètode *toString('G')* per obtenir la seva informació. Cridarem als mètodes *getX()* i *getY()* de la classe *Estat* per obtenir les coordenades de l'estat, i als mètodes *getXOrigen()*, *getYOrigen()*, *getXDesti()*, *getYDesti()*, *getXControl()* i *getYControl()* de la classe *Transicio* per obtenir els punts necessaris per poder dibuixar una transició.

Per obtenir la informació necessària pel corrector, l'*applet* crida al mètode *toString('C')* de la classe *meuCanvas* que a la vegada crida al mètode *toString('C')* de la classe *Automat*. El caràcter 'C' indica que volem la informació referent al corrector. Per cada estat cridem al mètode *toString('C')* obtenint la informació dels estats. Per cada transició també cridem el seu mètode *toString('C')* per obtenir la seva informació.

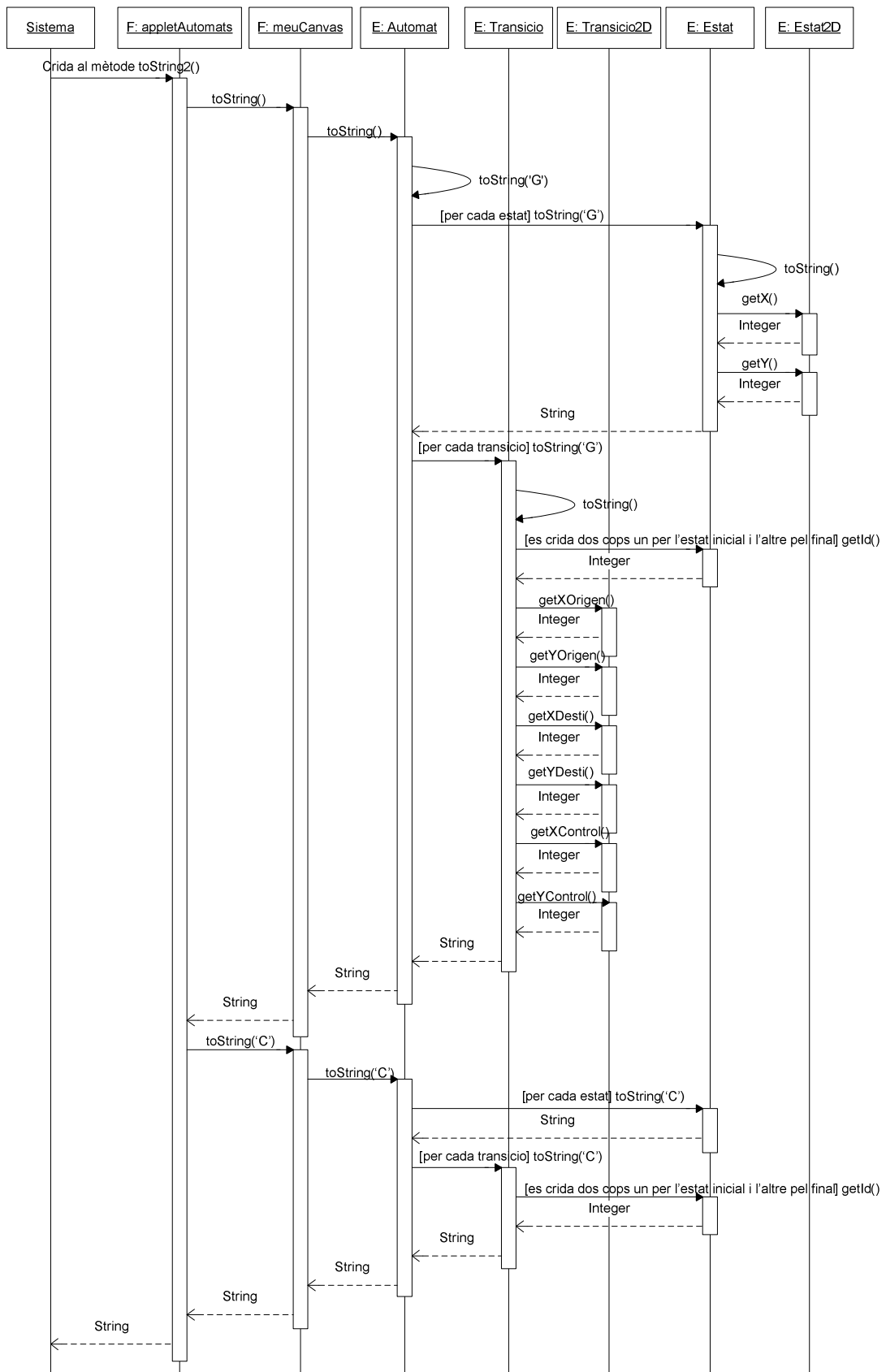


Fig. 5-36 Diagrama de seqüència per obtenir la cadena que defineix el contingut de l'editor.

5.5.12 Restaurar contingut de l'editor

En el següent diagrama es mostren les classes que intervenen a l'hora de restaurar el contingut de l'editor a partir de la cadena de text obtinguda en el punt anterior.

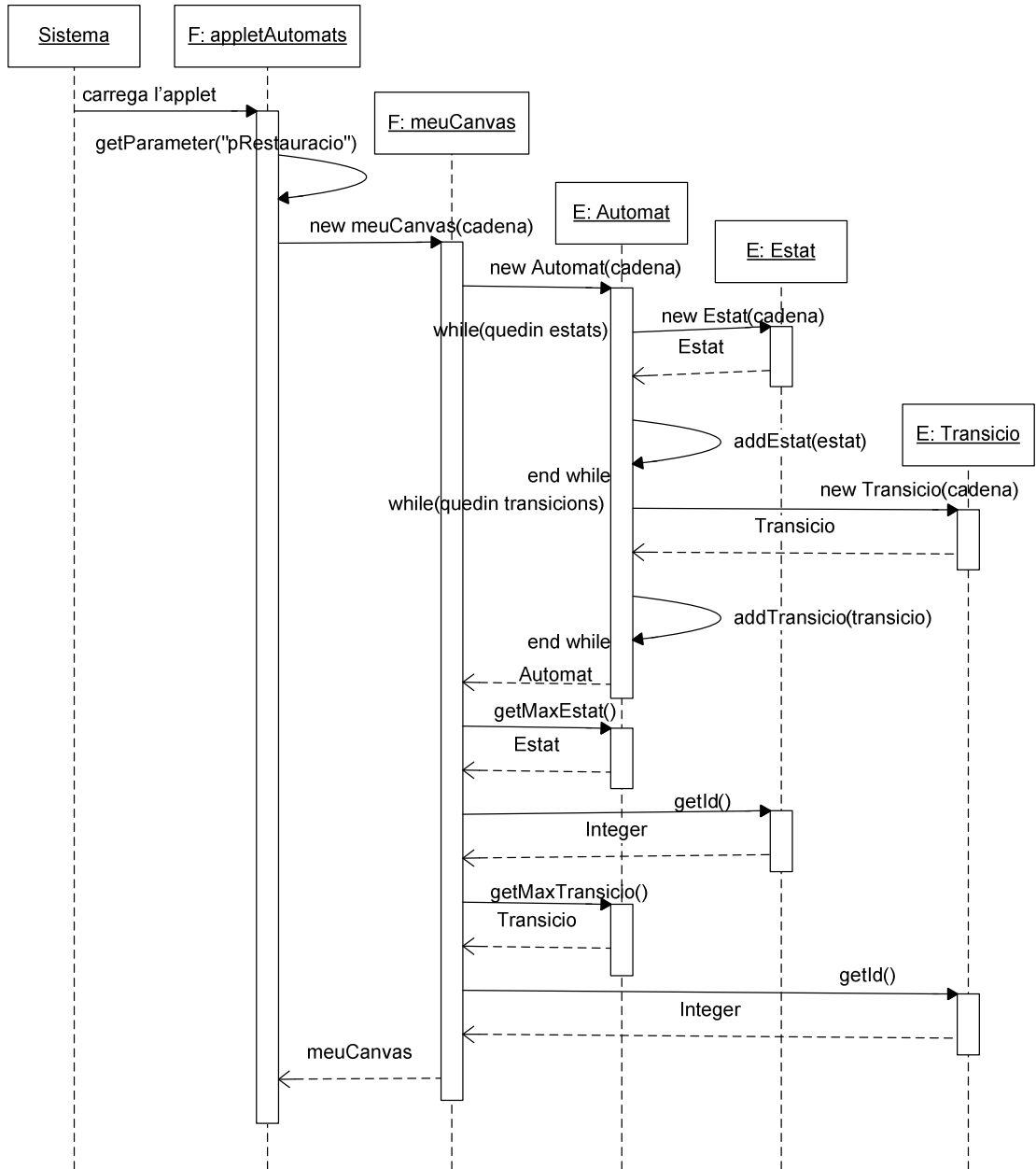


Fig. 5-37 Diagrama de seqüència per restaurar el contingut de l'editor.

El sistema carrega el nostre *applet*, l'*applet* al carregar-se executa el mètode *init()*, i és aquest qui crida al mètode *getParameter(...)*, obtenint el valor del paràmetre *pRestauracio*, inicialitzat prèviament pel sistema. L'*applet* crea la pissarra cridant al constructor amb paràmetre de la classe *meuCanvas*. La classe *meuCanvas* crea un nou autòmat a partir de la

cadena de text usant el constructor amb paràmetre. El constructor *Automat(cadena)* crea els estats i transicions continguts a la cadena cridant als constructors amb paràmetre de les classes *Estat* i *Transicio*. Per últim es criden als mètodes *getMaxEstat()* i *getMaxTransicio()* de la classe *Automat* per inicialitzar alguns paràmetres de la classe *meuCanvas*.

5.5.13 Configurar l'idioma de l'editor

El següent diagrama mostra les classes que intervenen i els missatges que s'intercanvien per inicialitzar el diccionari de l'idioma.

El sistema carrega el nostre *applet*, l'*applet* al carregar-se executa el mètode *init()*, i és aquest qui crida al mètode *getParameter(...)*, obtenint el valor del paràmetre *pIdioma*, inicialitzat prèviament pel sistema. La nomenclatura que segueix la cadena de text *pIdioma* s'explicarà en el següent capítol. L'*applet* crea una instància de la classe *SingletonTraductor* cridant al mètode *getSingleton()*, i després l'inicialitza cridant al mètode *setTraductor(pIdioma)*.

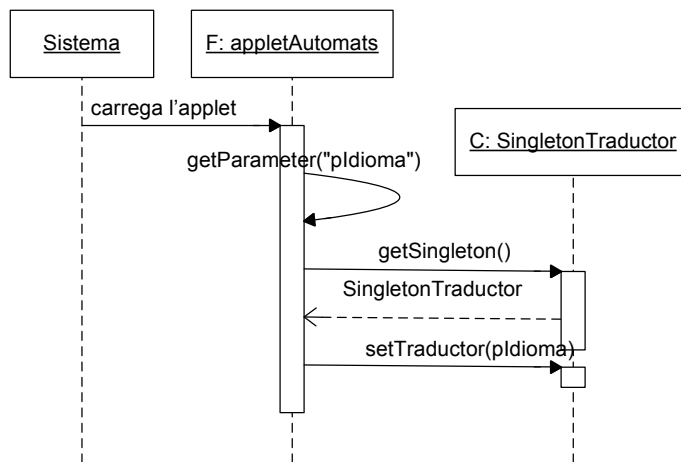


Fig. 5-38 Diagrama de seqüència per inicialitzar el diccionari de l'idioma.

6 Implementació

En aquest capítol es comentaran les característiques més importants de les interfícies i correctors implementats.

6.1 Interfície

Tal i com ja s'ha explicat en altres capítols d'aquesta memòria, l'objectiu principal d'aquest projecte és afegir a la plataforma ACME un corrector de problemes d'autòmats finits i de pila. Per tal de dur a terme això a calgut implementar una interfície que és la que es mostra en la següent figura.

The screenshot shows the ACME PROJECTE web interface. The top navigation bar includes 'LLENGUATGE, GRA...' and 'INTRODUCCIÓ A L...'. The user is logged in as 'NOM ALUMNE'. The left sidebar contains 'ACTIVITATS' (AFD I, AFND I, APD I, APND I) and 'UTILITATS' (Mail al tutor, Canviar password, Canviar dades, Ajuda). A calendar for August 2008 is visible. The main content area displays 'EXERCICI: 2 de l'activitat AFND I' with the problem statement: 'Trobeu un AFND amb epsilon-moviments que reconegui el llenguatge de les paraules formades per zeros i uns que comencen amb dos zeros o acaben amb dos uns.' Below this is a large text input area for the solution, with 'Zoom' and 'Estat' buttons. To the right is a 'Propietats de l'estat' form with fields for 'Nom', 'Tipus estat' (set to 'Estat inicial'), and an 'Eliminar' button. A 'Corregir' button is at the bottom. A table at the bottom shows the exercise status:

Activitat	N. Exercici	Estat	Errors de Resultat	Errors Sintàctics	Data límit
2	2	No Result	0	0	19/10/2009

Fig. 6-1 Interfície d'exercicis d'autòmats.

La interfície de l'anterior figura es pot dividir en tres parts: enunciat del problema (part A de color blau), introducció i enviament de la solució (part B de color lila), i zona de consulta (part C de color vermell).

Deixant de banda aquestes tres parts observem que a la part superior ens apareix informació sobre l'assignatura, el tema, l'exercici i el nom de l'usuari. A la part esquerra de la pantalla tenim un menú amb totes les activitats de l'assignatura, un altre menú d'utilitats i un calendari on s'hi poden posar avisos.

Enunciat del problema

En aquesta part de la interfície s'hi mostra l'enunciat del problema que prèviament s'haurà sortejat. La manera en què es visualitza i es sorteja l'enunciat és igual a la d'altres tipus de problemes ja existents a la plataforma ACME, i per tant no haurem d'implementar res.

Trobeu un AFND amb ϵ -moviments que reconegui el llenguatge de les paraules formades per zeros i uns que comencen amb dos zeros o acaben amb dos uns.

Fig. 6-2 Interfície d'exercicis d'autòmats. Zona de l'enunciat.

Introducció i enviament de la solució

Aquesta part de la interfície proporciona a l'alumne les eines necessàries per poder introduir i enviar la solució del problema proposat. Està formada d'un editor on l'alumne podrà dibuixar la solució i del botó corregir, que polsant sobre ell s'envia la solució que hi hagi a l'editor en aquell moment. En el punt 6.1.1 s'explica en més detall la interfície de l'editor.



Al polsar sobre el botó *Corregir* no només enviem la solució sinó que s'inicia el procés de correcció, segons el tipus d'autòmat s'executarà uns dels quatre processos correctors. Un cop el sistema ha corregit el problema es mostra el resultat a través d'una de les pàgines estàndard de l'ACME.

Si el resultat de la correcció és incorrecte es mostra l'error dins un requadre de color vermell (veure *figura 6-3*). L'alumne podrà polsar sobre el botó continuar i tornarà a la pantalla de la *figura 6-1*. En aquesta pantalla l'alumne visualitzarà l'última solució enviada i tindrà l'opció de modificar-la i tornar-la enviar. El professor també tindrà aquestes opcions, però amb la diferència que si fa alguna modificació aquesta no es quedarà guardada.

Si el resultat és correcte es mostra el mot CORRECTE dins d'un requadre de color blau (veure *figura 6-4*). L'alumne podrà polsar sobre el botó continuar i tornarà a la pantalla principal de l'activitat (veure *figura 6-9*). En aquesta pantalla l'alumne podrà veure els problemes de l'activitat, i saber en quin estat es troben (resolts o no), i el nombre de solucions errònies que ha enviat.

The screenshot shows the ACME PROJECT web interface. On the left is a navigation menu with sections 'ACTIVITATS' (containing AFD I, AFND I, APD I, APND I) and 'UTILITATS' (containing Mail al tutor, Canviar password, Canviar dades, Ajuda). The main content area displays the text 'Estàs a » ACME » Inici » Llenguatge, Gramàtiques i Automàta » AFND I » Exercici: 2' and 'La solució enviada és:'. Below this is a finite automaton diagram with states e0 through e6. State e0 is the start state, and e3 and e6 are final states. Transitions are labeled with 0, 1, and λ . A zoom window is visible in the bottom left of the diagram area. Below the diagram, the text 'El resultat de la correcció és:' is followed by a red error message box containing the text 'ERROR: rebutja la paraula 1011 que hauria d'acceptar'. A green 'Continuar' button is located below the error message.

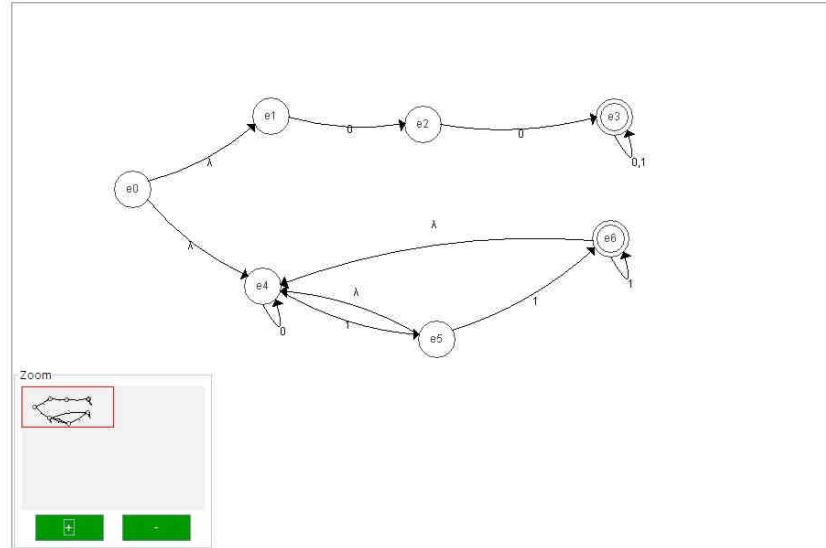
Fig. 6-3 Resultat correcció d'un problema. Solució incorrecta.

This screenshot shows the same ACME PROJECT interface as Fig. 6-3, but with a different result. The text 'La solució enviada és:' is followed by the same finite automaton diagram. Below the diagram, the text 'El resultat de la correcció és:' is followed by a blue message box containing the text 'Correcte'. A green 'Continuar' button is located below the message box.

Fig. 6-4 Resultat correcció d'un problema. Solució correcta.

Tot seguit es mostren quatre exemples més de correcció a on l'alumne ha enviat una solució incorrecta. En la primera imatge l'alumne no ha posat cap estat inicial. A la segona imatge ha introduït menys estats del compte. A la tercera imatge li falten transicions per introduir. A la quarta imatge no ha posat cap estat final.

La solució enviada és:



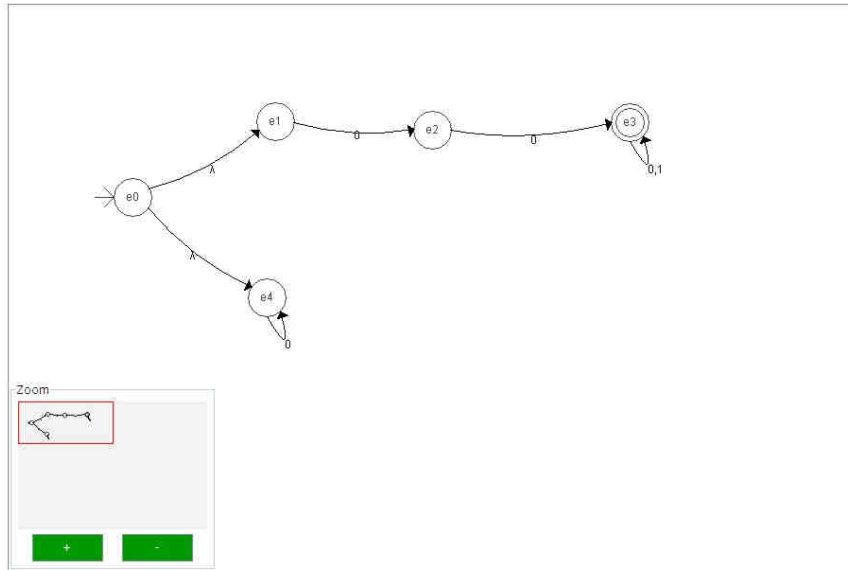
El resultat de la correcció és:

ERROR: l'autòmat de la solució proposada no té estat inicial

Continuar

Fig. 6-5 Resultat correcció d'un problema. Solució incorrecta.

La solució enviada és:



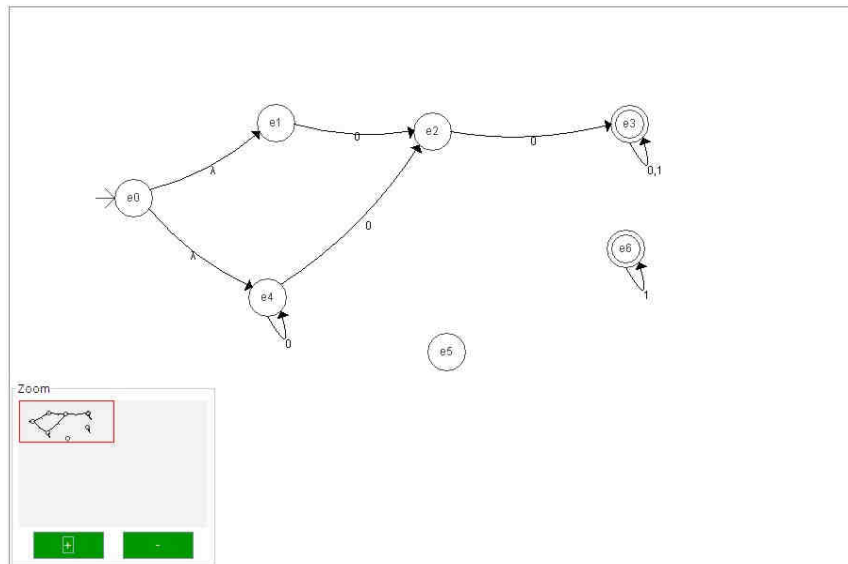
El resultat de la correcció és:

ERROR: no coincideix el nombre d'estats. Professor: 7 --
Alumne: 5

Continuar

Fig. 6-6 Resultat correcció d'un problema. Solució incorrecta.

La solució enviada és:



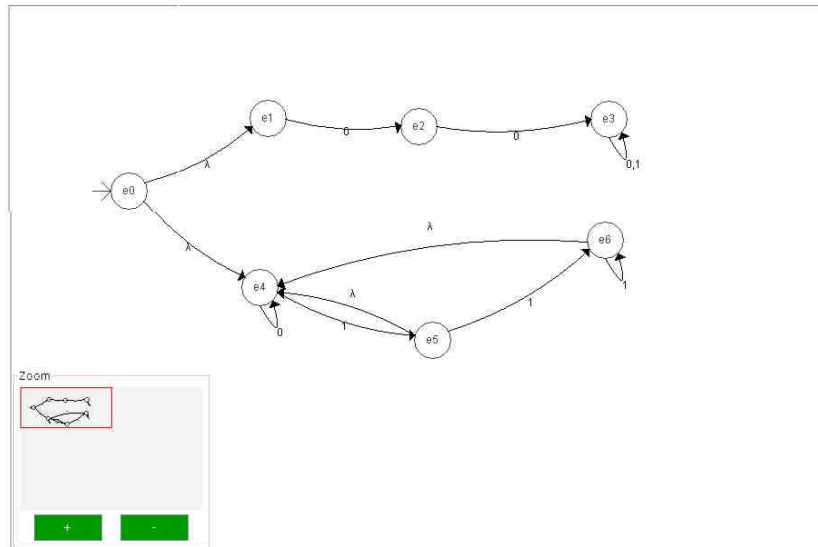
El resultat de la correcció és:

ERROR: rebutja la paraula 011 que hauria d'acceptar

Continuar

Fig. 6-7 Resultat correcció d'un problema. Solució incorrecta.

La solució enviada és:



El resultat de la correcció és:

ERROR: rebutja la paraula 00 que hauria d'acceptar

Continuar

Fig. 6-8 Resultat correcció d'un problema. Solució incorrecta.

LLENGUATGE, GRA... INTRODUCCIÓ A L...

NOM ALUMNE

ACTIVITATS

- * AFD I
- * **AFND I**
- * APD I
- * APND I

UTILITATS

- * Mail al tutor
- * Canviar password
- * Canviar dades
- * Ajuda

<< Agost 2008 >>

DL	DT	DC	DI	DV	DS	DG
						3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Estàs a » ACME » Inici » Llenguatge, Gramàtiques i Autòmats » AFND I

ACTIVITAT 2 - AFND I

Data límit	19/10/2009 23:59		
N. Exercici	Estat	Errors de Resultat	Errors Sintàctics
1	No Result	0	0
2	Result	2	0
3	No Result	0	0

Departament d'Informàtica i Matemàtica Aplicada - Escola Politècnica Superior - Universitat de Girona

Fig. 6-9 Pantalla principal d'una activitat.

Zona de consulta

En aquesta zona (veure requadre vermell *figura 6-1*) l'alumne podrà consultar el número de problema que estar resolent, el número d'activitat al que pertany el problema, l'estat del problema, el número d'errors de resultat i sintàctics del problema i la data límit de resolució del problema.

Activitat	N. Exercici	Estat	Errors de Resultat	Errors Sintàctics	Data límit
2	1	No Resolt	0	0	19/10/2009

Fig. 6-10 Interfície d'exercicis d'autòmats. Zona de consulta

En aquesta zona també hi ha tota una sèrie d'icones que permeten a l'usuari dur a terme tota una sèrie d'accions.



Fig. 6-11 Interfície d'exercicis d'autòmats. Botons/Icones zona de consulta.

La icona en forma de llapis i paper permet a l'usuari consultar totes les solucions que ha anat enviant del problema en qüestió. Això és possible gràcies a que l'ACME s'ha anat guardant tota la informació enviada per part de l'alumne, així com també es guarda l'hora de l'enviament i el resultat de la correcció. A la *figura 6-12* s'hi pot veure la pantalla de consulta de les solucions enviades. En aquesta pantalla es mostra la data i hora en que s'ha enviat la solució, el resultat i l'error que ha donat. En cas que la solució enviada sigui correcta es mostra la solució enviada. Si la solució enviada no és correcta i es vol veure el que s'ha enviat s'ha pulsar amb el ratolí sobre el text *Veure la solució parcial* i s'obrirà una nova finestra amb el resultat enviat (veure *figura 6-13*).

La icona en forma d'impressora permet a l'usuari imprimir l'enunciat del problema.

Les icones en forma de fletxa permeten a l'usuari navegar pels diferents problemes de l'activitat.

ACME PROJECTE

Nom: Alumne aquestes són les solucions que has enviat a l'exercici 2 de l'activitat 2:

Data	Solució Enviada	Resultat
3/8/2008 01:21:59	1 Veure la solució parcial 1.1	Incorrecte
	2 ERROR: rebutja la paraula 1011 que hauria d'acceptar	Incorrecte
3/8/2008 01:34:14	1 Veure la solució parcial 2.1	Incorrecte
	2 ERROR: rebutja la paraula 1011 que hauria d'acceptar	Incorrecte

3/8/2008 01:48:57

Zoom

+ -

Tancar

Correcte

Fig. 6-12 Veure solucions enviades d'un problema.

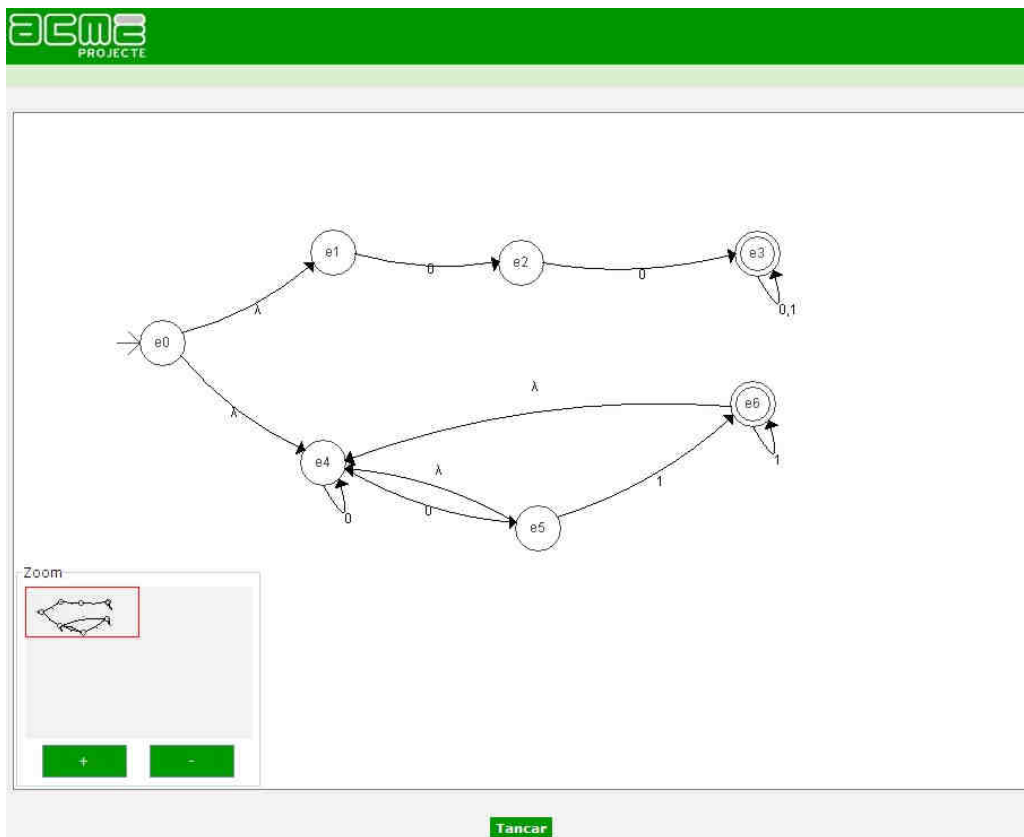


Fig. 6-13 Veure el detall d'una solució enviada.

Per tal de visualitzar les solucions enviades, les finestres o pantalles que es poden veure a les figures 6-12 i 6-13 porten integrat l'editor d'autòmats, però es mostra en mode lectura. Les úniques accions que pot dur a terme l'usuari són augmentar i disminuir el zoom.

Cal dir que totes les funcionalitats de consulta que ens ofereix aquesta zona de la pantalla ens la proporciona la plataforma ACME, seguint el mateix estil que a la resta de mòduls.

6.1.1 Característiques de l'editor d'autòmats

En aquest punt explicarem les principals característiques de l'editor d'autòmats. Des d'un bon principi aquest editor ha estat pensat per ser usat pels problemes d'autòmats finits i de pila, però a l'hora de dissenyar-lo i implementar-lo s'ha fet el màxim de genèric, de manera que es pugui aprofitar per altres assignatures en el futur, com per exemple per dibuixar diagrames d'estats de les màquines seqüencials de *Moore* o *Mealy* de l'assignatura d'introducció als computadors.

En la pròxima figura podrem veure la pantalla de l'editor que la podem dividir en tres zones: la pissarra, zona de zoom i zona de controls.

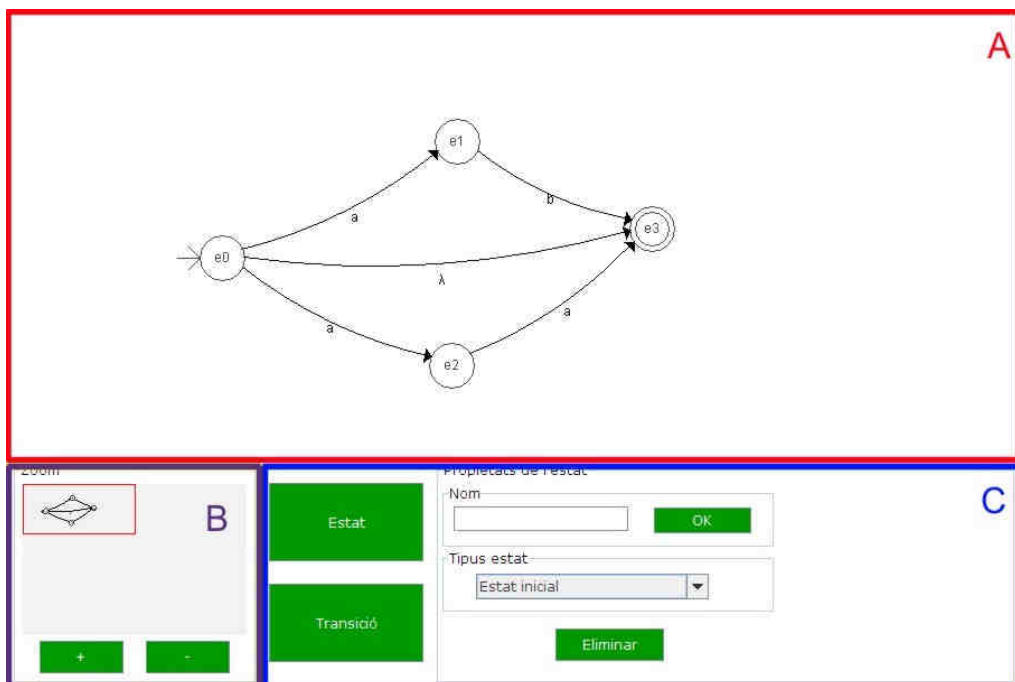


Fig. 6-14 Vista general de l'editor d'autòmats.

La pissarra

Correspon a la zona marcada de color vermell amb una A. En aquesta zona l'usuari podrà veure i dibuixar els diagrames d'estats dels autòmats.

Zona de zoom

Correspon a la zona marcada de color lila amb una B. En aquesta zona l'usuari podrà augmentar o disminuir el zoom de la pissarra.

Zona de controls

Correspon a la zona marcada de color blau amb una C. En aquesta zona l'usuari podrà canviar les propietats dels estats i de les transicions.

L'aspecte d'aquesta zona canvia segons si l'element seleccionat és un estat, una transició d'un autòmat finit o una transició d'un autòmat de pila. Si l'editor es crida en mode de només visualitzador llavors la zona C no es mostra i d'aquesta manera l'usuari no pot modificar el contingut de l'editor.

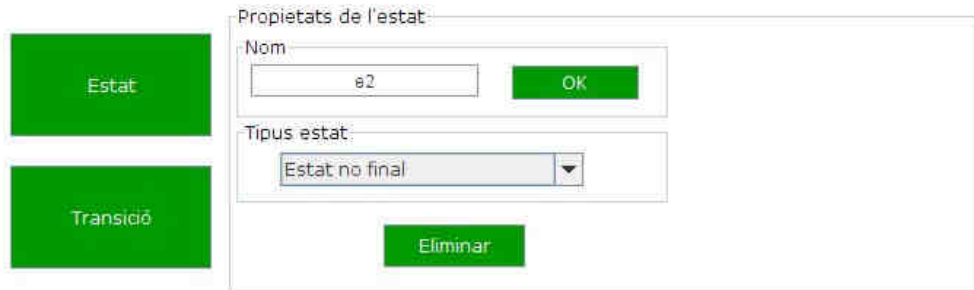


Fig. 6-15 Zona C de l'editor. Dades de l'estat seleccionat.

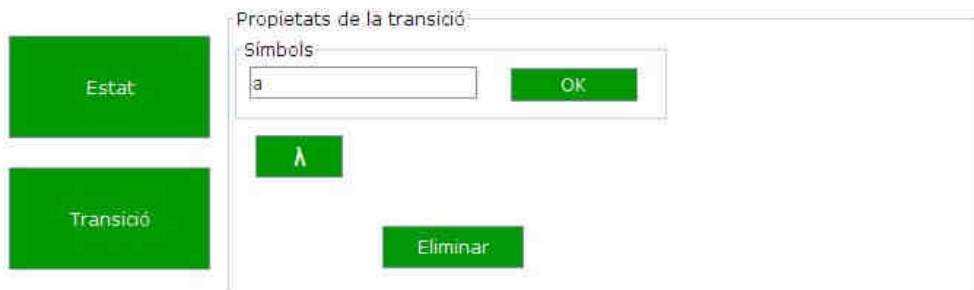


Fig. 6-16 Zona C de l'editor. Dades de la transició seleccionada.

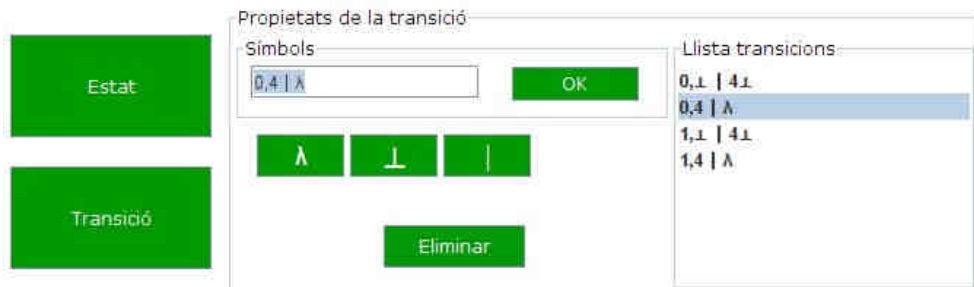


Fig. 6-17 Zona C de l'editor. Dades de la transició seleccionada d'un autòmat de pila.

Expliquem doncs quines opcions té l'usuari davant de l'editor.

Afegir un estat

Per afegir un estat l'usuari ha de polsar sobre el botó *Estat*, aquest per indicar que es troba activat quedarà de color blau cel. Amb aquest botó activat quan el punter del ratolí es mou per sobre de la pissarra es mostra la figura d'un estat que va seguint al punter del ratolí. Per acabar d'afegir l'estat només cal fer clic en algun punt de la pissarra i l'estat s'hi quedarà fixat. Si finalment no es volgués afegir l'estat tornariem a polsar sobre el botó *Estat* quedant aquest de color verd, que és el seu estat natural.



Fig. 6-18 Botó d'afegir estat en estat natural i activat.



Fig. 6-19 Imatge d'un estat en procés d'afegit i un cop afegit.

Un cop afegit l'estat aquest quedarà pintat de color gris i serà l'estat actual. L'usuari podrà completar el procés d'afegir un estat posant-li un nom (per defecte tindrà el nom *ei*, on *i* es el número de l'estat) o modificant el tipus de l'estat fent ús dels controls de la zona C.

L'usuari podrà afegir tants estat com vulgui repetint el procés descrit anteriorment.

Afegir una transició

Per afegir una transició l'usuari haurà de polsar sobre el botó *Transició*, aquest per indicar que es troba activat quedarà de color blau cel. Amb aquest botó activat primer es farà clic sobre l'estat origen i després sobre l'estat destí. Si l'estat origen i destí és el mateix farem clic dos cops sobre el mateix estat (farem dos clics simples i no un doble clic). Si un cop activat el botó *Transició* no es volgués afegir cap transició tornariem a polsar sobre el botó *Transició* quedant aquest de color verd. Si volem avortar l'operació un cop ja s'ha seleccionat l'estat origen podem fer clic a qualsevol zona de la pissarra on no hi hagi cap estat.



Fig. 6-20 Botó d'afegir transició en estat natural i activat.



Fig. 6-21 Imatge procés d'afegir una transició. Estat origen seleccionat.

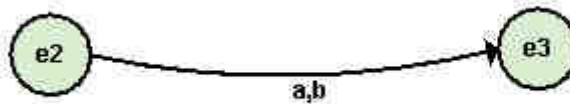


Fig. 6-22 Imatge d'una transició un cop afegida.

Un cop afegida la transició quedarà marcada en negreta i els estats d'origen i destí pintats de color gris, i passarà a ser la transició actual. L'usuari podrà completar el procés introduint els símbols que reconeix la transició a partir dels controls de la zona C.

Si estem treballant amb autòmats no deterministes haurem de poder introduir lambda transicions, per fer-ho polsarem sobre el següent botó:



Al posar aquest botó s'insertirà una lambda a la casella de text *Símbols* davant de la posició del cursor.

Si treballem amb autòmats de pila també tindrem la necessitat d'insertir el símbol de pila buida i la barra vertical, ho farem amb els següents botons:



La barra vertical també es podrà afegir des del teclat amb l'ajuda de la tecla *Alt GR*.

Seleccionar un objecte de la pissarra

L'usuari podrà seleccionar un estat o una transició de la pissarra fent clic sobre d'ell amb el ratolí. Quan el punter del ratolí passi per sobre d'un objecte prendrà la forma d'una mà.



Si es selecciona un estat aquest quedarà pintat de color gris, si es selecciona una transició aquesta quedarà marcada en negreta i els seus estats origen i destí pintats de color gris.

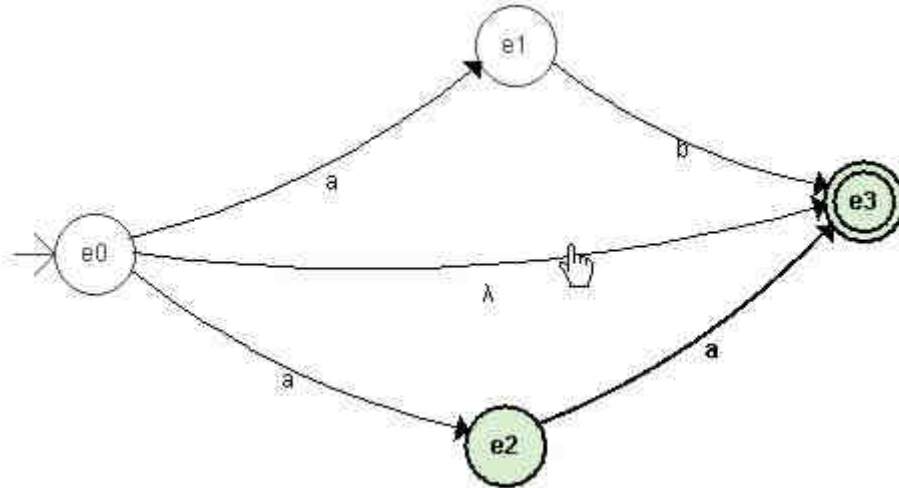


Fig. 6-23 Imatge d'un autòmat amb una transició seleccionada.

Moure objectes de la pissarra

L'usuari pot moure de lloc els estats de l'autòmat, per fer-ho ha de polsar sobre l'estat amb el botó esquerra i sense deixar-lo moure el punter del ratolí fins arribar a la posició de la pissarra desitjada, llavors deixar anar el botó esquerra. El resultat serà el desplaçament de l'estat cap al nou punt junt amb les transicions que surten o entren de l'estat. L'usuari mai podrà moure una transició.

Eliminar objecte seleccionat

Quan l'usuari vulgui eliminar un estat o una transició primer l'haurà de seleccionar i després polsar sobre el botó *Eliminar* de la zona C o sobre la tecla *Supr* del teclat.



Si l'usuari elimina un estat del que surten o entren una o diverses transicions, aquestes també s'eliminaran.

Modificar objecte seleccionat

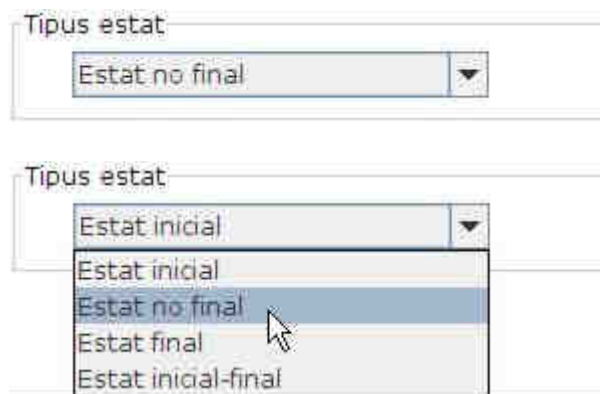
Quan l'usuari vulgui modificar les dades d'un estat o d'una transició primer l'haurà de seleccionar i després modificar les dades fent ús dels controls de la zona C.

D'un estat podem modificar el nom i el tipus:

- Per modificar el nom hem d'escriure el nou nom a la casella *Nom* i polsar el botó *OK* o la tecla *Return*. El nom de l'estat ha de ser un nom curt, d'entre 1 i 4 caràcters de longitud, ja que el nom es mostra a l'interior del cercle que representa l'estat, i si és més llarg sortirà a fora. Si es vol es pot deixar en blanc.

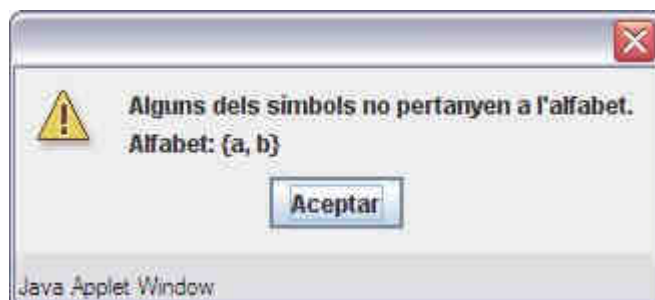


- Per modificar el tipus de l'estat cal seleccionar el nou tipus de la llista desplegable *Tipus Estat*.



D'una transició només en podem modificar els símbols que reconeix. La forma d'introduir els símbols és diferent segons el tipus d'autòmat.

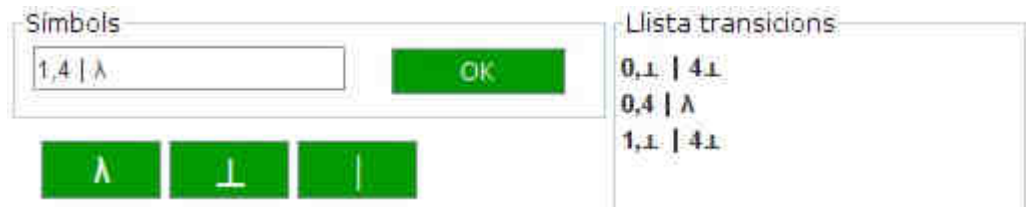
- Pels AFD's i AFND's introduïm els diferents símbols a la casella de text *Símbols* separats per comes, i després polsem sobre el botó *OK* o la tecla *return*. En cas d'introduir un símbol que no pertanyi a l'alfabet de l'autòmat l'editor donarà un missatge d'avís.



- Pels autòmats de pila la forma d'introduir els símbols reconeguts per una transició és una mica més complexa, ja que a part dels símbols de l'alfabet de l'autòmat també tenim els símbols de l'alfabet de la pila. Un exemple de transició per un autòmat de pila seria $0, \perp | 4 \perp$, que vol dir que si llegim un 0 i la pila esta buida llavors avancem cap al nou estat i empilem 4 a la pila. En lloc del símbol de pila buida podríem tenir un símbol de l'alfabet de la pila i representaria el cim de la pila. Si ens trobem davant de $0, 0 | \lambda$ vol dir que si llegim un 0 i al cim de la pila tenim un 0 llavors avancem cap al nou estat i desempilem el símbol 0. Si s'introdueix un símbol que no pertany a l'alfabet de l'autòmat o a l'alfabet de la pila l'editor donarà un avis.



Per introduir els símbols escriurem a la casella de text *Símbols* el símbol que llegim, el símbol del cim de la pila i l'acció sobre la pila seguint el format dels exemples anteriors i polsarem sobre el botó *OK* o la tecla *Return*. La casella de text quedarà en blanc i els símbols introduïts es mostraran a la llista de l'esquerra amb la resta de símbols.



Si volguéssim modificar alguns dels símbols el seleccionariem de la llista, es mostraria a la casella *Símbols*, el modificariem i polsariem el botó *OK* o la tecla *return*. Si deixem la casella *Símbols* en blanc i polsem *OK* s'eliminarà el símbol seleccionat de la llista *Llista transicions*.

Posar i treure zoom

L'usuari podrà augmentar o disminuir el zoom amb els botons de la zona B. Per augmentar el zoom cal polsar el botó +, i per disminuir-lo el botó -.



Tenim fins a set nivells diferents de zoom que l'usuari podrà aplicar en funció de les seves necessitats, per defecte sempre n'hi ha dos d'aplicats.

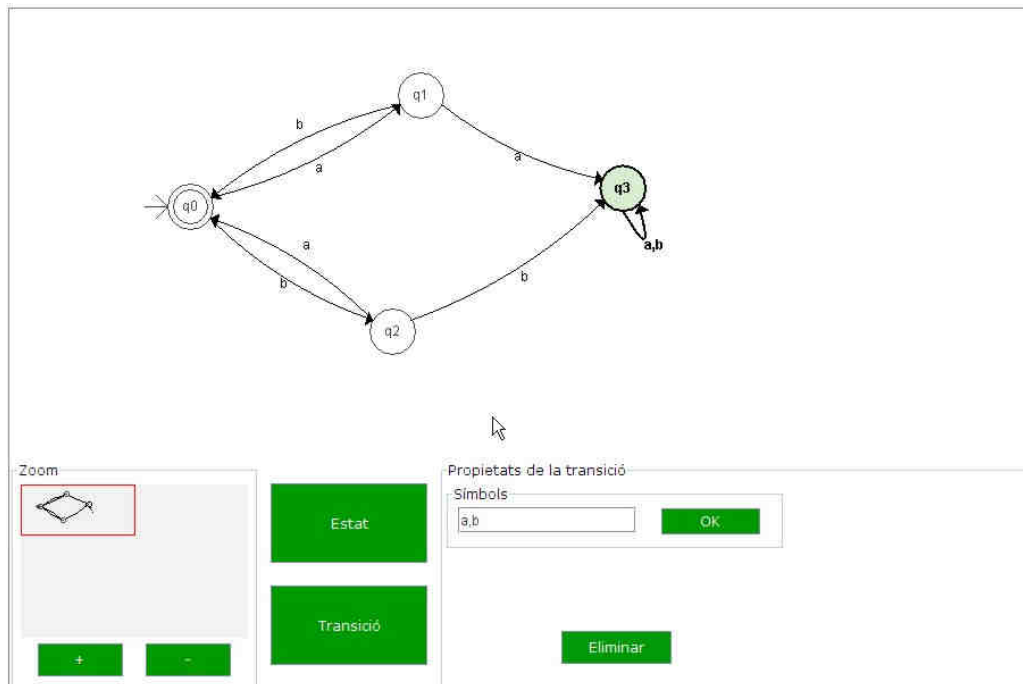


Fig. 6-24 Vista de l'editor amb el zoom aplicat per defecte.

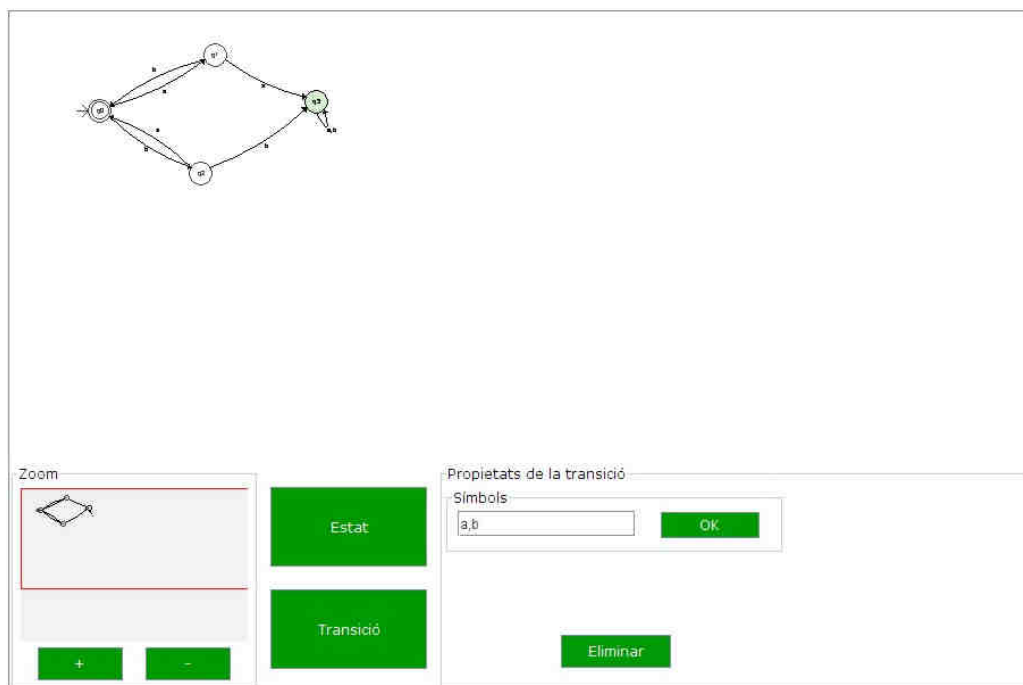


Fig. 6-25 Vista de l'editor sense cap nivell de zoom aplicat.

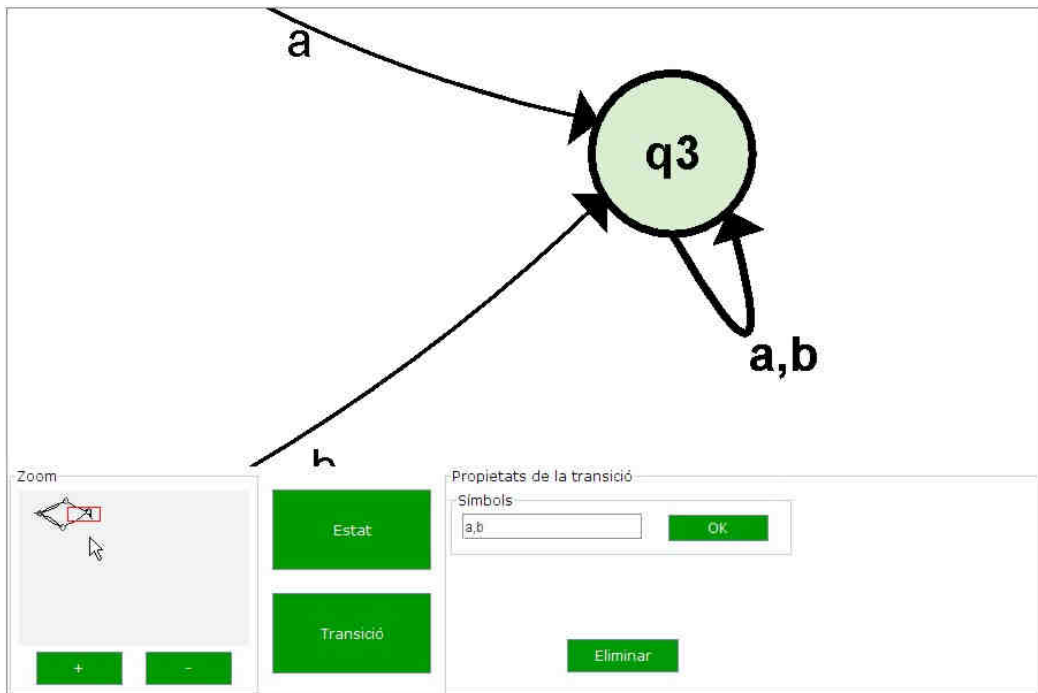
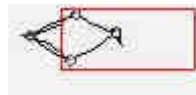


Fig. 6-26 Vista de l'editor amb els set nivells de zoom aplicats.

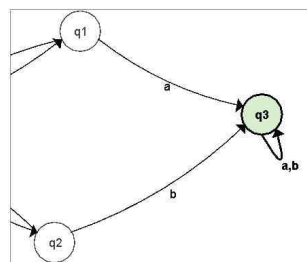
Fer translacions

Quan el contingut de la pissarra no es visualitzi sencer per tenir masses nivells de zoom aplicats o per ser massa gran, l'usuari podrà enfocar la zona de la pissarra que vulgui movent el requadre vermell de la zona B. A la pissarra només es mostra allò que queda dins d'aquest requadre.



Per moure el requadre s'ha de fer clic sobre d'ell amb el botó dret del ratolí, i sense deixar anar el botó arrossegar el requadre fins on sigui necessari.

En la posició en que es troba el requadre vermell de l'anterior imatge el contingut que veuríem a la pissarra seria el següent:



6.2 Transformació, traducció i reconstrucció d'una solució a través de l'editor

La plataforma ACME guarda les solucions enviades pels alumnes en forma de cadena de text, per tant hem de ser capaços de transformar el contingut del nostre editor a una cadena de text i restaurar-lo també a partir d'una cadena de text.

Per aquest motiu en moltes de les classes hem sobrecarregat el mètode *toString()* i implementat un constructor del tipus *NomClasse(s String)*. Amb el mètode *toString* podrem obtenir la representació de l'objecte en una cadena de text. Amb els constructors podem reconstruir un determinat objecte a partir d'una cadena de text.

La cadena que comunica l'editor amb l'ACME tindrà la següent forma:

CadenaEditor <####>CadenaCorrector

La marca <####> ens servirà per separar la informació de l'editor de la informació que rebrà el nucli corrector. A la part dreta de la cadena hi tindrem la informació de l'autòmat necessària per poder-lo corregir (tipus autòmat, estats, transicions,...), mentre que a la part esquerra hi tindrem tota aquesta informació més la informació necessària per poder dibuixar l'autòmat. A més a més les dues parts tindran algunes diferències de sintaxis per tal d'adaptar-se a les necessitats de l'editor i del nucli corrector.

6.2.1 Representació del contingut de l'editor

Tal com ja hem explicat en el punt anterior el contingut de l'editor s'haurà de poder transformar i restaurar des d'una cadena de text. Aquesta cadena seguirà la següent nomenclatura:

TipusAutomat<t#a%>Automat

En el pròxims tres punts s'explica amb més detall com representarem un autòmat, amb els seus estats i les seves transicions, amb una cadena de text.

6.2.1.1 Representació de l'autòmat

Per representar un autòmat amb una cadena de text farem servir la nomenclatura del punt anterior a on la cadena *TipusAutomat* informarà el tipus d'autòmat i podrà tenir els valors AFD, AFND, APD i APND, i la cadena *Automat* contindrà l'autòmat en si.

La cadena *Automat* seguirà la següent nomenclatura:

SA<#a#>SAP<#a#>E<#a#>T

- A la cadena *SA* hi tindrem una llista amb els símbols de l'alfabet de l'autòmat separats per la marca *<#s#>*. La cadena finalitzarà amb aquesta marca. Per exemple, amb l'alfabet {a,b,c} tindrem la cadena:

a<#s#>b<#s#>c<#s#>

- La cadena *SAP* contindrà una llista amb els símbols de l'alfabet de la pila separats per la marca *<#sp#>*. Aquesta cadena també finalitzarà amb aquesta marca. Vegem la cadena que tindríem per l'alfabet {A,B,C}.

A<#sp#>B<#sp#>C<#sp#>

- La cadena *E* contindrà la llista d'estats dels quals estar compost l'autòmat. Aquesta llista seguirà la següent nomenclatura:

<#e#>Estat₁<#e#>Estat₂<#e#>Estat₃... <#e#>Estat_N

- I per últim tenim la cadena *T* que contindrà una llista amb les transicions que formen part de l'autòmat. Les diferents transicions estaran separades per la marca *<#t#>*. Seguirà la següent nomenclatura:

<#t#>Transicio₁<#t#>Transicio₂... <#t#>Transicio_N

6.2.1.2 Representació d'un estat

Per poder representar el contingut de l'editor amb una cadena de text cal poder representar també els estats com a una cadena de text.

La cadena de text que representarà un estat seguirà la següent nomenclatura:

Num<#e#>Nom<#e#>Tipus<#e#>X<#e#>Y

On:

- La cadena *Num* serà el número de l'estat o identificador.
- La cadena *Nom* serà el nom de l'estat.
- La cadena *Tipus* contindrà el tipus de l'estat.
- Les cadenes *X* i *Y* contindran un enter cadascuna d'elles que formaran el punt on s'ha de dibuixar l'estat.

6.2.1.3 Representació d'una transició

És evident que també necessitarem poder representar les transicions com a cadenes de text. La cadena de text dissenyada per representar una transició seguirà la següent nomenclatura:

Num<#t#>**Simbols**<#t#>**Ei**<#t#>**Ef**<#t#>**Xo**<#t#>**Yo**<#t#>**Xd**<#t#>**Yd**<#t#>**Xc**<#t#>**Yc**

On:

- La cadena *Num* serà el número de la transició o identificador.
- La cadena *Simbols* serà una llista amb els símbols que reconeix la transició. Els diferents símbols estaran separats per la marca <#s#>, així si tenim una transició que reconeix els símbols *a*, *b* i *c*, la llista resultant tindria el següent aspecte:

a <#s#>**b** <#s#>**c**

- La cadena *Ei* serà el número de l'estat origen de la transició.
- La cadena *Ef* serà el número de l'estat destí de la transició.
- Les cadenes *Xo* i *Yo* contindran un enter cadascuna d'elles que formarà el punt d'origen de la fletxa que representa la transició.
- Les cadenes *Xd* i *Yd* contindran un enter cadascuna d'elles que formarà el punt de destí de la fletxa que representa la transició.
- Les cadenes *Xc* i *Yc* contindran un enter cadascuna d'elles que formarà el punt de control de la fletxa que representa la transició.

6.2.2 Representació que rep el nucli corrector

Tal i com ja he dit més amunt la cadena de text que comunica l'editor amb la plataforma ACME està formada de dues parts, una primera part per poder guardar i restaurar el contingut de l'editor, i una segona part que serà la que rebrà el nucli corrector. Aquesta segona part és la que explicarem en aquest punt.

La nomenclatura que seguirà aquesta cadena serà la següent:

Tipus<t#a%>**SU**<t#a%> **SUP**<t#a%>**NE**<t#a%>**A**

On:

- La cadena *Tipus* informarà al corrector del tipus d'estat i contindrà un dels següents valors: AFD, AFND, APD o APND.
- La cadena *SU* contindrà la llista de símbols de l'alfabet que l'usuari ha usat per dibuixar l'autòmat, els diferents símbols estaran separats per la marca $\langle ll\#s\% \rangle$. Per exemple, si tenim l'alfabet $\{a,b,c,d,e\}$ i l'usuari acaba usant només els símbols a , c i e , la cadena resultant serà la següent:

$a\langle ll\#s\% \rangle c\langle ll\#s\% \rangle e$

- La cadena *SUP* contindrà la llista de símbols de l'alfabet de la pila que l'usuari ha usat per dibuixar l'autòmat, els diferents símbols estaran separats per la marca $\langle ll\#sp\% \rangle$. Si per exemple tenim un autòmat de pila amb un alfabet de pila com ara $\{0,1,2,3,4,5,6,7\}$, i l'usuari només usa els símbols 0 i 1 , la cadena resultant serà la següent:

$0\langle ll\#s\% \rangle 1$

- La cadena *NE* contindrà el nombre d'estats de l'autòmat que ha dibuixat l'usuari.
- Per últim la cadena *A* contindrà la resta d'informació de l'autòmat, bàsicament els estats i les transicions.

6.2.2.1 Representació de l'autòmat

La cadena que usarem per representar l'autòmat que acabarà rebent el nucli corrector tindrà la següent forma:

LlistaEstats $\langle \#a\# \rangle$ LlistaTransicions

- La cadena *LlistaEstats* contindrà la llista d'estats dels quals estar format l'autòmat. Aquesta llista seguirà la següent nomenclatura:

$\langle ll\#e\% \rangle \text{Estat}_1 \langle ll\#e\% \rangle \text{Estat}_2 \langle ll\#e\% \rangle \text{Estat}_3 \dots \langle ll\#e\% \rangle \text{Estat}_N$

- La cadena *LlistaTransicions* contindrà una llista amb les transicions que formen part de l'autòmat. Les diferents transicions estaran separades per la marca $\langle ll\#t\% \rangle$. Seguirà la següent nomenclatura:

$\langle ll\#t\% \rangle \text{Transicio}_1 \langle ll\#t\% \rangle \text{Transicio}_2 \dots \langle ll\#t\% \rangle \text{Transicio}_N$

6.2.2.2 Representació d'un estat

La cadena de text que representarà un estat seguirà la següent nomenclatura:

Num<#e#>Tipus

On:

- La cadena *Num* serà el número de l'estat o identificador.
- La cadena *Tipus* contindrà el tipus de l'estat.

Fixem-nos que pel corrector necessitem menys informació, només amb el número d'estat i el seu tipus ja en fem prou. No necessitem per res la informació referent a la representació gràfica.

6.2.2.3 Representació d'una transició

La cadena de text que rebrà el corrector i que representarà una transició tindrà la següent forma:

Simbols<#t#>Ei<#t#>Ef

On:

- La cadena *Símbols* serà una llista amb els símbols que reconeix la transició. Els diferents símbols estaran separats per la marca <#s#>, així si tenim una transició que reconeix els símbols *a*, *b* i *c*, la llista resultant tindria el següent aspecte:

a <#s#>b <#s#>c

- La cadena *Ei* serà el número de l'estat origen de la transició.
- La cadena *Ef* serà el número de l'estat destí de la transició.

Fixem-nos que aquesta cadena també té menys informació que la cadena que rep l'editor. Això és degut a que al corrector només li passarem la informació mínima i indispensable per poder corregir, en cap cas necessitarà saber la representació gràfica de la transició o el número que té aquesta, amb els símbols que reconeix i els estats d'origen i destí ja en farà prou.

6.2.3 Exemple de transformació

Anem a veure un exemple de transformació del contingut de l'editor a una cadena de text. Per fer-ho partim de l'autòmat de la següent imatge.

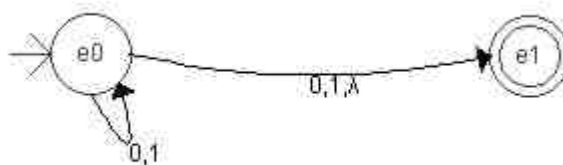


Fig. 6-27 Exemple de transformació d'un autòmat a una cadena de text.

La cadena resultant de la transformació serà:

```
AFND<#a%>0<#s%>1<#s%><#a%><#e%>0<#e%>e0<#e%>
>1<#e%>186<#e%>199<#e%>1<#e%>e1<#e%>3<#e%>405<#e%>
200<#a%><#t%>0<#t%>0<#s%>1<#t%>0<#t%>0<#t%>
>186<#t%>199<#t%>186<#t%>199<#t%>216<#t%>271<#t%>
1<#t%>0<#s%>1<#s%>\<#t%>0<#t%>1<#t%>186<#t%>
>199<#t%>405<#t%>200<#t%>295<#t%>219<###%>AFND<#t#
a%>1<#s%>0<#a%><#a%>2<#a%><#e%>0<#e%>1<#e%>
1<#e%>3<#a%><#t%>0<#s%>1<#t%>0<#t%>0<#t%>0<#t%>
>1<#s%>\<#t%>0<#t%>1
```

Separem la cadena anterior en dues, la primera amb la informació que rep l'editor i la segona amb la informació que rep el nucli corrector.

```
AFND<#a%>0<#s%>1<#s%><#a%><#e%>0<#e%>e0<#e%>
>1<#e%>186<#e%>199<#e%>1<#e%>e1<#e%>3<#e%>405<#e%>
200<#a%><#t%>0<#t%>0<#s%>1<#t%>0<#t%>0<#t%>
>186<#t%>199<#t%>186<#t%>199<#t%>216<#t%>271<#t%>
1<#t%>0<#s%>1<#s%>\<#t%>0<#t%>1<#t%>186<#t%>
>199<#t%>405<#t%>200<#t%>295<#t%>219
```

```
AFND<#a%>1<#s%>0<#a%><#a%>2<#a%><#e%>0<#e%>1
<#e%>1<#e%>3<#a%><#t%>0<#s%>1<#t%>0<#t%>0<#t%>
1<#t%>0<#s%>1<#s%>\<#t%>0<#t%>1
```

Aquí podem observar com la part de la cadena que s'enviarà cap al nucli corrector conté menys informació que la part que serveix per restaurar l'editor, ja que aquesta última conté informació de tipus gràfic que podem obviar a l'hora de corregir.

6.3 Configuració de l'idioma de l'editor

En altres capítols d'aquesta memòria ja s'ha comentat que la plataforma ACME és multilingüe i com a conseqüència l'editor d'autòmats també ho ha de ser.

La solució passa per enviar-li a l'editor una cadena de text amb les parelles de paraules o frases en l'idioma original i l'idioma que ha escollit l'usuari al entrar a l'ACME. Aquesta cadena serà de la forma:

paraula1&&&traduccio1&&¶ula2&&&traduccio2...

On:

- Les cadenes *paraula1*, *paraula2* contindran les paraules en l'idioma original.
- Les cadenes *traduccio1*, *traduccio2* contindran les traduccions corresponents.
- La marca **&&&** servirà per separar les diferents paraules.

A continuació es mostra una part de la cadena que rebria l'editor en cas que l'usuari escollís el castellà com a idioma.

```
Zoom&&&Zoom&&&Estat&&&Estado&&&Propietats de
l'estat&&&Propiedades del estado&&&Nom&&&Nombre
```

6.4 Mòduls correctors

Portem estona parlant gairebé només de l'editor i de la interfície, però cal recordar que per la generació d'aquest mòdul també ha estat necessari la implementació d'un nucli corrector format per quatre correctors, un corrector per cadascun dels quatre tipus d'autòmats a corregir: AFD, AFND, APD i APND.

En el capítol 5 d'aquesta memòria podem veure els diagrames d'activitat que descriuen els passos que segueixen cadascun dels quatre correctors, però en aquest punt explicarem amb més detall com es du a terme la correcció incorporant algun exemple que ho faci més entenedor.

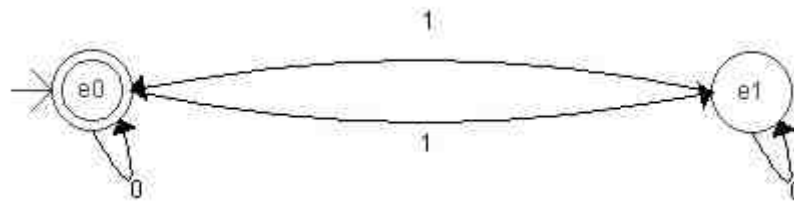
Com s'ha comentat en altres punts la cadena de text que rebrà la plataforma ACME estarà formada de dues parts, una amb la informació per restaurar l'editor i una segona pensada pel nucli corrector. Per tant el primer pas que haurà de fer el corrector serà capturar la segona part de la cadena i guardar-la en un *array*.

Per un altre costat el corrector haurà de recuperar els paràmetres introduïts pel professor, que ens serviran per corregir el problema, i guardar-los també en un *array*. Els paràmetres que el professor haurà entrat són: tipus de l'autòmat, alfabet, alfabet de la pila, número d'estats, llista de paraules acceptades i llista de paraules rebutjades.

El nucli corrector es basarà en la creació de la matriu de transicions associada a l'autòmat que haurà introduït l'usuari. Una matriu de transicions no és res més que una altra manera de representar un autòmat on:

- A les files hi tindrem els diferents estats.
- A les columnes els diferents símbols de l'alfabet de l'autòmat.
- L'estat inicial el marcarem amb el símbol \rightarrow .
- Els estats finals els marcarem amb el símbol $*$.
- Les caselles representaran les transicions.

Vegem un exemple de matriu de transicions creada a partir del diagrama de transicions.



f	0	1
$\rightarrow * e0$	e0	e1
e1	e1	e0

Fig. 6-28 Exemple de matriu de transicions.

Un cop creada la matriu de transicions el corrector validarà que la solució entrada per l'usuari accepti totes les paraules de la llista de paraules acceptades entrada pel professor, i que rebutgi totes les paraules de la llista de paraules rebutjades facilitada també pel professor.

El corrector per validar si una paraula és acceptada o no usarà el següent algorisme:


```

s:=s0;
c:=llegirCar(paraula);
mentre c <> EOF fer
    s:=moure(s,c);
    c:=llegirCar(paraula);
fi mentre
si s pertany a F llavors retorna cert
altrament retorna fals
fi si

```

Fig. 6-29 Algorisme simulació d'un AFD.

A on:

- s apuntarà a l'estat on ens trobem de l'autòmat, és a dir a una fila de la matriu. S s'inicialitza amb l'estat inicial.
- La variable *paraula* conté la paraula a validar, i la funció *llegirCar(...)* ens retorna el símbol a consumir. c contindrà en tot moment el símbol a consumir, és a dir una columna de la matriu.
- La funció *moure(s,c)* ens retornarà l'estat a on anem a parar des de l'estat s consumint el símbol c .
- F és el conjunt d'estats finals de l'autòmat.
- La constant EOF indica fi de paraula.

Si un cop hem gastat tots els símbols de la paraula *car* estem en un estat final retornem cert i acceptem la paraula, altrament retornarem fals i no acceptem la paraula.

Abans de començar a construir la matriu de transicions el corrector farà tota una sèrie de validacions, algunes necessàries per poder construir la matriu correctament, d'altres no necessàries per la matriu però que ens permetran informar més detalladament a l'usuari de l'error que ha comés al solucionar el problema. A continuació es mostren aquestes validacions.

- Es validarà que el tipus d'autòmat de la solució enviada sigui el mateix que el que ens va proporcionar el professor en el seu dia. En principi el tipus d'autòmat sempre coincidirà ja que l'usuari en cap moment el pot modificar, però si per algun motiu no coincidís la matriu de transicions quedaria molt construïda i per tant el problema mal corregit.
- Es validarà que el nombre d'estats de la solució enviada sigui el mateix que el nombre que ens va proporcionar el professor en el

seu dia. Aquesta validació serà útil quan l'enunciat del problema demani un autòmat mínim, ja que podríem donar per bo un autòmat que fos equivalent, però que no fos mínim.

- Es validaran els símbols de l'alfabet usats en la solució enviada. Aquesta validació serà diferent en funció del tipus d'autòmat a corregir.
- Es validarà que l'autòmat de la solució enviada tingui com a mínim i com a màxim un estat inicial. Aquesta validació no faria falta ja que al no tenir cap estat inicial no podríem començar a cercar per la matriu i per tant el resultat seria fals. Si es posa aquesta validació serà per indicar a l'usuari més detalladament l'error que ha comés.

El nucli corrector retornarà cap a l'ACME el següent *array*:

booleà	Error
--------	-------

A on:

- El booleà indicarà el resultat de la correcció. Cert si és correcte i fals si és incorrecte.
- Error contindrà un text informatiu de l'error comés. Si el resultat de la correcció és correcte contindrà espais.

Algunes d'aquesta validacions no es faran o es faran de forma diferent en funció del tipus d'autòmat a corregir. Això mateix també ens passarà amb la matriu de transicions i l'algorisme de simulació, en funció del tipus d'autòmat la matriu es construirà diferent i l'algorisme de simulació se'n veurà afectat. Aquestes diferències s'expliquen en els següents quatre punts.

6.4.1 Corrector d'AFD's

El codi del corrector implementat pels problemes d'AFD's segueix els passos indicats en el diagrama d'activitat de la figura 5-5 del capítol d'anàlisi i disseny.

Les validacions que farem pels AFD seran les següents:

- Validar que el nombre d'estats de la solució enviada sigui el mateix que el nombre d'estats que va introduir el professor en el seu dia. En cas que el professor introduís zero significa que la solució pot tenir qualsevol nombre d'estats.
- Validar els símbols de l'alfabet usats. Com que en un AFD de cada estat ha de sortir una transició per cada símbol de l'alfabet, el

resultat és que s'han d'usar tots els símbols de l'alfabet. Si no és així haurem de donar un error. Aquesta validació no seria necessària ja que ho detectaríem al moment d'emular l'autòmat amb la matriu de transicions, però d'aquesta forma l'alumne tindrà una informació més detallada sobre l'error que ha comés.

- Validar que l'autòmat tingui un i només un estat inicial.

Un cop fetes totes aquestes validacions passarem a construir la matriu de transicions que la farem servir per emular a un AFD. Al mateix temps que construïm la matriu validem que de cada estat en surtin tantes transicions com símbols té l'alfabet i que sigui determinista. Un exemple de matriu de transicions per un AFD és la de la figura 6-28.

Un cop construïda la matriu de transicions mirarem si l'autòmat de la solució introduïda per l'usuari accepta totes les paraules de la llista de paraules acceptades que ens va proporcionar el professor. Si es rebutja alguna de la paraules caldrà donar un error.

Per últim haurem de mirar que l'autòmat de la solució proposada per l'usuari rebutgi totes les paraules de la llista de paraules rebutjades que ens va proporcionar el professor. Si s'accepta alguna d'aquestes paraules caldrà donar un error.

6.4.2 Corrector d'AFND's

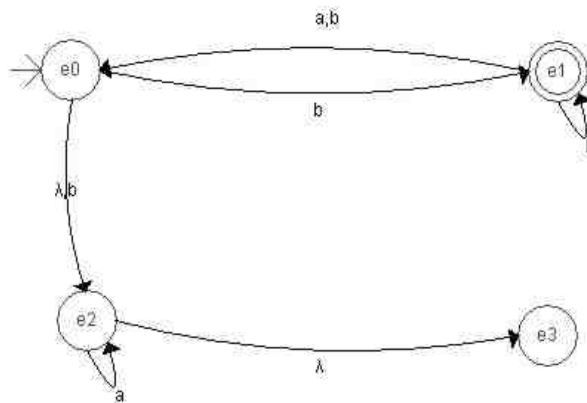
El codi del corrector implementat pels problemes d'AFND's segueix els passos indicats en el diagrama d'activitat de la figura *Fig. 5-6 Diagrama d'activitat procés de correcció d'un AFND* del capítol d'anàlisi i disseny.

Les validacions que farem pels AFND seran les següents:

- Validar que el nombre d'estats de la solució enviada sigui el mateix que el nombre d'estats que va introduir el professor en el seu dia. En cas que el professor introduís zero significa que la solució pot tenir qualsevol nombre d'estats.
- Validar els símbols de l'alfabet usats, que la solució enviada per l'usuari usi tots els símbols de l'alfabet. Si no és així haurem de donar un error. Aquesta validació no seria necessària ja que ho detectaríem al moment d'emular l'autòmat amb la matriu de transicions, però d'aquesta forma l'alumne tindrà una informació més detallada sobre l'error que ha comés.
- Validar que l'autòmat tingui un i només un estat inicial.

Un cop fetes totes aquestes validacions passarem a construir la matriu de transicions que la farem servir per emular a un AFND. Al tractar-se d'un AFND d'un mateix estat poden sortir de 0 a n transicions per un mateix símbol, per tant ens estalviem un parell de validacions.

Al tractar-se d'un AFND a cada casella de la matriu no hi tindrem un estat destí, si no que hi tindrem un vector d'estats destí. En la següent imatge podem veure un AFND representat en les formes de diagrama de transicions i matriu de transicions.



f	a	b	λ
$\rightarrow e0$		e1, e2	e2
* e1	e0	e0,e1	
e2	e2		e3
e3			

Fig. 6-30 Exemple de matriu de transicions d'un AFND

Podem observar que la matriu de transicions d'un AFD és lleugerament diferent a la matriu de transicions d'un AFND, per un costat tenim que a les caselles hi ha més d'un estat destí, i per l'altre ens apareixen les lambda transicions, per tant ens caldrà modificar també l'algorisme de la figura 6-29.

Les modificacions que farem a l'algorisme de la figura 6-29 seran:

- Primer de tot el transformarem d'iteratiu a recursiu, d'aquesta manera en quedarà més simple i intuïtiu.
- Farem ús de la tècnica de *backtracking* per tal buscar una solució entre els diferents estats d'una mateixa casella.

- Un cop hàgim provat tots els estats d'una casella sense èxit caldrà mirar si hi ha lambda transicions des de l'estat origen i intentar trobar una solució amb les lambda transicions.

Un cop construïda la matriu de transicions mirarem si l'autòmat de la solució introduïda per l'usuari accepta totes les paraules de la llista de paraules acceptades que ens va proporcionar el professor. Si es rebutja alguna de la paraules caldrà donar un error.

Per últim haurem de mirar que l'autòmat de la solució proposada per l'usuari rebutgi totes les paraules de la llista de paraules rebutjades que ens va proporcionar el professor. Si s'accepta alguna d'aquestes paraules caldrà donar un error.

6.4.3 Corrector d'APD's

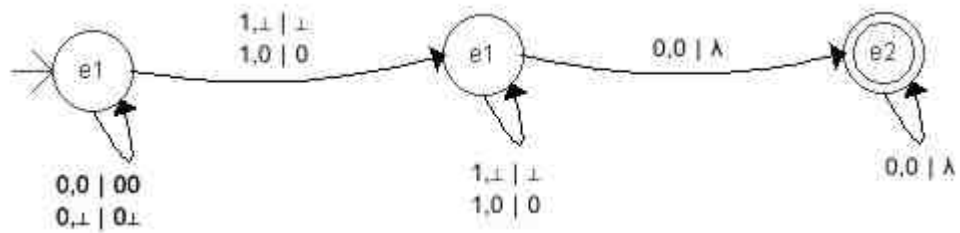
El codi del corrector implementat pels problemes d'APD's segueix els passos indicats en el diagrama d'activitat de la figura 5-7 del capítol d'anàlisi i disseny.

Les validacions que farem pels APD's seran les següents:

- Validar que el nombre d'estats de la solució enviada sigui el mateix que el nombre d'estats que va introduir el professor en el seu dia. En cas que el professor introduís zero significa que la solució pot tenir qualsevol nombre d'estats.
- Validar que els símbols usats a la pila pertanyin a l'alfabet de la pila.
- Validar que l'autòmat tingui un i només un estat inicial.

Un cop fetes totes aquestes validacions passarem a construir la matriu de transicions que la farem servir per emular a un APD. Al mateix temps que construïm la matriu validem que l'autòmat sigui determinista.

La matriu de transicions d'un APD serà bastant diferent a la d'un AFD. La diferència més gran serà que la matriu d'un APD tindrà tres dimensions en lloc d'una. Això es degut que a l'hora d'anar a buscar l'estat destí haurem de tenir en compte tres paràmetres: l'estat origen d'on partim, el símbol que estem consumint i el símbol del cim de la pila. A la següent imatge podem veure un APD representat en les formes de diagrama de transicions i matriu de transicions.



f	Símbol entrada	0		1		λ	
	Cim pila	0	\perp	0	\perp	0	\perp
	$\rightarrow e0$	(e0,0)	(e0,0)	(e1,)	(e1,)		
	e1	(e2, λ)		(e1,)	(e1,)		
	* e2	(e2, λ)					

Fig. 6-31 Exemple de matriu de transicions d'un APD

Podem observar que la matriu de transicions d'un APD és molt diferent a la d'un AFD. Deixant de banda que té tres dimensions, a cada casella i tenim una tupla de dos elements, el primer element indica l'estat destí i el segon l'acció a fer sobre la pila: si hi ha un símbol de la pila empilem, si hi ha una lambda desempilem i si no hi ha res deixem la pila tal com estar. Per tant també ens caldrà modificar l'algorisme de simulació.

En l'exemple de la anterior figura no tenim cap lambda transició, però un APD pot tenir-ne.

Per desenvolupar l'algorisme corrector partirem de l'algorisme corrector d'un AFND, l'adaptarem per que tracti una matriu de tres dimensions en lloc de una de dues dimensions, i haurem de tenir en compte que a cada casella a part de l'estat destí hi tindrem l'acció a fer sobre la pila.

Un cop construïda la matriu de transicions mirarem si l'autòmat de la solució introduïda per l'usuari accepta totes les paraules de la llista de paraules acceptades que ens va proporcionar el professor. Si es rebutja alguna de la paraules caldrà donar un error.

Per últim haurem de mirar que l'autòmat de la solució proposada per l'usuari rebutgi totes les paraules de la llista de paraules rebutjades que ens va proporcionar el professor. Si s'accepta alguna d'aquestes paraules caldrà donar un error.

6.4.4 Corrector d'APND's

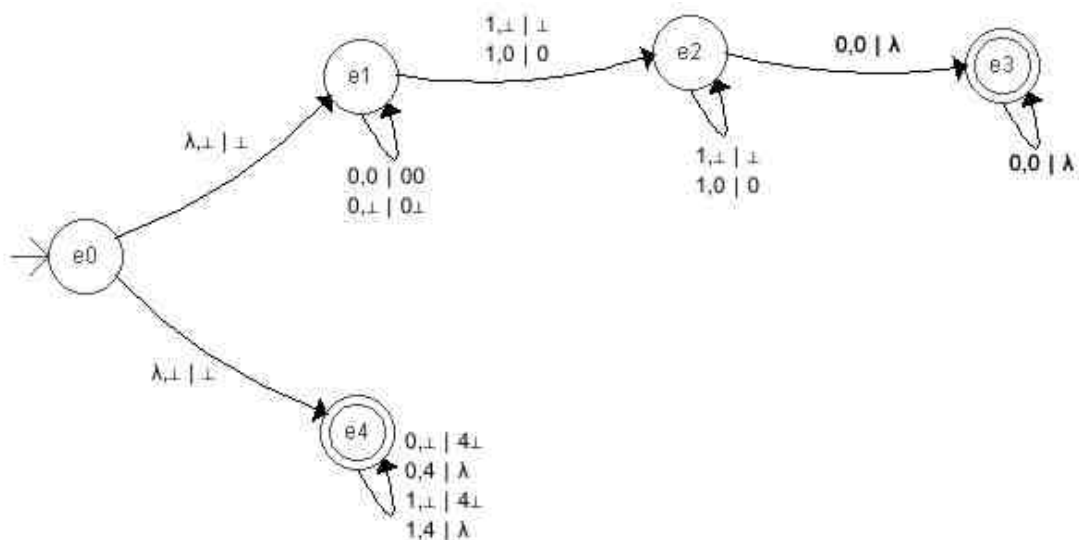
El codi del corrector implementat pels problemes d'APND's segueix els passos indicats en el diagrama d'activitat de la figura 5-8 del capítol d'anàlisi i disseny.

Les validacions que farem pels APND's seran les següents:

- Validar que el nombre d'estats de la solució enviada sigui el mateix que el nombre d'estats que va introduir el professor en el seu dia. En cas que el professor introduís zero significa que la solució pot tenir qualsevol nombre d'estats.
- Validar que els símbols usats a la pila pertanyin a l'alfabet de la pila.
- Validar que l'autòmat tingui un i només un estat inicial.

Un cop fetes totes aquestes validacions passarem a construir la matriu de transicions que la farem servir per emular a un APND, evidentment no caldrà validar que sigui determinista.

La matriu de transicions d'una APND és bastant semblant a la d'un APD, la diferència recau en que a cada casella en lloc de tenir una tupla tindrem un vector de tuples. En la següent imatge podem veure un APND representat en les formes de diagrama de transicions i matriu de transicions.



f	Símbol entrada	0			1			λ		
	Cim pila	0	4	\perp	0	4	\perp	0	\perp	4
	$\rightarrow e0$								(e1,),(e4,)	
	e1	(e1,0)		(e1,0)	(e2,)	(e2,)				
	e2	(e3, λ)			(e2,)		(e2,)			
	* e3	(e3, λ)								
	* e4		(e4, λ)	(e4,4)		(e4, λ)	(e4,4)			

Fig. 6-32 Exemple de matriu de transicions d'un APND.

Per emular un APND necessitarem fer ús de la tècnica del *backtracking* per tal d'explorar les diferents opcions d'una mateixa casella. Com que hem sigut previsors i l'algorisme que emulava un APD ja porta incorporada aquesta tècnica podrem aprofitar el mateix algorisme.

Un cop construïda la matriu de transicions mirarem si l'autòmat de la solució introduïda per l'usuari accepta totes les paraules de la llista de paraules acceptades que ens va proporcionar el professor. Si es rebutja alguna de la paraules caldrà donar un error.

Per últim haurem de mirar que l'autòmat de la solució proposada per l'usuari rebutgi totes les paraules de la llista de paraules rebutjades que ens va proporcionar el professor. Si s'accepta alguna d'aquestes paraules caldrà donar un error.

7 Tipus 27 – Autòmats finits

En aquest capítol veurem quines són les especificacions que hauran de seguir els problemes d'aquest nou mòdul per poder-los penjar a la plataforma ACME i els passos que haurà de seguir el professor per tal de crear nous problemes.

7.1 Definició dels enunciats dels problemes

Per aquest nou tipus de problema tindrem un fitxer per problema, és a dir, que cada vegada que es vulgui crear un problema s'haurà de crear un fitxer amb el tipus de problema, l'enunciat i el paràmetres necessaris per poder corregir el problema.

L'estructura que haurà de seguir aquest fitxer s'explicarà més endavant.

7.2 Definició dels problemes

Per redactar un nou problema es crearà un fitxer de text pla que haurà de seguir una determinada estructura. Cada fitxer representarà un nou problema a donar d'alta i a sortejar. El nom del fitxer no cal que segueixi cap format en concret, tot i que s'aconsella que sigui el màxim de descriptiu possible. Un exemple de nom podria ser *ProblemaAPD01.txt*.

7.3 Implementació dels problemes

Per tal de poder reaprofitar codi, principalment de verificació, i mantenir la coherència amb els altres mòduls es mantindrà el mateix esquema de tipus de fitxer adaptant-lo als requeriments del nou mòdul. Cada fitxer es trobarà dividit en tres parts, capçalera, enunciat i definició del paràmetres per a la correcció. En els següents tres punts s'expliquen les característiques d'aquests tres apartats.

7.3.1 Capçalera del fitxer

La capçalera dels problemes d'autòmats serà molt simple, igual que en altres tipus de problemes. Només hi caldrà indicar el tipus de problema amb un número, en el nostre cas serà el número 27.

27

Fig. 7-1 Exemple de capçalera d'un fitxer de problemes.

7.3.2 Enunciat del problema

En els nostres fitxers de problemes només es s'hi podrà definir un sol problema, però un mateix problema podrà tenir l'enunciat redactat de varies formes. Per marcar l'inici i el final de cadascuna de les redaccions de l'enunciat ho farem amb els *metatags* `%<E>` i `%</E>`. Tot text que es trobi entre aquestes dues marques serà mostrat per la plataforma ACME com a enunciat.

```
%<E>
Trobar l'autòmat finit determinista que accepta el
llenguatge representat per l'expressió regular
(ab+ba)*
%</E>
```

Fig. 7-2 Exemple d'enunciat d'un fitxer de problemes.

7.3.3 Definició dels paràmetres per a la correcció

Per últim ens quedarà introduir la solució del problema. Per marcar l'inici i final de la solució ho farem amb els *metatags* `%<SOLUCIO>` i `%</SOLUCIO>`. Tot text que es trobi entre aquestes dues marques serà considerat part de la solució, i si no compleix les especificacions que ha de complir una solució el procés de verificació donarà un error.

La solució estarà formada per una llista de paràmetres amb els seus valors corresponents. Aquests paràmetres els farà servir el nucli corrector per validar les solucions introduïdes per l'usuari. Millor vegem-ne un exemple.

```
%<SOLUCIO>
TIPUS AFD
ALFABET a,b
ESTATS 4
ACCEPTA
abababbaba
$
baabba
REBUTJA
aa
bb
aba
abbbaa
a
%</SOLUCIO>
```

Fig. 7-3 Exemple de solució d'un fitxer de problemes.

En funció del tipus d'autòmat sobre el que tracti el problema alguns d'aquests paràmetres poden variar.

7.4 Exemples complets de problemes

En aquest punt veurem quatre exemples complets de problemes d'autòmats, un exemple per cada tipus de problema a corregir.

7.4.1 Exemple complet d'un problema d'un AFD

A continuació es mostra un exemple complet d'un problema d'autòmats, concretament d'un AFD. Les parts imprescindibles que es tenen en compte a l'hora de fer la verificació es ressalten en negreta.

```

27
%<E>
Trobeu l'AFD mínim que reconeix el llenguatge de les
paraules formades per zeros, uns i dosos que representen un
nombre ternari divisible per tres (la paraula buida no
pertany al llenguatge).
%</E>

%<SOLUCIO>
TIPUS AFD
ALFABET 0,1,2
ESTATS 4
ACCEPTA
10
20
000
120
20120
REBUTJA
$
1
012
201
10202
0220101
%</SOLUCIO>

```

Fig. 7-4 Exemple complet d'un fitxer de problemes d'un AFD.

Els paràmetres *TIPUS*, *ALFABET* i *ESTATS* sempre han de tenir algun valor (per això estan marcats en negreta) ja que si no el procés de verificació donarà un error. A més a més el valor del paràmetre *ESTATS* ha de ser un nombre natural, i el paràmetre *TIPUS* només pot rebre el valor AFD. Els paràmetres *ACCEPTA* i *REBUTJA* corresponen a la llista de paraules acceptades i rebutjades i poden estar buides. Per últim indicar que el símbol \$ (dòlar) representa la paraula buida.

7.4.2 Exemple complet d'un problema d'un AFND

A continuació es mostra un exemple complet d'un problema d'autòmats, concretament d'un AFND. Les parts imprescindibles que es tenen en compte a l'hora de fer la verificació es ressalten en negreta.

```

27

%<E>
Trobeu un AFND amb épsilon-moviments que reconegui el
llenguatge de les paraules formades per zeros i uns que
comencen amb dos zeros o acaben amb dos uns.
%</E>

%<SOLUCIO>
TIPUS AFND
ALFABET 0,1
ESTATS 7
ACCEPTA
00
001
011
0001
1011
01011
01010111
REBUTJA
$
0
1
01
101
1101
1100
1001
0100
%</SOLUCIO>

```

Fig. 7-5 Exemple complet d'un fitxer de problemes d'un AFND.

Els paràmetres *TIPUS*, *ALFABET* i *ESTATS* sempre han de tenir algun valor (per això estan marcats en negreta) ja que si no el procés de verificació donarà un error. A més a més el valor del paràmetre *ESTATS* ha de ser un nombre natural, i el paràmetre *TIPUS* només pot rebre el valor AFND. Els paràmetres *ACCEPTA* i *REBUTJA* corresponen a la llista de paraules acceptades i rebutjades i poden estar buides. Per últim indicar que el símbol \$ (dòlar) representa la paraula buida.

7.4.3 Exemple complet d'un problema d'un APD

A continuació es mostra un exemple complet d'un problema d'autòmats, concretament d'un APD. Les parts imprescindibles que es tenen en compte a l'hora de fer la verificació es ressalten en negreta.

```

27

%<E>
Trobeu un APD que accepti les paraules de la forma  $0^n 1^m$ 
 $0^n$  amb  $m>0$ ,  $n \geq 0$ .

Podeu triar l'alfabet de la pila dins del conjunt de símbols
 $P=\{0,1,2,3,4\}$ . No cal fer-los servir tots.

%</E>

%<SOLUCIO>
TIPUS APD
ALFABET 0,1,2
ALFABET PILA 0,1,2,3,4
ESTATS 3
ACCEPTA
010
01110
00100
1
REBUTJA
00
01
10
0010
001000
000101000
%</SOLUCIO>

```

Fig. 7-6 Exemple complet d'un fitxer de problemes d'un APD.

Els paràmetres *TIPUS*, *ALFABET*, *ALFABET PILA* i *ESTATS* sempre han de tenir algun valor (per això estan marcats en negreta) ja que si no el procés de verificació donarà un error. A més a més el valor del paràmetre *ESTATS* ha de ser un nombre natural, i el paràmetre *TIPUS* només pot rebre el valor APD. Els paràmetres *ACCEPTA* i *REBUTJA* corresponen a la llista de paraules acceptades i rebutjades i poden estar buides. Per últim indicar que el símbol \$ (dòlar) representa la paraula buida.

7.4.4 Exemple complet d'un problema d'un APND

A continuació es mostra un exemple complet d'un problema d'autòmats, concretament d'un APND. Les parts imprescindibles que es tenen en compte a l'hora de fer la verificació es ressalten en negreta.

```

27

%<E>
Trobeu un APND que accepti les paraules sobre l'alfabet
S={0,1,+,*,(,)} que representen expressions regulars
correctes com, per exemple, (0*+(11)*)*.

Podeu triar l'alfabet de la pila dins del conjunt de símbols
P={0,1,+,.,*,(,)}.

%</E>

%<SOLUCIO>
TIPUS APND
ALFABET 0,1,+,.,*,(,)
ALFABET PILA 0,1,+,.,*,(,)
ESTATS 3
ACCEPTA
0
1
(0+1)*
(0+1)*(0+1)*
((0*+(11)*)*
((0))
(((0+1)(0+1))(1+11))*
REBUTJA
()
0++1
(01+)*
(0+1))
%</SOLUCIO>

```

Fig. 7-7 Exemple complet d'un fitxer de problemes d'un APND.

Els paràmetres *TIPUS*, *ALFABET*, *ALFABET PILA* i *ESTATS* sempre han de tenir algun valor (per això estan marcats en negreta) ja que si no el procés de verificació donarà un error. A més a més el valor del paràmetre *ESTATS* ha de ser un nombre natural, i el paràmetre *TIPUS* només pot rebre el valor APND. Els paràmetres *ACCEPTA* i *REBUTJA* corresponen a la llista de paraules acceptades i rebutjades i poden estar buides. Per últim indicar que el símbol \$ (dòlar) representa la paraula buida.

8 Integració a la plataforma ACME

Un cop desenvolupada la interfície, els nuclis correctors i el verificador que analitzarà que l'estructura dels problemes especificats pel professor sigui correcta, cal integrar-ho tot a la plataforma ACME.

Per fer-ho ens ha calgut familiaritzar-nos amb el funcionament de l'ACME. Alguns dels punts que hem hagut de conèixer sobre l'ACME són: les parts que ha de tenir qualsevol mòdul de la plataforma, com interactuen els usuaris amb la plataforma, què és un tipus de problema i quines parts té, com es creen els problemes i com es creen els seus enunciats i es sortegen, com es corregeixen els problemes, l'estructura interna de directoris de la plataforma, i per últim els llenguatges de programació que es poden utilitzar amb la plataforma.

La plataforma ACME va ser dissenyada des del primer moment per poder ser ampliada sense gaire dificultat. Té tota una sèrie d'eines que fan que la integració d'un nou mòdul sigui senzilla i el màxim de transparent.

Per començar amb la integració cal identificar el nostre tipus de problema. Un tipus de problema és un conjunt d'exercicis que tenen unes particularitats semblants entre ells, per exemple els problemes de tipus test, que tots tenen una solució bona i la resta són dolentes.

Qualsevol tipus de problema de la plataforma ACME hereta de la classe *Problema*, i per tant heretarà tot un conjunt de mètodes que li aportaran les funcionalitats necessàries per integrar-se a la plataforma, es a dir que es pugui verificar, sortejar, corregir,... Aquests mètodes a la vegada poden variar per adaptar-se a les necessitats de cadascun dels tipus de problema.

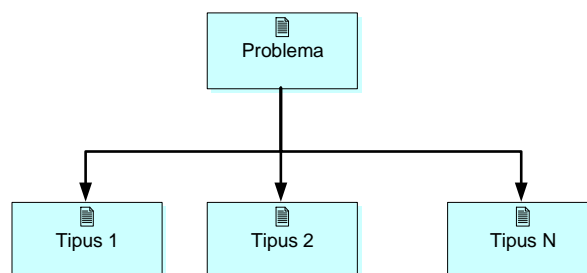


Fig. 8-1 Esquema dels tipus de problemes.

Per la correcció d'exercicis d'autòmats finits i de pila no hem trobat cap tipus de problema que s'adapti a les nostres necessitats, és per això que hem hagut de crear-ne un de nou, el tipus 27.

Per crear aquest nou tipus de problema hem hagut de donar suport a cadascun dels següents punts:

- **Pujada del problema.** El professor escriu el problema seguint les pautes especificades i el dona d'alta a l'ACME. Si el professor no té permisos ho haurà de fer l'administrador. Les pautes a seguir s'han explicat a l'anterior capítol.
- **Verificació.** El sistema ha de validar la correctesa del problema proposat pel professor. Si és correcte, s'incorporarà a la base de dades de l'ACME, altrament es refusarà i s'informarà de l'error. El procés de verificació s'ha explicat al capítol 5.
- **Sorteig.** Durant el sorteig es fa una adjudicació aleatòria d'enunciats de problemes als alumnes. Aquest punt no ens ha portat gaire feina ja que la forma de sorteig és molt semblant a la d'altres tipus de problemes ja existents a l'ACME.
- **Interacció amb l'usuari.** Aquesta part permet a l'usuari entrar les solucions dels problemes. Pot ser molt diferent en funció de la naturalesa de cada tipus de problema. En el nostre cas és la part que ens ha dut més feina. D'aquesta part se'n parla detalladament als capítols 5 i 6.
- **Correcció.** Cada tipus de problema té associat un corrector, que corregeix la solució enviada per l'usuari i li dona una resposta de forma immediata. El procés de correcció s'ha explicat en els capítols 5 i 6.
- **Visualització.** El sistema permet a l'usuari consultar l'estat d'un determinat problema i les solucions enviades fins al moment amb el resultat de la correcció. El procés de visualització s'explica en els capítols 5 i 6.

8.1 Integració del tipus de problema 27

Ja hem explicat que el tipus de problema 27 hereta de la classe *Problema*, i que aquesta proporciona tota una sèrie de mètodes que permeten al tipus 27 adaptar-se a la plataforma ACME sense cap tipus de problema.

En aquest punt veurem aquells mètodes que s'han hagut de modificar a l'hora de desenvolupar el tipus de problema 27.

Verificar_fitxer

Aquest mètode li passem el fitxer d'enunciats i ens retorna un *array* amb el conjunt d'enunciats col·locat de la manera en que el tipus de problema requereix.

En el nostre cas com que només tindrem un problema per fitxer ens retornarà un *array* d'una sola posició amb l'enunciat i solució del problema. Aquest mètode també valida l'estructura del fitxer, és per això que nosaltres l'hem hagut d'adaptar a l'estructura d'un fitxer de problemes d'autòmats.

Form_enviar_solucio i Form_enviar_solucio_profe

Són els mètodes que carreguen els formularis que serveixen perquè l'alumne i el professor pugin enviar la solució al corrector.

Aquests dos mètodes els hem modificat perquè carreguessin el nostre applet, és a dir, l'editor d'autòmats.

També ha sigut necessari crear la funció *getResposta()* en *JavaScript* que és l'encarregada de cridar al mètode *toString2()* implementat en l'*applet*. Amb aquest mètode obtenim la cadena de caràcters que defineix el contingut de l'editor, i que permet enviar la resposta de l'usuari cap a l'ACME.

Corregir_Problema

Aquest mètode li passem el problema i les solucions enviades per l'usuari, i ens retorna la solució global i totes les solucions corregides que hagi pogut enviar l'usuari. En el nostre cas, com que només tenim una possible solució, només envia el resultat de la correcció, cert o fals, i en cas de ser erroni l'error comés.

És obvi que aquest mètode l'hem hagut de modificar per tal que cridi al nostre nucli corrector.

Mostrar_Solucions_Enviades

Aquest mètode s'encarrega de mostrar la solució enviada per l'usuari en el moment en què es mostra el resultat de la correcció. L'hem hagut de modificar per tal que cridés al nostre editor d'autòmats.

Visualitzar_Solucio

Amb aquest mètode l'usuari pot visualitzar cadascuna de les solucions enviades del problema. També l'hem hagut de modificar per tal que cridés al nostre editor d'autòmats.

9 Posta en marxa

En aquest capítol veurem com es farà la posta en marxa d'aquest nou mòdul dins de la plataforma ACME i es facilitarà la informació necessària per tal de poder fer proves del nou mòdul des de la plataforma ACME.

9.1 Incorporació de la nova aplicació

Per incorporar el nou mòdul a la plataforma ACME caldrà copiar els diferents fitxers que el componen als diferents directoris del sistema. Caldrà tenir en compte la funcionalitat de cada fitxer per copiar-lo a un o altre directori.

Es copiaran de la següent manera:

- Al directori */www/models* hi trobem els diferents fitxers que implementen els diferents tipus de problemes que suporta la plataforma ACME. En aquest directori hi copiarem el fitxer *tipus_27.php*. Aquest fitxer conté la implementació pel tipus de problemes que s'ha afegit pels problemes d'autòmats finits i de pila.
- Al directori */www/applet* hi trobem els diferents *applets* que formen part d'algunes de les interfícies de la plataforma. En aquest directori hi copiarem el fitxer *automats.jar*. Aquest fitxer conté totes les classes necessàries pel funcionament de l'*applet* implementat.
- Al directori */www/traduccio* hi trobem els fitxers que donen suport al multilingüisme de la plataforma ACME. Per incorporar el nostre nou mòdul caldrà modificar almenys els fitxers *catala.php* i *castella.php*, introduint els nous textos de la nova interfície traduïts al català i al castellà.

Després de copiar i modificar tots aquests fitxers faltará donar d'alta el nou tipus de problema a la base de dades de l'ACME i associar-li el corrector implementat en aquest projecte.

9.2 Com fer proves

Si es vol provar l'aplicació cal seguir els següents passos:

1. Accedir a la plataforma ACME a través de l'adreça web <http://acme.udg.edu>
2. Identificar-se com a alumne o com a professor. Per identificar-se com a alumne es pot entrar amb algun d'aquests quatre usuaris: *alumne1*, *alumne2*, *alumne3* o *alumne4*. Per identificar-se com a professor es pot entrar amb algun d'aquests dos usuaris: *profe1* o *profe2*. Tots aquests usuaris com a clau tenen la paraula *acme*.
3. Fer clic sobre l'assignatura Llenguatges, gramàtiques i autòmats (LGA) i escollir una de les quatre activitats: AFD's, AFND's, APD's i APND's.

A través dels diferents problemes de les quatre activitats es poden provar les diferents funcionalitats que ofereix el nou mòdul de l'ACME.

No s'ha cregut convenient crear cap manual donada la facilitat d'ús de la interfície implementada.

10 Conclusions

L'objectiu principal d'aquest Projecte Final de Carrera era elaborar un mòdul d'autòmats finits per la plataforma ACME. Els resultats obtinguts satisfan els objectius inicials, havent desenvolupat un mòdul que permet generar i corregir problemes d'autòmats finits i de pila, tant deterministes com no deterministes.

En el moment d'escollir el projecte vaig tenir en compte varis factors. Un d'aquests factors era haver d'enfrontar-me a nous llenguatges de programació, a noves eines i a nous entorns en general. En aquest sentit crec que també he complert els objectius.

S'ha implementat un editor d'autòmats finits i de pila, que permet introduir els autòmats en la seva forma de diagrama de transicions. La seva elaboració no ha sigut trivial, i es correspon a la part del projecte que m'ha portat més feina i a la vegada també ha sigut la part que m'ha aportat més didàcticament. Per desenvolupar l'editor he usat l'entorn de desenvolupament *Eclipse*, i per dissenyar la interfície m'ha sigut de gran ajuda l'editor *Visual Editor* que porta integrat l'*Eclipse*. El funcionament d'aquestes dues eines era desconegut per mi fins ara. També he après a treballar amb els paquets *Swing* i *AWT* de *Java*.

També s'ha implementat el nucli corrector que hi ha al darrera de l'editor. La implementació d'aquest nucli s'ha dut a terme amb el llenguatge *PHP*, fet que m'ha permès adquirir uns mínims coneixements de *PHP* i de programació web en general, ja que mai havia treballat en programació de pàgines web dinàmiques.

Tant l'editor com el nucli corrector no serien res sense la corresponen integració en el sistema ACME, fet que m'ha obligat a obtenir un coneixements previs sobre el funcionament i estructuració interns de l'ACME.

A nivell personal ha sigut una experiència del tot satisfactòria. M'ha aportat tot allò que crec que ha d'aportar un Projecte Final de Carrera: per una banda aprendre i conèixer noves eines i entorns, i per l'altra aprofundir en els coneixements adquirits durant la carrera. També és gratificant saber que aquest projecte es posarà en funcionament i que servirà d'ajuda tant a professors com a alumnes d'assignatures de les carreres d'informàtica de la Universitat de Girona, i que no es quedarà en un recó omplint-se de pols.

Finalment agrair la col·laboració a totes aquelles persones que han fet possible aquest projecte.

11 Possibles millores i treballs futurs

Durant la realització del projecte s'han anat trobant noves idees i propostes per tal de millorar i/o ampliar aquest mòdul corrector d'autòmats. A continuació es nombren algunes d'aquestes possibles futures millores o ampliacions.

- Quan l'usuari aplica un cert nivell de zoom o fa una translació del contingut de la pissarra, aquests no queden guardats enlloc i quan es torna entrar hi ha les opcions de zoom per defecte. Podríem guardar les opcions de zoom i carregar-les quan l'usuari tornés a accedir-hi.
- Afegir la funcionalitat que l'usuari pugui accedir a les propietats dels objectes de la pissarra (nom, tipus, símbols,...) fent clic sobre d'ells amb el botó dret del ratolí accedint a un menú emergent. D'aquesta manera la edició d'un autòmat seria més àgil i amena.
- Permetre a l'usuari modificar la trajectòria de la fletxa que representa una transició. D'aquesta manera alguns diagrames de transicions quedarien més entenedors.
- Aquest mòdul es podria ampliar creant altres correctors per nous tipus de problemes de les assignatures de LGA o TALLF. Alguns d'aquests tipus de problemes són: problemes d'expressions regulars, problemes de llenguatges, problemes de gramàtiques,... Per aquests nous problemes caldria veure si es pot aprofitar el nostre mòdul o part d'aquest, o potser es podria aprofitat alguns del mòduls ja existents a l'ACME, o simplement caldria fer un mòdul nou.
- Una altra possible ampliació d'aquest mòdul seria trobar la forma de poder-lo visualitzar i executar a través d'una PDA, un Pocket PC, un telèfon mòbil, etc.

12 Índex de figures

Fig. 4-1 Diagrama de casos d'ús de context.	19
Fig. 4-2 Diagrama de casos d'ús per resoldre exercicis d'autòmats.....	20
Fig. 4-3 Diagrama de casos d'ús per introduir la solució d'un problema.....	21
Fig. 4-4 Diagrama de casos d'ús per afegir un objecte a la pissarra.	22
Fig. 4-5 Diagrama de casos d'ús per afegir un estat.....	22
Fig. 4-6 Diagrama de casos d'ús per afegir una transició.	23
Fig. 4-7 Diagrama de casos d'ús per modificar la vista de la pissarra.	23
Fig. 4-8 Diagrama de casos d'ús per obtenir i establir els paràmetres de l'editor.....	24
Fig. 4-9 Diagrama de casos d'ús per establir els paràmetres de l'editor.	24
Fig. 5-1 Diagrama de paquets implementació editor.....	34
Fig. 5-2 Paquets utilitzats del paquet java.	35
Fig. 5-3 Paquets utilitzats del paquet javax.	35
Fig. 5-4 Diagrama de classes de l'editor.	38
Fig. 5-5 Diagrama d'activitat procés de correcció d'un AFD	51
Fig. 5-6 Diagrama d'activitat procés de correcció d'un AFND.....	52
Fig. 5-7 Diagrama d'activitat procés de correcció d'un APD	53
Fig. 5-8 Diagrama d'activitat procés de correcció d'un APND.....	54
Fig. 5-9 Diagrama d'activitat procés d'obtenció de la cadena que defineix el contingut de l'editor.....	55
Fig. 5-10 Diagrama d'activitat procés de restauració dels paràmetres de l'editor.....	57
Fig. 5-11 Diagrama d'activitat procés de validació d'un fitxer de problemes.	58
Fig. 5-12 Diagrama de seqüència general per refrescar la pissarra i la zona de zoom.	61
Fig. 5-13 Diagrama de seqüència per afegir un estat. Activació del boto 'Estat'.	62
Fig. 5-14 Diagrama de seqüència per afegir un estat. Selecció del punt de la pissarra.	63
Fig. 5-15 Diagrama de seqüència per afegir un estat. Inserció de l'estat a la pissarra.....	64
Fig. 5-16 Diagrama de seqüència per afegir una transició. Activació del botó 'Transició'.	65
Fig. 5-17 Diagrama de seqüència per afegir una transició. Selecció de l'estat origen.....	66
Fig. 5-18 Diagrama de seqüència per afegir una transició. Selecció de l'estat destí i inserció de la transició.	67
Fig. 5-19 Diagrama de seqüència per seleccionar un objecte de la pissarra.....	69
Fig. 5-20 Diagrama de seqüència per moure un objecte de la pissarra. Pulsar sobre l'objecte.....	70
Fig. 5-21 Diagrama de seqüència per moure un objecte de la pissarra. Arrastrar l'objecte.	71

Fig. 5-22 Diagrama de seqüència per moure un objecte de la pissarra. Deixar el botó dret del ratolí.	72
Fig. 5-23 Diagrama de seqüència per eliminar un objecte de la pissarra. Eliminar un estat. ..	72
Fig. 5-24 Diagrama de seqüència per eliminar un objecte de la pissarra. Eliminar una transició.	73
Fig. 5-25 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar el nom d'un estat.	74
Fig. 5-26 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar el tipus d'un estat.	74
Fig. 5-27 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar els símbols d'una transició.	75
Fig. 5-28 Diagrama de seqüència per modificar un objecte de la pissarra. Afegir un símbol a una transició.	76
Fig. 5-29 Diagrama de seqüència per modificar un objecte de la pissarra. Eliminar un símbol d'una transició.	77
Fig. 5-30 Diagrama de seqüència per modificar un objecte de la pissarra. Modificar un símbol d'una transició.	78
Fig. 5-31 Diagrama de seqüència per augmentar el zoom de la pissarra.	79
Fig. 5-32 Diagrama de seqüència per disminuir el zoom de la pissarra.	80
Fig. 5-33 Diagrama de seqüència per modificar la vista de la pissarra.	81
Fig. 5-34 Diagrama de seqüència per modificar la vista de la pissarra. Pulsar botó dret del ratolí.	81
Fig. 5-35 Diagrama de seqüència per modificar la vista de la pissarra. Arrossegar el requadre de la vista.	82
Fig. 5-36 Diagrama de seqüència per obtenir la cadena que defineix el contingut de l'editor.	84
Fig. 5-37 Diagrama de seqüència per restaurar el contingut de l'editor.	85
Fig. 5-38 Diagrama de seqüència per inicialitzar el diccionari de l'idioma.	86
Fig. 6-1 Interfície d'exercicis d'autòmats.	87
Fig. 6-2 Interfície d'exercicis d'autòmats. Zona de l'enunciat.	88
Fig. 6-3 Resultat correcció d'un problema. Solució incorrecta.	89
Fig. 6-4 Resultat correcció d'un problema. Solució correcta.	89
Fig. 6-5 Resultat correcció d'un problema. Solució incorrecta.	90
Fig. 6-6 Resultat correcció d'un problema. Solució incorrecta.	91
Fig. 6-7 Resultat correcció d'un problema. Solució incorrecta.	91
Fig. 6-8 Resultat correcció d'un problema. Solució incorrecta.	92
Fig. 6-9 Pantalla principal d'una activitat.	92
Fig. 6-10 Interfície d'exercicis d'autòmats. Zona de consulta	93
Fig. 6-11 Interfície d'exercicis d'autòmats. Botons/Icones zona de consulta.	93
Fig. 6-12 Veure solucions enviades d'un problema.	94

Fig. 6-13 Veure el detall d'una solució enviada	95
Fig. 6-14 Vista general de l'editor d'autòmats	96
Fig. 6-15 Zona C de l'editor. Dades de l'estat seleccionat	97
Fig. 6-16 Zona C de l'editor. Dades de la transició seleccionada	97
Fig. 6-17 Zona C de l'editor. Dades de la transició seleccionada d'un autòmat de pila	97
Fig. 6-18 Botó d'afegir estat en estat natural i activat	98
Fig. 6-19 Imatge d'un estat en procés d'afegit i un cop afegit	98
Fig. 6-20 Botó d'afegir transició en estat natural i activat	98
Fig. 6-21 Imatge procés d'afegir una transició. Estat origen seleccionat	99
Fig. 6-22 Imatge d'una transició un cop afegida.....	99
Fig. 6-23 Imatge d'un autòmat amb una transició seleccionada	100
Fig. 6-24 Vista de l'editor amb el zoom aplicat per defecte.....	103
Fig. 6-25 Vista de l'editor sense cap nivell de zoom aplicat	103
Fig. 6-26 Vista de l'editor amb els set nivells de zoom aplicats	104
Fig. 6-27 Exemple de transformació d'un autòmat a una cadena de text.....	110
Fig. 6-28 Exemple de matriu de transicions	112
Fig. 6-29 Algorisme simulació d'un AFD	113
Fig. 6-30 Exemple de matriu de transicions d'un AFND	116
Fig. 6-31 Exemple de matriu de transicions d'un APD	118
Fig. 6-32 Exemple de matriu de transicions d'un APND	120
Fig. 7-1 Exemple de capçalera d'un fitxer de problemes	121
Fig. 7-2 Exemple d'enunciat d'un fitxer de problemes	122
Fig. 7-3 Exemple de solució d'un fitxer de problemes.....	122
Fig. 7-4 Exemple complet d'un fitxer de problemes d'un AFD	123
Fig. 7-5 Exemple complet d'un fitxer de problemes d'un AFND.....	124
Fig. 7-6 Exemple complet d'un fitxer de problemes d'un APD	125
Fig. 7-7 Exemple complet d'un fitxer de problemes d'un APND	126
Fig. 8-1 Esquema dels tipus de problemes	127

13 Bibliografia

Monografies

- Ferri Rabasa, Francesc Josep. Teoria d'autòmats i llenguatges formals. València: Publicacions Universitat de València, 2004. (Educació; 75; Materials) [84-370-1806-4].
- Martin, John C. Introduction to languages and the theory of computation. International edition 1991. Nova York: McGraw-Hill, 1991. [0-07-100851-9].
- Schildt, Herbert. Traduït per M^a Luz Morcillo Punzano. La biblia de Java 2 v5.0. [The Complete Reference Java J2SE 5 Edition]. Madrid: Ediciones Anaya Multimedia (Grupo Anaya, S.A), 2005. [84-415-1865-3].
- Brogden, Bill. Traduït per Javier Saralegui Sánchez i Joaquín Siabra Fraile. Manual fundamental de Java. [Discover Java]. Madrid: Ediciones Anaya Multimedia, S.A., 1997. [84-415-0274-9].
- Coggeshall, John. Traduït per Patricia Scott Peña i María Jesus Fernández Vélez. La biblia de PHP 5 [PHP 5 Unleashed]. Madrid: Ediciones Anaya Multimedia (Grupo Anaya, S.A), 2005. [84-425-1845-9].

Pàgines Web

- Documentació de la plataforma JDK 5.0.
<http://java.sun.com/j2se/1.5.0/docs/index.html>.
- Manual de *PHP*.
<http://es2.php.net/manual/es/>.
- Enciclopèdia lliure en línia.
<http://www.wikipedia.org>.
- Documentació sobre el servidor *Apache*.
<http://www.apache.org>.