Generació automàtica de codi ANSI C a partir del diagrama d'una màquina d'estats

Pere Sitjà Rius
Enginyeria en Informàtica

Anna Queralt Calafat

14-01-2008

Generació automàtica de codi ANSI C a partir del diagrama d'una màquina d'estats.

PFC -Tècniques avançades d'enginyeria del programari.

El projecte consisteix en la implementació d'una aplicació que generi el codi en ANSI C de manera automàtica a partir del diagrama d'una màquina d'estats, més concretament del model generat en format XMI XML Metada Interchange (XML d'intercanvi de metadades) per l'aplicació gràfica de disseny de programari argoUML0.24, aquesta aplicació és independent de la plataforma, de codi obert i gratuïta.

Aquesta eina facilitarà les tasques de disseny, desenvolupament i posterior manteniment de les aplicacions desenvolupades en ANSI C, més concretament en sistemes encastats. El codi generat serà el descrit en els diagrames de màquines d'estats, que fan referència al funcionament dinàmic d'aquestes aplicacions. També permetrà, als desenvolupadors d'aquestes aplicacions, tenir una perspectiva jeràrquica de més alt nivell i una visió molt més clara de l'estructura del programari en aplicacions més o menys complexes, tenir els desenvolupaments més ràpids i amb menys cost, així com facilitar-ne el manteniment i una possible reutilització. Només una petita part de codi haurà de ser escrita pels desenvolupadors, concretament la que permet la interacció entre la màquina d'estats i el maquinari i d'altres parts de l'aplicació.

Per la realització del projecte s'han realitzat les tasques definides en les etapes d'anàlisi, disseny, implementació i test de l'aplicació. En l'etapa d'anàlisi s'han inclòs les tasques de comprensió d'una màquina d'estats i la seva codificació en llenguatge C com a tasques prèvies a l'anàlisi de l'aplicació.

L'aplicació s'ha desenvolupat amb llenguatge Java per tal de tenir un codi executable independent del sistema operatiu i no se l'ha dotat de cap interfície gràfica de comunicació amb l'usuari per tal de facilitar la seva integració en els entorns de programació IDE Integrated Development Environment (Entorn de treball integrat.

L'aplicació rep el nom de MASGEC (MArc d'eStats GEnerador de Codi).

Índex de continguts

Índex de figures

1. Introducció.

1.1. Justificació.

En el desenvolupament d'aplicacions en sistemes encastats la generació del programari és una part del projecte molt important, però que normalment no està acompanyada d'un bon mètode de desenvolupament, sobre tot en la part d'anàlisi i disseny del programari. Tots els esforços es concentren en la implementació, tenint com a resultat d'aquesta pràctica un manteniment molt costós i una reutilització pràcticament nul·la, això passa sobre tot en les empreses amb equips de desenvolupament reduïts, on una o dues persones realitzen totes les tasques del projecte.

El poder disposar d'una eina que generi codi directament d'un diagrama de màquina d'estats permetrà als desenvolupadors d'aquestes aplicacions tenir una perspectiva jeràrquica de més alt nivell i una visió molt més clara de l'estructura del programari.

L'aplicació genera el codi amb llenguatge C, perquè la majoria dels microcontroladors utilitzats en sistemes encastats tenen un compilador de llenguatge C pel desenvolupament del programari, a partir de l'eina de desenvolupament gcc de GNU.

No té cap interfície gràfica de comunicació amb l'usuari per tal de facilitar la integració d'aquesta eina en els entorns de programació IDE Integrated Development Environment (entorn de treball integrat), que normalment faciliten l'ús d'aplicacions en línia externes amb pas de paràmetres.

1.2. Objectius.

Obtenir una aplicació que, a partir d'un model de diagrama de màquina d'estats generat per l'aplicació gràfica de disseny de programari, generi el codi en ANSI C de manera automàtica, per tal que mitjançant un compilador i un enllaçador, obtenir el codi executable per un processador o controlador determinat.

Sense entrar amb més detalls, aquesta pot ser una feina molt més gran que la que aquest projecte pot aspirar, per tant, l'objectiu principal d'aquest projecte és ajudar a tenir una base perquè, en futurs treballs, l'eina que aquí s'inicia sigui realment un instrument útil i perquè no, imprescindible pel desenvolupament del programari de sistemes encastats. Tampoc pretenem implementar totes les funcionalitats que donen els diagrames d'estats definits per l'UML, sinó que ens centrarem en els elements que creiem imprescindibles per realitzar la majoria d'aplicacions en sistemes encastats.

Per això, destacarem principalment les tasques inicials de l'etapa d'anàlisi:

- La descripció de les possibilitats d'una màquina d'estats.
- Interpretar el fitxer XMI generat a partir del diagrama d'estats.
- Implementar una màquina d'estats en codi C.

1.3. Enfocament i mètode seguit.

Per la realització del PFC s'ha dividit el projecte en les etapes clàssiques en el cicle de vida d'un projecte informàtic: anàlisi, disseny, implementació i proves.

En l'etapa d'anàlisi hem inclòs les tasques prèvies que hem cregut necessàries per tal de poder assolir els objectius marcats, doncs aquestes ens han permès obtenir una visió completa del problema que es plantejava en el projecte, així com poder definir millor les tasques de les següents etapes. Aquestes tasques han estat l'exploració de les possibilitats d'un diagrama de màquina d'estats, l'elecció de l'eina per realitzar els diagrames a l'hora d'obtenir un fitxer de sortida en format XMI i la implementació de la màquina d'estats amb llenguatge C.

L'aplicació s'ha dividit en tres sistemes:

- Model de màquina d'estats. Aquest sistema representarà la màquina d'estats del diagrama dibuixat.
- L'analitzador. Que ha partir del fitxer d'entrada XMI generarà el model de màquina d'estats.
- El generador de codi. Què ha partir del model de màquina d'estats generarà el codi en ANSI C.

L'analitzador i el generador compartiran el model de maquina d'estats que hem dissenyat. D'aquesta manera amb petites modificacions en la part analitzadora podrem acceptar d'altres formats de fitxers d'entrada sense haver de tocar res del sistema generador. Per la mateixa raó i de manera independent podrem crear d'altres generadors de codi.
Hem usat l'entorn de programació Eclipse i JDK-6 com eines per implementar el codi en Java.
Les proves han consistit en realitzar un seguit de diagrames de màquines d'estat que contemplessin les diferents combinacions, si més no les més comuns, i realitzar la generació de codi.
Durant l'etapa de proves també s'ha realitzat el manual d'usuari i s'ha començat la memòria.

1.4. Planificació del projecte.

Les etapes del projecte són:

- Anàlisi de l'aplicació. Especificació de les característiques més importants de l'aplicació. Destaquem les següents tasques:
  - Definició de les possibilitats dels diagrames d'estats.
  - Elecció de l'aplicació gràfica de disseny de programari.
  - Interpretació del fitxer XMI.
  - Implementació d'una màquina d'estats amb llenguatge C.
  - Definició del generador de codi.
- Disseny. Etapa per realitzar el disseny que satisfaci les especificacions establertes en la fase de anàlisi. Les tasques principals són:
  - Disseny del model de màquina d'estats.
  - Disseny de l'analitzador.
  - Disseny del generador de codi.
- Implementació de l'aplicació. Etapa per codificar en llenguatge Java. Les tasques principals són:
  - Implementació del model de màquina d'estats.
  - Implementació de l'analitzador.
  - Implementació del generador de codi.
- Proves. Etapa per realitzar les proves més convenients per tal de garantir el funcionament correcte de l'aplicació. Les tasques principals són:
  - Proves de l'aplicació sobre diferents diagrames de màquines d'estats.
- Documentació. Etapa per realitzar les tasques de documentació. Les tasques principals són:
  - Realització del manual d'usuari
  - Realització de la memòria.
  - Realització de la presentació.

Les fites coincideixen amb les PACs.
PAC1 28-09-2007. En aquesta data tindrem la descripció i la planificació del projecte.
PAC2 29-10-2007. En aquesta data tindrem les especificacions i el disseny de l'aplicació acabades.
PAC3 17-12-2007. En aquesta data tindrem l'aplicació implementada i estarem realitzant les proves. També es tindrà un proposta de manual d'usuari.

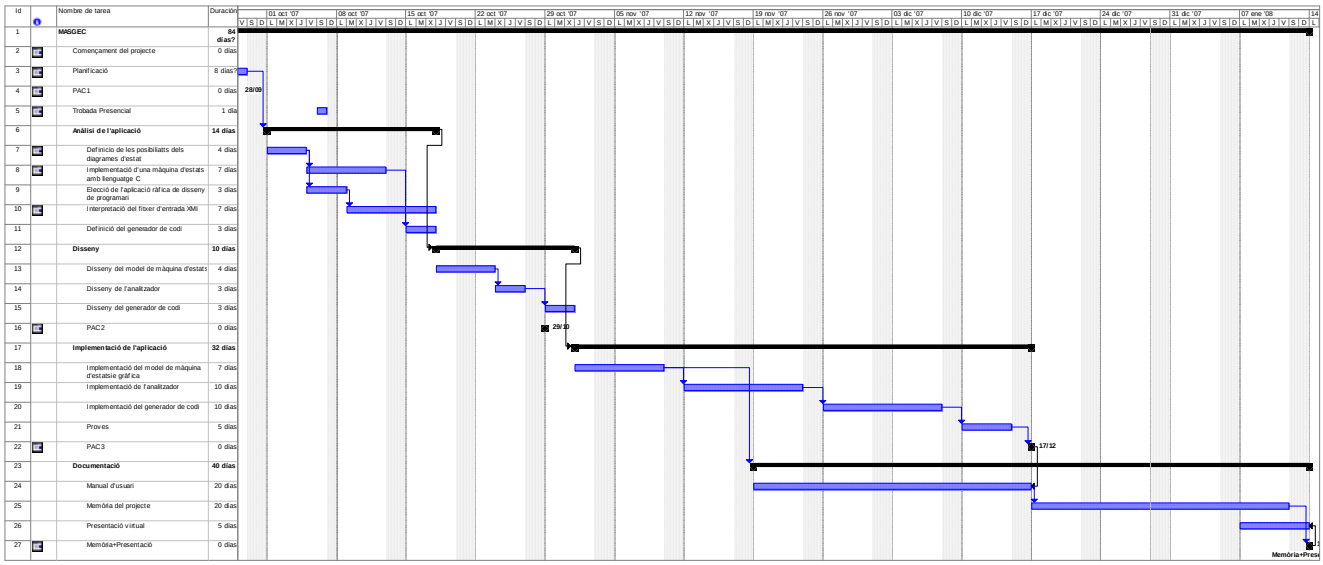


Figura 1. Calendari del projecte.

1.5. Productes obtinguts.

L'aplicació MASGEC es subministrarà en un fitxer comprimit on estaran tots els fitxers necessaris per què l'aplicació funcioni correctament, distribuïts en els directoris corresponents., També hi serà el manual d'usuari i un fitxer "Readme.txt" amb les instruccions més elementals per la seva instal·lació.

El manual d'usuari que ha de tenir com a punts més destacats:

- Una guia de com instal·lar i executar l'aplicació.
- Una petita introducció als diagrames d'estats i les seves possibilitats sobre tot en el desenvolupament de sistemes encastats.
- La informació de com aconseguir l'aplicació gràfica de disseny de software ArgoUML0.24.

1.6. Descripció dels altres capítols de la memòria.

En el capítol 2 hi trobarem la descripció de les tasques realitzades durant l'anàlisi de l'aplicació. Principalment les prèvies a la definició de l'aplicació com són les possibilitats d'un diagrama d'estats, la codificació d'una màquina d'estats en codi C i la interpretació del fitxer XMI generat per l'ArgoUML0.24.

En el capítol 3 hi trobarem les tasques realitzades durant l'etapa de disseny. Principalment l'obtenció dels diagrames de classes dels diferents sistemes que formaran part de l'aplicació.

En el capítol 4 hi trobarem les tasques realitzades durant l'implementació. Bàsicament la forma com s'han acabat implementant les classes obtingudes durant la fase de disseny.

En el capítol 5 hi trobarem un exemple de maquina d'estats implementada.

En el capítol 6 hi trobarem les conclusions del projecte.

En el capítol 7 hi trobarem la valoració econòmica del projecte.

2. Anàlisi de l'aplicació.

2.1. Definició de les possibilitats d'un diagrama d'estats.

2.1.1. Diagrama d'estats

Una màquina d'estats és l'especificació del funcionament dinàmic d'un model. Consisteix en un nombre finit d'estats i en una col·lecció de transicions, a més, només pot estar en un estat a l'hora.
Un diagrama d'estats pot estar format per diferents components:
Estats, transicions i reaccions d'estats.
En la Figura 2 es pot observar un exemple de màquina d'estats amb la majoria d'elements que la poden composar.

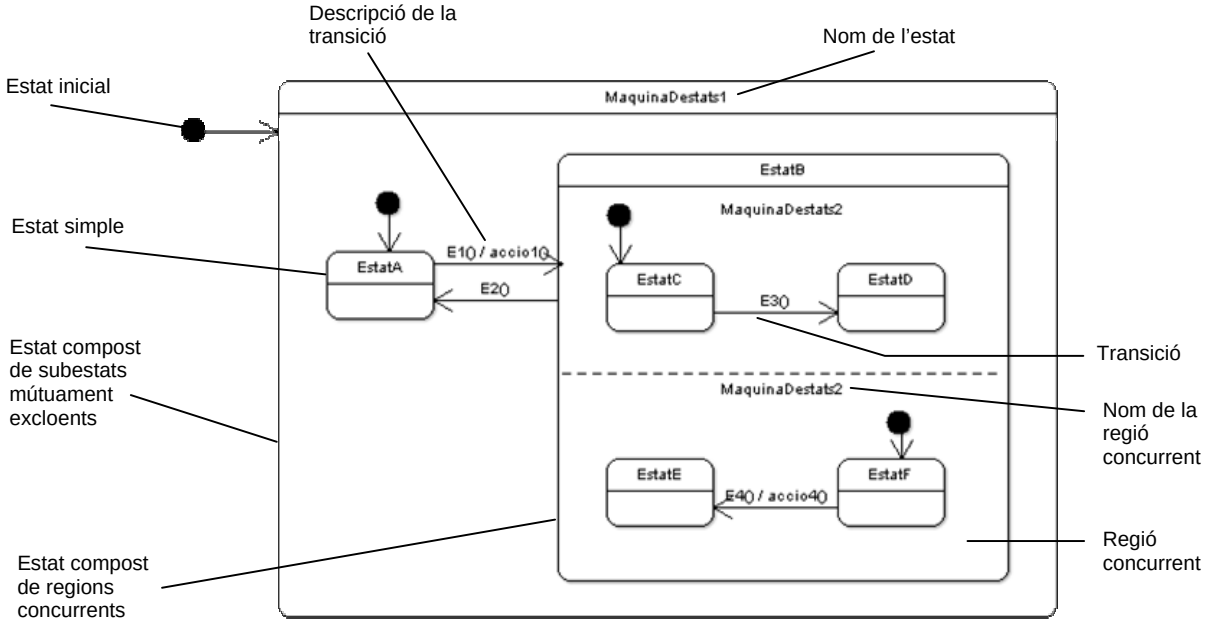


Figura 2. Exemple de diagrama de maquina d'estats.

2.1.2. Estats.

Un estat és una condició o situació que durant la vida d'un objecte satisfà alguna condició, fa alguna activitat o espera algun esdeveniment.
En general, un estat es representa com un rectangle arrodonit i que pot arribar a tenir dos compartiments.

- El compartiment del nom. En aquest compartiment s'hi mostra el nom de l'estat.
- El compartiment de les reaccions d'estat. En aquest compartiment hi pot haver una llista de les reaccions d'estat de l'estat.
- El compartiment de subestat. En aquest s'hi poden trobar dos zones ben diferenciades: la de les reaccions d'estat on s'hi pot trobar una llista de les reaccions d'estat de l'estat i una altre on hi poden haver-hi subestats. Aquestes dues zones són opcionals, que hi siguin o no, depèn de com es composa l'estat.

Els estats es poden subdividir en estats i pseudoestats.
Com estats tenim l'estat simple, l'estat compost i la regió.
Com pseudoestats tenim l'estat inicial, l'estat final, l'estat historia poc profunda i l'estat historia profunda. S'anomenen pseudoestats pel fet de que no poden tenir totes les característiques dels estats, per exemple l'estat inicial mai pot tenir una transició d'entrada.
En la Figura 3 es poden observar els diferents compartiments d'un estat.

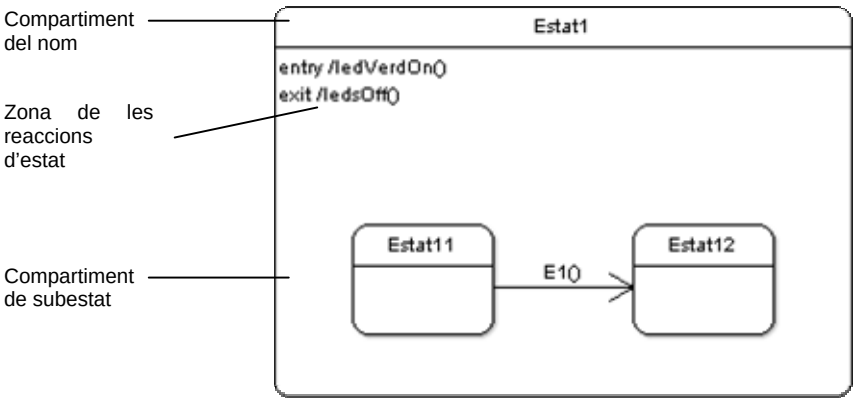


Figura 3. Exemple dels compartiments d'un estat.

2.1.2.1. Estat simple.

És l'estat de més baix nivell en la jerarquia d'estats i no pot tenir cap altre estat o regió, però si que pot tenir reaccions d'estat.

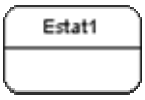


Figura 4. Exemple d'estat simple.

2.1.2.2. Estat compost.

És l'estat que està composat per altres estats, d'una manera concurrent mitjançant les regions i d'una manera mútuament excloent mitjançant combinacions d'estats simples i/o estats compostos.

Un estat compost pot tenir diferents regions concurrents, cadascuna de elles representen maquines d'estats i estaran actives sempre que l'estat compost que les conté estigui actiu. Les regions concurrents estan separades per una línia discontinua.

En la Figura 5 i en la Figura 6 és pot observar un estat compost concurrent i un estat compost mútuament excloent respectivament.

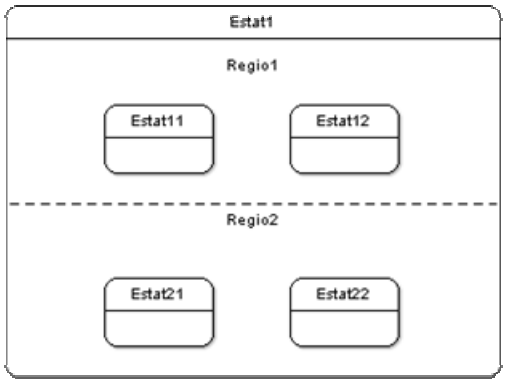


Figura 5. Exemple d'estat compost concurrent.

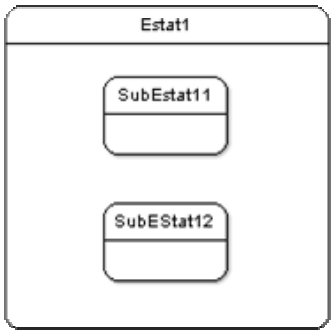


Figura 6. Exemple d'estat compost mútuament excloent.

2.1.2.3. Regió.

Les regions són usades per representar maquines d'estat, forment part d'un estat compost i s'usen com a regions concurrents en un estat superior o en un estat compost.
En l'editor argoUML les regions estan referides com a "concurrent region".
En la
Figura 5 es pot veure un exemple de regions concurrents.

2.1.2.4. Estat inicial.

És la manera d'indicar el començament, es a dir, l'estat que primer s'executarà en una maquina d'estats. Només té una transició que s'executa sempre i per tant no té cap disparador que la condicioni.

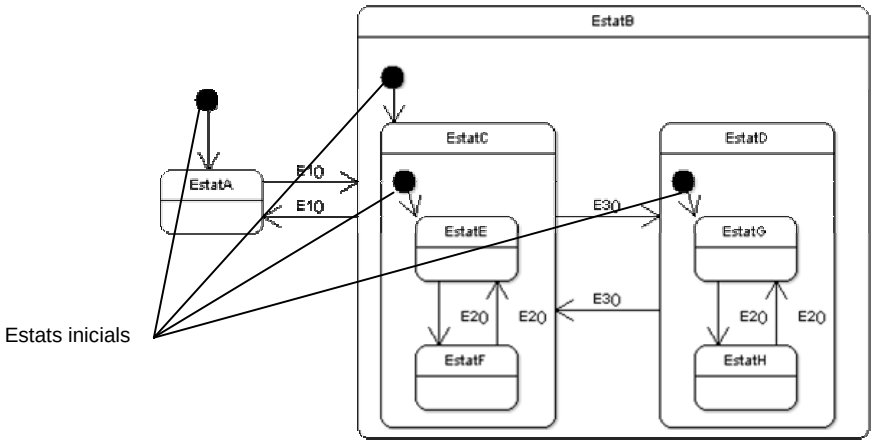


Figura 7. Exemple d'estat inicial.

En l'exemple de la Figura 7 l'estat de defecte és l'estat 1 doncs és el senyalat per l'estat inicial del diagrama. Això significa que l'automat es començara a executar per aquest estat. Quan l'esdeveniment E1 arribi, la màquina d'estats passara a l'estat B que a l'hora començara per l'estat C (que és el senyalat per l'estat inicial). Aquest estat C té com estat inicial l'E.

2.1.2.5. Estat historial superficial.

Substitueix l'estat inicial quan a més d'indicar l'estat inicial es vol indicar que la màquina d'estats s'ha de recordar de l'estat on estava cada cop que es reactivi. Quan la maquina d'estats es reactiva se'n recordarà de l'estat que estava actiu en la darrera desactivació i entrarà directament a aquest estat sense passar per l'estat d'inici.

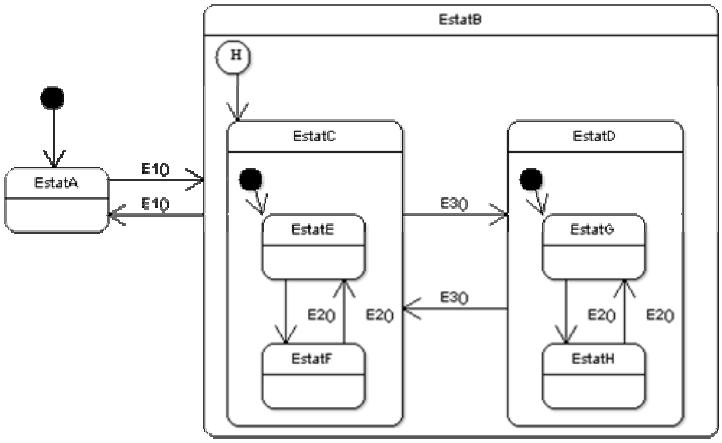
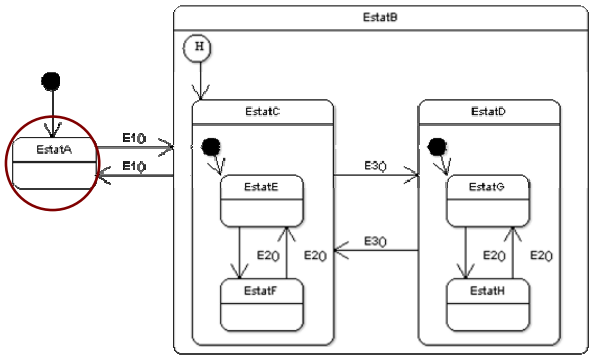


Figura 8. Exemple de l'ús de l'estat historial superficial.

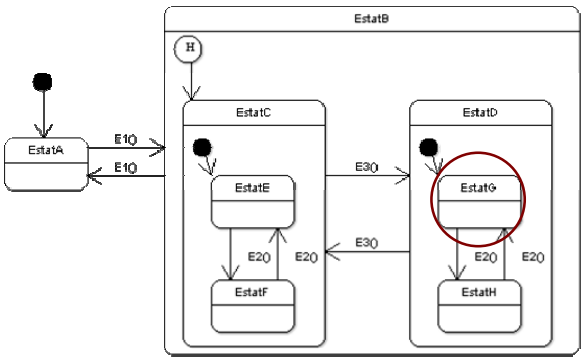
La Figura 8 mostra el mateix diagrama que la Figura 6 però amb una petita diferència a nivell de dibuix i que es que l'estat B conté un estat historial superficial i no un estat inicial. El primer cop que s'entra a l'estat B el resultat és el mateix que en la Figura 7.

La sèrie de diagrames de la Figura 9 són el resultat de tenir la següent seqüència d'esdeveniments després del reset inicial. E1(), E3(), E2(), E1(), E1().

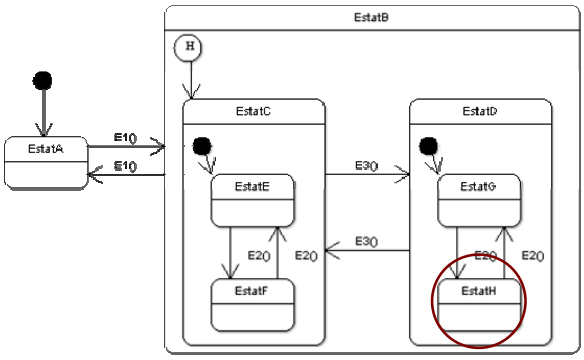
Reset de la màquina d'estats ---> La màquina d'estat es troba a l'estat A



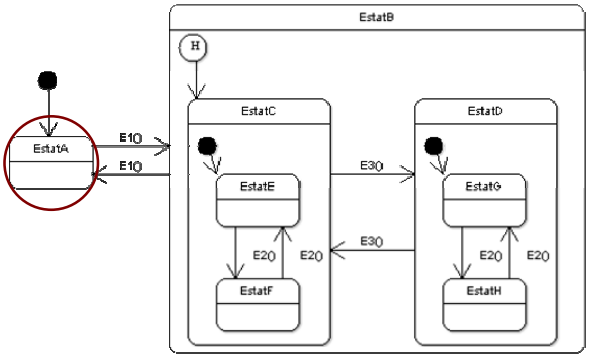
Esdeveniment E3 és capturat ---> La màquina d'estat entra a l'estat D que té com a defecte l'estat G.



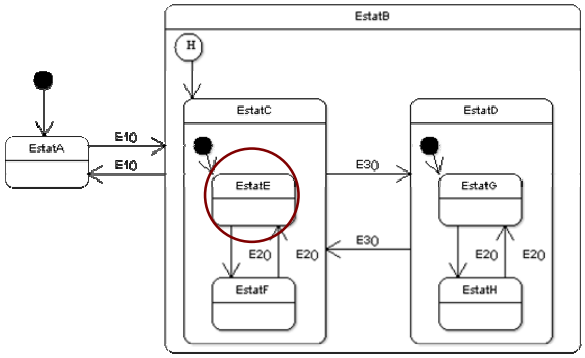
Esdeveniment E2 és capturat ---> La màquina d'estat passa de l'estat G a l'estat H.



Esdeveniment E1 és capturat ---> La màquina d'estat passa de l'estat A.



Esdeveniment E1 és capturat ---> La màquina d'estat entra a l'estat B que té com a defecte l'estat C. L'estat C representa una màquina d'estats amb l'estat de defecte l'E.



Esdeveniment E1 és capturat ---> La màquina d'estat entra a l'estat B, però coma aquest té un estat historial superficial entrarà a l'estat D.

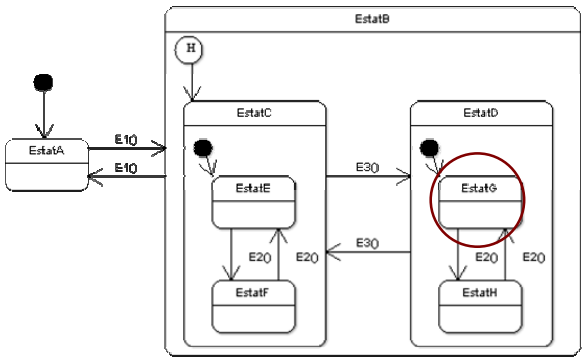


Figura 9. Efecte d'usar un estat historial superficial.

La diferència entre l'ús d'un estat inicial i un estat historial superficial es pot veure des del punt de vista de que quan la màquina d'estats torna a entrar a l'estat B, l'estat inicial és l'estat D i no el C. Això és perquè l'estat B té l'estat històric que recorda l'estat que estava quan va ser abandonat.

2.1.2.6. Estat historial profund.

El principi de funcionament és el mateix que el d'un estat historial superficial, però amb la diferència que el profund força que totes les submàquines d'estats que estan per sota han de tenir memòria històrica. És una manera de propagar el historial a totes les submàquines a partir d'un estat determinat.

Un exemple de l'ús d'un estat historial profund és el de la Figura 10. Aquest és el mateix exemple que el de un estat historial superficial de la Figura 8, però en aquest diagrama aquest estat s'ha canviat per un estat historial profund.

Assumint la mateixa seqüència d'esdeveniments, és adir E1(), E3(), E2(), E1(), E1(). L'efecte serà que després de haver reentrat a l'estat B, la màquina d'estats recordarà l'últim estat que estava en tota la jerarquia d'estats. Això vol dir que en el nostre cas, l'estat de la reentrada seria l'H.

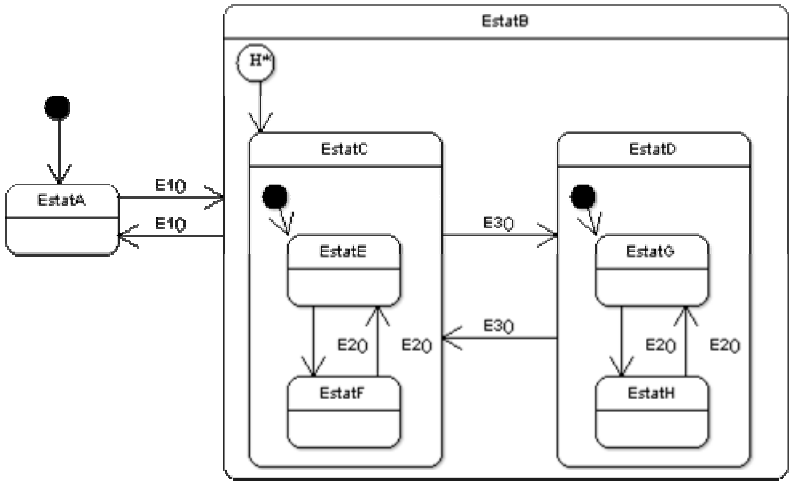


Figura 10. Exemple de l'ús d'un estat historial profund.

2.1.2.7. Estat final.

Estat per indicar el final d'una maquina d'estat. Aquest estat no té transicions de sortida i un cop s'hi ha entrat no s'entorna a sortir.

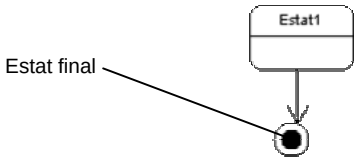
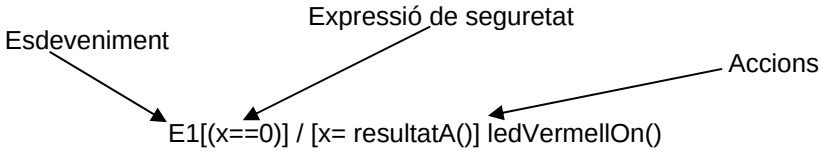


Figura 11. Exemple d'estat final

2.1.3. Transicions.

La transició és la relació entre dos estats, indicant quin és l'estat origen, on surt la fletxa i quin és l'estat destí, on arriba la fletxa quan és produeix un determinat esdeveniment.

La descripció està formada per dos parts, la de les condicions i la de les accions, separades per una barra inclinada '/'.



Aquesta transició és podria llegir com:

if
l'esdeveniment E1 ha ocorregut i la variable x és igual a 0
aleshores
s'assigna a la variable x el valor de retorn de la funció resultatA()
executa la funció ledVermellOn()
endif

Figura 12. Exemple de transició.

2.1.4. Disparadors.

Els disparadors és poden classificar en dos grups:

- Disparadors èxplicits:
  - Esdeveniment "Event". Un esdeveniment és alguna cosa que passa en l'entorn extern a la màquina d'estat.
  - Un esdeveniment és considerat com una entrada momentània (com pot ser una tecla premuda, un final de carrera activat, etc), que ha de ser capturat i guardat abans de que pugui ser valorat per la màquina d'estats.
- Disparadors implícits:
  - Aquests queden definits com a reaccions d'estats i són la de l'entrada "Entry", la de la sortida "Exit" i la de fer mentre s'està en l'estat "Do".

2.1.5. Expressions de seguretat.

Es troba en les condicions d'una transició i perquè aquesta és dispari totes les expressions de seguretat s'han de complir (ser veritat). Aquestes fan referència a variables globals del sistema.

2.1.6. Reaccions d'estat.

Les reaccions d'estat són com les transicions, però amb la diferència de que no comporten un canvi d'estat. Les reaccions poden ser activades en els següents casos:

- Quan s'entra en un estat. "Reacció d'entrada".
- Mentre s'està en un estat. "Reacció fer".
- Quan s'abandona un estat. "Reacció de sortida".
- Per un esdeveniment, però sense abandonar l'estat. "Reacció interna".

2.1.6.1. Reacció fer.

És una transició que es disparada per un esdeveniment però no s'abandona l'estat, es a dir no té estat destí. Estan composades de la mateixa manera que les transaccions ordinàries. En aquest cas, al no implicar un abandonament de l'estat no s'executen les reaccions d'entrada i de sortida.

2.1.6.2. Reacció d'entrada.

És l'acció que serà executada cada cop que s'entri a l'estat. S'identifica amb la paraula "entry" i no cal cap més condició per ser executada.

2.1.6.3. Reacció de sortida.

És l'acció que serà executada cada vegada que es surt de l'estat. S'identifica amb la paraula "exit" i no cal cap més condició per ser executada.

2.2. Elecció de l'aplicació gràfica de disseny de programari.

Després d'una cerca més o menys exhaustiva per Internet i d'haver valorat diferents eines decidim que l'aplicació gràfica de disseny que emprarem serà l'ArgoUML 0.24 que té com a característiques més destacades:

- Complir amb l'estàndard UML. D'aquesta manera es podran representar les màquines d'estats a traves dels diagrames d'estat d'aquest estàndard.
- L'opció de poder exportar els diagrames en un format XMI. D'aquesta manera es podrà analitzar el diagrama d'estats per poder-ne generar el codi.
- Independent del sistema operatiu.
- Codi obert. Encara que hem decidit que l'aplicació no sigui un plug-in de l'eina de disseny gràfic en aquest projecte, és una possibilitat que en el futur és pot contemplar.
- Gratuït.

2.3. Implementació d'una maquina d'estats.

A l'hora d'implementar una màquina d'estats hi han dos qüestions per resoldre: com representar els estats i com interaccionaran aquets amb l'entorn.

Hi ha diferents tècniques per representar una màquina d'estats, una de les més comuns és l'encadenament d'estaments "switch case". Aquesta tècnica però, presenta l'inconvenient de que si el diagrama d'estats creix l'encadenament d'estaments "switch case" s'esdevé illegible, doncs totes les pàgines del fitxer font són molt semblants.

L'altre tècnica és l'ús de taules. Aquesta tècnica consisteix en la creació de tres taules:

- Una taula unidimensional amb cada una de les funcions que ha de gestionar els esdeveniments de cada estat, és a dir, l'índex de la taula és l'estat actual.
- Una taula bidimensional amb cada una de les accions que ha de fer cada estat i per cada esdeveniment, és a dir: un índex, el de les files, és l'estat actual i l'altre, el de les columnes, és l'esdeveniment actual.
- Una taula bidimensional amb el proper estat segons l'estat actual i l'esdeveniment que s'ha tractat. També amb l'estat actual com un índex, el de les files i l'esdeveniment actual com l'altre índex el de les columnes.

Per omplir aquestes taules es tracta de contemplar per cada estat:

- Una funció pel tractament dels esdeveniments que provocaran el disparo corresponent. Cada una d'aquestes funcions seran un element de la taula d'esdeveniments.
- Una funció per executar les accions que s'indiquin per cada esdeveniment produït. Cada una d'aquestes funcions seran un element de la taula d'accions.

2.3.1. Taules per implementar els estats.

Hem decidit implementar la tècnica de les taules que, a part de tenir el codi molt més clar, ens permet tractar millor les reaccions d'estat i els pseudoestats inicial, final, historial poc profund i historial profund.

Aquestes taules tindran les dimensions fixes i arrodonides al màxim nombre de transicions per estat, es a dir si un estat té una transició i un altre dos transicions la taula d'accions i la de propers estats d'aquesta màquina tindrà dos columnes en tots els seus estats. A més, al nombre màxim de transicions per estat, s'afegirà un esdeveniment més per representar la situació de que no s'hagi produït cap esdeveniment contemplat en l'estat actual. Resumint si un diagrama té 5 estats i l'estat amb més esdeveniments en té 2 la taula unidimensional per a controlar els esdeveniments tindrà dimensió 5 i les taules per les accions i els propers estats seran de 5 files per 3 columnes.

Prenem com exemple per implementar, el diagrama de maquina d'estats representat per la Figura 13, i ens concentrarem en l'estat "ledBlau".

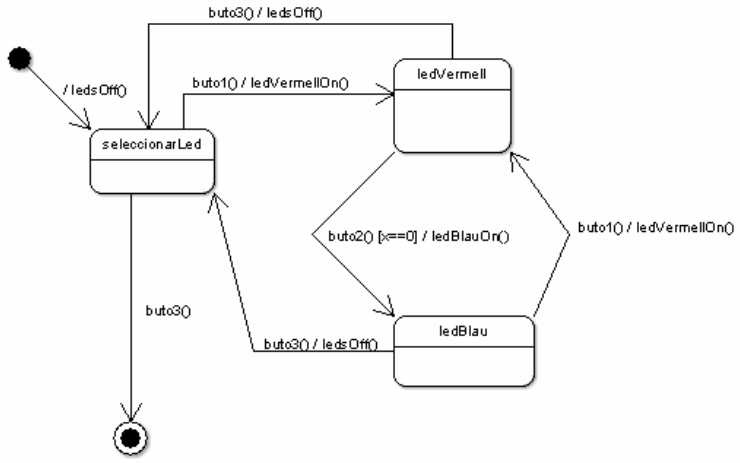


Figura 13. Exemple de maquina d'estats per implementar en codi C.

L'estat "ledBlau" té dos transaccions.
Una desencadenada per l'esdeveniment "buto1()" que executarà l'acció "ledVermellOn" i la màquina d'estats passarà a l'estat "ledVermell".
Una altra desencadenada per l'esdeveniment "buto3()" que executarà l'acció "ledsOff()" i la màquina d'estats passarà a l'estat "seleccionarLed".
El nom de la màquina d'estats és "me_cntrlLed" i el nom del contenidor per defecte és "controlLlums".
Per tant, el codi per aquest estat haurà de ser el següent, tal com es pot observar en la Figura 14:

- Una funció "getEvent_ledBlau". Per esbrinar si algun dels possibles esdeveniments que poden provocar alguna reacció en l'estat "ledBlau" s'han produït. D'aquesta funció també observem que la màquina d'estat treballarà amb una estructura de dades com a pas de paràmetres i que un dels camps d'aquesta estructura, "event", ens indicarà l'esdeveniment que s'ha produït, aquest camp serà del tipus enter "int". Els paràmetres de retorn de la funció són constants, generades de manera automàtica i seran els índexs que ens indicaran l'acció a executar segons l'esdeveniment produït, com podem observar a part dels dos que surten del diagrama n'hi ha un tercer per indicar que no s'ha produït cap d'esdeveniment.Una funció "acció_ledBlau_BUTO1" per executar les accions quan l'esdeveniment "BUTO1" s'hagi produït, en aquest cas la funció "ledVermellOn". En la funció hi ha tres comentaris que ens indiquen les accions que s'hi poden arribar a executar, el "effect" que ja l'hem comentat, el "entry" on hi hauran les accions que s'hagin de fer en l'entrada de l'estat, les especificades en la reacció d'estat "entry" i el "exit" on hi hauran les accions que s'hagin de fer al abandonar l'estat anterior. Les especificades en la reacció d'estat "exit", en aquest cas de l'estat "ledVermell". La funció també té l'estructura de dades com a paràmetre d'entrada.
- Una funció "accio_ledBlau_BUTO3" per executar les accions quan l'esdeveniment "BUTO3" s'hagi produït, en aquest cas la funció "ledsOff".

```
//Estat: ledBlau
int getEvent_ledBlau(struct tagDcntrlLed *d_cntrlLed)
{
    //Esdeveniments
    if(d_cntrlLed->event==EVENT_butol)
        return DISP_0;
    if(d_cntrlLed->event==EVENT_buto3)
        return DISP_1;
    return DISP_2;
}

void accio_ledBlauD0(struct tagDcntrlLed *d_cntrlLed)
{
    //Exit ledBlau

    //Effect
    ledVermellOn();

    //Entry ledVermell
}

void accio_ledBlauD1(struct tagDcntrlLed *d_cntrlLed)
{
    //Exit ledBlau

    //Effect
    ledsOff();

    //Entry seleccionarLed
}
```

Figura 14. Exemple del codi generat per l'estat "ledBlau".

Les taules pel diagrama de la Figura 13 s'ompliran de la següent manera:

- La taula "tGetNouEvent_controlLlums" amb les funcions per gestionar els esdeveniments per cadascun dels estats del diagrama.
- La taula "tEstats_controlLlums" amb les funcions de les accions per cada estat i per cada esdeveniment. Els esdeveniments que no s'usen en un estat o no tenen accions declarades fan servir la funció "accioSxExNull", que és la funció sense cap acció per defecte.
- La taula "tProperEstat_controlLlums" amb els índexs dels propers estats per cada estat i per cada esdeveniment. Els esdeveniments que no es fan servir en un estat tenen per defecte l'índex del mateix estat.

En la Figura 15 hi han les tres taules generades a partir del diagrama d'exemple de la Figura 13:

```
int (*const tGetNouEvent_controlLlums []) (void *) = {
    getEvent_ledVermell, getEvent_ledBlau, getEvent_iniciControlLed,
    getEvent_seleccionarLed, getEvent_final
};
void (*const tEstats_controlLlums [][TOTAL_EVENTS_cntrlLed]) (void *) = {
    {accio_ledVermellD0, accio_ledVermellD1, accioSxExNull},
    {accio_ledBlauD0, accio_ledBlauD1, accioSxExNull},
    {accio_iniciControlLedD0, accioSxExNull, accioSxExNull},
    {accio_seleccionarLedD0, accio_seleccionarLedD1, accioSxExNull},
    {accioSxExNull, accioSxExNull, accioSxExNull}
};
int const tProperEstat_controlLlums [][TOTAL_EVENTS_cntrlLed] = {
    {ESTAT_ledBlau, ESTAT_seleccionarLed, ESTAT_ledVermell},
    {ESTAT_ledVermell, ESTAT_seleccionarLed, ESTAT_ledBlau},
    {ESTAT_seleccionarLed, ESTAT_iniciControlLed, ESTAT_iniciControlLed},
    {ESTAT_final, ESTAT_ledVermell, ESTAT_seleccionarLed},
    {ESTAT_final, ESTAT_final, ESTAT_final}
};
```

Figura 15. Exemple de les taules per representar un màquina d'estats.

Com es pot deduir del codi de la Figura 14 i de la Figura 15, també és generarà un fitxer capçalera, d'extensió ".h" i per cada màquina d'estats, on hi estaran declarades totes les constants i l'estructura de dades que és necessiten per la implementació de la màquina d'estats.
En l'estructura de dades a més de la variable "event", que ja s'ha comentat, és pot observar la variable "estat_controlLlums" que és l'índex de l'estat actual.

```
//Dimensió de les taules de la màquina d'estas
#define TOTAL_EVENTS_cntrlLed   3

//Posició dels disparadors en les taules
//Valors retornats per les funcions "getEvents"
#define DISP_0  0
#define DISP_1  1
#define DISP_2  2

//Posició dels estats en les taules
#define ESTAT_ledVermell    0
#define ESTAT_ledBlau   1
#define ESTAT_iniciControlLed   2
#define ESTAT_seleccionarLed    3
#define ESTAT_final 4

struct tagDcntrlLed {
    int event;
    int estat_controlLlums;
};
```

Figura 16. Exemple de les declaracions per una màquina d'estats.

Finalment ens caldrà una funció per realitzar en un sol pas les tres accions en que s'ha dividit un cicle de la màquina d'estats. L'anomenarem "automata" i tindrà com paràmetres d'entrada el punter a l'estructura de dades, el punter a la taula de les funcions per tractar els esdeveniments, el punter a la taula de les accions a fer segons l'esdeveniment tractat, el punter a la taula del proper estat segons l'esdeveniment tractat i el punter a la variable de l'estat actual.
Aquesta funció, igual que la funció d'acció nul·la "accioSxExNull", serà comú per totes les màquines d'estats i per tant les posarem en un fitxer a part que s'anomenarà "automata.c". Aquestes funcions és poden veure en la Figura 17.

```
//
void accioSxExNull(void *x)
{
}
//
void automata(void *params,
              int (*const taulaGetNouEvent[])(void*),
              void (*const taulaAccions[](void*),
              int const taulaProperEstat[],
              int *estat)
{
    int event;

    event = taulaGetNouEvent [*estat] (params);
    taulaAccions [event] (params);
    *estat = event = taulaProperEstat [event];
}
```

Figura 17. Funcions del fitxer "automata.c".

El fitxer "automata.c" serà un fitxer comú a totes les màquines d'estat i per tant, contemplarem la possibilitat de crear-lo una sola vegada i afegir-lo a les llibreries de desenvolupament del sistema encastat.

2.3.2. Interacció amb l'entorn.

Aquesta altra qüestió es la encarregada de facilitar la comunicació de la màquina d'estats generada de manera automàtica amb els esdeveniments, les variables usades en les expressions de seguretat i amb l'execució de les accions.

La manipulació dels esdeveniments i les accions la fan els controladors de dispositius (device drivers), per exemple transformant el senyal físic produït al prémer un botó en un esdeveniment contemplat per la màquina d'estats, o convertint una crida de funció en un senyal elèctric per encendre un led (light emitter diode) diode emissor de llum o accionar un motor. Aquests controladors de dispositius seran implementats pel desenvolupador del sistema encastat.
Per poder atendre tots els esdeveniments que es poden produir en el sistema encastat, s'implementarà una cua d'esdeveniments. Bàsicament és tractarà de una llista FIFO (First Input First Output) primer d'entrar primer de sortir, que de passada ens servirà com interfície d'entrada per què l'entorn pugui notificar els esdeveniments que es produeixin. Aquesta cua constarà de tres funcions i una estructura de dades:

- Funció "initCuaEvent", per inicialitzar la cua. Tindrà com a paràmetre d'entrada el punter de l'estructura de dades de la cua.
- Funció "getEvent", per capturar el primer esdeveniment que ha entrat en la cua. Tindrà com a paràmetre d'entrada el punter de l'estructura de dades de la cua i retornarà el valor del primer esdeveniment que hagi entrat en la cua.
- Funció "addEvent", per afegir un esdeveniment a la cua. Tindrà com a paràmetres d'entrada l'esdeveniment que serà del tipus enter i el punter de l'estructura de dades de la cua.

El fitxer que contindrà les funcions de manegament de la cua d'esdeveniments rebrà el nom de "cuaEvents.c" i aquest és un clar exemple de fitxer que és pot voler crear una sola vegada, per incorporar-lo a les llibreries de desenvolupament del sistema encastat i per tant també només es crearà amb l'opció seleccionada.
Els fitxers "automata.c" i "cuaEvents.c" seran els fitxers de les APIs (Aplication Programming Interface) Interfície de programació d'aplicacions de la màquina d'estats implementada i la seva creació serà opcional.

```
#define LLARGADA_EVENTS     15

#define CAP_EVENT           0
#define CUA_EVENT_PLENA     -1

struct tCuaEvents {
    //Contenidor dels esdeveniments que es van produint.
    int events[LLARGADA_EVENTS];
    //Apuntador d'entrada.
    int in;
    //Apuntador de sortida.
    int out;
};

//Fitxer: cuaEvents.c
int addEvent(int, struct tCuaEvents *);
int getEvent(struct tCuaEvents *);
void initCuaEvent(struct tCuaEvents *);
```

Figura 18. Declaració de les funcions de la cua d'esdeveniments.

Per accedir a les variables usades en les expressions de seguretat, caldrà que l'entorn proveeixi dels accessos per aquestes, bàsicament "gets" i "sets". Per exemple, si la variable és del tipus enter "int" i el seu nom és "x", l'aplicació hi accedirà cridant la funció "getX" per obtenir el seu valor i "setX" per assignar-li un valor determinat.
Les accions escrites pel desenvolupador del sistema encastat seran cridades pel autòmat, tal com estiguin escrites dins del diagrama de la maquina d'estats.

2.3.3. Fitxers de codi d'una màquina d'estats.

Combinant els fitxers generats de manera automàtica i descrits en els apartats anteriors, amb els que el desenvolupador ha escrit manualment per tal de tenir el codi per controlar els dispositius. Així tindrem tota l'aplicació completa i compilant i enllaçant tot el codi es podrà obtenir un fitxer amb el "firmware" del sistema encastat.

En la Figura 19 es pot veure l'esquema de una estructura de fitxers d'una aplicació completa.

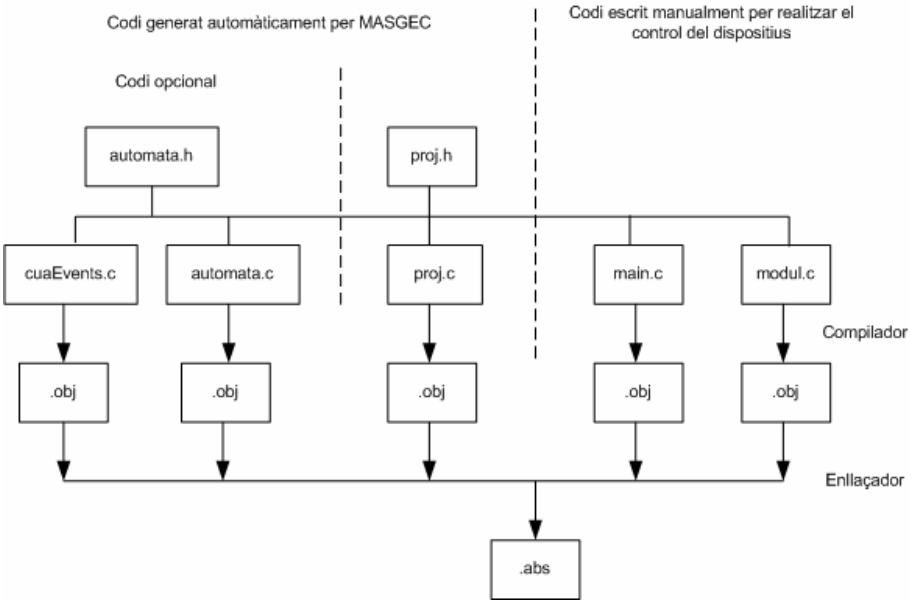


Figura 19. Estructura dels fitxers de codi per una màquina d'estats.

2.4. Interpretació del fitxer XMI generat per l'ArgoUML0.24.

Per poder interpretar el contingut d'un document XMI generat per l'ArgoUML0.24 hem dibuixat diferents models, per tal de veure com els diferents components d'una màquina d'estats intervenien. D'aquesta manera disposarem de tots els elements que volem modelar en la nostra aplicació i que més tard implementarem en codi ANSI C.

2.4.1. Elements del fitxer XMI.

El document XMI generat per l'ArgoUML0.24, com tot document XML ben format, a més alt nivell té dues parts, la capçalera i la instància de document.

En la capçalera hi apareix informació de la codificació del document, en aquest cas UTF-8.

La instància de document representa el contingut real del document i en aquest cas està constituïda pels elements XMI.header i XMI.content:

XMI.header. És el contenidor de la capçalera amb els següents elements:

- **XMI.documentation.** Conté una sèrie d'elements per identificar l'eina que s'utilitza per generar el document i la versió de l'aplicació.
- **XMI.metamodel.** Té com atributs el nom i la versió del model emprat.

XMI.content. És el contenidor del document del model UML.

- **UML:model.** Model del diagrama dibuixat. Amb els atributs:
  - xmi.id. Identificador de l'element.
  - name. Nom del model.
  - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
  - isRoot. Per indicar si l'element és l'arrel. Només pot tenir els valors "true" o "false".
  - isLeaf. Per indicar si l'element és una fulla. Només pot tenir els valors "true" o "false".

    - isAbstract. Per indicar si l'element és abstracte. Només pot tenir els valors "true" o "false".
- **UML:Namespace.ownedElement.** És l'element de UML:model i fa de contenidor dels elements: UML:StateMachine, UML:CallEvent i UML:Comment.
- **UML:StateMachine.** Contenidor dels elements que formen una màquina d'estats: UML:StateMachine.top i UML:StateMachine.transitions.
  Amb els següents atributs:
    - xmi.id. Identificador de l'element.
    - name. Nom de la màquina d'estats.
    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
- **UML:StateMachine.top.** Contenidor dels estats que formen part de la màquina d'estats: UML:CompositeState, UML:SimpleState i UML: Pseudostate.
- **UML:CompositeState.** Element per representar un estat compost o una regió.
  Pot tenir com elements: UML:StateVertex.outgoing, UML:StateVertex.incoming, UML:State.entry, UML:State.exit, UML:State.internalTransition i UML:CompositeState.subvertex.
  Amb els atributs:
    - xmi.id. Identificador de l'element.
    - name. Nom de l'estat compost o regió.
    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
    - isConcurrent. Per indicar si l'element és concurrent. Només pot tenir els valors de "true" o "false".
- **UML:SimpleState.** Element per representar un estat simple. Pot tenir com elements: UML:StateVertex.outgoing, UML:StateVertex.incoming, UML:State.entry, UML:State.exit i UML:State.internalTransition.
  Amb els atributs:
    - xmi.id. Identificador de l'element.
    - name. Nom de l'estat.
    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
- **UML:Pseudostate.** Element per representar un pseudoestat. Només pot tenir com element: UML:StateVertex.outgoing.
  Amb els atributs:
    - xmi.id. Identificador de l'element.
    - name. Nom del pseudo-estat.
    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
    - kind. Per indicar el tipus de pseudoestat que només pot ser estat inicial, estat amb historia poc profunda i estat amb historia profunda. Només pot tenir els valors de "initial", "shallowHistory" i "deepHistory".
- **UML:FinalStateUML.** Element per representar un estat final. Només pot tenir com element : UML:StateVertex.incoming.
  Amb els atributs:
    - xmi.id. Identificador de l'element.
    - name. Nom del pseudoestat.
    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
- **UML:StateMachine.transitions.** Contenidor de les transicions que formen part de la màquina d'estats. Conté elements del tipus UML:Transition.

- **UML:Transition.** Element per representar una transició definida en la màquina d'estats. Pot tenir com elements: UML:Transition.effect, UML:Transition.trigger, UML:Transition.source i UML:Transition.target.
  Amb els atributs:
  - xmi.id. Identificador de l'element.
  - name. Nom de la transició.
  - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
- **UML:Transition.effect.** Element per representar l'acció que s'ha de fer en la transició. És el contenidor de l'element UML:CallAction.
  Amb els atributs:
  - xmi.id. Identificador de l'element.
  - name. Nom de l'acció.
  - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
- **UML:CallAction.** Element contenidor d'una acció durant una transició. Conté l'element UML:Action.script.
  Amb els atributs:
  - xmi.id. Identificador de l'element.
  - name. Nom del contenidor de l'acció.
  - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
  - isAsynchronous. Per indicar la sincronització. Només pot tenir els valors de "true" o "false".
- **UML:Action.script.** Contenidor del que s'ha de fer com acció d'una transició. Conté l'element UML:ActionExpression.
- **UML:ActionExpression.** Element definidor de l'expressió de l'acció d'una transició.
  Amb els atributs:
  - xmi.id. Identificador de l'element.
  - lenguage. Llenguatge de programació en que s'ha escrit el cos de l'acció.
  - body. Cos de l'acció a realitzar.
- **UML:Transition.trigger.** Element que identifica l'esdeveniment que provoca la transició, és el disparador. Conté l'element UML:CallEvent que és la referència a l'element que representa l'esdeveniment.
- **UML:CallEvent.** Element contingut en l'element UML:Transition.trigger per fer referència a l'esdeveniment disparador de la transició.
  Té l'atribut:
  - xmi.idref. Referència al identificador de l'esdeveniment.
- **UML:Transition.source.** Element contenidor de la referència a l'element que és l'origen de la transició i segons l'element aquest podrà ser UML:Pseudostate, UML:SimpleState i UML:CompositeState. Tindran com atribut
  - xmi.idref. Referència al identificador de l'element origen, més concretament a l'atribut xmi.id d'aquest.
- **UML:Transition.target.** Element contenidor de la referència de l'element que és el destí la transició i segons l'element aquest podrà ser UML:Pseudostate, UML:SimpleState i UML:CompositeState. Tindran com atribut:
  - xmi.idref. Referència al identificador de l'element destí, més concretament a l'atribut xmi.id d'aquest.
- **UML:CallEvent.** Element per representar un esdeveniment definit en la màquina d'estats.
  Amb els atributs:
  - xmi.id. Identificador de l'element.
  - name. Nom de l'esdeveniment.

    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
- **UML:Comment.** Element per representar un comentari en el diagrama. Pot contenir l'element UML:Comment.annotatedElement.
  Amb els atributs:
    - xmi.id. Identificador de l'element.
    - name. Nom del comentari.
    - isSpecification. Per indicar si l'element és una especificació. Només pot tenir els valors de "true" o "false".
    - body. Cos del comentari. Cada línia acaba amb el caràcter "line feed" (ascii #10) seguit pel caràcter punt i coma (ascii #59).
- **UML:Comment.annotatedElement.** Contenidor de l'element que està associat al comentari. L'element que conté pot ser UML:SimpleState o . UML:CompositeState. Tindran com atribut:
    - xmi.idref. Referència al identificador de l'element associat a l comentari, més concretament a l'atribut xmi.id d'aquest.

2.4.2. Exemple de un fitxer XMI generat per l'ArgoUML0.24.

La màquina d'estats dibuixada és la representada en la Figura 20. Per no tenir un fitxer XMI molt extens hem realitzat un diagrama de màquina d'estats molt elemental:

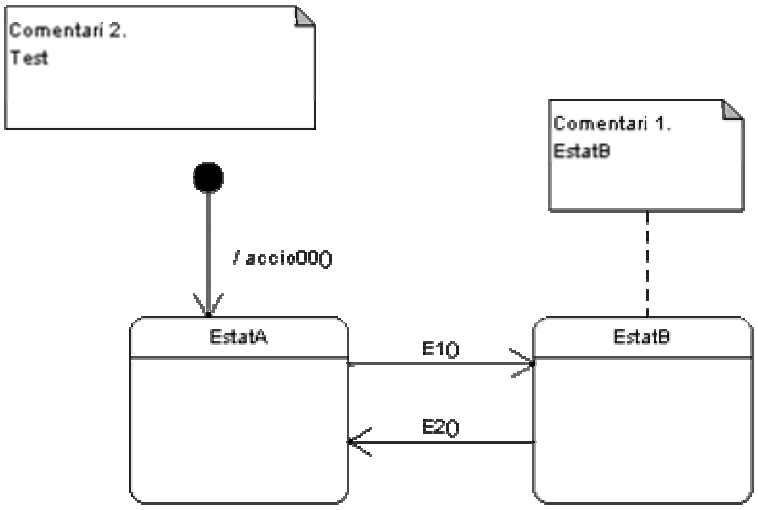


Figura 20. Màquina d'estats per poder interpretar el fitxer XMI.

L'ArgoUML0.24 ens mostra l'arbre del model.

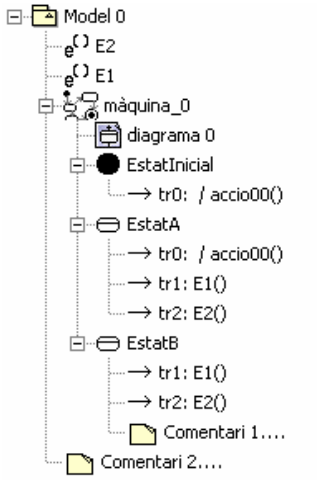


Figura 21. Arbre XMI.

I finalment el contingut del fitxer XMI, tal com es pot veure a la Figura 23, que l'obtenim de l'opció "File/Export XMI" des del menú principal de l'aplicació argoUML0.24.

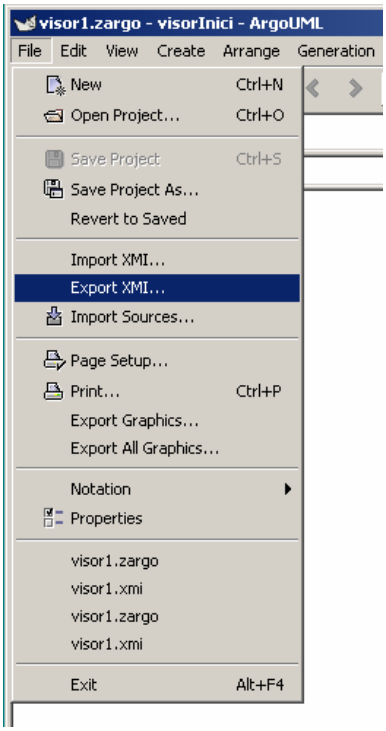


Figura 22. Opció d'exportació a un fitxer en format XMI.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Fri Dec 28 18:28:03 CET 2007'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>ArgoUML (using Netbeans XMI Writer version 1.0)</XMI.exporter>
      <XMI.exporterVersion>0.24(5) revised on $Date: 2006-11-06 19:55:22 +0100 (Mon, 06 Nov 2006) $ </XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML" xmi.version="1.4"/></XMI.header>
  <XMI.content>
    <UML:Model xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:0000000000000868'
      name = 'Model 0' isSpecification = 'false' isRoot = 'false' isLeaf = 'false'
      isAbstract = 'false'>
      <UML:Namespace.ownedElement>
        <UML:StateMachine xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:0000000000000869'
          name = 'màquina_0' isSpecification = 'false'>
          <UML:StateMachine.top>
            <UML:CompositeState xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086A'
              name = 'contenidor principal' isSpecification = 'false' isConcurrent = 'false'>
              <UML:CompositeState.subvertex>
                <UML:Pseudostate xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086B'
                  name = 'EstatInicial' isSpecification = 'false' kind = 'initial'>
                  <UML:StateVertex.outgoing>
                    <UML:Transition xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086D'/>
                  </UML:StateVertex.outgoing>
                </UML:Pseudostate>
                <UML:SimpleState xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086C'
                  name = 'EstatA' isSpecification = 'false'>
                  <UML:StateVertex.outgoing>
                    <UML:Transition xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088F'/>
                  </UML:StateVertex.outgoing>
                  <UML:StateVertex.incoming>
                    <UML:Transition xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086D'/>
                    <UML:Transition xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:0000000000000890'/>
                  </UML:StateVertex.incoming>
                </UML:SimpleState>
                <UML:SimpleState xmi.id = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088E'
                  name = 'EstatB' isSpecification = 'false'>
                  <UML:ModelElement.comment>
                    <UML:Comment xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:0000000000000892'/>
                  </UML:ModelElement.comment>
                  <UML:StateVertex.outgoing>
                    <UML:Transition xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:0000000000000890'/>
                  </UML:StateVertex.outgoing>
                  <UML:StateVertex.incoming>
                    <UML:Transition xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088F'/>
                  </UML:StateVertex.incoming>
                </UML:SimpleState>
              </UML:CompositeState.subvertex>
            </UML:CompositeState>
          </UML:StateMachine.top>
          <UML:StateMachine.transitions>
            <UML:Transition xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086D'
              name = 'tr0' isSpecification = 'false'>
              <UML:Transition.effect>
                <UML:CallAction xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000088A'
                  name = 'a0' isSpecification = 'false' isAsynchronous = 'false'>
                  <UML:Action.script>
                    <UML:ActionExpression xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:0000000000000895'
                      language = '' body = 'accio00()'/>
                  </UML:Action.script>
                </UML:CallAction>
              </UML:Transition.effect>
              <UML:Transition.source>
                <UML:Pseudostate xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086B'/>
              </UML:Transition.source>
              <UML:Transition.target>
                <UML:SimpleState xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086C'/>
              </UML:Transition.target>
            </UML:Transition>
            <UML:Transition xmi.id = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088F'
              name = 'tr1' isSpecification = 'false'>
              <UML:Transition.trigger>
                <UML:CallEvent xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:0000000000000880'/>
              </UML:Transition.trigger>
              <UML:Transition.source>
                <UML:SimpleState xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086C'/>
              </UML:Transition.source>
              <UML:Transition.target>
                <UML:SimpleState xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088E'/>
              </UML:Transition.target>
            </UML:Transition>
            <UML:Transition xmi.id = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:0000000000000890'
              name = 'tr2' isSpecification = 'false'>
              <UML:Transition.trigger>
                <UML:CallEvent xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000087F'/>
              </UML:Transition.trigger>
              <UML:Transition.source>
                <UML:SimpleState xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088E'/>
              </UML:Transition.source>
              <UML:Transition.target>
                <UML:SimpleState xmi.idref = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000086C'/>
              </UML:Transition.target>
            </UML:Transition>
          </UML:StateMachine.transitions>
        </UML:StateMachine>
        <UML:CallEvent xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:000000000000087F'
          name = 'E2' isSpecification = 'false'/>
        <UML:CallEvent xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:0000000000000880'
          name = 'E1' isSpecification = 'false'/>
        <UML:Comment xmi.id = '-64--88-1-10--26ef299a:1171ffle7fb:-8000:00000000000008BE'
          name = 'ComentariGeneral' isSpecification = 'false' body = 'Comentari 2.&#10;Test'/>
        <UML:Comment xmi.id = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:0000000000000892'
          name = 'ComentariEstatB' isSpecification = 'false' body = 'Comentari 1.&#10;EstatB'>
          <UML:Comment.annotatedElement>
            <UML:SimpleState xmi.idref = '-64--88-1-10--4ba79a12:1171215b80b5:-8000:000000000000088E'/>
          </UML:Comment.annotatedElement>
        </UML:Comment>
      </UML:Namespace.ownedElement>
    </UML:Model>
  </XMI.content>
</XMI>
```

Figura 23. Contingut d'un fitxer XMI

2.5. Definició de l'aplicació.

Amb les tasques prèvies hem obtingut prou informació per la definició de l'aplicació i una primera decisió és fer-la modular. D'aquesta manera facilitarem possibles millores i ampliacions futures. Per tant la dividim en quatre subsistemes suficientment independents:

- Un subsistema de màquina d'estats que ha de poder representar qualsevol diagrama de màquina d'estats.
- Un subsistema d'entrada per transformar el fitxer XMI a un model de màquina d'estats, com aquest fitxer és un document XML utilitzarem un analitzador DOM Document object model (model d'objectes d'un document), per tal de poder accedir ràpidament i tantes vegades com es vulgui a la informació dels elements del document XML..
- Un subsistema de sortida per transformar el model de màquina d'estats a codi ANSI C.
- Un subsistema per generar els fitxers de les API.

L'aplicació no tindrà interfície gràfica amb l'usuari, només atendrà el pas de paràmetres en línia. Els paràmetres reconeguts seran:

- -f <arg> Fitxer d'entrada del model en format XMI. Aquest és obligatori, en cas de no ser-hi l'aplicació donarà informació de les comandes esperades i pararà la seva execució.
- -v L'aplicació va indicant els passos que es realitzen. Aquest és opcional, per defecte no donarà cap informació dels passos realitzats.
- -a Per generar les llibreries associades a la màquina d'estats. Aquest és opcional, per defecte no és generaran les APIs.
- -o Directori destí dels fonts creats. Aquest és opcional, en cas de no ser-hi, el directori destí serà el mateix que el del fitxer d'entrada.

Quan es faci l'execució de l'aplicació s'haurà de passar-li el nom del fitxer XMI.

L'aplicació té una sèrie d'opcions que es passaran en la línia de comandes quan s'hagi d'executar.

Com la única interacció entre l'usuari i el sistema és l'activació de l'aplicació no hem cregut oportú reflectir-lo com tal.

Hem identificat com classes "analyzer", "cgenerator", "apiautomata", "document", "maquinaestats" (objectes del domini) i "documentbuiderfactory" i "documentbuilder" com objectes del món exterior.

El model del domini és aquest:

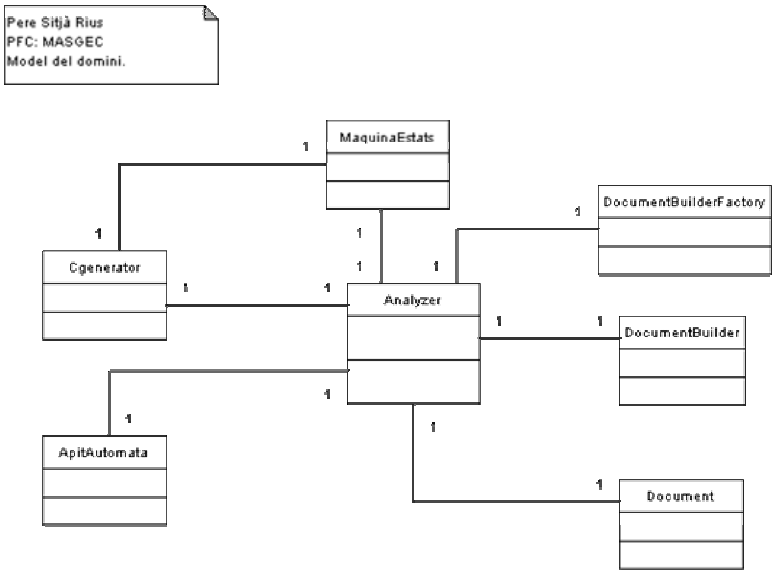


Figura 24. Model del domini.

3. Disseny.

3.1. Paquets de l'aplicació.

Descomponem el programari en tres paquets:
El paquet "analyzer". On anirà la classe "analyzer".
El paquet "generator". On anirà la classe "Cgenerator" i "ApiAutomata".
El paquet "maquinaEstats". On anirà les classes "MaquinaEstats", "Estat", "EstatCompost", "EstatFinal", "EstatiInicial", "HistoriaPocProfunda", "HistoriaPeofunda", "Comantari", "Transicio", "Event", "Accio" i "Guard".
Aquests tres paquets quedaran englobats en un únic paquet que rebrà el nom de l'aplicació "MASGEC".

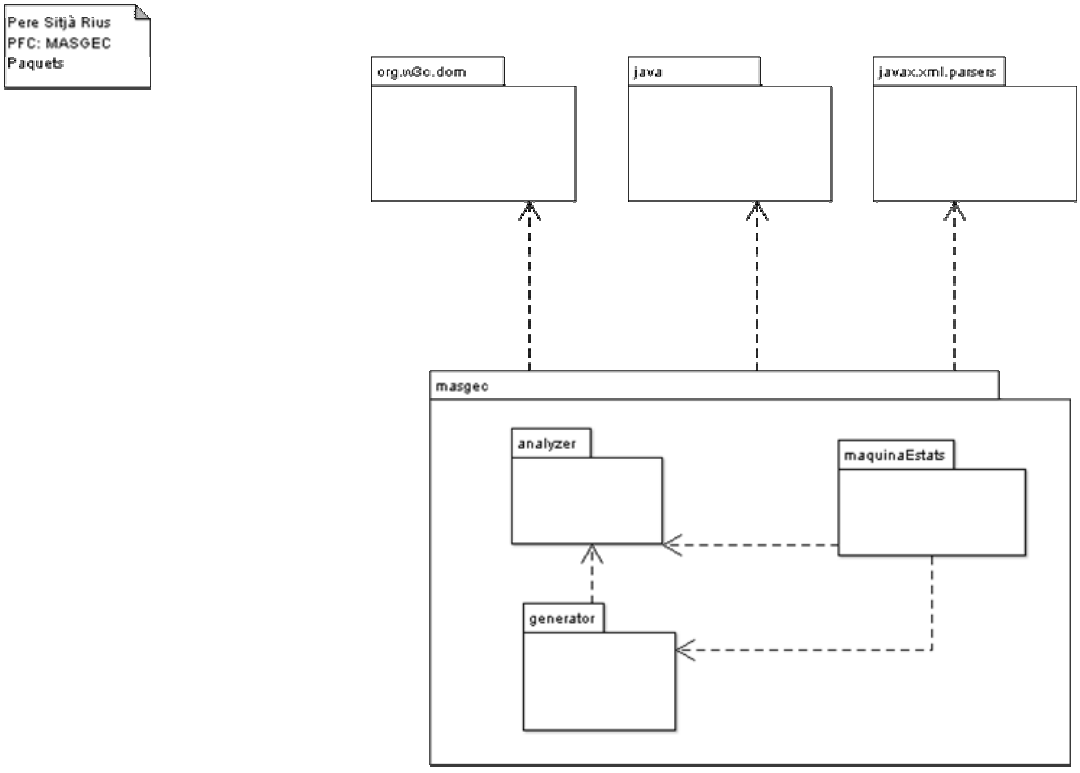


Figura 25. Diagrama de paquets de MASGEC.

3.2. Disseny del model de la màquina d'estats.

A partir de les especificacions de l'anàlisi i de l'estructura del fitxer XMI obtenim un model de màquina d'estats, per poder representar els diagrames de màquines d'estat, amb les següents classes:

- **Classe "Event".** Per modelar un esdeveniment. Tindrà com atributs el "nom" i l'"id" que serà el seu codi identificador. Tots dos estan declarats en el fitxer XMI d'entrada i seran del tipus "string".
- **Classe "Guard".** Per modelar una expressió de seguretat. Tindrà com atributs el "nom", l'"id" i l'"expressio" que contindrà l'expressió de seguretat. Tots tres estan declarats en el fitxer XMI i seran dels tipus "string".
- **Classe "Accio".** Per modelar una acció. Tindrà com atributs el "nom", l'"id" i el "cos" que contindrà l'expressió de l'acció a realitzar. Tots tres estan declarats en el fitxer XMI i serna del tipus "string".
- **Classe "Transicio".** Per modelar una transició. Tindrà com atributs el "nom" i l'"id". Tindrà referències a les classes "Event", "Guard" i "Accio".

- **Classe “Estat”.** Per modelar un estat. Tindrà com atributs el “nom” i l’”id”. Tindrà referències a les classes “Transicio” , “Accio” i “Comentari”.
- **Classe “EstatCompost”.** És una subclasse d’estat i és per modelar un contenidor d’estats. Tindrà un atribut “isConcurrent” de tipus booleà i servirà per indicar si l’estat compost és concurrent. Tindrà referències a les classes “Estat”, “EstatCompost” i “Comentari”.
- **Classe “EstatFinal”.** És una subclasse d’estat i és per modelar un estat final que només ha de tenir una transició d’origen. Tindrà una referència a la classe “Transicio”.
- **Classe “PseudoEstat”.** És una superclasse abstracta de totes les classes que només poden tenir una transició de destí. Tindrà una referència a la classe “Transició”.
- **Classe “HistoriaProfunda”.** És una subclasse de Pseudoestat i és per modelar un estat de historia profunda.
- **Classe “HistoriaPocProfunda”.** És una subclasse de Pseudoestat i és per modelar un estat de historia poc profunda.
- **Classe “EstatInicial”.** És una subclasse de Pseudoestat i és per modelar un estat inicial.
- **Classe “Comentari”.** És per modelar un comentari en el diagrama i només pot estar relacionat amb un estat o amb la màquina d’estats. Tindrà com atributs el “nom” l’”id” i “cos” que contindrà les línies del comentari. Tots tres atributs seran del tipus “string”. Tindrà referències a les classes “Estat” i “MaquinaEstats”.
- **Classe “MaquinaEstats”.** Per fer de contenidor d’una màquina d’estats a partir d’un estat compost, tal com ens demostra la descripció d’un fitxer XMI. Tindrà com atributs el “nom” i l’”id”. Tindrà referències a les classes “EstatCompost” i “Comentari”.

Les relacions entre aquestes classes és poden veure en el diagrama de la Figura 26.

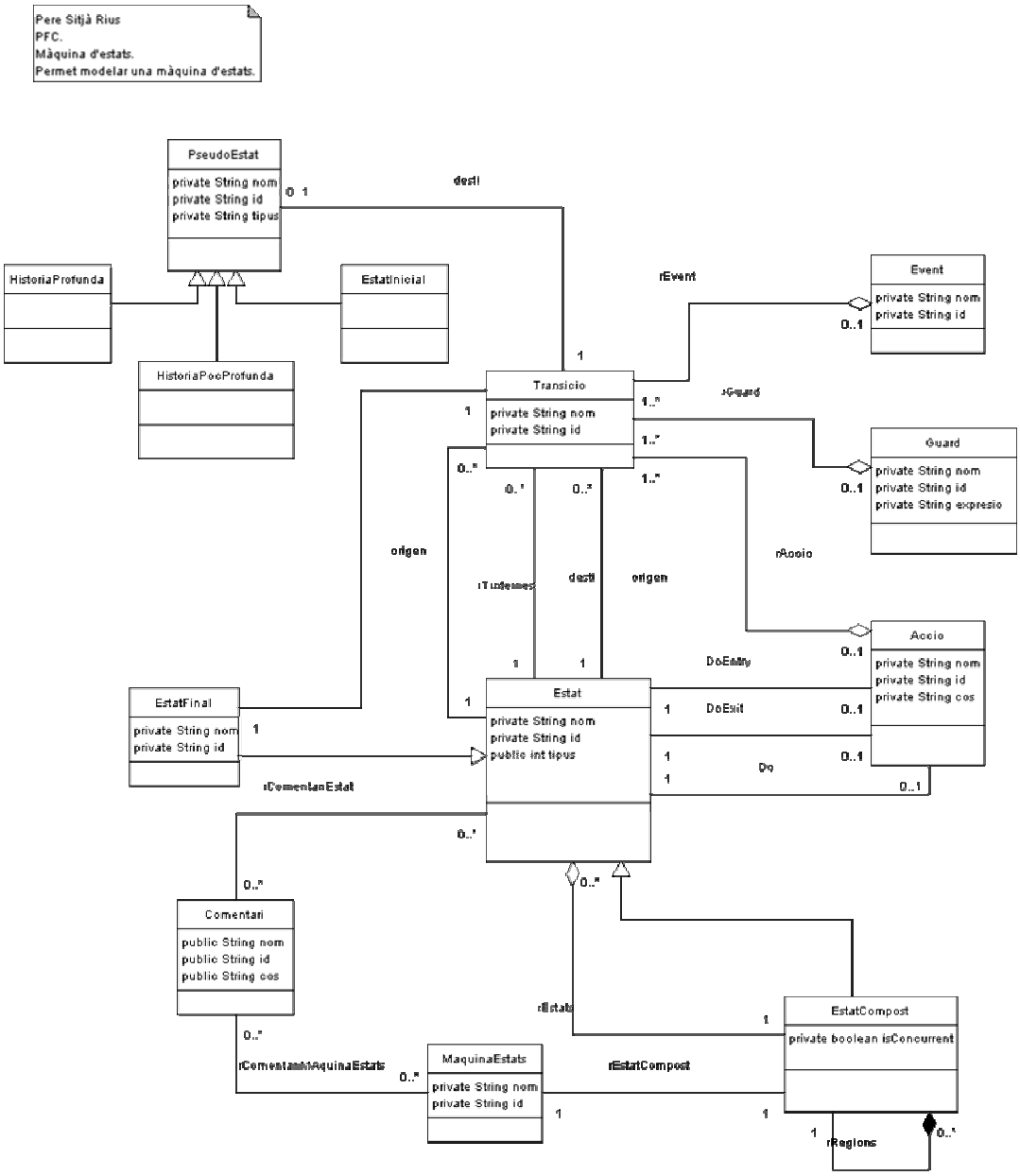


Figura 26. Diagrama de classes del model de la màquina d'estats.

3.3. Disseny del subsistema analitzador.

És la part de l'aplicació que controlarà tot el procés, les seves principals tasques seran:

- Crear un document DOM a partir del fitxer XML d'entrada. Amb les classes "DocumentBuilderFactory" i "DocumentBuilder".
- Generar el model de màquina d'estat a partir d'un document DOM.
- Generar les llibreries APIs amb la classe "ApiAutomata".
- Generar el codi en ANSI C a partir del model de màquina d'estats amb la classe "Cgenerator".

Les classes "DocumentBuilderFactory" i "DocumentBuilder" formen part del paquet "javax.xml.parsers".

La classe principal d'aquest subsistema és "Analyzer".

Figura 27. Diagrama de la classe del subsistema analitzador.

3.4. Disseny del subsistema generador de codi.

És la part encarregada de generar el codi en ANSI C amb les següents classes:

- Classe "Cgenerator". És l'encarregada de transformar el model de màquina d'estats a codi.
- Classe "ApiAutomata". És l'encarregada de generar les llibreries API que donen suport al codi generat.

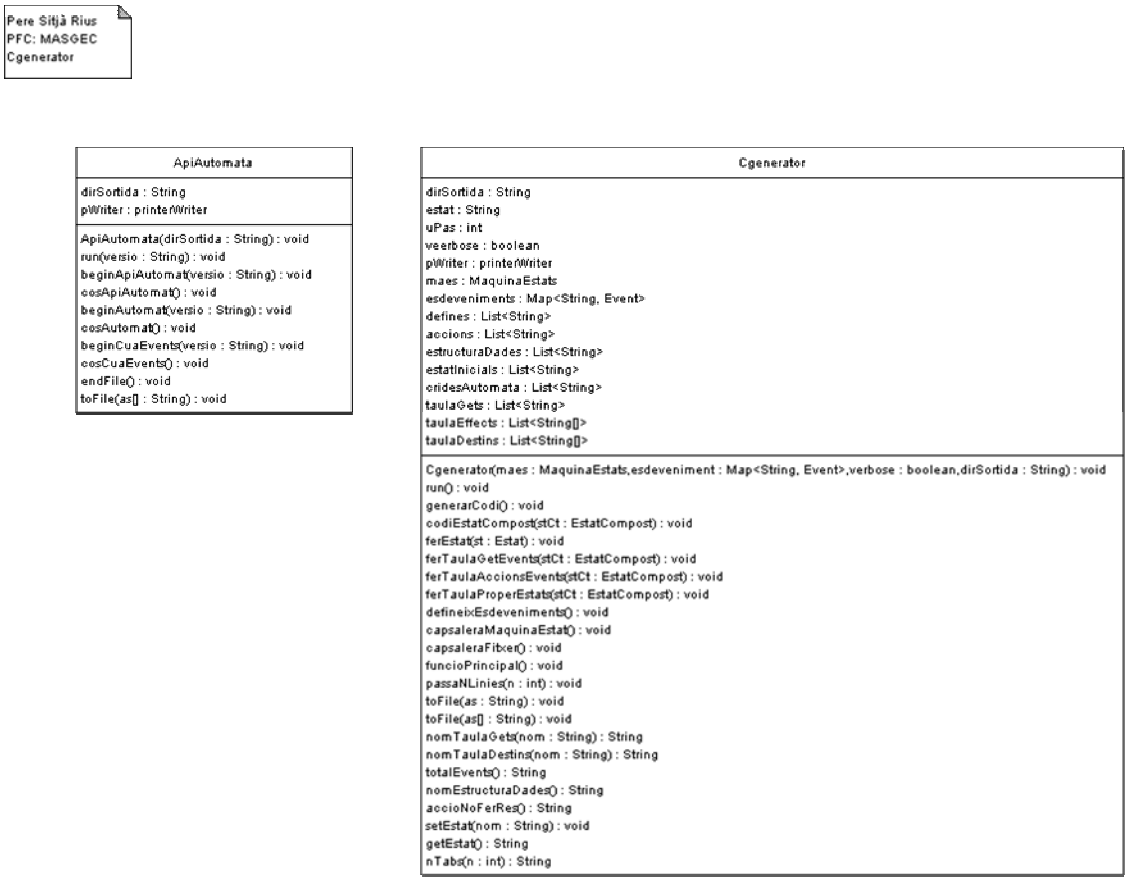


Figura 28. Diagrama de les classes del subsistema generador de codi.

4. Implementació.

En aquesta etapa hem realitzat la codificació els subsistemes detallats en l'etapa de disseny. Només remarquem les consideracions més importants que hem tingut en compte a l'hora de realitzar la codificació. Com documentació d'aquesta etapa ens remetríem al document generat de manera automàtica per la utilitat "javadoc".

4.1. Implementació del model de màquina d'estats.

La implementació del model de màquina d'estats s'ha fet seguint el diagrama del mateix model en la fase de disseny, però amb la següent consideració a l'hora de codificar les classes del model:

- La classe "PseudoEstat" pot ser una especialització de la classe "Estat", igual com ho és la classe "EstatFinal". Per tant les dos podrien ser la mateixa. Per fer més fàcil l'implementació del model hem fet que la mateixa classe "Estat" en funció de la variable "tipus" s'identifiqués per cada una de les possibles especialitzacions. Aquesta és una part que ja bé diferenciada en el contingut del fitxer XMI.

A les totes classes d'aquest subsistema hi hem posat el mètode "toString" per facilitar el llistat. A la classe "Estat" hi hem posat el mètode "equals" per facilitar la comparació. També hem contemplat totes els mètodes necessaris per garantir la cerca d'un objecte en concret dins del model, com poden ser: "cercarEstat", "cercarEstatCompost".

4.2. Implementació del subsistema analitzador.

La implementació del subsistema analitzador ha consistit en la codificació de la classe "analyzer".

Cal remarcar que en aquesta classe hi ha el mètode estàtic "main" que serveix per realitzar l'entrada des del sistema operatiu corresponent.

4.3. Implementació del subsistema generador de codi.

L'implementació del sistema ha consistit en la codificació de les classes "Cgenerator" i l"ApiAutomata" del subsistema generador de codi.

5. Proves.

5.1. Exemple de màquina d'estats amb estat compost.

En aquest exemple volem provar la generació de codi amb un estat compost amb les reaccions d'estat declarades i un subestat format per b1 i b2. La Figura 29 mostra el diagrama tal com ha estat dibuixat en el argoUML0.24.

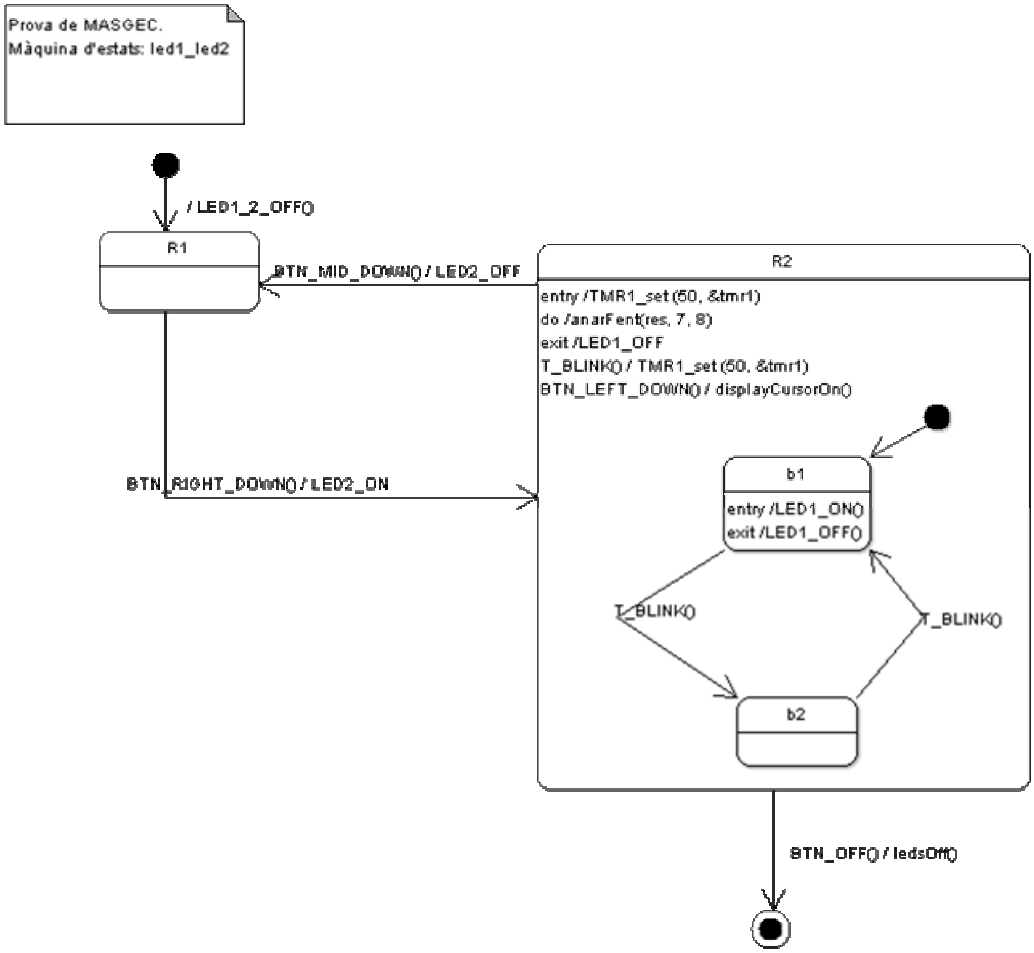


Figura 29. Exemple de màquina d'estats amb estat compost.

Per fer més seguidor el codi generat i poder comentar les parts d'aquest hem dividit el codi generat en diferents figures.

En la Figura 30 s'hi pot veure el codi generat per l'estat final que en aquest exemple s'ha anomenat "feinaFeta". Aquest només té l'esdeveniment per defecte i cap acció a fer, doncs al tractar-se d'un estat final no es sortirà del mateix fins un que es provoqui un "reset" al sistema. També s'hi pot veure el codi de l'estat "R1" que té la funció "R1D0" per realitzar l'acció "LED2_ON" durant la transició, la inicialització de la variable estat del subsistema "b1b2" i finalment es pot observar l'acció d'entrada "TMR1_set" de l'estat "R2".

```
/*
 * Codi generat automaticament per MASGEC V0.1
 *
 * Fitxer d'origen     :    E:\Pere\UOC\wsp_pfc\fonts_c\rlr2\rlr2.xmi
 * Màquina d'estats    :    ledl_led2
 * Data                :    Wed Dec 12 10:49:54 CET 2007
 * Total d'estats      :    8
 * Total de transicions:    5
 **/

#include    "ledl_led2.h"

//rlr2
//Estat: feinaFeta 3
int getEvent_feinaFeta(struct tagDrlr2 *d_rlr2)
{
    //Esdeveniments
    return DISP_0;
}


//Estat: R1 1
int getEvent_R1(struct tagDrlr2 *d_rlr2)
{
    //Esdeveniments
    if(d_rlr2->event==EVENT_BTN_RIGHT_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_R1D0(struct tagDrlr2 *d_rlr2)
{
    //Effect
    LED2_ON();
    d_rlr2->estat_R2= ESTAT_iniciblb2;
    //Entry R2
    TMR1_set(50, &tmr1);
}
```

Figura 30. Exemple estat compost. Estats "feina_feta" i "R1".

En la Figura 31 tenim el codi generat per estat inicial "iniciR1R" i té una funció a fer de manera incondicional "inici_R1R2D0" amb l'acció "LED1_2_OFF".

```
//EstatInicial: iniciR1R2 0
int getEvent_iniciR1R2(struct tagDrlr2 *d_rlr2)
{
    //Esdeveniments
    return DISP_0;
}

void accio_iniciR1R2D0(struct tagDrlr2 *d_rlr2)
{
    //Effect
    LED1_2_OFF();
}
```

Figura 31. Exemple estat compost. Estat "iniciR1R2".

En la Figura 32 tenim el codi generat per l'estat "R2", es pot observar en la funció "getEvent_R2" el tractament dels esdeveniments que ha d'atendre l'estat a més de la reacció d'estat "Do" (Fer) "anaFent". També és pot observar la reacció d'estat "Exit" (Sortir) en les funcions de les transicions que abandonen l'estat "R2".

```
//Estat Compost: R2 2
int getEvent_R2(struct tagDrlr2 *d_rlr2)
{
    //Do
    anarFent(res, 7, 8);
    //Esdeveniments
    if(d_rlr2->event==EVENT_BTN_MID_DOWN)
        return DISP_0;
    if(d_rlr2->event==EVENT_T_BLINK)
        return DISP_1;
    if(d_rlr2->event==EVENT_BTN_OFF)
        return DISP_2;
    if(d_rlr2->event==EVENT_BTN_MID_DOWN)
        return DISP_3;
    return DISP_4;
}

void accio_R2D0(struct tagDrlr2 *d_rlr2)
{
    //Exit
    LED1_OFF();
    //Effect
    LED2_OFF();
}

void accio_R2D1(struct tagDrlr2 *d_rlr2)
{
    //Effect
    TMR1_set (50, &tmr1);
}

void accio_R2D2(struct tagDrlr2 *d_rlr2)
{
    //Exit
    LED1_OFF();
    //Effect
    ledsOff();
}

void accio_R2D3(struct tagDrlr2 *d_rlr2)
{
    //Effect
    displayCursorOn();
}
```

Figura 32. Exemple estat compost. Estat "R2".

En la Figura 33 es poden observar les taules pels estats "iniciR1R2", "R1", "R2" i "feinaFeta". Només destacar que les funcions que no tenen accions s'han completat amb la funció "accioSxExNull".

```
int (*const tGetNouEvent_rlr2 []) (void *) = {
    getEvent_feinaFeta, getEvent_R1, getEvent_iniciR1R2, getEvent_R2
};
void (*const tEstats_rlr2 [][TOTAL_EVENTS_rlr2]) (void *) = {
    {accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull},
    {accio_R1D0, accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull},
    {accio_iniciR1R2D0, accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull},
    {accio_R2D0, accio_R2D1, accio_R2D2, accio_R2D3, accioSxExNull}
};
int const tProperEstat_rlr2 [][TOTAL_EVENTS_rlr2] = {
    {ESTAT_feinaFeta, ESTAT_feinaFeta, ESTAT_feinaFeta, ESTAT_feinaFeta,
    ESTAT_feinaFeta},
    {ESTAT_R2, ESTAT_R1, ESTAT_R1, ESTAT_R1, ESTAT_R1},
    {ESTAT_R1, ESTAT_iniciR1R2, ESTAT_iniciR1R2, ESTAT_iniciR1R2, ESTAT_iniciR1R2},
    {ESTAT_R1, ESTAT_R2, ESTAT_feinaFeta, ESTAT_R2, ESTAT_R2}
};
```

Figura 33. Exemple estat compost. Taules dels estats "R1R2".

En la Figura 34 es pot observar les funcions dels estats “inicib1b2”, “b1” i “b2” que formen part de l’estat compost “R2”.

```
//R2
//Estat: b2 1
int getEvent_b2(struct tagDrlr2 *d_rlr2)
{
    //Esdeveniments
    if(d_rlr2->event==EVENT_T_BLINK)
        return DISP_0;
    return DISP_1;
}

void accio_b2D0(struct tagDrlr2 *d_rlr2)
{
    //Entry b1
    LED1_ON();
}


//Estat: b1 0
int getEvent_b1(struct tagDrlr2 *d_rlr2)
{
    //Esdeveniments
    if(d_rlr2->event==EVENT_T_BLINK)
        return DISP_0;
    return DISP_1;
}

void accio_b1D0(struct tagDrlr2 *d_rlr2)
{
    //Exit
    LED1_OFF();
}

//EstatInicial: iniciblb2 2
int getEvent_iniciblb2(struct tagDrlr2 *d_rlr2)
{
    //Esdeveniments
    return DISP_0;
}

void accio_iniciblb2D0(struct tagDrlr2 *d_rlr2)
{
    //Entry b1
    LED1_ON();
}
```

Figura 34. Exemple estat compost. Estats “b1”, “b2” i “inicib1b2”.

En la Figura 35 es poden observar les taules pels estats “inicib1b2”, “b1”, “b2”.

```
int (*const tGetNouEvent_R2 []) (void *) = {
    getEvent_b2, getEvent_b1, getEvent_iniciblb2
};
void (*const tEstats_R2 [][TOTAL_EVENTS_rlr2]) (void *) = {
    {accio_b2D0, accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull},
    {accio_b1D0, accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull},
    {accio_iniciblb2D0, accioSxExNull, accioSxExNull, accioSxExNull, accioSxExNull}
};
int const tProperEstat_R2 [][TOTAL_EVENTS_rlr2] = {
    {ESTAT_b1, ESTAT_b2, ESTAT_b2, ESTAT_b2, ESTAT_b2},
    {ESTAT_b2, ESTAT_b1, ESTAT_b1, ESTAT_b1, ESTAT_b1},
    {ESTAT_b1, ESTAT_iniciblb2, ESTAT_iniciblb2, ESTAT_iniciblb2, ESTAT_iniciblb2}
};
```

Figura 35. Exemple estat compost. Taules dels estats “R2”.

En la Figura 36 es pot observar la funció principal del diagrama de màquina d'estats, on hi ha la inicialització de la cua d'esdeveniments "iniciCuaEvents" i la inicialització de la variable estat del sistema "r1r2".
Dins del bucle "for" hi ha l'accés al primer esdeveniment amb la funció "getEvent", l'execució

```
void rlr2(void)
{
    struct tCuaEvents cuaEvents;
    struct tagDrlr2 d_rlr2;

    initCuaEvent(&cuaEvents);

    d_rlr2.estat_rlr2= ESTAT_iniciR1R2;
    for(;;){

        d_rlr2.event= getEvent(&cuaEvents);
        //Crides a l'automata
        automata(&d_rlr2,
                tGetNouEvent_rlr2,
                tEstats_rlr2[d_rlr2.estat_rlr2],
                tProperEstat_rlr2[d_rlr2.estat_rlr2],
                &d_rlr2.estat_rlr2);
        if(d_rlr2.estat_R2==ESTAT_R2){
            automata(&d_rlr2,
                    tGetNouEvent_R2,
                    tEstats_R2[d_rlr2.estat_R2],
                    tProperEstat_R2[d_rlr2.estat_R2],
                    &d_rlr2.estat_R2);
        }
    }
}
```

Figura 36. Exemple estat compost. Funció principal.

En la Figura 37 hi trobem el fitxer de capçalera on es poden observar la declaració de les constants que l'aplicació ha utilitzat per la implementació així com la declaració de l'estructura de dades de la màquina d'estats amb les variables:

- "event". Per poder disposar del valor de l'esdeveniment a tractar en tots els subpassos d'un cicle.
- "estat_r1r2". Indica l'estat actual del sistema "R1R2".
- "estat_R2". Indica l'estat actual del sistema "R2" que està format pels estats "inicib1b2", "b1" i "b2".

```
/*
 * Codi generat automaticament per MASGEC V0.1
 *
 * Fitxer d'origen     :    E:\Pere\UOC\wsp_pfc\Diagrames\xmi\rlr2.xmi
 * Màquina d'estats    :    ledl_led2
 * Data                :    Wed Dec 12 21:59:07 CET 2007
 * Total d'estats      :    8
 * Total de transicions:    5
 **/

//Esdeveniments retornats per la cua d'esdeveniments
#define EVENT_BTN_OFF   1
#define EVENT_T_BLINK   2
#define EVENT_BTN_RIGHT_DOWN    3
#define EVENT_BTN_MID_DOWN  4
#define EVENT_BTN_LEFT_DOWN 5

//Dimensió de les taules de la màquina d'estas
#define TOTAL_EVENTS_rlr2   5

//Posició dels disparadors en les taules
//Valors retornats per les funcions "getEvents"
#define DISP_0  0
#define DISP_1  1
#define DISP_2  2
#define DISP_3  3
#define DISP_4  4

//Posició dels estats en les taules
#define ESTAT_b2    0
#define ESTAT_bl    1
#define ESTAT_iniciblb2 2

struct tagDrlr2 {
    int event;
    int estat_rlr2;
    int estat_R2;
};
```

Figura 37. Exemple estat compost. Capçalera.

5.2. Exemple de màquina d'estats amb estat compost concurrent.

En aquest exemple volem provar la generació de codi amb un estat compost recurrent format per les regions "D1D2" i "L1L2", aquesta darrera amb un estat inicial de historia poc profunda. La Figura 38 mostra el diagrama tal com ha estat dibuixat en el argoUML0.24.

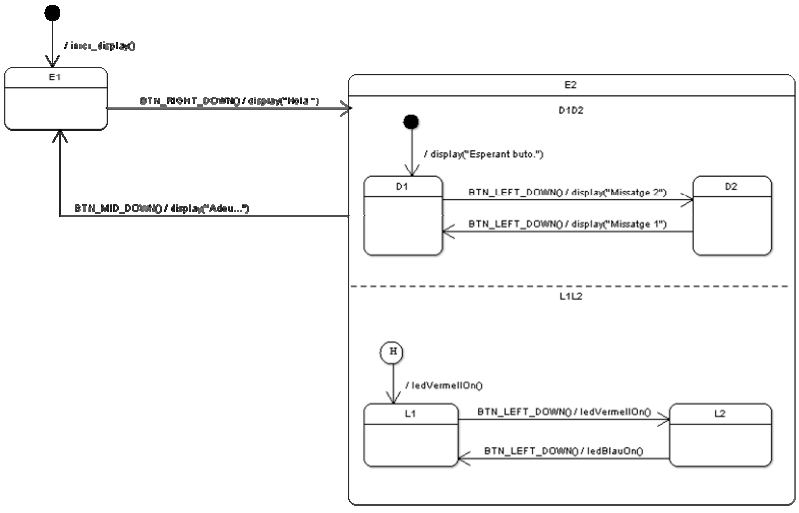


Figura 38.Exemple de màquina d'estats amb estat compost recurrent.

En aquest exemple volem provar la generació de codi amb un estat compost recurrent format per les regions “D1D2” i “L1L2”, aquesta darrera amb un estat inicial de historia poc profunda.

En la Figura 39 s’hi pot veure el codi generat pel sistema “E1E2”. Ressaltem la inicialització de la variable “estat_D1D2” a l’acció “accio_E1D0”. Així cada vegada que s’entri al sistema “D1D2” es començarà per l’estat d’inici “iniciD1D2”.

```
/*
 * Codi generat automaticament per MASGEC V0.1
 *
 * Fitxer d'origen     :    E:\Pere\UOC\wsp_pfc\fonts_c\visor1\visor1.xmi
 * Màquina d'estats    :    visorInici
 * Data                :    Sun Jan 13 12:07:18 CET 2008
 * Total d'estats      :    12
 * Total de transicions:    2
 **/

#include    "visorInici.h"

//visor1
//Estat: E1 0
int getEvent_E1(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    if(d_visor1->event==EVENT_BTN_RIGHT_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_E1D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    display("Hola.");
    d_visor1.estat_D1D2= ESTAT_iniciD1D2;
}


//EstatInicial: iniciE1E2 1
int getEvent_iniciE1E2(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    return DISP_0;
}

//Estat Compost: E2 2
int getEvent_E2(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    if(d_visor1->event==EVENT_BTN_MID_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_E2D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    display("Adeu...");
}


int (*const tGetNouEvent_visor1 []) (void *) = {
    getEvent_E1, getEvent_iniciE1E2, getEvent_E2
};
void (*const tEstats_visor1 [][TOTAL_EVENTS_visor1]) (void *) = {
    {accio_E1D0, accioSxExNull},
    {accio_iniciE1E2D0, accioSxExNull},
    {accio_E2D0, accioSxExNull}
};
int const tProperEstat_visor1 [][TOTAL_EVENTS_visor1] = {
    {ESTAT_E2, ESTAT_E1},
    {ESTAT_E1, ESTAT_iniciE1E2},
    {ESTAT_E1, ESTAT_E2}
};
```

Figura 39. Exemple estat compost recurrent. Sistema “E1E2”.

En la Figura 40 s'hi pot veure el codi generat pel subsistema "L1L2".

```
//E2
//Estat Compost: L1L2 1
//Estat: L1 0
int getEvent_L1(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    if(d_visor1->event==EVENT_BTN_LEFT_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_L1D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    ledVermellOn();
}


//EstatInicial: inicL1L2 2
int getEvent_inicL1L2(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    return DISP_0;
}

void accio_inicL1L2D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    ledVermellOn();
}
//Estat: L2 1
int getEvent_L2(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    if(d_visor1->event==EVENT_BTN_LEFT_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_L2D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    ledBaluOn();
}


int (*const tGetNouEvent_L1L2 []) (void *) = {
    getEvent_L1, getEvent_inicL1L2, getEvent_L2
};
void (*const tEstats_L1L2 [][TOTAL_EVENTS_L1L2]) (void *) = {
    {accio_L1D0, accioSxExNull},
    {accio_inicL1L2D0, accioSxExNull},
    {accio_L2D0, accioSxExNull}
};
int const tProperEstat_L1L2 [][TOTAL_EVENTS_L1L2] = {
    {ESTAT_L2, ESTAT_L1},
    {ESTAT_L1, ESTAT_inicL1L2},
    {ESTAT_L1, ESTAT_L2}
};
```

Figura 40. Exemple estat compost recurrent. Subsistema "L1L2".

En la Figura 41 s'hi pot veure el codi generat pel subsistema "D1D2".

```
//Estat Compost: D1D2 0
//D1D2
//Estat: D1 0
int getEvent_D1(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    if(d_visor1->event==EVENT_BTN_LEFT_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_D1D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    display("Missatge 2");
}


//EstatInicial: iniciD1D2 2
int getEvent_iniciD1D2(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    return DISP_0;
}

void accio_iniciD1D2D0(struct tagDvisor1 *d_visor12)
{
    //Effect
    display("Esperant buto.");
}
//Estat: D2 1
int getEvent_D2(struct tagDvisor1 *d_visor1)
{
    //Esdeveniments
    if(d_visor1->event==EVENT_BTN_LEFT_DOWN)
        return DISP_0;
    return DISP_1;
}

void accio_D2D0(struct tagDvisor1 *d_visor1)
{
    //Effect
    display("Missatge 1");
}


int (*const tGetNouEvent_D1D2 []) (void *) = {
    getEvent_D1, getEvent_iniciD1D2, getEvent_D2
};
void (*const tEstats_D1D2 [][TOTAL_EVENTS_D1D2]) (void *) = {
    {accio_D1D0, accioSxExNull},
    {accio_iniciD1D2D0, accioSxExNull},
    {accio_D2D0, accioSxExNull}
};
int const tProperEstat_D1D2 [][TOTAL_EVENTS_D1D2] = {
    {ESTAT_D2, ESTAT_D1},
    {ESTAT_D1, ESTAT_iniciD1D2},
    {ESTAT_D1, ESTAT_D2}
};
```

Figura 41. Exemple estat compost recurrent. Subsistema "D1D2".

En la Figura 42 s'hi pot veure el codi generat del la funció principal del diagrama de màquina d'estats. Ressaltem la inicialització de la variable "estat_iniciE1E2" del sistema principal "E1E2" i de la variable "estatL1L2" del subsistema "L1L2", aquesta s'inicialitza aquí pel fet de tenir un estat historia poc profunda, doncs és la manera de que no es reinicialitzi amb l'estat inicial "iniciL1L2" quan es torni passar a l'estat "E2". També es pot observar la implementació de la concurrència que consisteix en l'execució en el mateix cicle del autòmats del subsistema "D1D2" i del subsistema "L1L2", això queda implementat en la crida als dos autòmats dins de la sentencia "if".

```
void visor1(void)
{
    struct tCuaEvents cuaEvents;
    struct tagDvisor1 d_visor1;

    initCuaEvent(&cuaEvents);

    d_visor1.estat_visor1= ESTAT_iniciE1E2;
    d_visor1.estat_L1L2= ESTAT_iniciL1L2;
    for(;;){

        d_visor1.event= getEvent(&cuaEvents);
        //Crides a l'automata
        automata(&d_visor1,
                tGetNouEvent_visor1,
                tEstats_visor1[d_visor1.estat_visor1],
                tProperEstat_visor1[d_visor1.estat_visor1],
                &d_visor1.estat_visor1);
        if(d_visor1.estat_visor1==ESTAT_E2){
            automata(&d_visor1,
                    tGetNouEvent_L1L2,
                    tEstats_L1L2[d_visor1.estat_L1L2],
                    tProperEstat_L1L2[d_visor1.estat_L1L2],
                    &d_visor1.estat_L1L2);
            automata(&d_visor1,
                    tGetNouEvent_D1D2,
                    tEstats_D1D2[d_visor1.estat_D1D2],
                    tProperEstat_D1D2[d_visor1.estat_D1D2],
                    &d_visor1.estat_D1D2);
        }
    }
}
```

Figura 42. Exemple estat compost recurrent. Funció principal.

En la Figura 43 hi trobem el fitxer de capçalera on es poden observar la declaració de les constants que l'aplicació ha utilitzat per la implementació així com la declaració de l'estructura de dades de la màquina d'estats amb les variables:

- "event". Per poder disposar del valor de l'esdeveniment a tractar en tots els subpassos d'un cicle.
- "estat_visor1". Indica l'estat actual del sistema "E1E2".
- "estat_L1L2". Indica l'estat actual del subsistema "L1L2", que està format pels estats "iniciL1L2", "L1" i "L2".
- "estat_D1D2". Indica l'estat actual del subsistema "D1D2", que està format pels estats "iniciD1D2", "D1" i "D2".

```
/*
 * Codi generat automaticament per MASGEC V0.1
 *
 * Fitxer d'origen     :    E:\Pere\UOC\wsp_pfc\fonts_c\visor1\visor1.xmi
 * Màquina d'estats    :    visorInici
 * Data                :    Sun Jan 13 12:07:18 CET 2008
 * Total d'estats      :    12
 * Total de transicions:    2
 **/

//Esdeveniments retornats per la cua d'esdeveniments
#define EVENT_BTN_RIGHT_DOWN    1
#define EVENT_BTN_MID_DOWN  2
#define EVENT_BTN_LEFT_DOWN 3

//Dimensió de les taules de la màquina d'estas
#define TOTAL_EVENTS_visor1 2

//Posició dels disparadors en les taules
//Valors retornats per les funcions "getEvents"
#define DISP_0  0
#define DISP_1  1

//Posició dels estats en les taules
#define ESTAT_E1    0
#define ESTAT_iniciE1E2 1
#define ESTAT_E2    2

//Posició dels estats en les taules
#define ESTAT_L1    0
#define ESTAT_iniciL1L2 1
#define ESTAT_L2    2

//Posició dels estats en les taules
#define ESTAT_D1    0
#define ESTAT_iniciD1D2 1
#define ESTAT_D2    2

struct tagDvisor1 {
    int event;
    int estat_visor1;
    int estat_L1L2;
    int estat_D1D2;
};
```

Figura 43. Exemple estat compost recurrent. Capçalera.

6. Conclusions.

Els objectius marcats s'han aconseguit, obtenir un codi en ANSI C a partir d'un diagrama de màquina d'estats. Com sempre, al principi es pot tenir una ideà, més o menys gran del tema que s'ha de desenvolupar, però a mida que s'hi va entrant, cada vegada més, els horitzons es van eixamplant. L'objectiu inicial s'ha assolit però durant la realització s'han obert d'altres com poden ser:

- Realitzar un anàlisi de les coherències del diagrama d'estats dibuixats a partir del model de màquina d'estats generat.
- Implementar la resta de funcionalitats de les màquines d'estats, definits per l'UML, que s'han obviat, "forks", "unions", etc.
- Fer que l'aplicació sigui un "plug-in" de l'ArgoUML0.24" si aquesta eina segueix sent valida per la realització dels diagrames.
- Explorar d'altres formats d'entrada, com pot ser el SCXML "State Chart Extensible Markup Language" desenvolupat pel consorci w3c, que en aquest cas també es podria considerar de sortida.
- Considerar alguna opció de cara a tenir diferents codis optimitzats en funció del volum de codi o del temps d'execució.

7. Valoració econòmica.

Només hem fet una valoració aproximada del cost del projecte, considerant un sol rol per realitzar totes les etapes del projecte i prenen un preu mig de 25 €/hora.
La durada total del projecte ha estat de 84 dies, segons el calendari del projecte, amb un promig de 2 hores diàries, tenim un total de 168 hores.
El cost econòmic serà de 168 hores * 25 €/hora = 4200 €.
Si comptem la jornada de 8 hores tenim que en total són 21 dies laborables els necessaris per l'obtenció de l'aplicació.

Dies laborables	Hores*dia	Preu de l'hora (€)	Total d'hores	Cost total (€)
21	8	25	168	4200

Figura 44. Valoració econòmica aproximada.

Glossari.

UML (Unified Modeling Lenguage). Llenguatge unificat de modelatge. Es un llenguatge gràfic per visualitzar, especificar, construir i documentar un sistema de programari. Compta amb diversos tipus de diagrames per mostrar els diferents aspectes de les entitats representades.

XML (eXtensible Markup Language). Llenguatge de marques extensible. Metallenguatge extensible d'etiquetes. No és un llenguatge en concret sinó una manera de definir llenguatges per a diferents necessitats. Té el propòsit de convertir-se en l'estàndard pel intercanvi d'informació estructurada entre diferents plataformes.

XMI (XML Metadata Interchange). És una especificació per al intercanvi de models de UML entre diferents eines de modelat.

DOM (Document Object Model) Model d'Objectes de un Document. Jerarquia de classes que permeten de representar un document electrònic en memòria. Ofereix mètodes per a navegar i manipular l'estructura.

ANSI C. Estandardització del llenguatge C per part del Institut Americà d'Estàndards (American National Standarts Institute).

Compilador. (compiler). És un programa que tradueix un programa (codi font) escrit amb un llenguatge de programació a un altre llenguatge màquina (codi objecte).

Depurador. (debugger). És un programa que permet depurar i netejar d'errors un altra programa.

Enllaçador (linker). És un programa que pren els fitxers de codi objecte creats pel compilador i la informació necessària dels recursos senyalats (llibreria) per enllaçar tot el codi objecte a fi d'obtenir un fitxer executable.

Firmware. És el programa que està en un dispositiu de maquinari com pot ser un microcontrolador o una flash ROM. Estaria situat entre el programari i el maquinari, Aquest que fora del seu context no te cap utilitat.

ROM (Read Only Memory) Memòria solament d'escriptura. Memòria que no perd les dades quan és queda sense alimentació. Normalment és elèctricament esborrable i regravable.

Device driver (Control de dispositius). És el programa que permet la interacció entre el programari (normalment escrit amb llenguatges d'alt nivell) i un maquinari concret. Són totalment dependents del maquinari.

Sistema encastat. (Embedded system) és un sistema informàtic d'ús específic, que és encapsulat totalment pel dispositiu que controla. Un sistema encastat té requisits específics i realitza tasques predefinides, a diferència d'un ordinador personal d'ús general.

API (Application Programming Interface) Interfície de programació d'aplicacions. És un conjunt de funcions i procediments que ofereix una llibreria determinada per ser utilitzat per un altre programari com una capa d'abstracció.

LED (Light Emitting Diode) Diode emissor de llum. És un dispositiu semiconductor (diode) que emet llum quasi-monocromàtica, quan és polaritzat de forma directa i es travessat per un corrent elèctric.

Plug-in. És una aplicació informàtica que interactua amb un altre aplicació per aportar-li una funció o utilitat especifica. També es podria definir, com una manera de poder expandir els programes de forma modular de tal manera que es podessin afegir funcionalitats sense afectar les existents ni complicar el desenvolupament del programa principal.

IDE (Integrated Development Environment) Entorn de treball integrat. És una aplicació composta per un conjunt d'eines per facilitar la feina a un programador. Aquestes eines poden ser un editor de codi, un compilador, un enllaçador i un depurador.

Bibliografia.

David Harel and Michal Politi. Modeling Reactive Systems with Statcharts. The Statemate Approach. McGraw-Hill, 1998
F.Wagner, R.Schmuki, T.Wagner, P.Wolstenholme. Modeling Software with Finite States Machines. A Practical Approach. Auerbach Publications, 2006

Webs consultades.
Web de consulta general.
[http://ca.wikipedia.org/wiki]

Web que comercialitzen productes semblants.
[http://www.stateworks.com]
[http://www.iar.com/vs]

Web per descarregar l'aplicació gràfica pel disseny de programari argoUML0.24
[http://argouml.tigris.org]

Annex A. Manual de l'usuari.

Només fer l'observació que el manual que acompanyarà l'aplicació és una mica més extens doncs s'hi ha afegit una petita introducció als diagrames d'estats i una explicació de com aquests són implementats. Aquí no hi són per ser redundants i no fer la memòria més extensa, doncs són resums dels apartats 2.1 i 2.3 d'aquesta memòria.

1. Introducció

1.1. Què és MASGEC

MASGEC és una aplicació de consola per generar el codi en ANSI C de manera automàtica a partir d'un diagrama d'una màquina d'estats, més concretament del model generat en format XMI XML Metadata Interchange (XML d'intercanvi de metadades) per l'aplicació gràfica de disseny de software ArgoUML 0.24 que és una aplicació gratuïta, independent de la plataforma i de codi obert.

D'aquesta manera MASGEC vol facilitar més la tasca de disseny, implementació i manteniment de les aplicacions desenvolupades en ANSI C i més concretament en els sistemes encastats.

MASGEC genera codi pel funcionament dinàmic de l'aplicació, la descrita en els diagrames de maquines d'estats i només una petita part de codi haurà de ser escrita pel desenvolupador, concretament la que permet la interacció entre la màquina d'estats i el maquinari i d'altres parts de l'aplicació. Gracies a aquesta independència amb el maquinari fa que el codi generat per MASGEC sigui transportable a diferents plataformes de maquinari.

MASGEC s'executa sota l'entorn Java i per tant és independent al sistema operatiu.

2. Instal·lació

2.1. Requeriments del sistema

Requeriments mínims del sistema:

- Qualsevol sistema operatiu que suporti Java.
- Java 2 JRE o JDK a partir de la versió 1.4. [www.**java**.com/download/].
- argoUML 0.24 o qualsevol aplicació gràfica de disseny de programari que permeti exportar els models en format XMI. [http://www.argouml.org/].
- 5 MB d'espai lliure en el disc dur.

2.2. Opcions d'instal·lació

Cal assegurar que es té instal·lat Java 2 JRE.

Crear un directori per instal·lar MASGEC, per exemple "C:\masgec" i copiar-hi el fitxer empaquetat "masgec.zip".

Canviar del directori actual al directori creat per instal·lar MASGEC.

Desempaquetar el contingut del fitxer "masgec.zip" en el directori actual. L'estructura del directori resultant ha de ser aquesta:

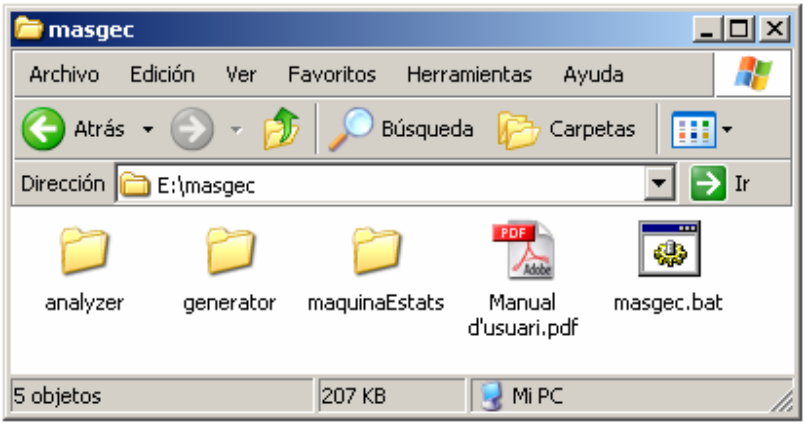


Figura 45. Contingut del directori de l'aplicació MASGEC després de d'instal·lació.

2.3. Comandes de línea

Per fer corre MASGEC des de la línea de comandes tenim tres possibilitats:
Teclejar:

java analyzer.Analyzer –f fitxer d'entrada(xmi)

O bé utilitzant el fitxer "batch" que acompanya a l'aplicació:

masgec –f fitxer d'entrada(xmi)

O podem escriure un fitxer "batch" amb les comandes i opcions necessàries.
Les opcions que contempla MASGEC són:

```
masgec -f xmi_input_file [-v] [-a] [-o path]
     -f <arg>    fitxer del model en format xmi
     -v          l'aplicació va indicant els passos que realitza
     -a          per generar les llibreries associades a l'autòmat
     -o          directori destí dels fonts creats
```

Important: L'argument de "-f" "<arg>" sempre ha de ser l'adreça complerta del fitxer en format XMI. També cal recordar que si és vol cridar l'aplicació des del directori on hi ha el fitxer XMI del diagrama s'ha de afegir el "path" camí del "MASGEC" a la variable d'entorn "CLASSPATH". Per exemple, si tenim l'aplicació "MASGEC" en el directori "C:\MASGEC" escriuríem:

- **C:\SET CLASSPATH = %CLASSPATH%;C:\MASGEC**