Títol: Simulació multiagent de la Girona medieval emprant SDLPS

Volum: 1

Alumne: Javier Millan Mendez

Director/Ponent: Pau Fonseca i Casas

Departament: EIO (Estadística i Investigació Operativa)

DADES DEL PROJECTE

Títol del Projecte: Simulació multiagent de la Girona Medieval emprant SDLPS

Nom de l'estudiant: Javier Millan Mendez
Titulació: Enginyeria Informàtica
Crèdits: 37,5
Director/Ponent: Pau Fonseca i Casas
Departament: EIO

MEMBRES DEL TRIBUNAL

President: Josep Casanovas Garcia

Vocal: Angel Olivé Duran

Secretari: Pau Fonseca i Casas

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data: 27/06/2011

Continguts

1 Motivació

El propòsit del projecte és la construcció d'un model de simulació d'una ciutat medieval emprant el simulador SDLPS (P. Fonseca i Casas, SDL distributed simulator 2008). El punt de partida és una versió preliminar construïda durant el desenvolupament d'un projecte predecessor, el projecte *Lemuria* (Rello 2009). El model evolucionat d'aquest model preliminar s'anomenarà **civiSDL**.

El model ha de ser parametritzable amb la ciutat de Girona en aquesta època. D'aquesta forma definim un "input" d'exemple i alhora el model és utilitzable en un futur per altres ciutats.

S'opta per una ciutat medieval per simplificar els processos (la vida en una ciutat del passat aparentment és més simple de modelar) i per conèixer tant un estat inicial com final de les variables del model, per mitjà de la documentació històrica.

L'objectiu final de totes les evolucions que es facin sobre aquest projecte és estudiar el comportament de la ciutat modelada davant diferents situacions, així com l'evolució de les variables que s'estudien. Això ens permet tant explicar el perquè de situacions passades com predir les futures.

El SDLPS és un software en desenvolupament. Paral·lelament un objectiu del projecte és treure rendiment del simulador, localitzar errors i promoure possibles millores per tal de que evolucioni.

2 La simulació i els models socials

Podem veure la simulació com l'experimentació amb un model basat en la realitat. Els objectius es poden centrar en:

- Descriure el comportament del model davant determinats inputs.
- Construir i confirmar o rebutjar hipòtesis a partir d'aquest comportament.
- Predir successos en base als resultats obtinguts.

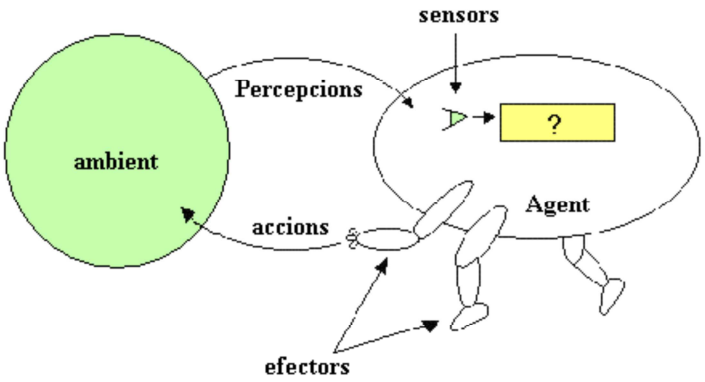
Un model social no resulta fàcil d'acotar, ja que el món que representen es ampli i s'han de marcar els límits. El comportament humà està influït per nombrosos factors i les societats formen xarxes d'esdeveniments que poden resultar interminables.

Per qualsevol model de simulació és imprescindible formular una sèrie d'hipòtesis simplificadores que delimitin el sistema i permetin una abstracció factible del món real.

Per sort l'ésser humà s'ha dedicat al llarg de la seva història a estudiar el món que l'envolta i la

seva pròpia conducta. Això facilita el fet de plasmar en un model de simulació una imatge ben propera a la realitat. Sense dubte és una feina que requereix coneixement de fonts multidisciplinàries per obtenir bons resultats.

El model d'una organització urbana s'ha de veure com un sistema multiagent on la gran part són intel·ligents: reben estímuls d'altres agents, actuen en conseqüència i adquireixen coneixement (P. Fonseca i Casas, SDL, A Graphical Language Useful to Describe Social Simulation Models 2008)



Il·lustració 1: esquema d'un agent intel·ligent

3 Antecedents

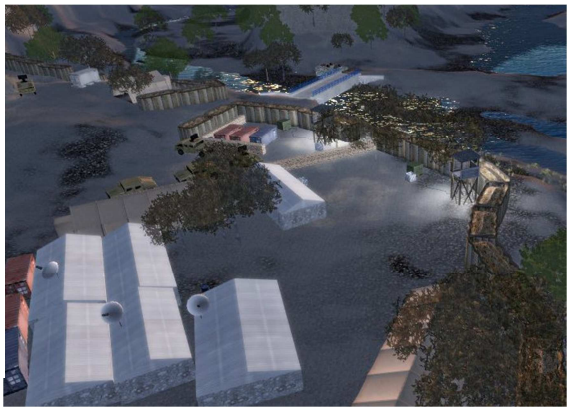
Una gran majoria d'exemples en simulacions socials i en concret de ciutats, els podem trobar al món dels videojocs. Aquests sistemes combinen la interacció de l'usuari amb una intel·ligència que permet al sistema actuar amb independència.

Potser l'exemple més famós són els *Sims*. El comportament humà n'és el factor més destacable.

D'altres casos més centrats en la civilització humana són jocs *com SimCity, Age of Empires...*

Il·lustració 2: Imatge del terreny al joc Age of Empires

Si parlem d'aplicacions a la vida real, l'exèrcit dels Estats Units també dedica part dels seus recursos tecnològics per investigar amb models socials que intenten predir escenaris a situacions de conflicte o per l'entrenament de les seves tropes (DARPA, U.S. Army looking for social computing technology 2009)



Il·lustració 3: imatge de simulador del exèrcit americà

4 Àmbits d'estudi

La modelització d'una ciutat pot ser quasi bé infinita, tenint en compte tots els factors o àmbits d'estudi que pot contemplar una organització urbana. Per això s'ha de definir l'abast del model; els següents punts descriuen els àmbits d'estudi que s'inclouen al projecte:

- **Demografia**

  Volem estudiar diverses dades sobre la població de la ciutat. Podem observar el temps de vida de les persones, les taxes de naixement i mort, els efectes de l'alimentació en la salut, la forma de crear noves famílies...

- **Terreny**

  Simularem el terreny amb una graella de propietats, tant urbanes com rurals. S'espera veure com la ciutat creix, si tendeix cap a l'urbanisme o cap a l'explotació rural o fins i tot quins efectes provoca el clima sobre el territori de la ciutat.

- **Producció**

  Les famílies podran decidir quin producte exploten per sobreviure. Volem veure quins productes són els més escollits, com canvien les tendències segons el preu, l'experiència que van adquirint els ciutadans en cada àmbit de treball i d'altres factors interessants.

- **Economia**

  Simularem un mercat on les famílies poden comprar i vendre els productes. Segons aquests moviments els preus variaran a l'alça o a la baixa. El poder adquisitiu de cada família serà molt divers.

- **Migració - Immigració**

  El model ens ha de permetre veure com algunes famílies decideixen abandonar la ciutat per cercar millor prosperitat o en canvi famílies noves hi busquen lloc.

5 Objectius tecnològics

Podem resumir els objectius del projecte en aquests punts:

- **Simplificació i modulació del model**

  El nou model ha de tenir una implementació molt estructurada i modular per tal de que la seva comprensió sigui senzilla i l'ampliació més còmoda.

- **Implementació multiagent de les Unitats Familiars del sistema**

  Les famílies del sistema han d'estar representades per instàncies d'un mateix agent, descentralitzant el control de tot el mòdul d'Unitats Familiars. S'ha de seguir la mateixa línia de disseny pels Membres.

- **Implementació com Autòmat Cel·lular del terreny del sistema**

  El terreny de la ciutat ha d'estar representat per un agent de tipus autòmat cel·lular, que ens permetrà operar amb més flexibilitat sobre cada porció de terreny (cel·la de l'autòmat).

- **Reducció del codi extern al SDL**

  S'ha de reduir tant com sigui possible l'ús de codi extern implementat en C ; s'han d'implementar les seves funcionalitats dins els diagrames SDL.

- **Ampliació i millora de continguts del model**

  S'ha de millorar la intel·ligència de la qual disposa el model per realitzar les operacions, així com introduir nous conceptes que ampliïn la representació de la ciutat dins el model.

6 Implementació en Specification and Description Language

SDL (*Specification and Description Language*) és un llenguatge d'especificació pensat per a sistemes distribuïts, que inclou la descripció del comportament, estructura i dades del sistema descrit. Està definit per la *International Telecomunicacions Union–Telecommunications Standardization Sector (**ITU–T**)* sota la recomanació Z.100 (P. Fonseca i Casas, Formalismes III, SDL s.f.).

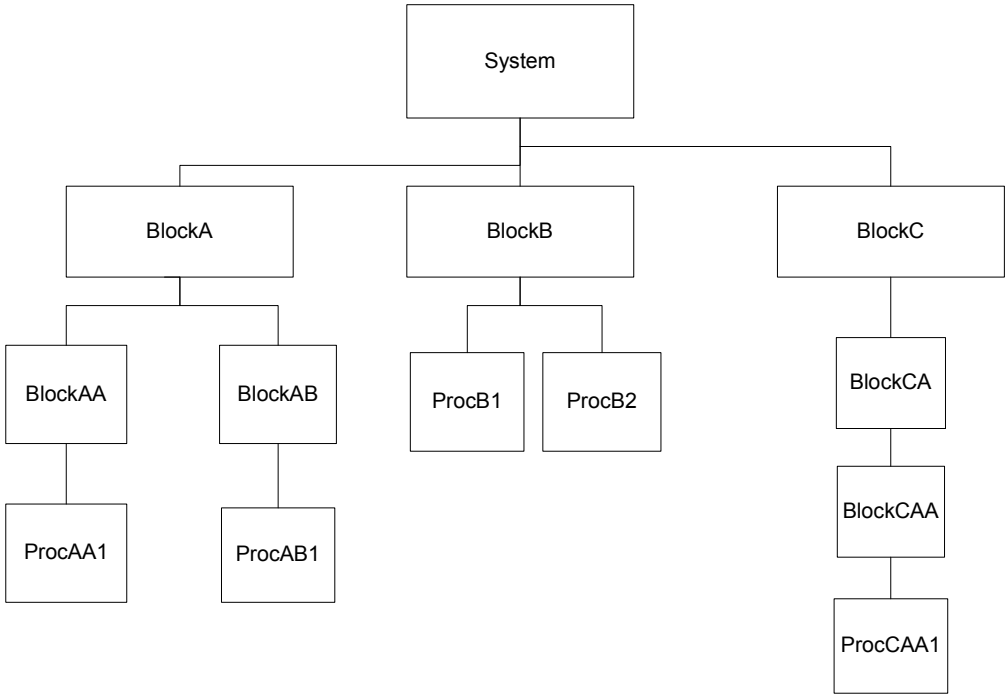
Un sistema implementat en SDL basa el funcionament en l'enviament i rebuda de senyals (*signals*). Aquests *signals* son rebuts per processos que actuen en conseqüència.

6.1 Nivells

Un sistema SDL està estructurat jeràrquicament. A continuació es descriuen de major a menor nivell dins la jerarquia:

- **Sistema:** conté tota la representació SDL. Pot estar composat de diversos blocs.
- **Bloc:** pot contenir diversos blocs o processos que es poden executar de forma concurrent. Serveixen per organitzar el sistema de forma distribuïda i jeràrquica. Els processos només poden compartir el mateix nivell de profunditat amb processos.
- **Procés:** element que conté una màquina finita d'estats extesa, la qual canvia el seu estat en funció dels *signals* que rep per part d'altres processos del sistema.
- **Procedure:** fragment de codi dins un procés que pot ésser cridat des de el mateix procés.

Utilitzarem la nomenclatura *agent* per referir-nos a un sistema, bloc o procés.



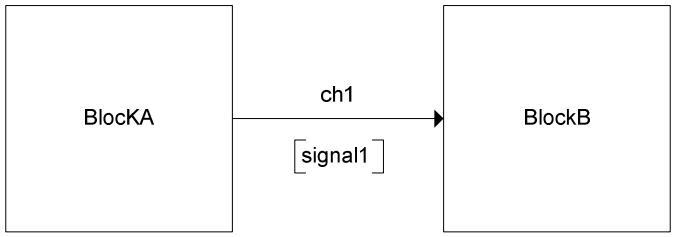
Il·lustració 4: exemple d'estructura de sistema

6.2 *Channels* i *signals*

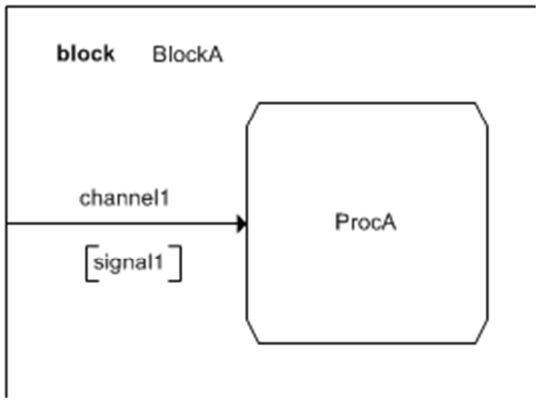
Els processos es poden comunicar per mitjà de senyals (*signals*). Els *signals* s'identifiquen per un nom. Cada *signal* pot ser enviat via un canal.

Els canals (*channels*) comuniquen dos agents i tenen assignats un o més *signals*, per on poden ser enviats.

Un *signal* comença i acaba el seu trajecte en un procés, travessant els blocs necessaris, sempre que estiguin correctament comunicats per *channels*.



Il·lustració 5: Blocs comunicats per un canal

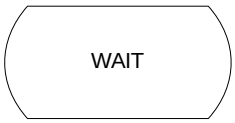


Il·lustració 6: connexió bloc – procés

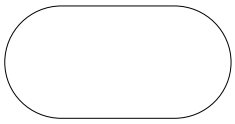
6.3 Processos

Els processos estan formats per una sèrie d'elements que determinen com funcionen. A continuació queden descrits:

Estat: representa un estat en els que pot quedar el procés.



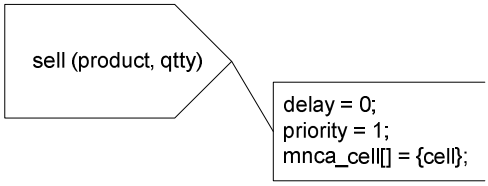
Start: estat inicial del procés, del qual l'execució surt sense necessitat de rebre cap *signal*.



Input: element que representa la recepció del *signal* amb el nom que descriu. Totes les branques d'un estat estan lligades a un input.

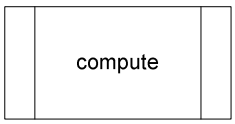
Output: sortida de *signal*. Especifica els següents paràmetres:

- **Nom:** nom del *signal*.
- **Delay:** temps de retard respecte l'instant actual en el que s'enviarà el *signal*.
- **Paràmetres d'usuari:** altres valors que es poden enviar juntament amb el *signal*.
- **Prioritat:** prioritat davant altres senyals que incideixen al mateix estat.
- **Cel·la:** En cas de que el procés que rep el signal és un autòmat cel·lular, hem d'especificar les cel·les a les quals està adreçat el signal

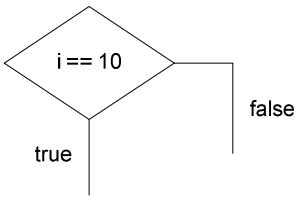


Task: fragment de codi a executar. Aquest codi pot operar amb les variables declarades dins el procés *(DCL's)* o fer crides a operacions declarades al codi extern.

Crida a Procedure: conté la crida a un dels *procedure* declarats al procés.



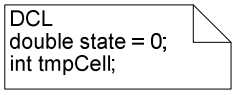
Decision: permet la declaració de condicions que bifurquen l'execució.



Create: crea una instància de l'agent que s'indica.

Delete: finalitza el procés. Elimina el procés del sistema.

DCL: variable declarada dins el procés.



6.4 Els autòmats cel·lulars

6.4.1 Teoria

Un autòmat cel·lular és un sistema n-dimensional de cel·les ubicades geomètricament on cada cel·la conté un estat escollit entre un alfabet finit. El comportament de totes les cel·les està determinat per un mateix procediment, que decideix l'estat en que cadascuna es troba segons variables i temps discrets.

Cada cel·la representa una unitat d'espai, que canvia d'estat segons passa el temps discretament segons una **funció de transició** per l'estat actual i segons l'estat de les cel·les veïnes (funció de veïnat).

6.4.2 El Joc de la Vida

Un exemple clàssic és el *Joc de la Vida,* de Conway (Wikipedia 2011). Consisteix en un taulell (autòmat cel·lular) on cada casella (cel·la) s'actualitza simultàniament. Cada cel·la té 8 cel·les veïnes i dos possibles estats: viva o morta. Les transicions es fan segons aquesta funció de veïnat:

1. Una cel·la morta amb exactament 3 veïnes vives neixen de nou.
2. Una cel·la viva amb 2 o 3 cèl·lules vives segueix viva, altrament mor o segueix morta.

Amb un estat inicial i només aquestes senzilles regles, s'han experimentat patrons molt interessants associables a patrons d'organització a la vida real.

6.4.3 Definició

Per al present treball emprarem una extensió del autòmat cel·lular autòmat cel·lular n-dimensional multicapa m:n-CA^k (Fonseca i Casas i Casanovas, Simplifying Gis Data Use Inside Discrete Event Simulation Model Through m:n-ac Cellular Automaton 2005).

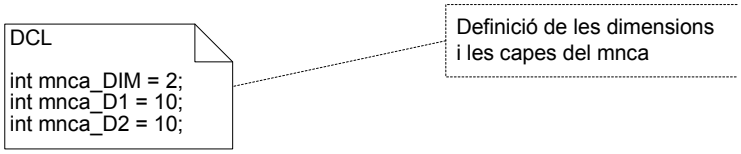
Podem notar un autòmat cel·lular n-dimensional i multicapa com **m:n-ac^k** on:

- **M**: nombre de capes
- **n:** dimensions de l'autòmat
- **K:** capes principals. Una capa principal es aquella que canvia d'estat durant el temps, i per tant ha de tenir un procés associat (funció de transició) que computi els canvis d'estat al pas del temps.

6.4.4 Aplicació al Model

Al model civiSDL, el mòdul que simula el terreny amb les propietats de les famílies (*Household*) està dissenyat com un autòmat cel·lular. Trobareu l'especificació completa a l'apartat de *Disseny i implementació.*

Podem desenvolupar autòmats cel·lulars amb SDL i executar-los a SDLPS. L'autòmat cel·lular es declara com un bloc que té assignat les variables de dimensió:



Il·lustració 7: definició de les variables d'un MNCA

Cada capa principal ha de tenir un bloc dins l'autòmat. Aquest bloc contindrà els processos que determinen l'estat de la capa.

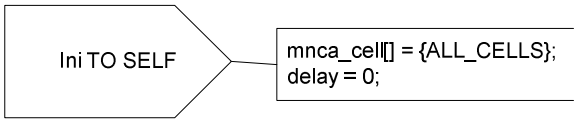
Les capes secundàries es declaren com variables multidimensionals dins l'autòmat.

Aquesta graella mostra a mode d'exemple els identificadors associats a cada cel·la de l'autòmat, en aquest cas de dimensió 2, on la primera component és 10 i la segona 10.

1	2	3	…			..	10
11	…					….	20
…							
							100

6.4.5 Comunicació

Els *signals* que fem arribar a un autòmat cel·lular han de tenir obligatòriament el paràmetre CELL indicat. Hem de determinar les cel·les destinatàries del *signal*. En cas de que voler fer un *broadcast* (*signal* a totes les cel·les) hem d'indicar com a valor del paràmetre l'etiqueta **ALL_CELLS.**



Il·lustració 8: *signal broadcast* a un *mnca*

Disposem d'una API amb funcions que ens ajuden a manegar l'autòmat cel·lular. Aquestes funcions es poden executar dins de qualsevol procés ubicat a una capa de l'agent:

- **MncaGetCurrentCell**
  obté el ID de la cel·la que estem tractant.
- **MncaGetCellValue (capa x, cel·la i)**
  retorna el valor de la capa x per la cel·la i.
- **MncaSetCellValue (capa x, cel·la i, valor v)**
  posa el valor v a la capa x per la cel·la i.

6.4.6 Fitxers de dades SIG

La informació de les capes de l'autòmat cel·lular està representada en format **Idrisi 32** (Clark Labs 2009)**.**

Idrisi32 és un sistema de informació geogràfica (SIG) que es basa en la representació de dades *raster*. El sistema té tres fitxers per cada capa:

1. **[nom_capa].doc**: fitxer que conté metadades de la capa.

2. **[nom_capa].img:** conté els valors **inicials** de la capa ordenats per cel·la, de forma que el primer valor correspon a la cel·la 1 i l'últim a la cel·la n x m, considerant un autòmat cel·lular de dimensions n x m.

3. **Resultats_[nom_capa].img**: amb la mateixa estructuració que els anteriors, contenen els valors de la capa corresponent tal i com van canviant al llarg de la simulació.

Tots els fitxers mencionats es troben a la carpeta **data** del model.

7 Construcció del model SDL

7.1 El software: SDLPS

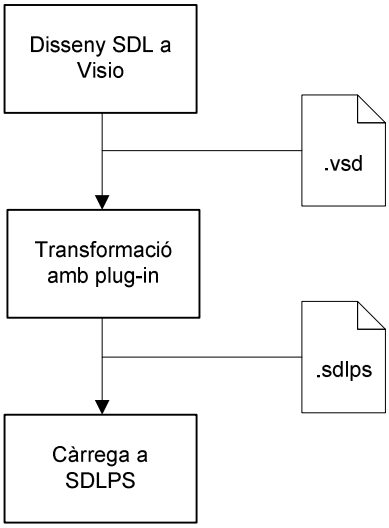
El SDLPS és un simulador que permet executar models dissenyats en SDL. Per fer-ho, utilitza una representació en XML del model dissenyat gràficament en SDL.

Per construir la part SDL del model seguim aquests dos passos:

1. **Disseny dels diagrames SDL amb Microsoft Visio 2010.** S'utilitza el *plug-in* **Sandrila,** que permet la construcció i validació dels diagrames de forma correcta.

2. **Transformació a XML.** Juntament amb *Sandrila*, s'utilitza una macro dissenyada per Visio que transforma el sistema de diagrames SDL en una representació en XML. El fitxer generat té extensió *.sdlps*.

El fitxer *.sdlps* s'utilitza per carregar el model al SDLPS.



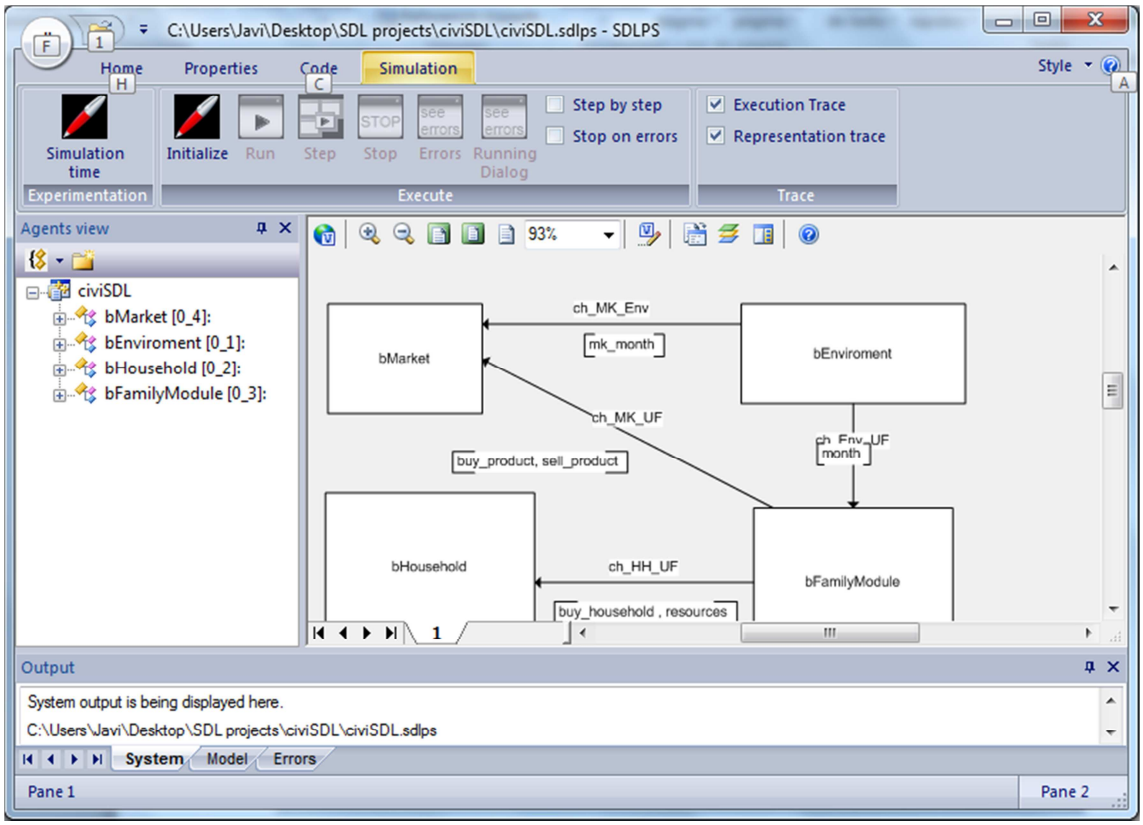
Il·lustració 9: procés de creació del model

A la figura següent es mostra la pantalla principal del SDLPS. Com es pot visualitzar, disposa d'una vista principal on es mostra el diagrama principal del Sistema.

Al panell esquerre podem veure l'arbre d'agents que forma el Sistema.

A la secció inferior hi és la pantalla de Output, que mostra els missatges d'error (o correctesa) de la compilació i *link*, així com de la execució del model.

La barra superior disposa de les operacions necessàries per fer funcionar el model. En l'apartat següent s'explicaran amb més detall, juntament amb els passos a seguir per executar el model.



Il·lustració 10: pantalla principal del SDLPS

7.2 Passos d'execució del Model

Els passos a efectuar per fer funcionar el model emprant SDLPS queden descrits en ordre a continuació:

1. **Càrrega del model** *(Home -> Open)*

   S'ha de seleccionar el fitxer *.sdlps* del model. Si és sintàcticament correcte, el model quedarà carregat al programa.

2. **Creació, compilació i link del Codi** *(Code-> Create, Compile and Link DLL)*

   Genera el codi C corresponent al fitxer *sdlps*, el compila juntament amb el codi extern i el *linka* en una única llibreria *.dll*.

3. **Establir temps d'execució** *(Simulation-> Simulation time)*

   Estableix el temps de simulació. Es pot determinar la durada o deixar-ho com infinit. Llavors la simulació s'hauria de detenir amb el botó Stop.

4. **Inicialitzar la simulació** *(Simulation -> Initialize)*

   Inicialitza el model. Consisteix en l'execució de tots els processos definits al SDL des de l'estat "*Start*" fins al canvi a un altre estat definit al procés.

5. **Executar la simulació**

   Podem seguir dos procediments:

   - *Execució contínua:* fent clic a *Run* s'executa el model fins al final.
   - *Pas a pas:* prèviament s'ha de marcar la casella *"Step by step".* El botó *Step* executa un pas de la simulació.

8 Arquitectura del Model

A continuació es presenten els diferents components que completen l'arquitectura del model civiSDL, juntament amb la representació en SDL.

8.1 Codi SDL

Tots els diagrames SDL estan situats a la carpeta **doc** del projecte. Quan executem la macro que genera el fitxer *.sdlps*, busca els diagrames a aquesta carpeta i deixa el fitxer creat a l'arrel del projecte.

8.2 El codi extern

El model de simulació CiviSDL necessitava un codi apart del contingut als diagrames SDL. D'aquesta manera es buscava un complement per donar-li més potència al model, definint variables i operacions que al SDL serien difícil de gestionar.

L'ús d'aquest codi es farà per mitjà de crides declarades dins elements *Task* dels processos SDL. No s'utilitzaran referències a variables entre el codi SDL i el codi extern, tota comunicació de variables es farà utilitzant paràmetres a les crides.

El SDLPS està basat en codi C. D'aquesta forma és fàcil integrar el codi extern si l'implementem en C, simplement hem de *linkar* les respectives llibreries del model en SDL i la resta de codi.

La utilització d'un llenguatge relativament limitat com C no és inconvenient per desenvolupar el codi extern. Recordem que SDL és un llenguatge orientat a objectes i no és precís tenir una estructura externa massa complexa per completar el model.

La filosofia de treball és que tot el pes de la simulació sigui portat pel codi en SDL, de forma que el codi extern només emmagatzema determinades variables i executa operacions dins l'àmbit semàntic en que són cridades des de el SDL.

El codi extern està situat dins la carpeta **SDLCodeExternal** del projecte. Per tal de que quedi linkat correctament, s'ha de tenir un fitxer [nom_del_model].h que conté tots els *includes* de la resta de fitxers de codi utilitzats, (en el nostre cas, civiSDL.h). D'aquesta forma el SDLPS compilarà i *linkarà* automàticament el codi extern.

8.3 Altres Inputs i outputs

Hem vist a l'apartat anterior que els fitxers en format Idrisi32 ens proporcionen una entrada i sortida de dades pel autòmat cel·lular. No obstant necessitem més inputs i outputs a banda per tal d'obtenir informació dels altres mòduls.

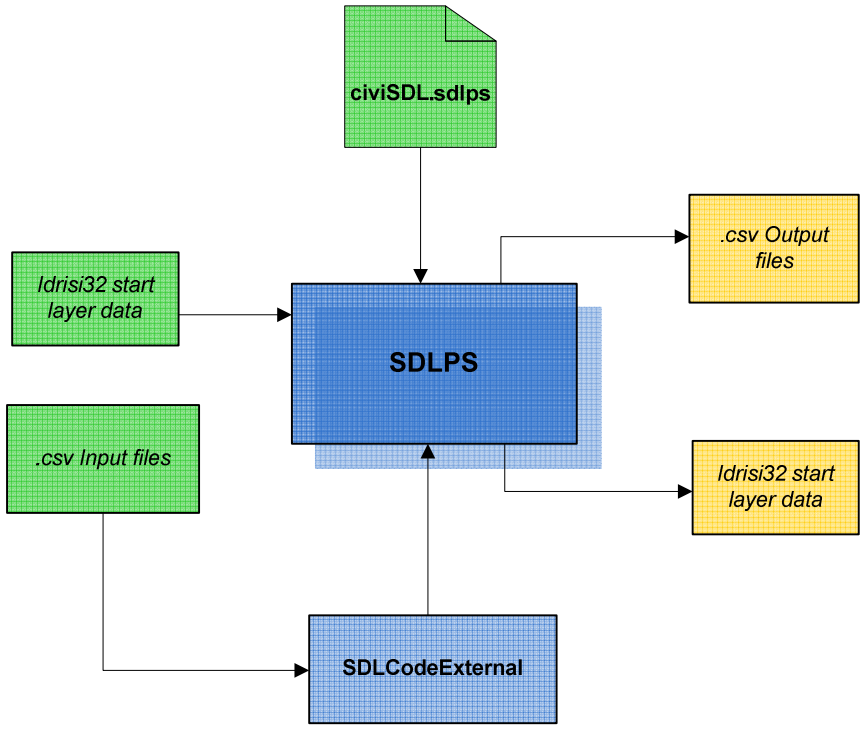
Aquesta entrada i sortida de dades es fan amb fitxers en format .csv que són llegits i escrits pel model per mitjà de funcions implementades al codi extern.

Els fitxers d'entrada i sortida es poden trobar a la carpeta **inputs i outputs** del projecte respectivament.

Per saber-ne més sobre cada fitxer en concret (format i contingut), consulteu la secció *Inputs i Outputs*

8.4 Diagrama del Sistema

A continuació es mostra un diagrama que resumeix els components que integren el model i com es relacionen:



Il·lustració 11: diagrama del sistema

9 Estructura del Model

A continuació s'expliquen resumidament els blocs principals en que està dividit el model. Cada bloc gestiona un àmbit de la ciutat i es correspon amb un bloc equivalent als diagrames SDL.

9.1 Els blocs del model

9.1.1 Bloc d' Entorn

Gestiona la variació del clima a la ciutat i marca el pas del temps, que és mesurat en mesos. Distribueix aquesta informació a la resta de blocs.

Al model rep el nom de **Enviroment.**

9.1.2 Bloc d'Unitats Familiars

Gestiona les famílies i els seus membres. Controlen els següents punts:

- Evolució de la vida
- Naixements
- Consum de productes
- Decisions
    - De producció: quin producte cultivem/fabriquem?
    - Financeres: compra i venda de productes
- Caràcter
- Evolució de la experiència i coneixement

Juntament amb aquests punts, un altre bloc intern gestiona les **relacions** entre solters. Es defineix com es creen les noves famílies i s'independitzen de les anteriors.

9.1.3 Bloc de Propietats

Modela el terreny que conforma la ciutat, les parcel·les del qual pertany a alguna família. Sobre cada porció de terreny tenim les següents dades:

- **Estat del terreny:** lliure, ocupat o en producció.
- **Producte que s'hi genera:** cultiu (blat, productes de l'hort...) o fabricació (sabates, roba, joies..).
- **Amo:** família que la posseeix
- **Qualitat:** factor que dona un valor afegit als productes que s'hi generen.
- **Treballadors i Experiència:** nombre de persones que treballen el producte que hi té assignat, així com l'experiència que acumulen per millorar la quantitat/qualitat del producte.

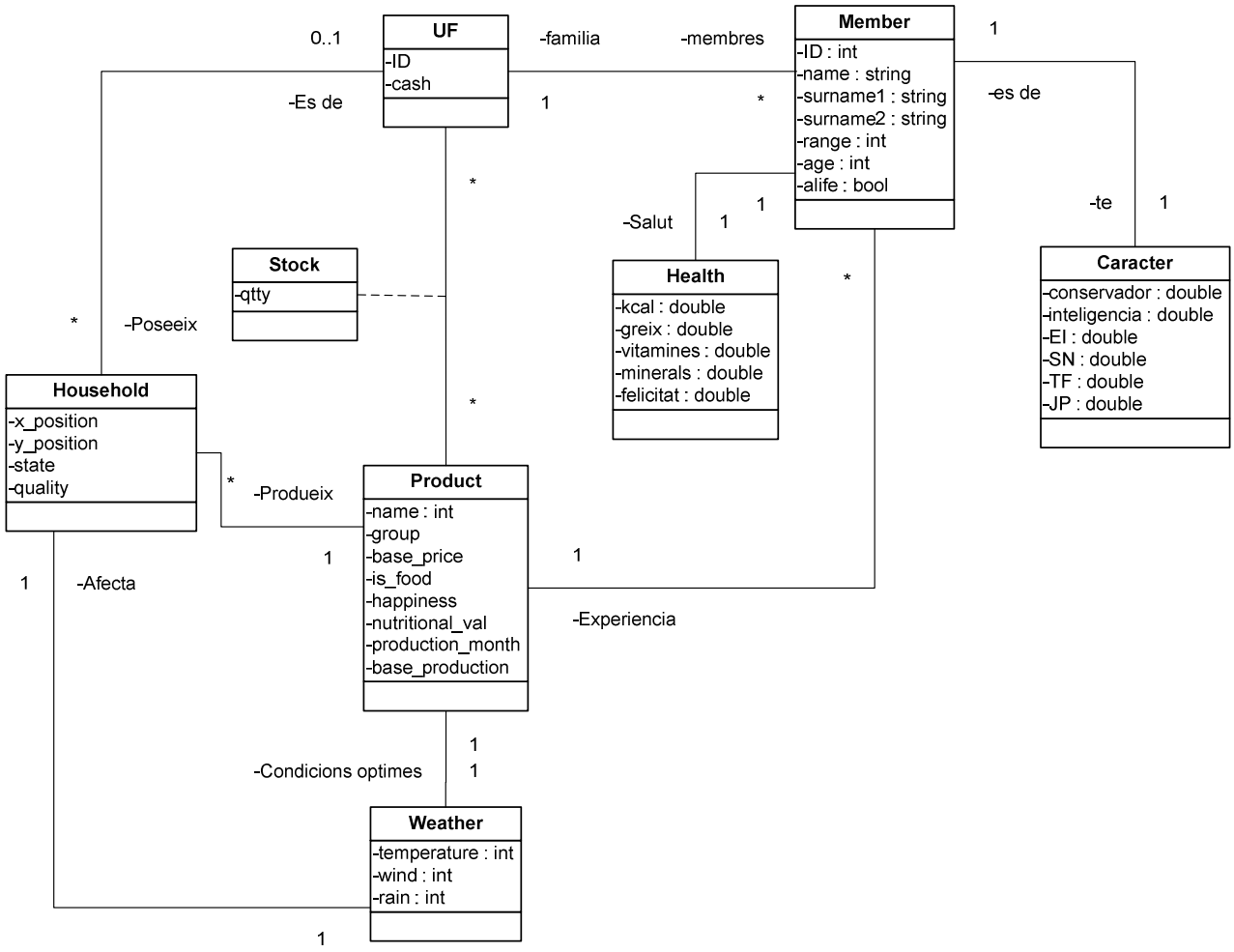
Les propietats de terreny es veuen afectades al clima. Durant tot l'any o en un mes en concret envia la producció a la família que la posseeix.

9.1.4 Bloc de Mercat

Gestiona la compra i venda de productes. Calcula l'oferta i la demanda de l'últim mes, així com la total des de l'inici de la simulació. Aquesta informació s'envia a les famílies per ajudar a prendre decisions financeres.

9.2 Model conceptual

El diagrama que es presenta a continuació representa a alt nivell d'abstracció els conceptes del món real que tenen presència al model civiSDL, així com les seves relacions.



Il·lustració 12: model conceptual

10 Hipòtesis Simplificadores

Les hipòtesis simplificadores són aquelles que ens ajuden a acotar el model, ja que no podem representar tot el món real en el sistema. Aquestes hipòtesis s'han de formular de tal forma que ens facilitin la feina i alhora siguin el més fidel possibles a la realitat. Estan classificades segons el bloc al que pertanyen.

10.1 Unitats Familiars

- N'hi ha tres franges d'edat:
  - **Infant:** entre 0 i 13 anys.
  - **Adolescent:** entre 13 i 17 anys.
  - **Adult:** 18 anys o més.

- Les persones es poden casar a partir dels 16 anys. Quan es forma una parella, han de buscar almenys una propietat on viure.
- Totes les parelles poden tenir fills.
- Les parelles no es poden divorciar.
- Existeixen tres rols diferents dins les famílies: cap de família, parella i resta de membres. El pes de les decisions queda ponderat entre el cap i la seva parella.
- La salut d'un membre depèn del que consumeix i de la edat.

10.2 Enviroment

- Els mesos duren 30 dies.
- El clima el determina la pluja, la temperatura i el vent. Es manté constant durant tot un mes.
- El clima sofreix variacions aleatòries que afecten a la qualitat de les parcel·les, però mai aquestes són catastròfiques.

10.3 Market

- Els preus dels productes baixen si el ràtio de oferta/demanda és >1; altrament baixen.
- El mercat accepta totes les compres i vendes de les Unitats Familiars.

10.4 Household

- Les parcel·les poden produir només un producte alhora.
- Una parcel·la pot canviar de producte, però perd la qualitat acumulada.
- Totes les propietats es veuen afectades de la mateixa manera per el clima.
- Els productes amb que es parametritza el sistema actualment són, segons ordre de necessitat:
    - Cereals
    - Verdures
    - Carns
    - Roba
    - Eines de la llar
    - Objectes d'oci o luxe

- Els membres consumeixen aquests productes en l'ordre plantejat. Si només tenen diners per comprar fins el producte "Verdures", no compraran ni consumiran de la resta de productes.
- La quantitat de producte que s'obté depèn de la qualitat del terreny, el nombre de treballadors i l'experiència acumulada dels treballadors en el producte.

11 Les capes de l'autòmat cel·lular

Com es comenta en anteriors apartats, el terreny de la ciutat modelada està representat amb un autòmat cel·lular, on cada cel·la representa una propietat del terreny . Cada atribut que se li vulgui atribuir a una propietat s'haurà de traduir amb una **capa** de l'AC.

La següent taula resumeix les capes que s'han contemplat al model, el seu comportament i els possibles valors que poden adquirir.

Nom	Descripció	Valors
State	Estat actual de la cel·la, en relació a si està en producció o lliure	1: En venda 2: En Producció
Owner	Unitat familiar propietària de la parcel·la de terreny	-1: Sense propietari Altrament, ID de la Unitat Familiar propietària
Product	Producte que s'obté actualment en el treball dins la cel·la	ID del producte produït
Quality	Qualitat acumulada del terreny, provocat per les inclemències del temps o les condicions del local (segons si és una propietat rural o urbana, respectivament)	Nombre indicador de tipus enter que valora la qualitat afegida
Workers	Treballadors que actualment contribueixen a la producció en la cel·la	Nombre de persones treballant
Experience	Experiència acumulada de la Unitat Familiar responsable en la cel·la amb el producte actual.	Nombre indicador de tipus enter que valora la experiència afegida

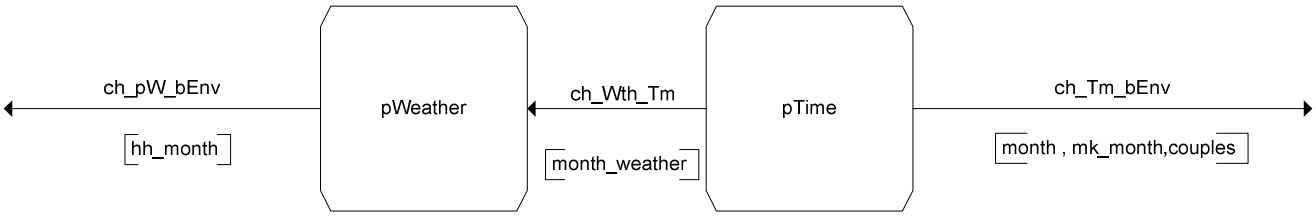
12 Disseny i implementació

12.1 Bloc Enviroment

El bloc *Enviroment* o Entorn gestiona la variació del clima a la ciutat i marca el pas del temps, que és mesurat en mesos. Aquesta informació és distribueix a la resta de blocs.

El bloc ***bEnviroment*** conté dos processos:

- ***pWeather*:** controla el clima.
- ***pTime*:** controla el pas del temps.

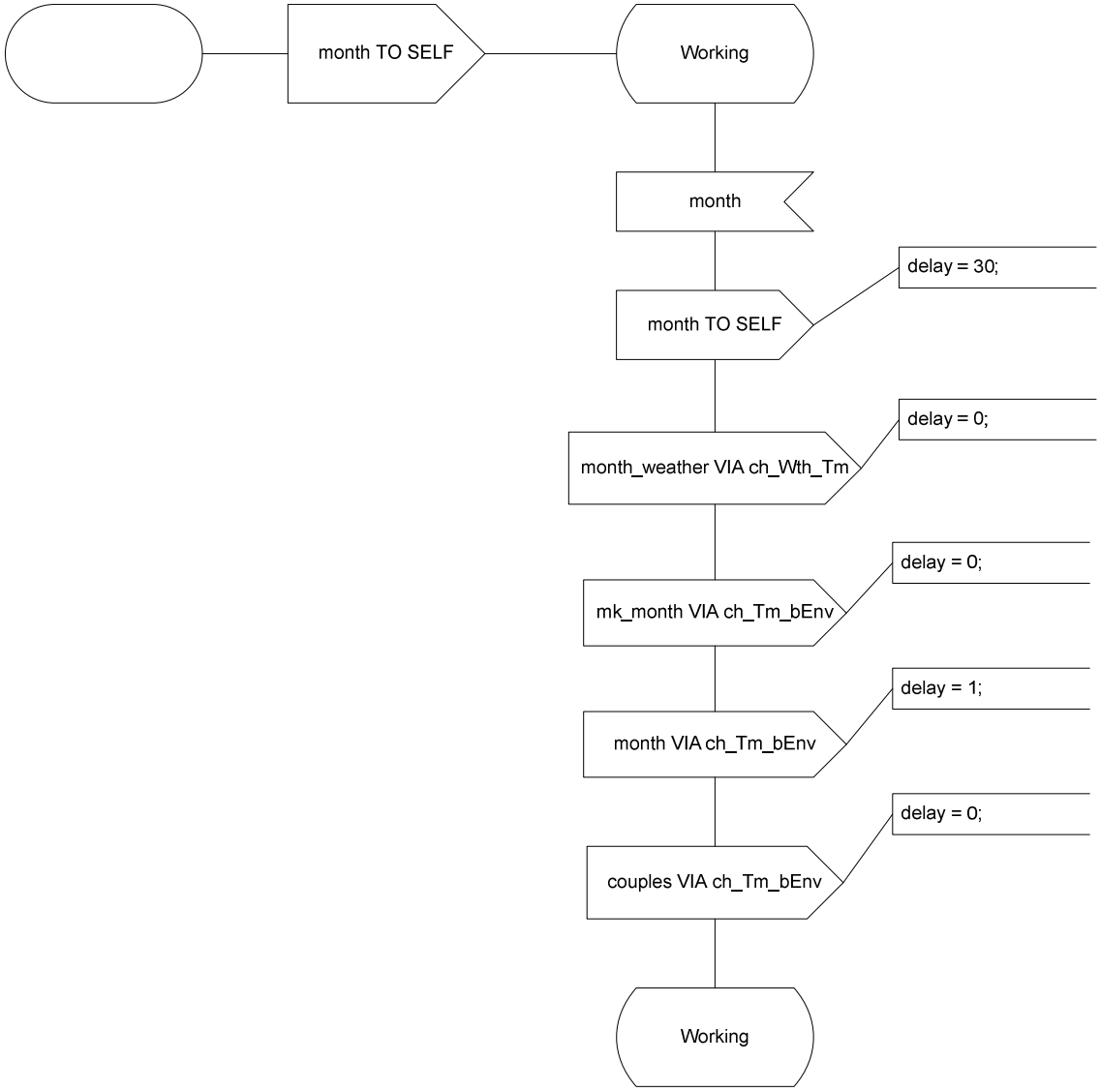


Il·lustració 13: bEnviroment

12.1.1 pTime

Computa els senyals que marquen cada mes. La idea és que amb un *delay* de 30 unitats de temps (cada unitat és un dia) el procés s'auto-envia un senyal *month*. Cada cop que rep aquest senyal ha passat un mes i s'envien senyals cap altres blocs del model per indicar aquest pas del temps.

La gestió del temps en SDL es basa en una extensió proposada a (Fonseca i Casas, Colls i Casanovas, Using Specification and Description Language to define and implement discrete simulation models 2010), (Fonseca i Casas, Colls i Casanovas, Representing Fibonacci function through cellular automata using Specification and Description Language 2010), (Fonseca i Casas, Colls i Casanovas, Towards a representation of environmenal models using specification and description language 2010) i que actualment està en procés d'incorporació en el llenguatge.



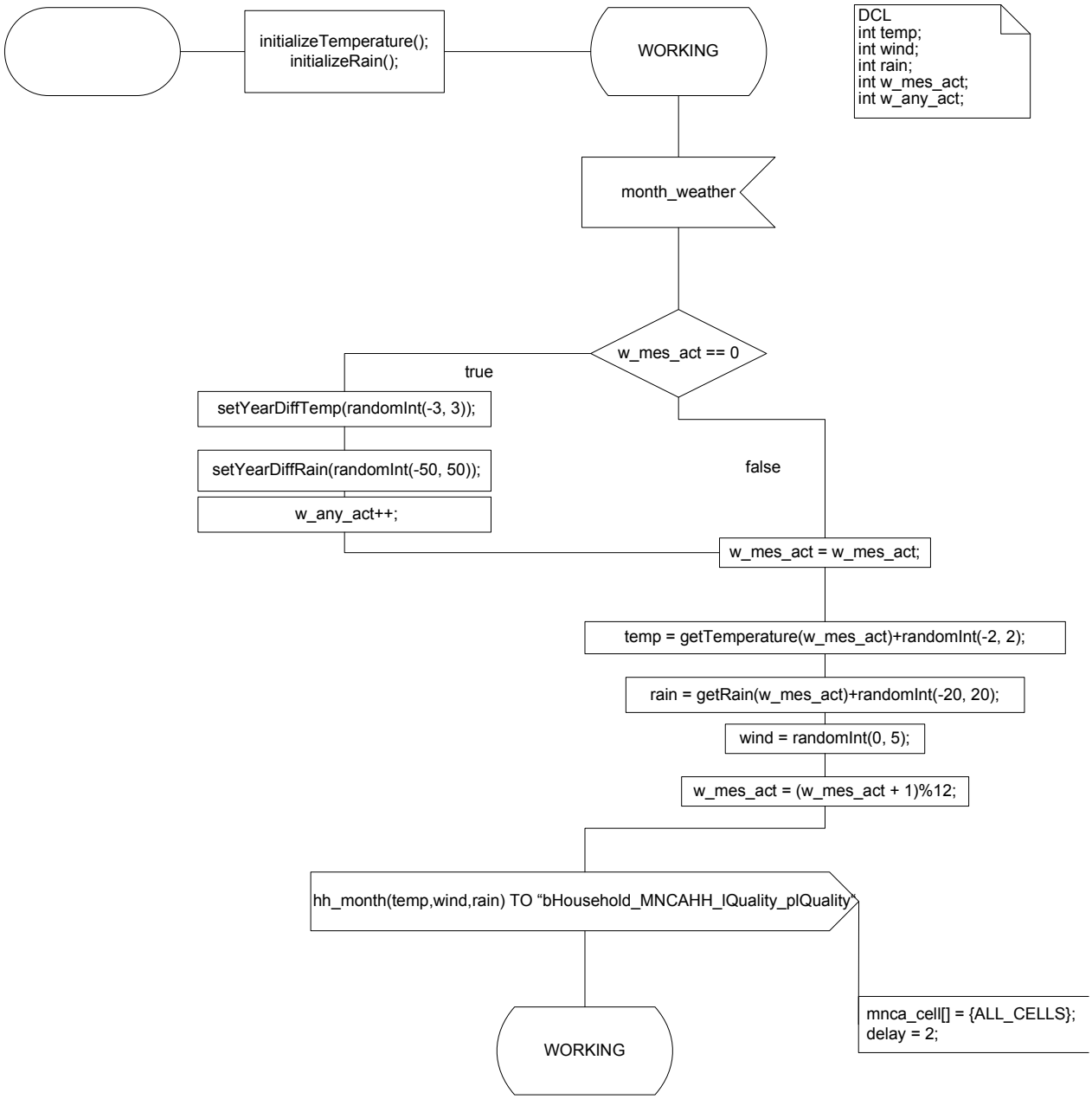
Il·lustració 14: pTime

A banda del *signal* auto-enviat de control, s'envien els següents:

Signal	Destí	Acció que provoca
month_weather	Procés de clima	Generació de clima mensual
mk_month	Procés de mercat	Actualització de variables de control del marcat mensual
month	Procés de Unitat Familiar	Inici de seqüència d'accions de les famílies
couples	Procés de control de relacions	Control de casaments

12.1.2 pWeather

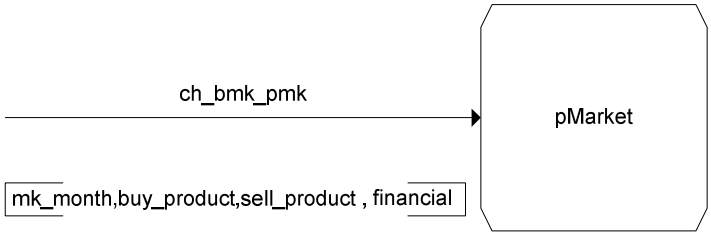
Calcula la temperatura, el vent i la pluja per cada mes. Es defineixen uns valors base per any i mes. Al inici de cada any es computa una desviació que s'afegirà o es restarà als valors base mensuals. Els resultats s'envien a totes les cel·les del bloc de propietats per tal de que es calculi com li afecta el clima.



Il·lustració 15: pWeather

12.2 Bloc Market

El bloc de Mercat **bMarket** gestiona la compra i venda de productes. Calcula l'oferta i la demanda de l'últim mes, així com la total des de l'inici de la simulació. Aquesta informació s'envia a les famílies per ajudar a prendre decisions financeres.



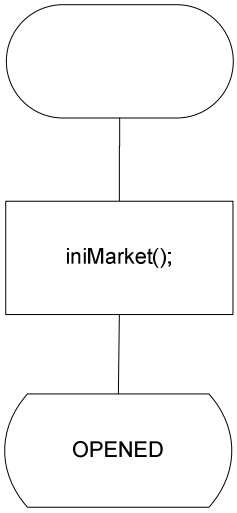
Il·lustració 16: bMarket

El bloc conté un únic procés que gestiona totes les operacions que ha de fer el mercat.

12.2.1 pMarket

Únic procés del mercat, controla la compra, venda i càlcul de preus segons l'oferta i la demanda.

El procés s'inicia amb la crida *iniMarket,* que posa els preus inicials als productes i inicialitza les variables d'oferta i demanda. Tot seguit el procés passa a l'estat OPENED.



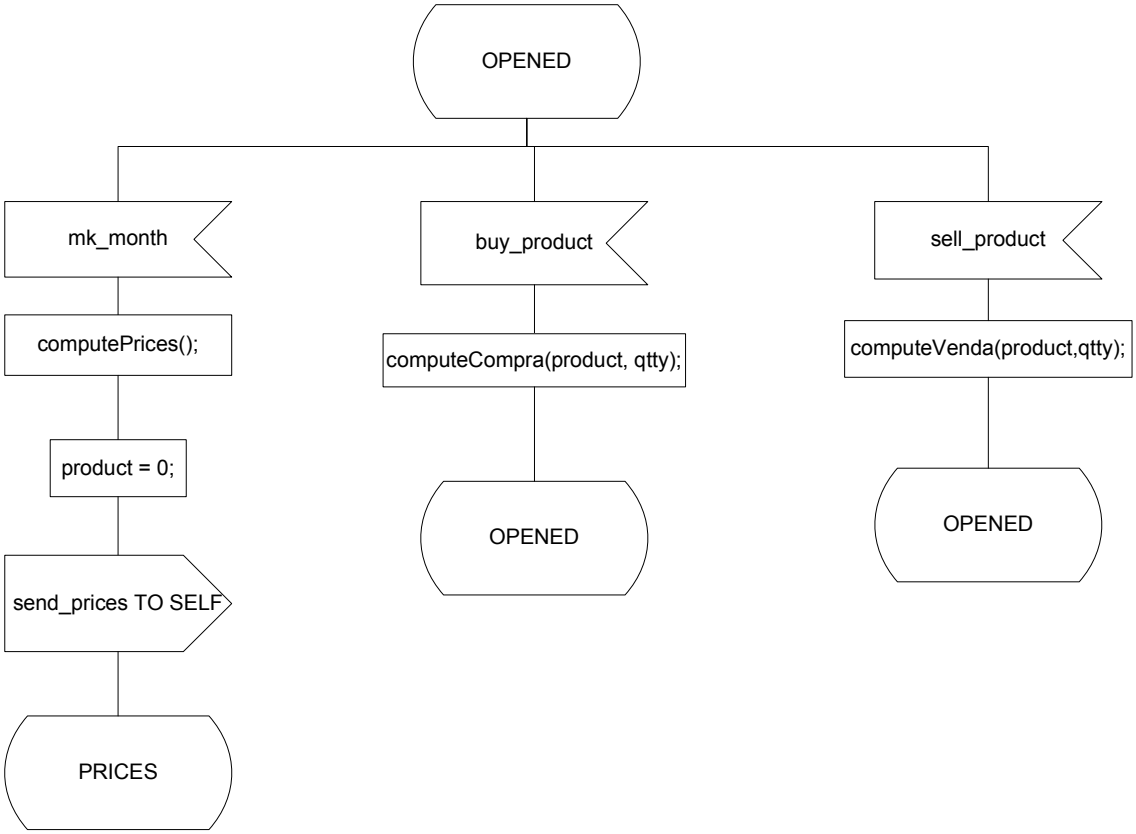
Il·lustració 17: pMarket, estat inicial

A l'estat OPENED el mercat pot rebre comandes de compra i venda amb els senyals *buy_product* i *sell_product*, respectivament.

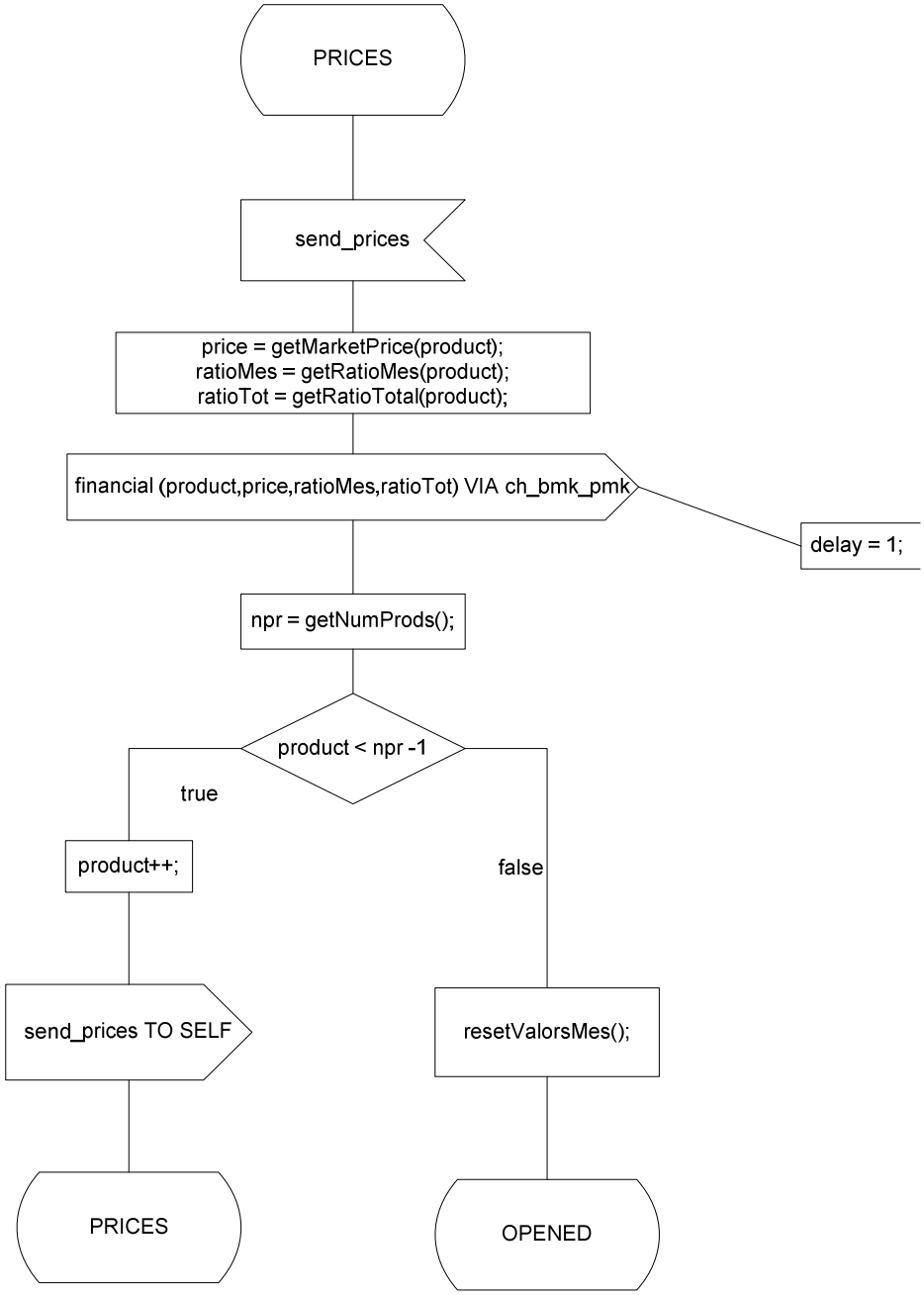
Les funcions *computeVenda* i *computeCompra* que s'executen a continuació actualitzen les variables d'oferta i demanda.

Cada mes es rep el senyal *mk_month* des del bloc Enviroment. Aquest senyal indica que s'han de calcular els preus dels productes, segons l'evolució de l'oferta i la demanda.

Un cop es calculen els preus amb la funció *computePrices,* el procés entra en un cicle iteratiu iniciat pel senyal *send_prices*, gestionat dins l'estat PRICES. Tot seguit s'explicarà el funcionament.



Il·lustració 18: pMarket, estat OPENED



Il·lustració 19: pMarket, estat PRICES

El procés s'inicia amb el senyal *send_prices* (enviat des de OPENED) amb la variable product a 0. S'obtenen els valors financers pel producte inicial i s'envien cap al bloc de les Unitats Familiars amb el senyal *financial*.

Tot seguit mirem si tenim més productes al sistema per enviar els preus. En cas de que sigui així incrementem la variable de control i repetim el procediment auto-enviant de nou el senyal *send_prices*. El procediment finalitza quan ja no queden més productes per tractar. Llavors es fa un *reset* als valors mensuals i tornem a l'estat OPENED, fins al mes vinent.

12.3 Bloc Unitats Familiars

12.3.1 Agents múltiples - Problemàtica

El bloc de les Unitats Familiars és potser el més complex del model. El fet de estar implementat amb múltiples agents fa que sigui un bloc sofisticat i alhora potent.

Durant la implementació del model, ens vam trobar amb un problema al SDLPS que no ens permetia comunicar-nos amb les instàncies creades a partir de l'operació CREATE de SDL.

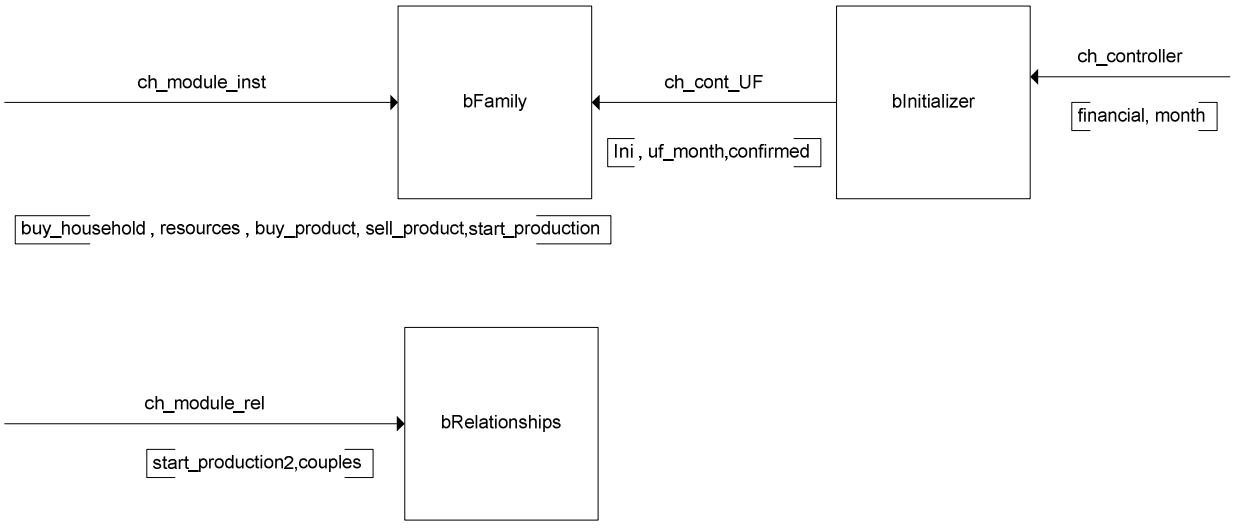
És un problema molt difícil de resoldre, ja que treballar amb instàncies d'agents comporta una gestió interna del SDLPS molt complexa, sobretot per controlar el bon funcionament dels senyals.

Com que aquesta funcionalitat està en desenvolupament, he hagut d'adaptar els processos **pActions** i **pMember** (els dos pensats per ser instanciables) per tal de que gestionin totes les famílies i membres del sistema.

Quan parlem d'aquests processos en el seu apartat corresponent, s'explicaran les adaptacions que s'han fet per resoldre aquest problema.

12.3.2 El bloc d'Unitat Familiar

El primer nivell del bloc UF ja conté 3 blocs amb funcionalitats diferents:

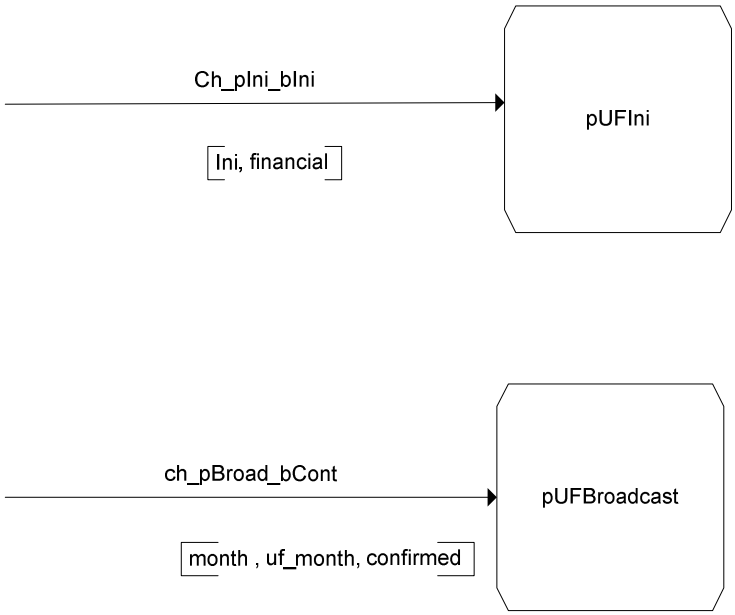


Il·lustració 20: bFamilyModule

Resumidament, *bFamily* conté la gestió de les Famílies i els Membres. El bloc *bInitializer* s'encarrega de facilitar el control de les famílies en general i *bRelationships* gestiona la creació de noves famílies. A continuació s'aprofunditza sobre el contingut de cada bloc.

12.3.3 bInitializer

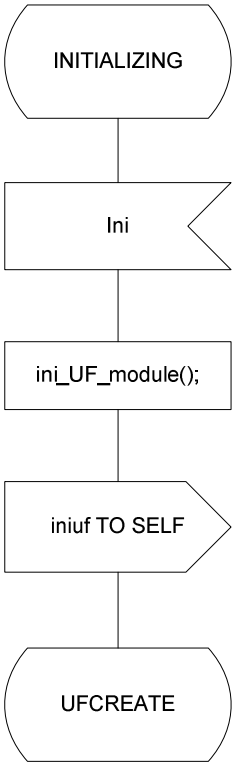
Aquest bloc va sorgir amb la necessitat d'un punt central que gestionés la creació de famílies, així com la comunicació centralitzada amb totes les que existeixen. Alhora s'aprofita per tasques comunes a tota la població de la ciutat.



Il·lustració 21: bInitializer

12.3.4 pUfIni

Inicialitza les famílies del model i gestiona tasques comunes per totes.



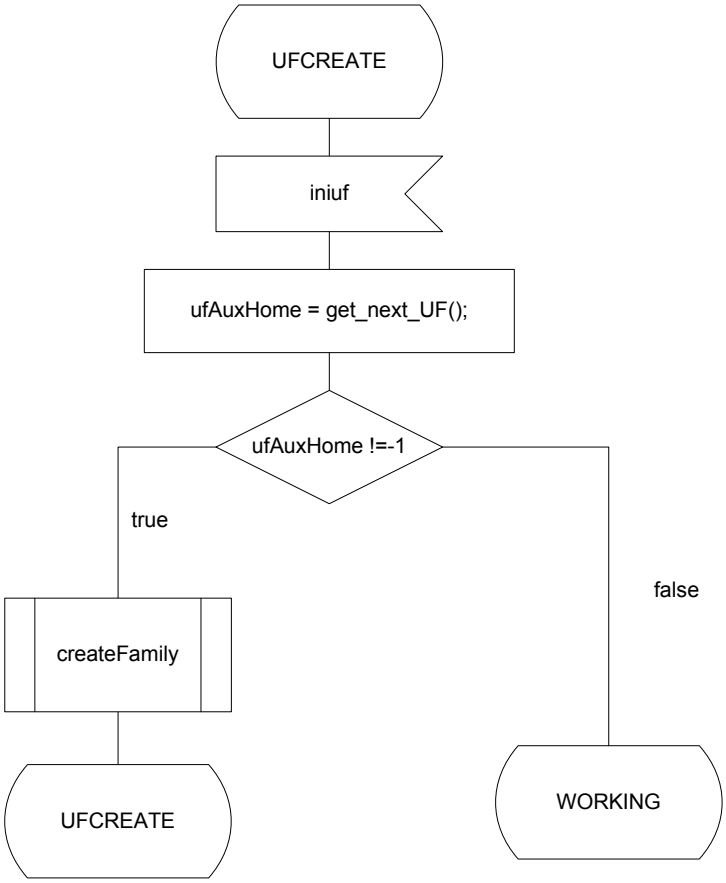
Il·lustració 22: pUFIni, estat INITIALIZING

La funció *ini_UF_Module* inicialitza les estructures de suport al codi extern relacionades amb les famílies. També llegeix els fitxers uf.csv i members.csv per crear les famílies i els membres del punt inicial.

De la mateixa manera, la funció *iniProductInfo* llegeix els fitxer products.csv per obtenir la informació dels productes. És necessària al mòdul UF per operacions com el consum de productes.

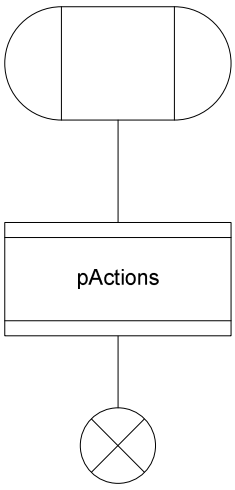
Un cop es fan les inicialitzacions, el procés passa a estat UFCREATE. Aquest estat gestiona la creació de tots els agents de **pActions** (procés que controla les accions de cada UF).

Com s'ha comentat a l'inici de l'apartat, aquest estat seria necessari només amb la implementació utilitzant la eina CREATE. Amb l'actual implementació no és necessari, tot i que el disseny és el següent:



Il·lustració 23: pUfIni, estat UFCREATE

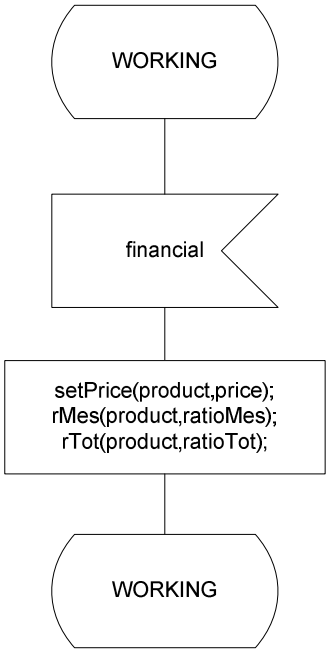
On el procedure *createFamily* és el següent:



Il·lustració 24: procedure createFamily

La seva única funció es crear una instància de *pActions*.

Finalment, tenim l'estat WORKING, on la seva funció es rebre la informació financera rebuda des de el bloc de mercat.



Il·lustració 25: pUFIni, estat WORKING

Rep informació financera per que les famílies la obtinguin i serveixi com a factor de decisió alhora de comprar productes o canviar-ne la producció.

Aquest últim *signal* és un exemple que es pot fer servir per futures implementacions. Enviem una informació que totes les famílies necessiten a un únic punt consultable per totes. Així ens evitem enviar cada cop tants *signals* com famílies n'hi ha.

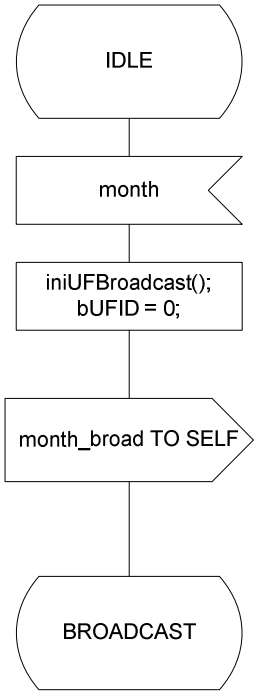
12.3.5 pUFBroadcast

Aquest procés ens proporciona un sistema per controlar totes les famílies del sistema i enviar-li un senyal. L'estat inicial només inicia el procés cap un estat passiu.

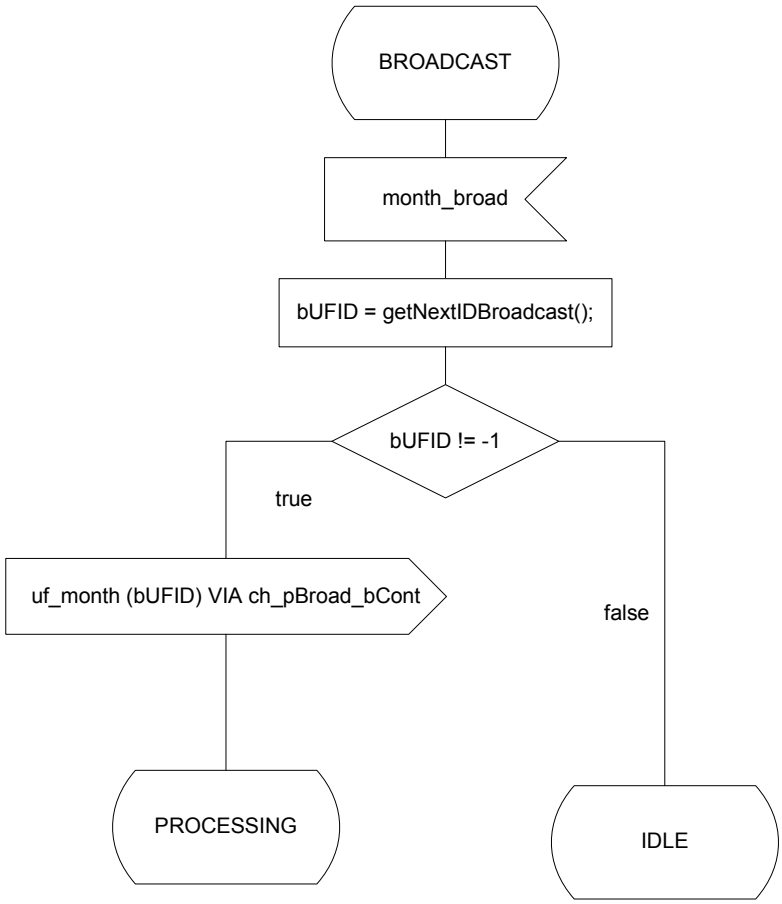
Il·lustració 26: pUFBroadcast, estat inicial

El *broadcast* s'inicia quan rebem el senyal *month* del control de temps. Com en anteriors diagrames, s'inicia el procés amb un senyal auto-enviat anomenat month_broad.

La funció *iniUFBroadcast* inicialitza les variables externes per portan el control del procés. El codi extern és el que gestiona la llista de ID's de UF actives al model.

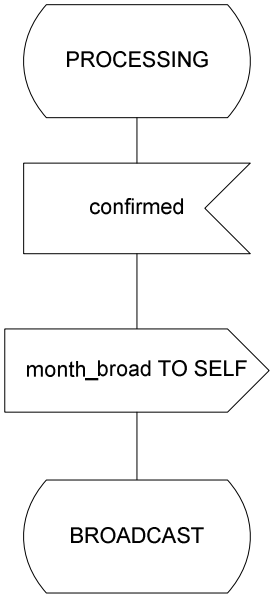


Il·lustració 27: pUFBroadcast, estat IDLE



Il·lustració 28: pUFBroadcast, estat BROADCAST

El codi extern ens proporciona per ordre els ID's de les Unitats Familiars actives. Ens retornarà un -1 quan ja no n'hi hagi més i per tant hem d'acabar el *broadcast*. El procés passa a estar PROCESSING:



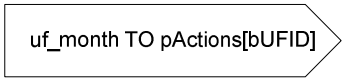
Il·lustració 29: pUFBroadcast, estat PROCESSING

Cada cop que enviem un senyal *uf_month* al bloc *pActions* per processar les accions mensuals de cada família, hem d'esperar a que aquest acabi i ens retorni el senyal *confirmed* per enviar el següent broadcast.

El motiu és que el procés *pActions* passa per diversos estats en la gestió mensual. Si enviem tot seguit un senyal d'inici per la família següent, ens trobarem que el senyal no podrà ser rebut encara per *pActions* i es perdrà.

Aquest sistema no és necessari si treballem amb instàncies múltiples, ja que podem tractar en paral·lel cada UF (tenim un *pActions* per família).

De la mateixa forma, si utilitzem instàncies múltiples, el senyal cap a *pActions* ha de tenir un aspecte com aquest:

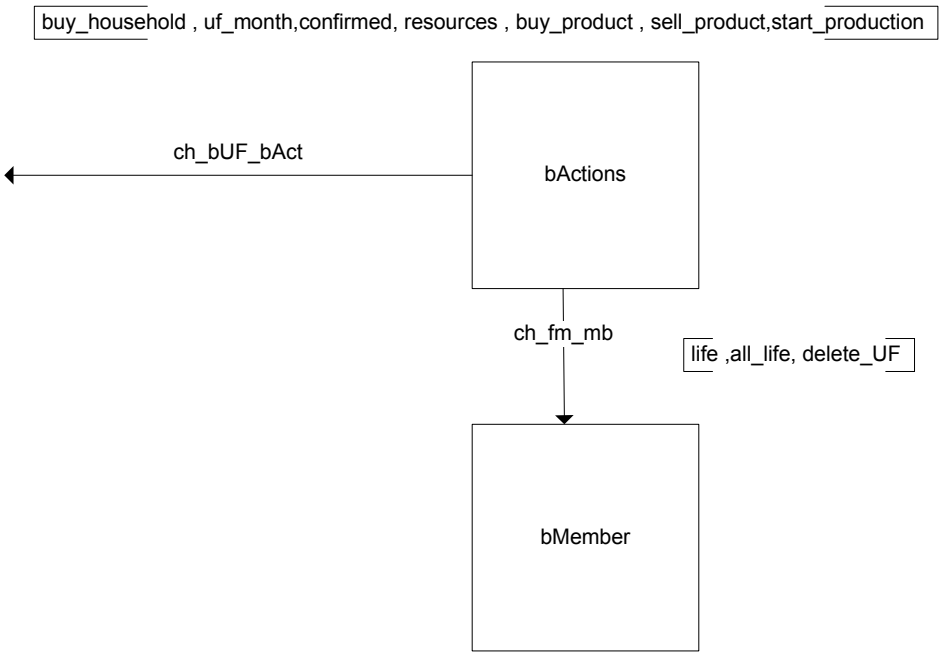


Il·lustració 30: senyal cap pActions en cas de instàncies múltiples

On entre claudàtors indiquem el ID de la UF. No cal passar-li com a paràmetre el ID, ja que serà un *DCL* del procés *pActions.*

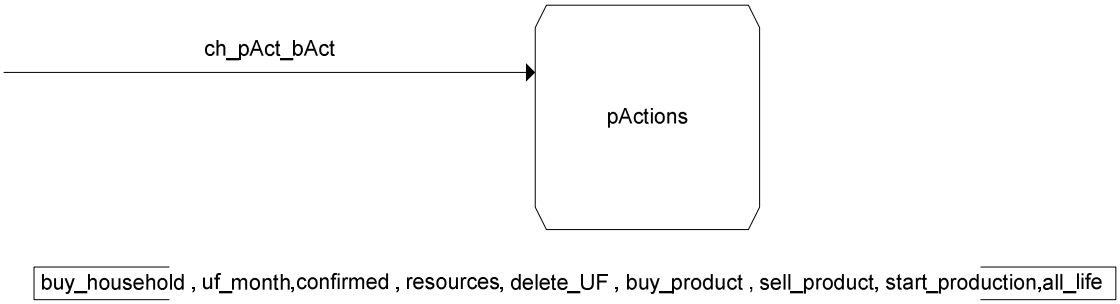
12.3.6 bFamily

Aquest bloc controla les Unitats Familiars i els Membres. Dins hi trobem un altre nivell amb dos blocs: *bActions i bMember.*

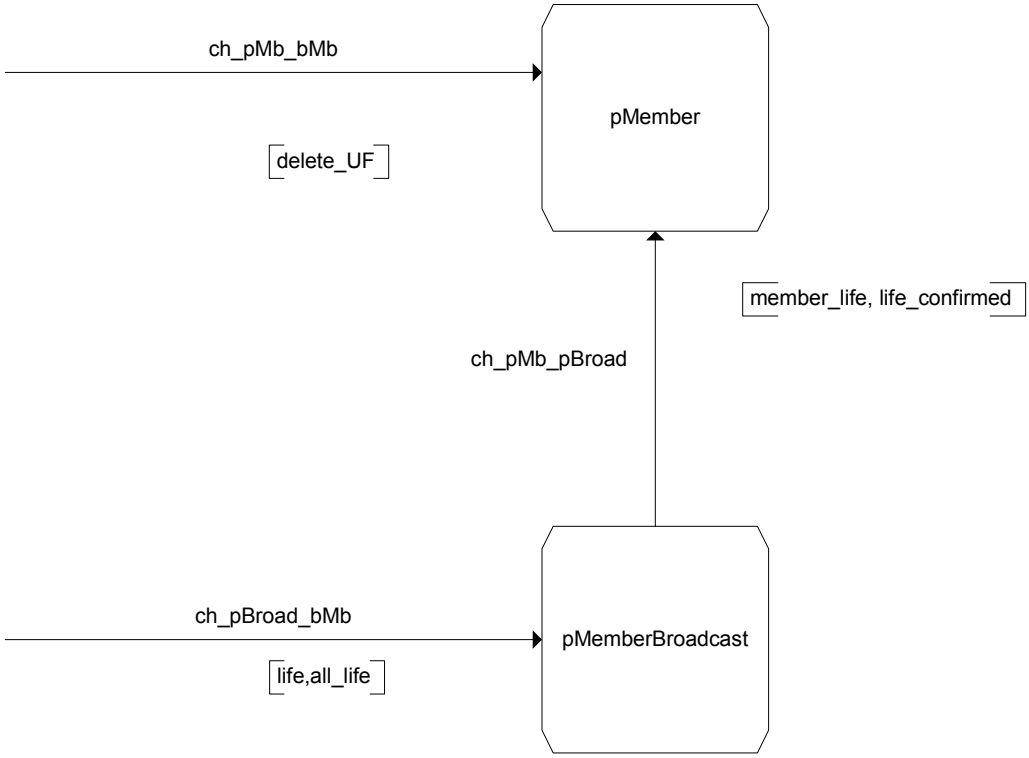


Il·lustració 31: bFamily

Aquests blocs contenen un procés *pActions* y *pMember* respectivament; tot seguit se n'explicaran els detalls.



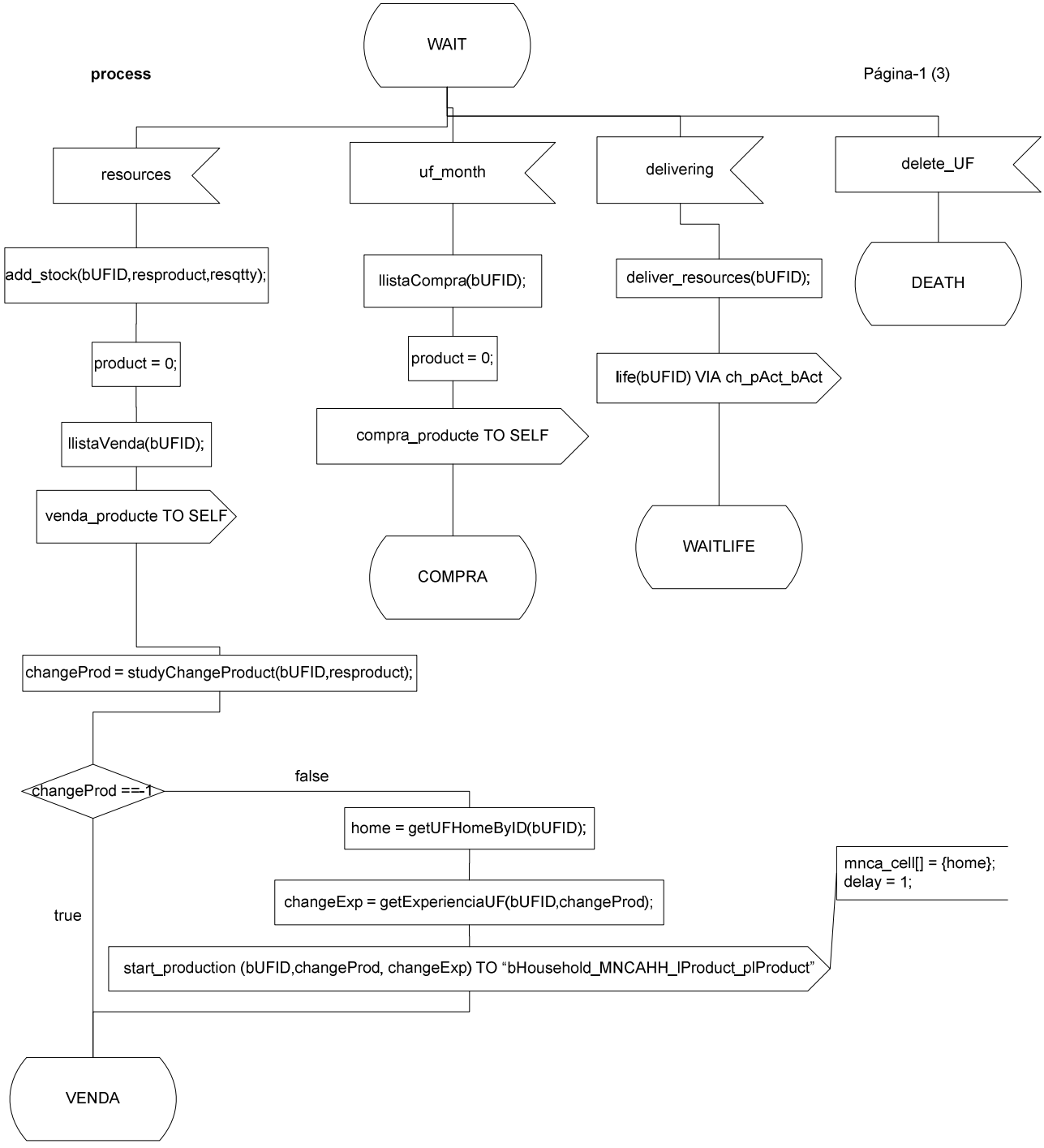
Il·lustració 32: bActions



Il·lustració 33: bMember

12.3.7 pActions

És el procés que computa totes les accions que pot dur a terme una unitat familiar. Passa per diversos estats segons la operació a executar, que ve determinada pels *signals* rebuts des de fora. L'estat WAIT recull diverses senyals que inicien aquestes tasques:



Il·lustració 34: pActions, estat WAIT

A continuació aprofundirem en el funcionament del procés segons la operació que es realitza.

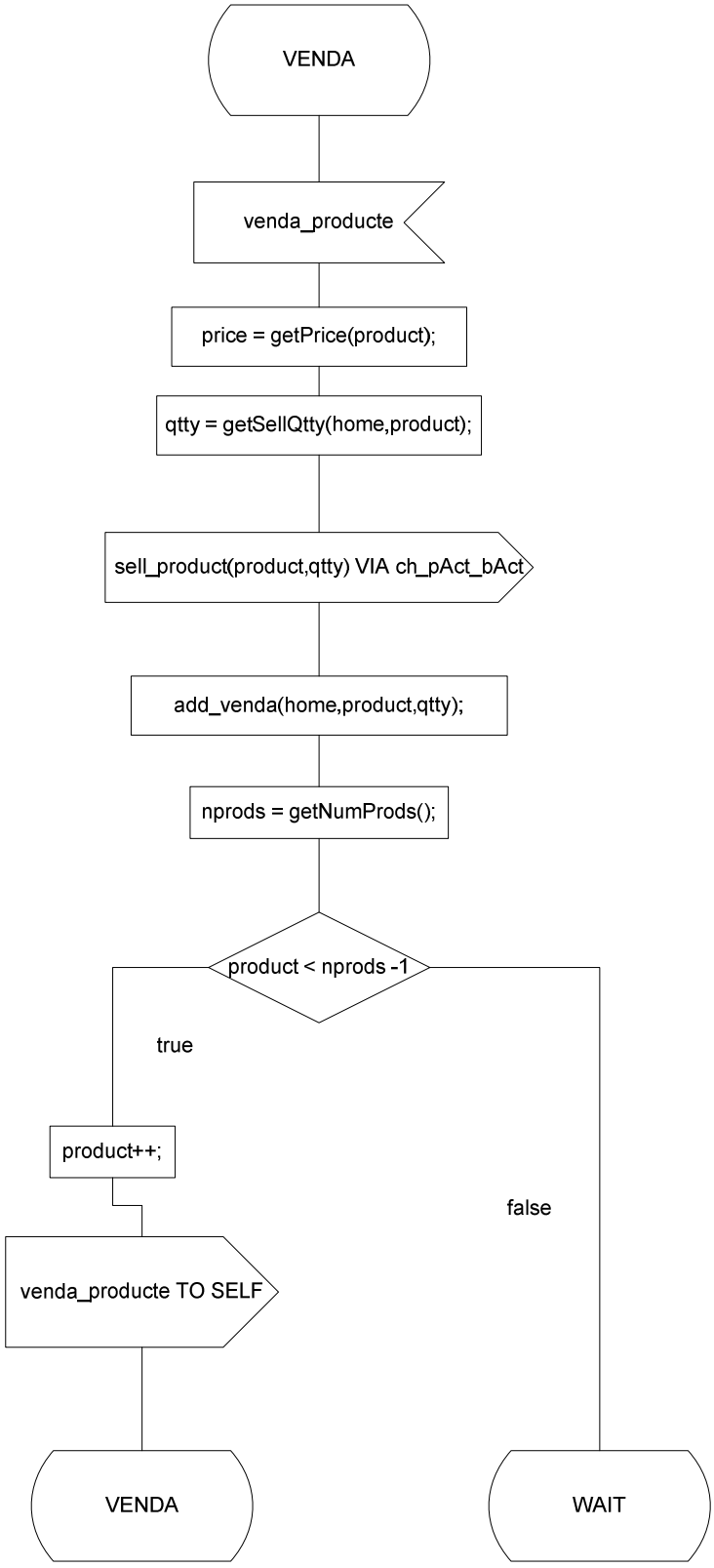
12.3.7.1 Recollida i venda de recursos

La senyal *resources* és enviada des del bloc *Household*, portant la producció del terreny que li pertany a la família. Aquests nous recursos s'afegeixen al *Stock* de la família.

Tot seguit s'executa la funció *llistaVenda*, que calcula el *Stock* sobrant per poder vendre'l. Per fer aquest càlcul s'utilitza un marge mensual, definit als fitxer de constants (per defecte,2). Si la família té excedents de més de x mesos, podrà vendre al mercat aquest sobrant.

En aquest punt s'estudia si a la família li convé canviar de producte. Si és així, es rep el ID del nou producte, amb l'experiència que pot tenir la família i envia el senyal *start_production* per fer el canvi i notificar-lo al autòmat cel·lular.

Per iniciar la venda de productes, el procés s'envia a sí mateix el senyal *venda_producte* i es passa el procés a l'estat VENDA.



Il·lustració 35: pActions, estat VENDA

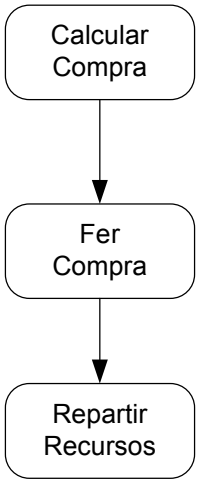
Com es pot veure al procés, s'itera amb aquesta senyal tants cops com productes n'hi hagi, amb la variable *product* com a control de la iteració. Amb la crida a la funció *add_venda* es constata la venda a les variables del codi extern de la família.

Per acabar, s'envia el senyal *sell_product* cap al bloc **Market,** amb el producte i la quantitat que es ven com a paràmetre.

Quan s'acaba la iteració, el procés torna a l'estat d'espera WAIT.

12.3.7.2 Consum de productes

Cada mes les famílies han de repartir entre els seus membres els recursos que necessita per mantenir un bon nivell de salut i felicitat. Resumidament, la seqüència que s'intenta reproduir al model seria la següent:



Il·lustració 36: seqüencia de consum

Les operacions s'inicien amb l'arribada del senyal *month*, rebut des del bloc **Enviroment.**

Es pot donar que el *Stock* de que disposa la família no és suficient per satisfer el consum mensual de la família. Per detectar això, utilitzem la funció *llistaCompra.* És possible que aquesta llista es passi de les necessitats i es comprin més productes dels necessaris (sobretot d'aquells que aporten felicitat com els productes de luxe). Depèn del caràcter dels membres de la família, si són més o menys conservadors.

Un cop s'ha computat la llista de la compra, cal fer-la comunicant-nos amb el bloc **Market**. El procediment és anàleg al de la Venda explicat a l'apartat anterior, solament que aquest cop els senyal involucrat s'anomena *sell_product* i s'opera sobre l'estat COMPRA.

Il·lustració 37: pActions, estat COMPRA

Un cop hem fet la compra, cal repartir els recursos entre els membres en proporció al que necessiten. Aquesta acció comença amb el senyal *delivering*, enviada a sí mateix pel procés quan acaba el procés de compra.

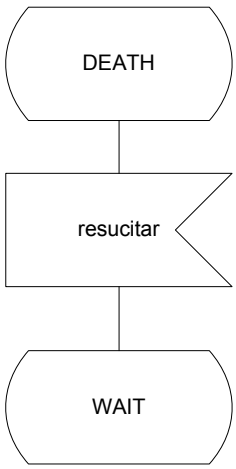
Els càlculs es fan a la funció externa *deliver_resources*, on es calcula i es posa a l'estructura *Member* (implementat al codi extern) de cada membre la quantitat de cada producte que consumirà. Al igual que amb la compra, és possible que un membre consumeixi més del necessari en productes que aporten més felicitat.

El següent pas consisteix en que cada membre "consumeixi" la seva part i es computi l'efecte a la salut. Això és responsabilitat del bloc **bMember** i s'inicia enviant la senyal *life* a cada membre de la família, per mitjà d'un *broadcast* que es detallarà als següents apartats.

12.3.7.3 Dissolució de la família

En cas de treballar amb instàncies múltiples, hem de terminar el procés *pActions* si la família s'ha de dissoldre. En l'actual implementació no és possible ja que el procés gestiona totes les famílies.

Quan tots els membres de la família moren o l'abandonen, s'ha de dissoldre. L'últim membre actiu indicarà des de **pMember** o **pRelationships** aquest fet amb la senyal *delete_UF*. Llavors passa a l'estat DEATH del qual no sortirà i per tant, no rebrà senyals d'acció.

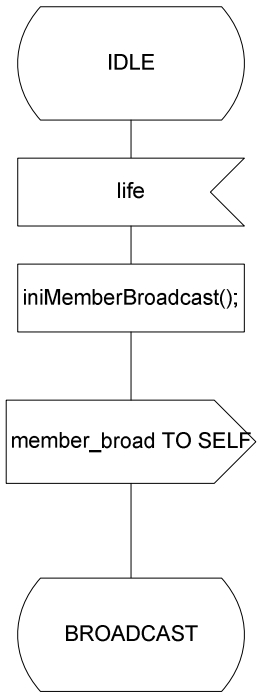


Il·lustració 38: pActions, estat DEATH

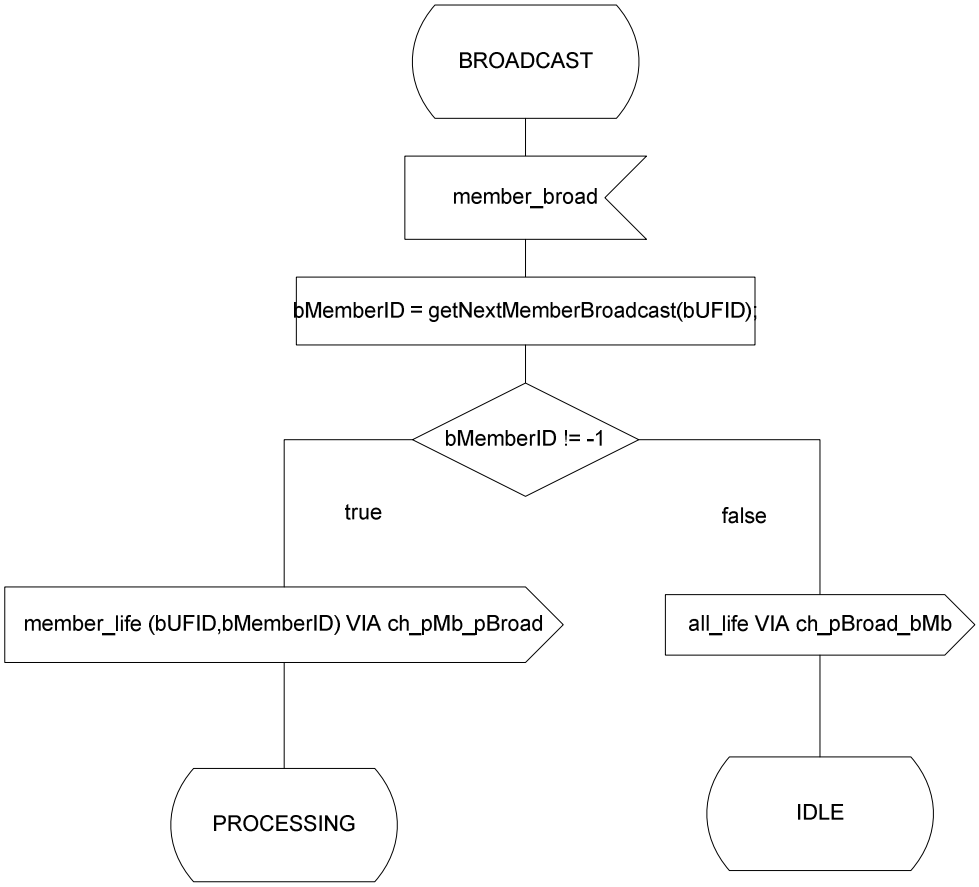
12.3.8 pMemberBroadcast

Com hem vist a l'anterior apartat, el procés *pActions* s'ha de poder comunicar amb tots els membres que hi pertanyen. Per això necessitem un *broadcast* que ens ho gestioni. El procediment és molt semblant al que hem vist anteriorment per les famílies, amb la única diferència de que el codi extern necessita també el ID de la família que estem tractant per obtenir només els *ID's* de membre relatius a la família en tractament.

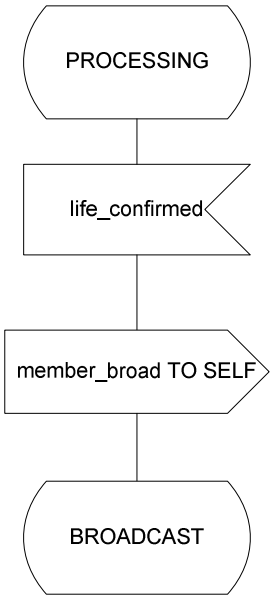
Il·lustració 39: pMemberBroadcast, estat inicial



Il·lustració 40: pMemberBroadcast, estat IDLE

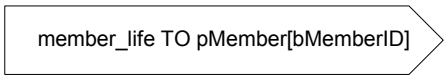


Il·lustració 41: pMemberBroadcast, estat BROADCAST



Il·lustració 42: pMemberBroadcast, estat PROCESSING

Si en futures implementacions utilitzem instàncies múltiples pels Membres, valen les mateixes premisses que s'han plantejat per les famílies: no cal fer l'espera per enviar el següent senyal *member_life ,* no es necessiten els paràmetres i el senyal ha de tenir un aspecte diferent:

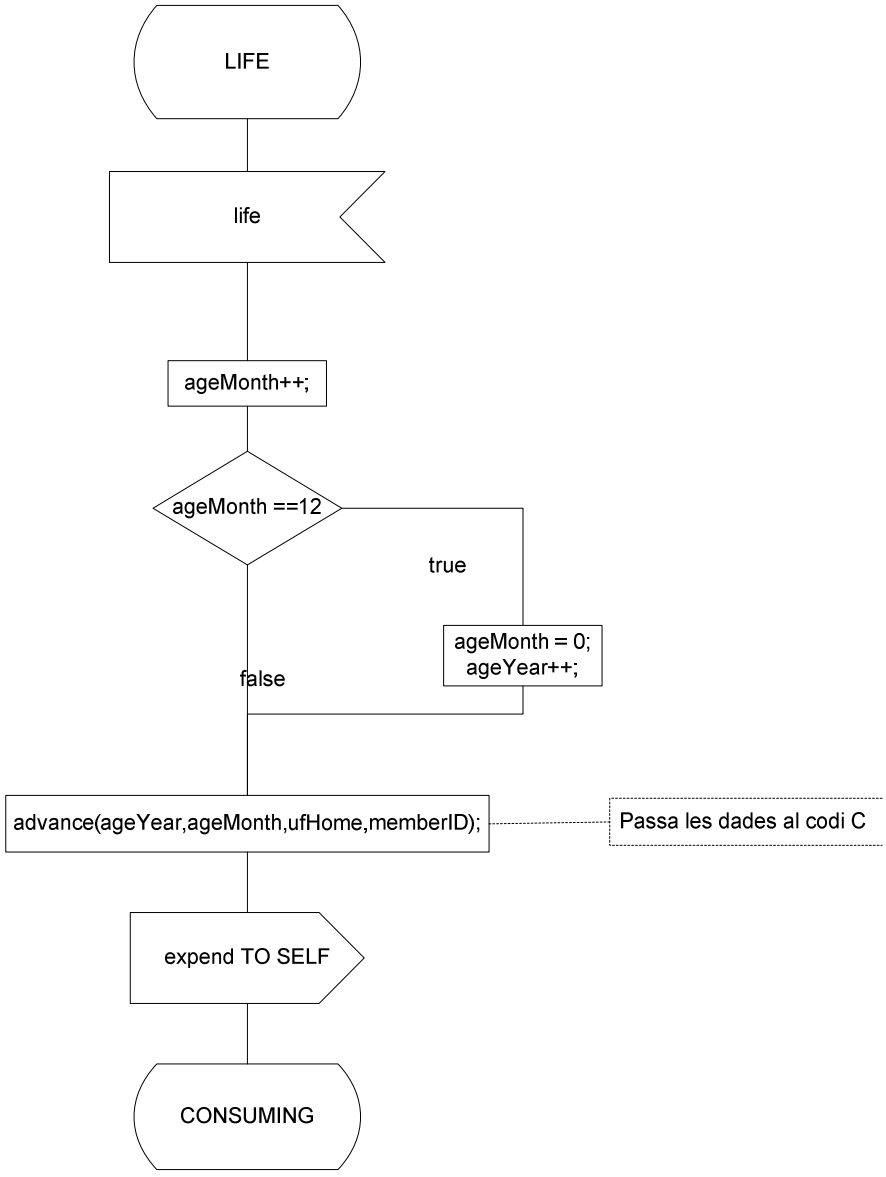


Il·lustració 43: senyal member_life es cas d'utilitzar instàncies múltiples

12.3.9 pMember

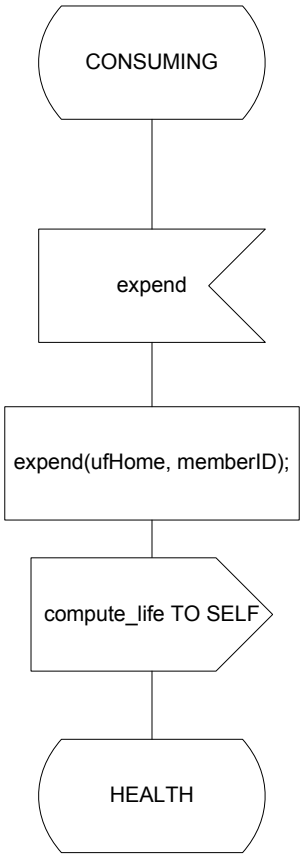
És el procés que controla els individus del model. Les accions que processa estan relacionades amb el consum, la salut i com a conseqüència la vida.

L'estat LIFE rep el senyal *life* des del bloc **pActions.** Com aquest senyal té periodicitat mensual, s'aprofita per actualitzar l'edat del membre. A continuació s'envia el senyal *expend* a ell mateix, que serà rebut a l'estat CONSUMING.



Il·lustració 44: pMember, estat LIFE

El consum es fa amb la funció *expend,* a partir de la quantitat per producte que havia estat calculat al procés **pActions**. Bàsicament el que fa és calcular els dèficits o superàvits de vitamines, greixos, minerals, kilocalories i felicitat; es fa en relació al consum que s'ha fet i el mínim requerit.

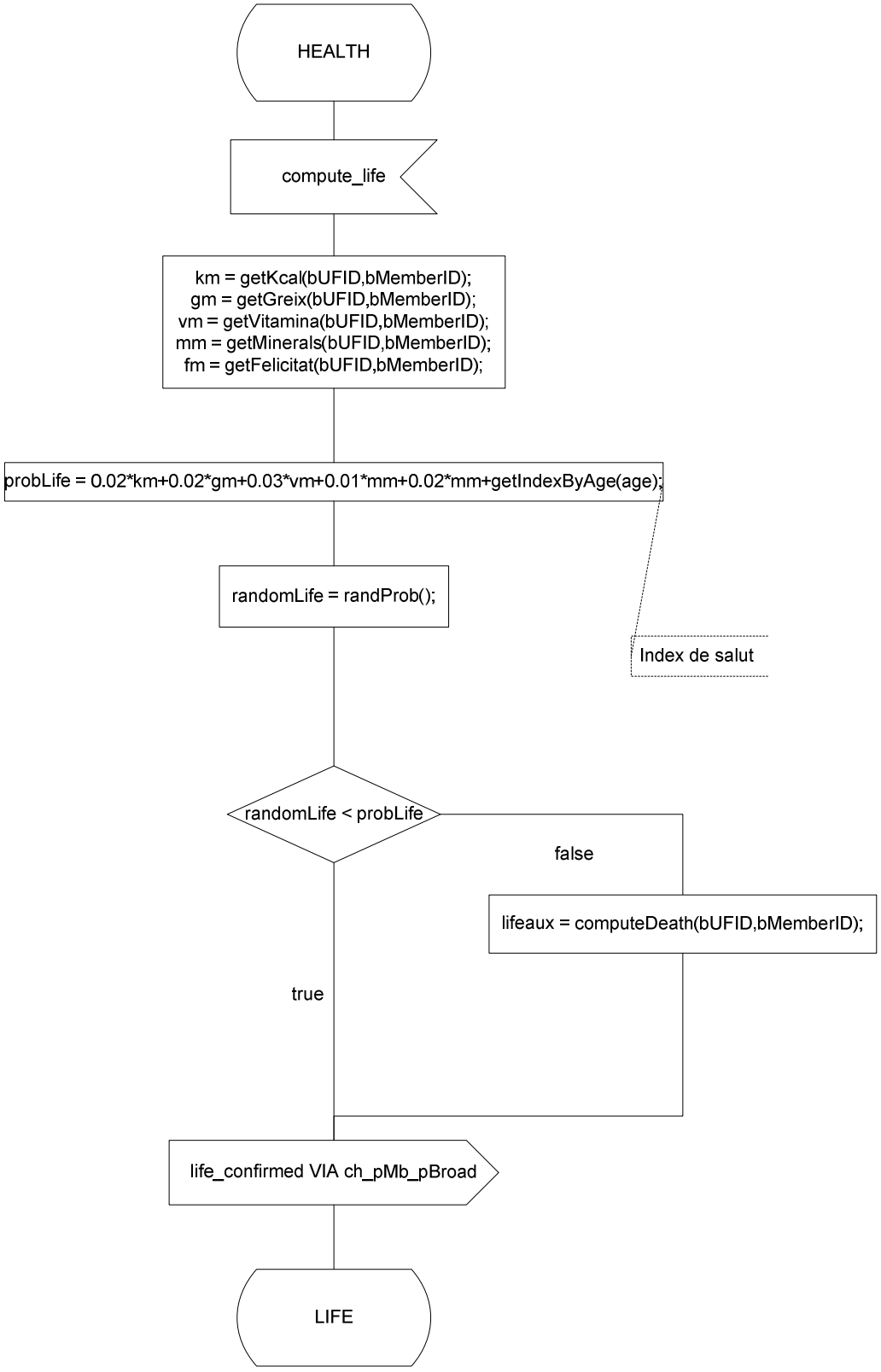


Il·lustració 45: pMember, estat CONSUMING

Cada membre té un ràtio de cadascun d'aquests factors, que varia segons el consum del mes. El valor òptim està al 1, tot i que aquest valor es pot superar i seria perjudicial per la salut global, excepte amb la felicitat.

Un cop s'ha fet aquest càlcul, s'envia la senyal *compute_life* a sí mateix i es posa a l'estat HEALTH, on es calcularà la salut a partir dels factors abans obtinguts.

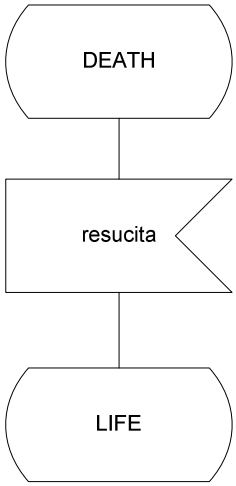
Per calcular el factor de salut es tenen en compte els ràtios esmentats, juntament amb l'edat i un component aleatori. La suma ponderada d'aquests factors decideix sobre la vida o mort del membre.



Il·lustració 46: pMember, estat HEALTH

En cas de mort, de la mateixa manera que passa amb les Unitats Familiars, hem de passar el procés a un estat DEATH si treballem amb instàncies de membres, cosa que no té sentit amb l'actual implementació, ja que el procés gestiona tots els membres.

En aquesta hipotètica implementació s'hauria de mirar si és l'últim membre de la família i enviar el senyal *delete_UF* al bloc **pActions**. El procés queda a l'estat DEATH d'on no sortirà si no arriba el senyal *resucita* .

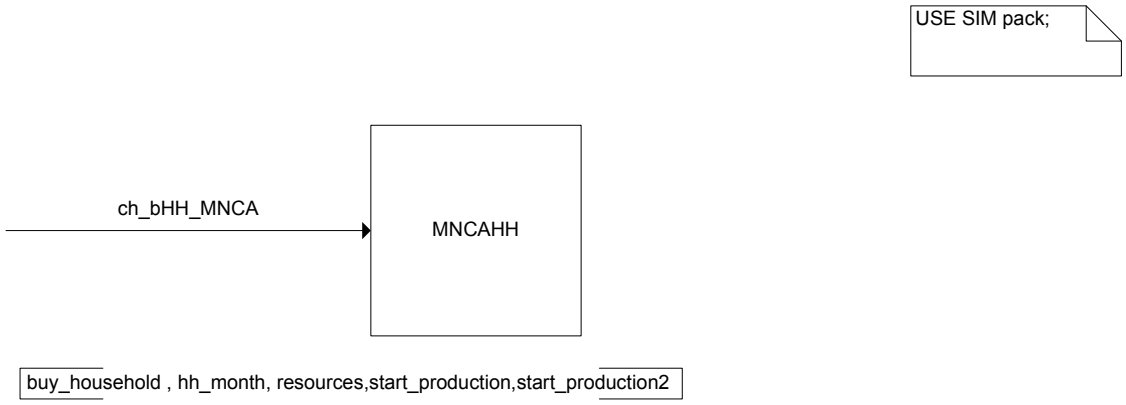


Il·lustració 47: pMember, estat DEATH

12.4 Bloc Household

El bloc *Household* modela les propietats de terreny de la ciutat com un autòmat cel·lular.

El bloc encapsula l'autòmat cel·lular, com es veu a la figura:

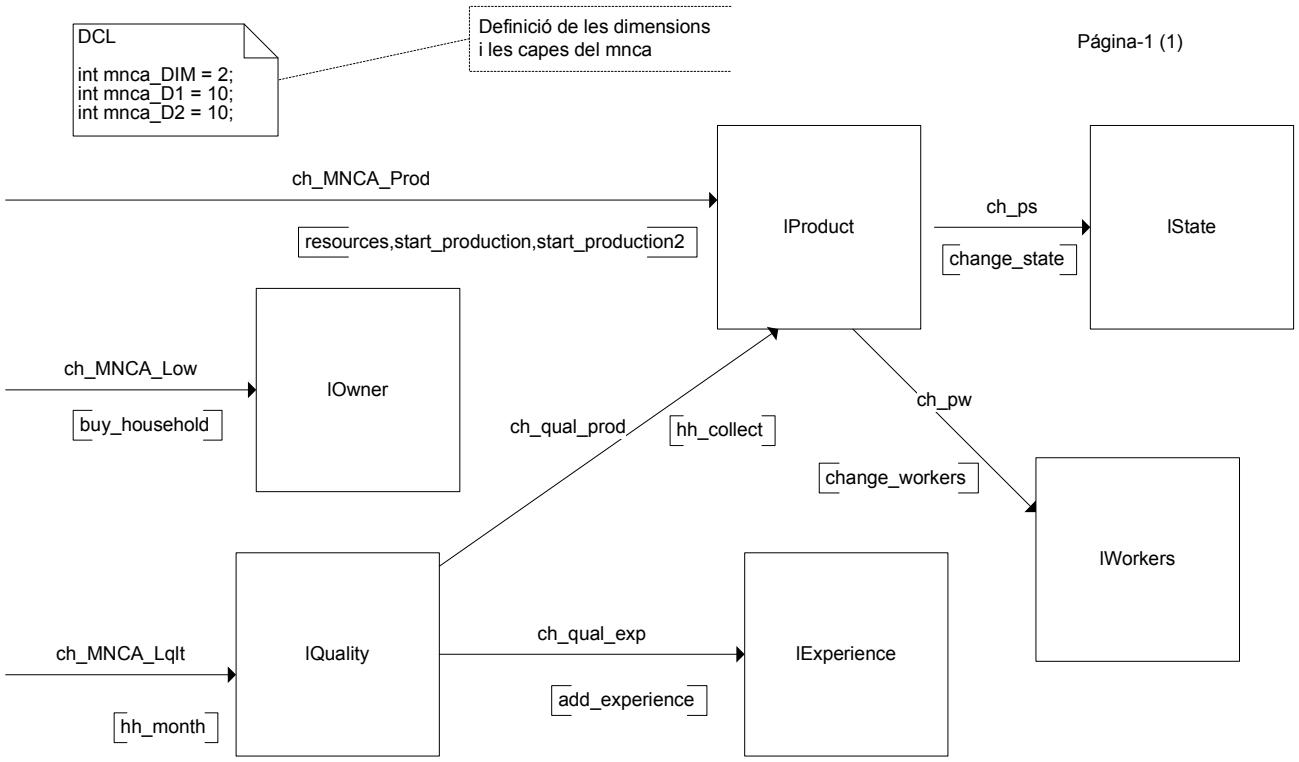


Il·lustració 48: bHousehold

L'autòmat conté capes principals i secundàries que contenen els valors que defineixen cada cel·la.

12.4.1 Capes principals

Totes les capes principals del agent han de tenir associat un bloc amb els processos que la controlen. Al nostre cas, utilitzem la nomenclatura *l[NOM_CAPA]* per definir el bloc i *pl[NOM_CAPA]* per definir el procés intern. Totes les capes estan contingudes al mateix nivell del bloc de l'autòmat.



Il·lustració 49: MNCA_HH

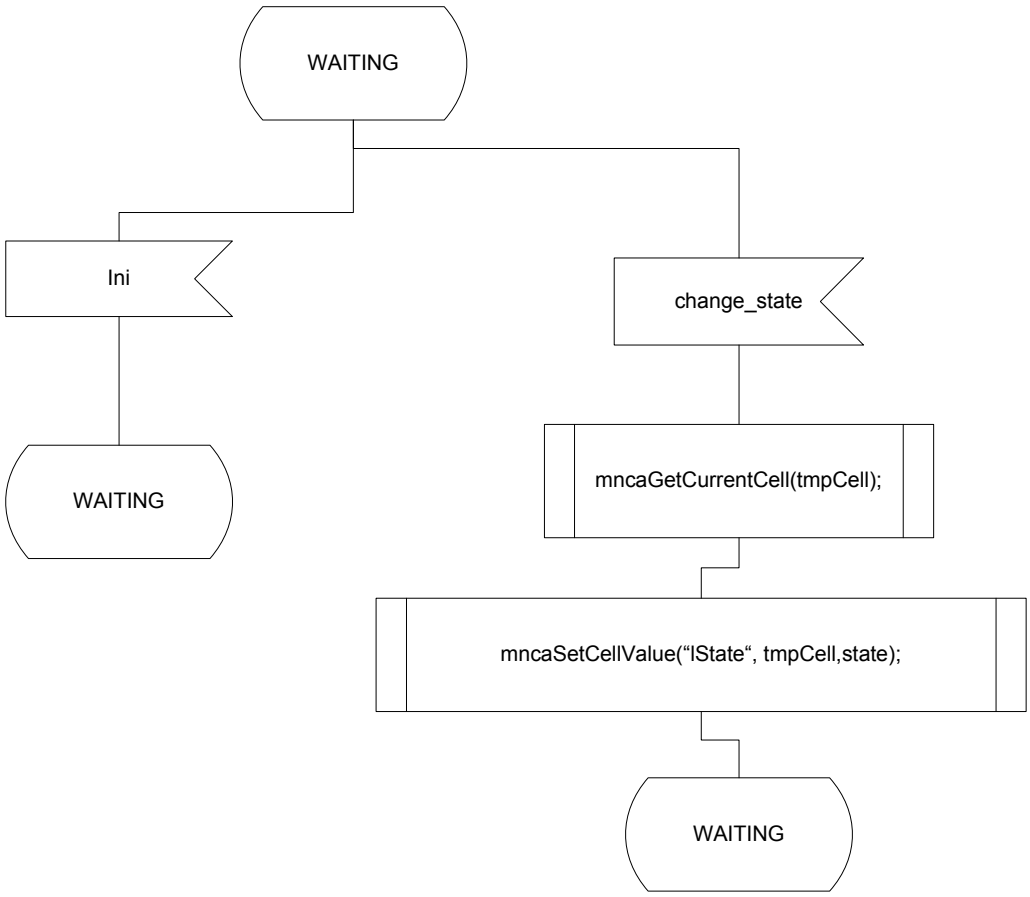
A continuació s'exposa el disseny de cadascuna de les capes principals.

12.4.1.1 State

Defineix l'estat de la propietat:

- 0 -> Lliure
- 1 -> En producció

El procés disposa del senyal *change_state* per canviar el valor de la capa des d'un altre procés.



Il·lustració 50: plState

12.4.1.2 Product

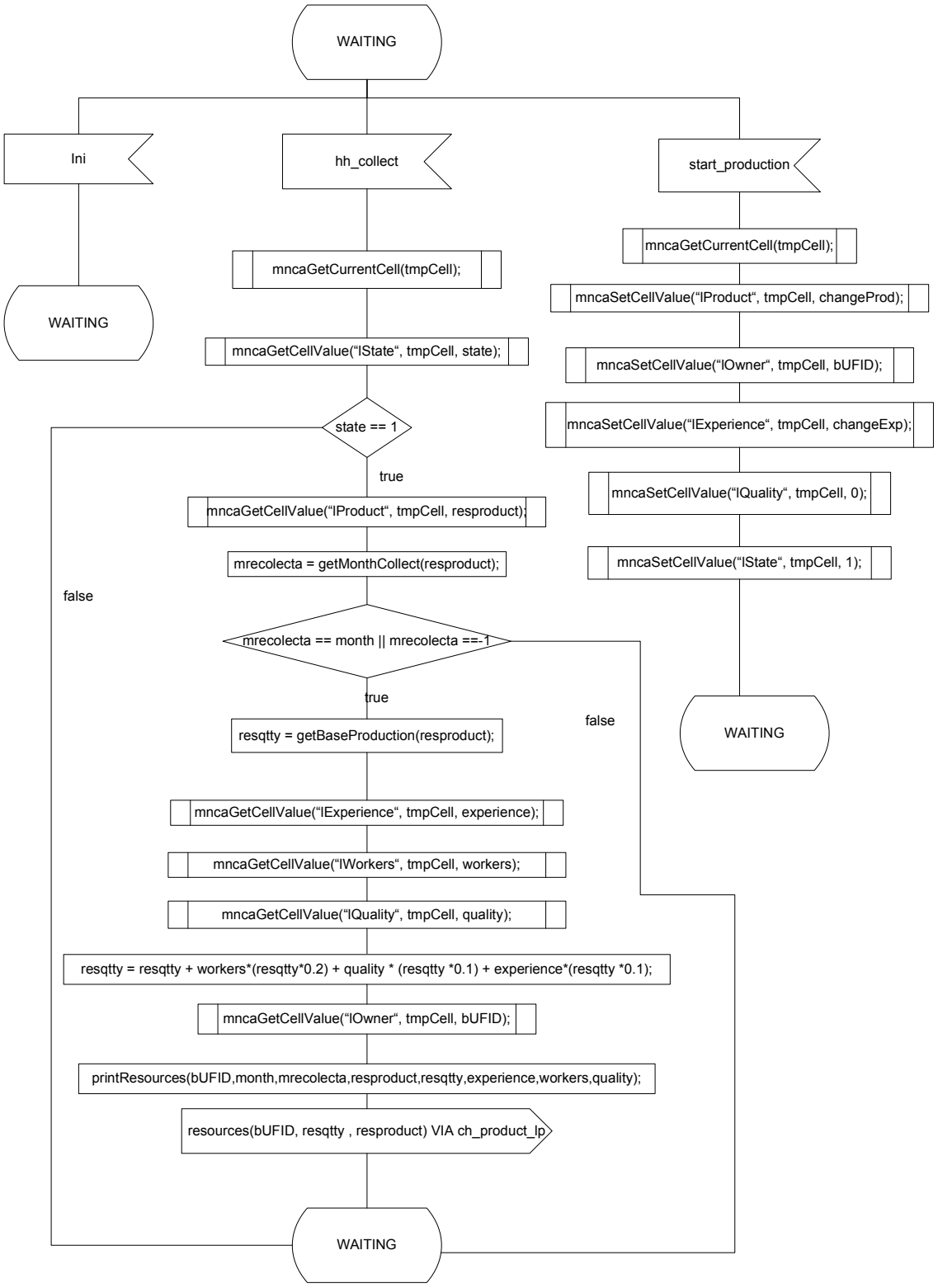
El valor de la capa representa el ID del producte associat.

Addicionalment, el procés s'encarrega de fer la recol·lecta de la quantitat de producte generat i enviar-la al seu amo.

Per fer aquesta tasca, primer es controla si el mes actual és de recol·lecta. Per calcular la quantitat de producte generat s'utilitza la producció base, que es veu augmentada o disminuïda segons una ponderació que fem amb:

- La quantitat de treballadors que hi treballen.
- La qualitat acumulada de la finca.
- L'experiència dels treballadors en el producte.

El resultat s'envia al bloc de Unitats Familiars amb el senyal *resources*.

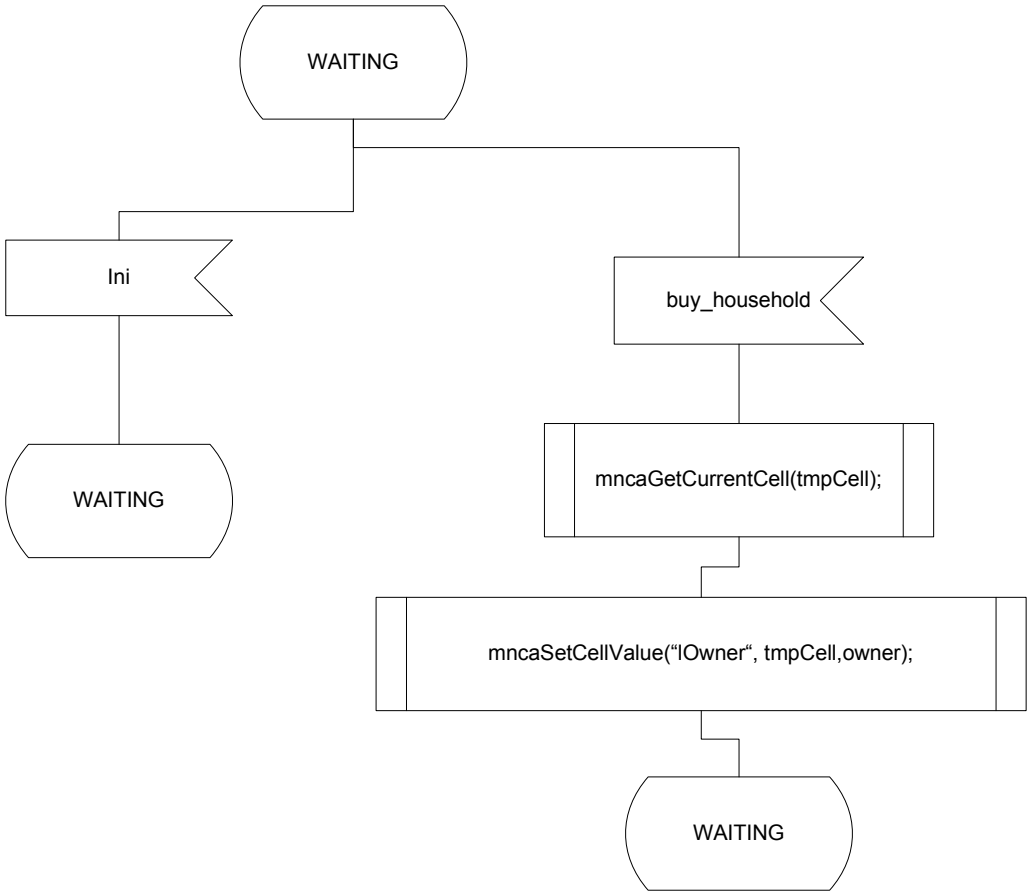


Il·lustració 51: plProduct

12.4.1.3 Owner

Defineix el propietari de la propietat, on el valor és el ID de la Unitat Familiar a la que pertany.

El procés disposa del senyal *change_owner* per canviar el valor de la capa des d'un altre procés.

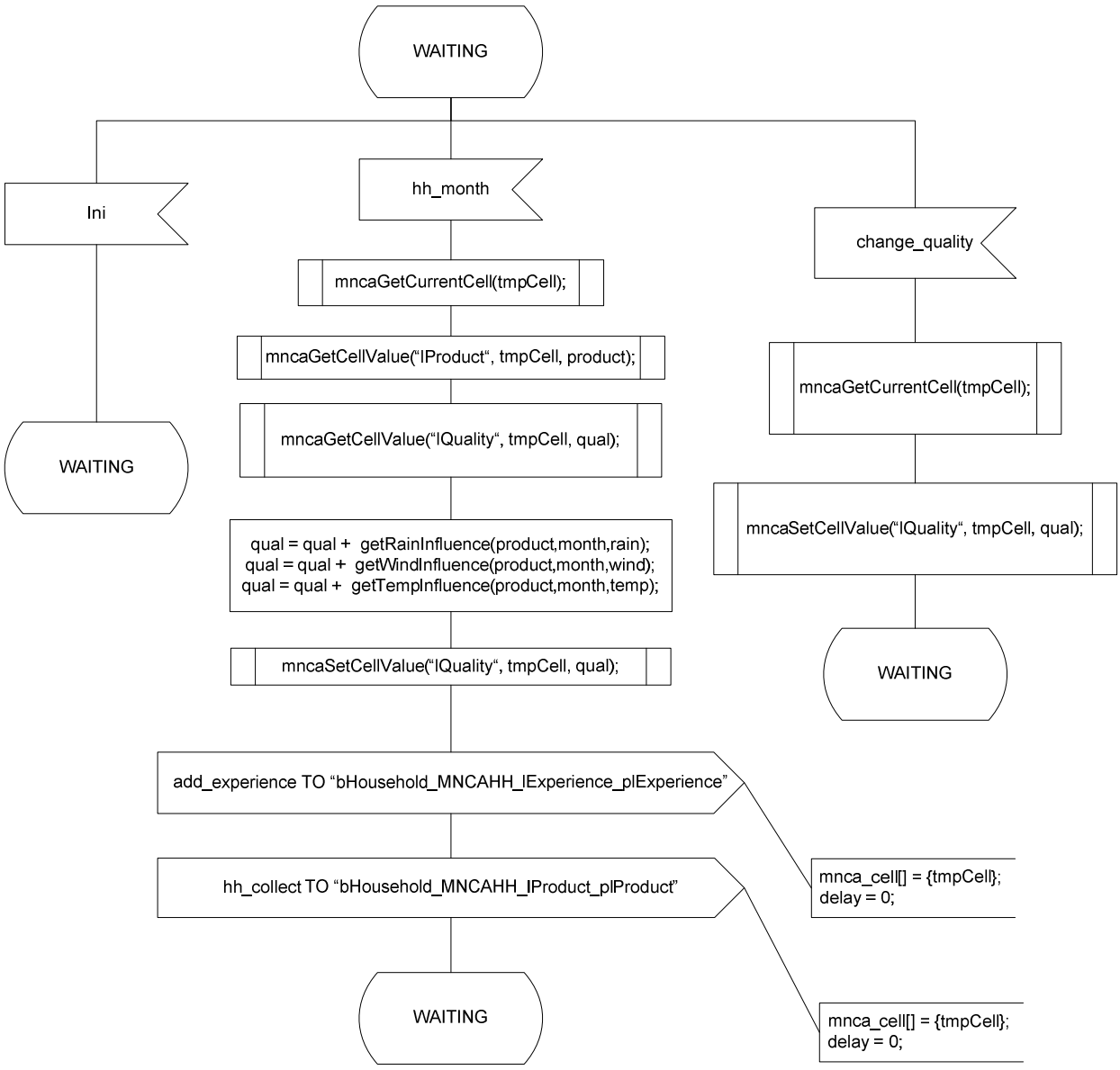


Il·lustració 52: plOwner

12.4.1.4 Quality

Defineix la qualitat del terreny, que afectarà a la quantitat de la producció.

La qualitat del producte la determinem segons les condicions climàtiques. Comparem la pluja, vent i temperatura actual amb la òptima per la producció. Si es situa dins un marge, s'augmenta la qualitat; en cas contrari es disminueix.



Il·lustració 53: plQuality

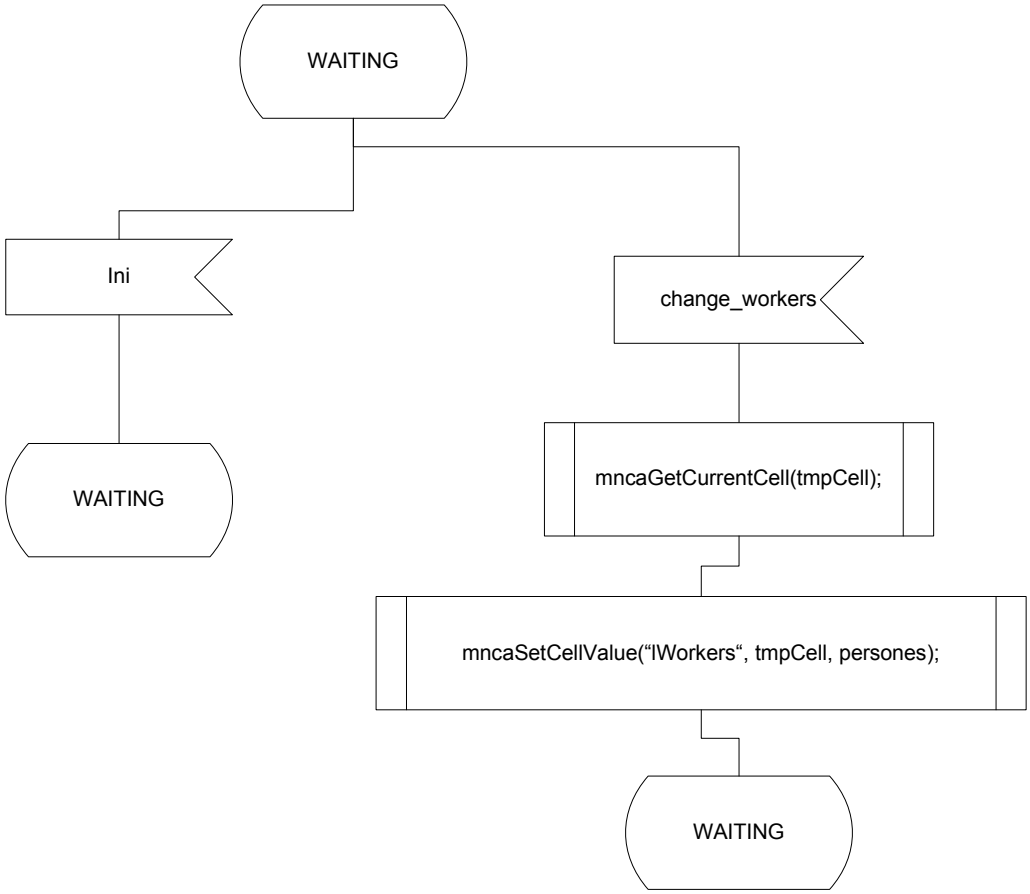
Addicionalment, enviem a la capa **Experience** el senyal *add_experience* per incrementar-la en una unitat.

Un cop s'han modificat els paràmetres, s'envia el senyal *hh_collect* a la capa de producte per calcular la producció del mes i enviar els recursos a la família.

12.4.1.5 Workers

Defineix el nombre de treballadors que contribueixen a la producció de la propietat.

El procés disposa del senyal *change_workers* per canviar el valor de la capa des d'un altre procés.



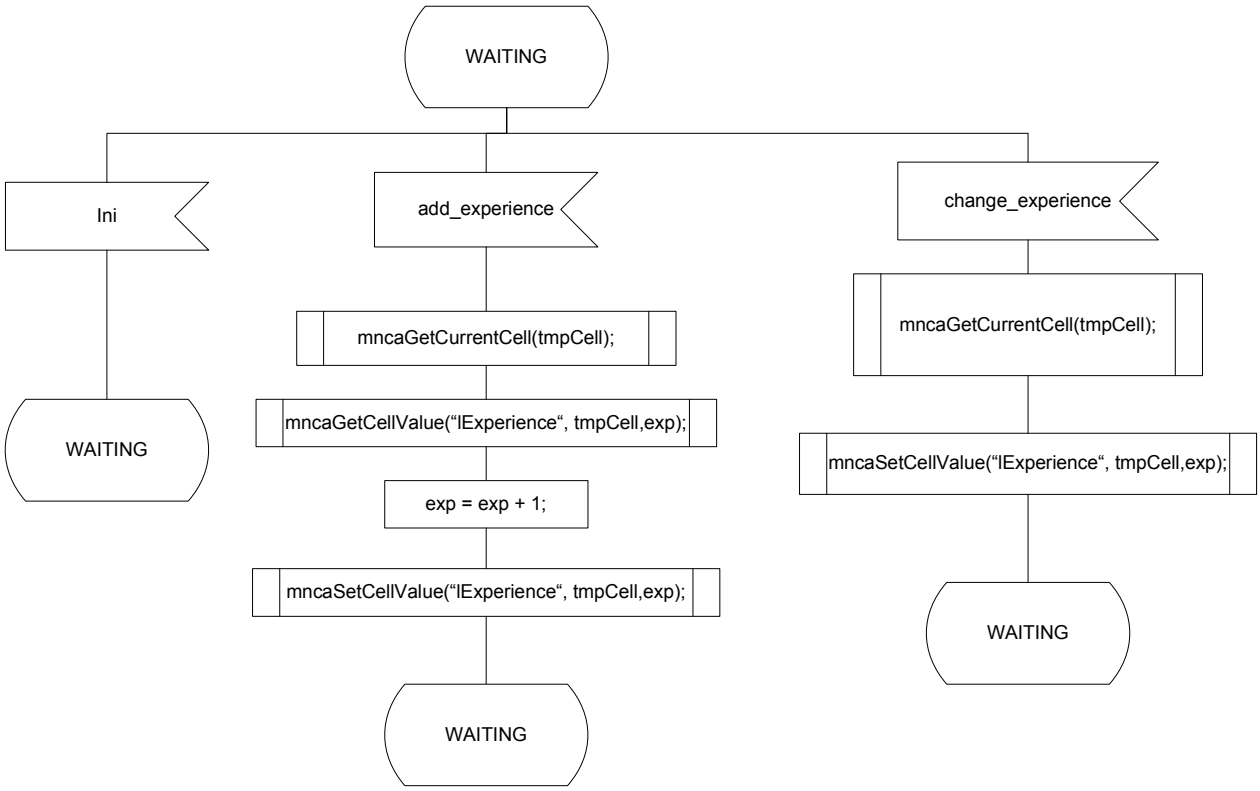
Il·lustració 54: plWorkers

12.4.1.6 Experience

Defineix la experiència acumulada dels treballadors de la propietat, que afectarà a la quantitat del producte recol·lectat.

El procés disposa del senyal *change_experience* per canviar el valor de la capa des d'un altre procés.

Addicionalment, disposa del senyal *add_experience* per afegir una unitat d'experiència i facilitar càlculs.



Il·lustració 55: plExperience

13 Entrada de dades

El model de simulació civiSDL és parametritzable. Això implica que hem de definir mecanismes per introduir dades al model que serveixin com a input de la simulació. Aquestes dades poden representar l'estat inicial del sistema, els agents que d'inici l'han de poblar o d'altres valors constants que defineixen el model.

13.1 Entrada del bloc Household

L'entrada i sortida del bloc *Household* són els valors de les diverses capes de l'autòmat cel·lular que el composa. Per tant, no cal cap gestió addicional amb aquest bloc, ja que el SDLPS llegeix i escriu aquests fitxers.

13.2 Entrada de la resta del model

Per introduir dades de la resta del model utilitzarem uns fitxers .csv definits. Cada fitxer té una funció al codi extern que el llegeix i construeix les estructures de dades necessàries.

13.2.1 Uf.csv

Conté la informació de les famílies inicials.

Camp	***Descripció***
ID	Identificador de la Unitat Familiar
Home	Propietat base on viu
Stockn	Quantitat inicial que posseeix del Producte n, per n ∈ [0… **num_productes**]

13.2.2 Member.csv

Conté els membres inicials del model. Han de pertànyer a alguna família que existeixi al fitxer uf.csv.

Camp	*Descripció*
ID	Identificador del Membre
UF	Unitat Familiar a la que pertany
Type	Pes dins la família: 0 -> cap de família; 1-> conyugue; >1 fill /germà
Age	Edat inicial
Gender	Gènere
Name	Nom
Surname	Cognom

13.2.3 Products.csv

Conté tota la informació bàsica dels productes sobre els que operarà el sistema

Camp	*Descripció*
ID	Identificador del Producte
Nom	Nom descriptiu
Group	Tipus de producte (alimentari, luxe..)
default price	Preu de sortida
isAliment	0-> no és aliment; 1-> és aliment
felicitat	Felicitat que aporta el consum d'una unitat
Kcal	Kcal que aporta el consum d'una unitat
greix	Greix que aporta el consum d'una unitat
vitamines	Vitamines que aporta el consum d'una unitat
minerals	Minerals que aporta el consum d'una unitat
mes recolecta	Mes del any en que es por recol·lectar, -1 si és cada mes
producció base	Producció per recol·lecta, sense tenir en compte bonus o penalitzacions
susceptible_clima	Indica si la producció es veu afectada pel clima
m_vent	Vent ideal per la producció al mes m, on m ∈ [0..11]
m_temperatura	Temperatura ideal per la producció al mes m, on m ∈ [0..11]
m_pluja	Pluja ideal per la producció al mes m, on m ∈ [0..11]

13.2.4 Consum.csv

Taula de consum per producte. Indica la quantitat mensual que ha de consumir un membre per tenir una bona salut, segons la franja d'edat a la qual pertany. Els productes han d'existir al fitxer products.csv.

Camp	***Descripció***
product	ID del producte
Infant	Consum mensual pels infants.
Adolescent	Consum mensual pels adolescents.
Adult	Consum mensual pels adults.

14 Sortida de dades

Les dades relatives a l'autòmat cel·lular són fàcils d'obtenir, ja que els fitxers Idrisi32 resultants ens donen la informació dels valors de les capes de cada cel·la en el moment final.

Tot i això ens interessaran obtenir dades dels altres mòduls i hem de definir un sistema per fer-ho. De la mateixa manera que amb els inputs, definim uns fitxers que descriuen l'evolució del sistema al llarg de la simulació.

14.1 compra.csv

Registra els moviments de compra de cada Unitat Familiar.

Camp	Descripció
ID_UF	Unitat Familiar que fa la operació
Product	ID del producte que es compra
Qtty	Quantitat de compra

14.2 venda.csv

Registra els moviments de compra de cada Unitat Familiar.

Camp	Descripció
ID_UF	Unitat Familiar que fa la operació
Product	ID del producte que es ven
Qtty	Quantitat de venda

14.3 Cash.csv

Registra l'evolució del poder adquisitiu (en unitats monetàries) de cada Unitat Familiar.

Camp	Descripció
ID_UF	Unitat Familiar de la xifra
Qtty	Nombre d'unitats monetàries

14.4 Preus.csv

Registra l'evolució dels preus dels productes quan canvien per ajustos del mercat.

Camp	Descripció
Product	Identificador del producte
Price	Nou preu del producte

14.5 Salut.csv

Registra l'evolució de la salut de cada membre en funció dels paràmetres de medició.

Camp	Descripció
UF_ID	Unitat Familiar del membre
MemberID	ID del membre
Age	Edat
Greix	Proporció d'estat en greix
Kcal	Proporció d'estat en kilocalories
Vitamines	Proporció d'estat en vitamines
Minerals	Proporció d'estat en minerals
Felicitat	Proporció d'estat en felicitat

14.6 Naixements.csv

Registra els naixements que n'hi han hagut a alguna Unitat Familiar del model

Camp	Descripció
UF_ID	Unitat Familiar on ha nascut
MemberID	ID del nou membre

14.7 Morts.csv

Reporta les morts del model.

Camp	Descripció
ID_UF	Unitat Familiar
IDMember	Membre
Age	Edat de la mort

14.8 Canvis.csv

Reporta els canvis de producte en la producció per Unitat Familiar.

Camp	Descripció
ID_UF	Unitat Familiar que fa el canvi
old_product	Antic producte
new_product	Nou producte
old_experience	Experiència acumulada amb l'antic producte
new_experience	Experiència acumulada amb el nou producte

14.9 Casaments.csv

Reporta les noves famílies formades a partir d'un casament.

Camp	Descripció
ID_UF	Nou ID de la nova Unitat Familiar
Home	Cel·la de terreny que posseiran
Member1	ID del membre 1
Member2	ID del membre 2

15 Etapes del projecte

Aquesta secció explica els detalls sobre cada etapa del desenvolupament del projecte.

15.1 Aprenentatge dels paradigmes i entorn de treball

Objectiu

El propòsit consistia en proporcionar-me una bona base sobre tots els conceptes teòrics i pràctics que s'emprarien en tot el desenvolupament del projecte.

Estat inicial

A l'inici d'aquesta fase no començava des de zero, ja que vaig poder cursar a les assignatures de Simulació i MEIO, en les que vaig aprendre alguns conceptes bàsics sobre el món de la simulació, així com detalls de SDL, el llenguatge d'especificació emprat al projecte.

El codi extern al model SDL està implementat en C. En aquest punt no vaig tenir cap problema ja que és un llenguatge que ja coneixia i s'utilitza a diverses assignatures de la carrera.

Desenvolupament

La metodologia de treball es va centrar en les reunions amb el director del projecte, que és qui em proporcionava tot el material i la informació necessària per poder dur a terme els desenvolupaments.

El primer pas de l'aprenentatge va consistir en aprofundir en el llenguatge d'especificació SDL. Per mi resultava una prioritat conèixer en el màxim detall totes les opcions i possibilitats del llenguatge. D'aquesta forma volia evitar que em frenés el fet no entendre alguna part del model ja fet o que no pugui explotar tota la potència del llenguatge en els meus futurs dissenys.

En aquest punt vaig començar a treballar amb el programa Microsoft Visio i el plug-in **Sandrila**. Aquestes dues eines permeten la creació de models SDL vàlids de forma senzilla.

Un cop assimilats els conceptes del llenguatge SDL, el següent pas consistia en començar a treballar amb SDLPS, el simulador que utilitzarem com a eina d'execució del model.

El programa SDLPS resulta intuïtiu de manegar. Amb les instruccions del director de projecte vaig poder comprendre tots el passos que es necessiten fer, des de la càrrega del Model fins la

seva execució. Al principi resulta una dificultat afegida el fet de que només podem trobar informació del programa dins el seu departament de desenvolupament.

El pas següent consistia en agafar el Model ja fet i entendre amb tot detall com es representen en XML tots els elements d'un model SDL.

Resultats i conclusions

Al final d'aquesta tasca ja tenia una bona base de gran part dels conceptes tècnics que necessitaria. Tot i això van quedar pendents de dominar alguns detalls de la implementació del model SDL en format .sdlps. Concretament pertanyen a la implementació d' autòmats cel·lulars i l'ús de l'operació **create** per generar instàncies d'agents. Com que aquests desenvolupaments tenen la seva pròpia fase, s'entén que aquests conceptes es treballarien en profunditat en el moment de començar-les.

Cal dir que aquesta fase es pot veure perllongada al llarg de tot el projecte, ja que els dubtes van anar sorgint. Al llarg de totes les reunions amb el director del projecte van quedar resolts.

15.2 Correcció del Model preliminar

Objectius

Al final d'aquesta tasca s'havia de disposar d'un model que compilés i quedés *linkat* amb el codi extern, obtenint com a resultat una llibreria SDLCode.dll executable dins l'entorn SDLPS. Aquest model hauria d'executar-se al SDLPS sense excepcions ni errors.

Estat Inicial

El model preliminar no compilava amb la versió actual de SDLPS. Els resultats de la primera càrrega donaven diversos errors al codi *sdlps* i al codi extern implementat en C. Com a conseqüència d'això la llibreria SDLCode.dll no era generada i per tant, no es podia executar el model.

Desenvolupament

Vaig procedir a solucionar els errors del model que impedien l'execució. Podem classificar els errors solucionats segons la seva procedència.

Errors al .sdlps

Al model sdlps n'hi havia errors com nom de variables o senyals incorrectament connectats.

Errors al codi extern

Com ja s'ha comentat en vàries ocasions, el codi extern en C és molt ampli, fet que dona peu a que contingui un cert nombre d'errors.

El programa SDLPS està pensat per ésser executat de forma distribuïda. Els processos poden estar a diferents màquines si així és indicat, per tal d'aconseguir un rendiment superior. Per aquest motiu no té sentit que el codi SDL treballi amb referències a memòria, ja que es pot incórrer en un error d'adreçament. Es va procedir a l'adaptació d'aquestes crides per tal de que només utilitzessin paràmetres de tipus simples.

15.2.1.1 Metodologia

Degut a la complexitat del model i a les poques opcions de debug de que disposava, vaig prendre la decisió en aquest punt de reconstruir el model bloc a bloc. Aquesta tasca va constar de les següents fases:

- Ordenació dels blocs(mòduls) segons dependència de la resta de mòduls.
- Creació d'un nou fitxer .sdlps del model, inicialment en blanc.
- Per tots els blocs, seguint l'ordre:

    1. Inserció del bloc dins el nou fitxer, comentant momentàniament els *signals* que van cap a blocs encara no inserits.

    2. Descomentar els *signals* dels blocs anteriorment inserits que apunten cap al nou bloc.

    3. Execució del sub-model creat.

D' aquesta forma vaig aconseguir l'execució satisfactòria del model de forma molt més eficient que treballant amb tot el model d'inici. També em va ajudar a entendre al 100% tot el contingut del model en el seu punt preliminar, ja que aquesta seria la base del nou model que hauria de construir.

Resultats i conclusions

A la finalització d'aquesta fase disposava d'un model que compila correctament i queda linkat amb el codi extern generant la llibreria SDLCode.dll. El model podia executar-se al SDLPS.

Tot i produir resultats no gaire significatius, aquesta etapa em va servir per entendre millor el model preliminar. D'aquesta forma podia començar a construir el nou model amb una bona base.

15.3 Implementació del autòmat cel·lular

15.3.1 Objectius

L'objectiu era dissenyar i implementar el mòdul *Household* com un autòmat cel·lular.

15.3.2 Estat Inicial

Al model preliminar, el mòdul *Household* estava implementat com un bloc SDL estàndard on, amb ajuda del codi SDL, s'iterava sobre totes les propietats amb un únic procés.

Aquesta implementació era molt millorable transformant el bloc en una autòmat cel·lular, ja que podríem veure el bloc com una graella bidimensional que simula el terreny i ens permet interactuar amb les cel·les (propietat o porció de terreny) de forma directa o en paral·lel.

15.3.3 Desenvolupament

Primerament es va fer un disseny del mòdul, decidint les capes que hauria de tenir i com són els processos que les controlen. El comportament final havia de ser semblant al implementat al model preliminar, però de forma eficient i millorada.

Un pas important dins d'aquesta etapa va consistir en implementar al SDL operacions que abans es feien al codi extern. Aquest pas és possible gràcies a que el mòdul s'implementa com un autòmat cel·lular, ja que ens permet treballar amb totes les cel·les sense cap estructura externa de suport.

15.3.4 Problemes

Com el SDLPS és una eina en desenvolupament, era comprensible que algunes operacions necessitaven ajustos per tal de que funcionessin bé. Inicialment no es podien enviar *signals* des de blocs externs cap un *mnca*, ja que el paràmetre CELL es perdia.

En les reunions amb el tutor li comunicava els possibles errors i l'equip encarregat de desenvolupar el SDLPS els solucionava, com va passar en aquest cas.

15.3.5 Resultats i Conclusions

S'obté el nou autòmat cel·lular que compleix les funcions de gestió del terreny y propietat del Sistema. Es comunica i s'integra satisfactòriament amb la resta de mòduls.

Des de el punt de vista del desenvolupador, resultarà molt més senzill ampliar aquest mòdul amb noves funcions i opcions ja que està perfectament estructurat, no té codi extern i és més clar d'entendre.

15.4 Implementació de les Unitats Familiars

15.4.1 Objectius

Modificar el mòdul de les Unitats Familiars per tal de que cadascuna estigui representada per un agent, així com els membres que la componen.

15.4.2 Estat inicial

En el model preliminar, les Unitats Familiars (UF) estaven gestionades per un únic procés que iterava sobre totes elles per fer qualsevol operació. El codi extern és bastant extens, per tal de gestionar totes les operacions.

Els Membres de la família no tenen representació en el codi SDL, solament apareixen en el codi extern.

15.4.3 Desenvolupament

Podem dividir el desenvolupament en dues etapes:

Agents per famílies

S'implementa la creació d'un procés per cada Família del model. Aquest procés realitza totes les accions relacionades amb la família que representa i es comunica amb els altres blocs seguint la mateixa filosofia.

Agent per membre

De la mateixa manera s'implementa un procés per cada Membre, que computa totes les accions que pot fer un individu al sistema, així com el control de la seva salut i vida.

15.4.4 Problemàtica

Com s'ha comentat a l'apartat de Disseny, no va se possible la comunicació dels *signals* quan un agent té més d'una instància, degut a problemes amb el SDLPS. Per això es va haver de muntar un sistema que simulés totes les famílies i membres amb un procés respectiu.

15.4.5 Conclusions

- El nou sistema de gestió és molt més modular i fàcil de gestionar. S'eviten iteracions complexes, amb els conseqüents errors que poden provocar.

- Aprofitem la potència del SDL com a llenguatge orientat a objectes.

- S'elimina codi extern, ja que molts processos iteratius sobre famílies/membres que estava implementat al codi C ara es pot passar als diagrames.

- Tot i que al final de la implementació no es va poder disposar de la operació CREATE funcionalment, s'espera que per evolucions futures del projecte s'implanti fàcilment si el SDLPS ja ho permet.

15.5 Canvi estructural del model complet

15.5.1 Objectius

Els objectius d'aquesta etapa es resumeixen en dos punts:

1. Traspassar el màxim de codi extern al codi SDL.
2. Delimitar la comunicació entre blocs a només per mitjà de *signals*.

A les següents seccions s'aclareix el significat d'aquestes seccions.

15.5.2 Estat Inicial

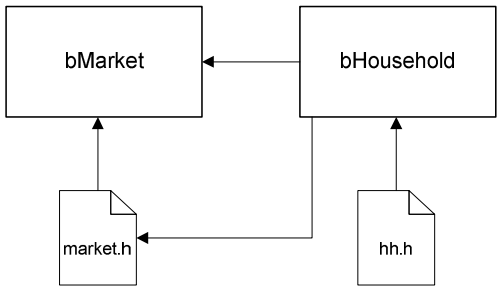
El model preliminar estava massa basat en el codi extern. S'aplicaven iteracions per operar sobre agents i es comunicaven els seus atributs sense passar pels diagrames SDL.

15.5.2.1 Codi extern

Com ja s'ha comentat en anteriors seccions, el codi extern del model preliminar és extens. Això presenta una dificultat afegida pel desenvolupador, ja que és més difícil d'abordar. Les operacions implementades al codi SDL són molt més fàcil de veure i analitzar, ja que tenim la representació amb diagrames.

15.5.2.2 Comunicació entre blocs

En determinats punts es fa referència per mitjà el codi extern a variables o funcions que en teoria pertanyen a un altre bloc. Ho podem veure amb el següent exemple:



Market.h conté el codi relacionat amb la gestió del mercat (bloc Market). En una funció, fa

referència directa a una variable del bloc *Household*.

És desitjable que un conjunt de funcions i variables en C només s'utilitzin des d'un mateix bloc. Aquests són els dos principals motius:

- **Claredat del Codi :** pel desenvolupador és molt més senzill entendre el Model si al codi extern n'hi ha una relació directa entre bloc – llibreria i no es barregen.

- **Permetre execució distribuïda:** SDLPS permet l'execució de processos en diferents màquines, de forma distribuïda. Si dos blocs situats a màquines diferents modifiquen el mateix espai de memòria es produiran incoherències. Això s'evita si una variable en C només és modificable des d'un sol bloc.

15.5.3 Desenvolupament i Conclusions

La feina a fer consistia en reconstruir pràcticament tot el model per tal d'aplicar la nova filosofia de treball. Aprofitant que s'havia de passar molt codi a SDL (objectiu 1), m'assegurava que tot el codi extern que inevitablement es mantenia, satisfeia l'objectiu 2.

Cal dir que com el blocs *Household* y UF es reconstruïen per els altres requisits, indirectament s'estava fent part de la feina corresponent a aquesta tasca.

S'afegeix la dificultat de que als blocs i processos només podem treballar amb tipus simples de dades *(int, double o char**), així com a l'enviament de paràmetres d'usuari amb els *signals*.

Aquestes consideracions van fer que fos inevitable mantenir un mínim de codi extern, tot i que va ser reduït en gran part. La filosofia de comunicació SDL – C que vaig seguir és la següent:

1. **Les operacions, gestionades per SDL.**
   S'intenta que tota la gestió d'estat dels agents, així com decisions de comportament davant events es gestiona des de el codi SDL. Per dir-ho d'una forma més clara, gestiona la "intel·ligència" del model.

2. **El codi C actua com base de dades.**
   Davant la impossibilitat de tenir estructures complexes a SDL, utilitzem el codi extern com a suport per guardar dades d'aquest tipus. S'implementen funcions que fan de get/set i computen petites operacions que no tenen una rellevància troncal per seguir l'execució del model.

No va ser una feina gens fàcil assolir els objectius enunciats. Reproduir el comportament del codi extern amb agents SDL és més sofisticat i complicat de plantejar.

Havia de tenir molt clar el disseny del model, ja que aquests canvis suposaven una xarxa de processos i *signals* molt més complexa que la del model preliminar. Sense oblidar que aquests agents podien ser instanciables (famílies i membres) o autòmats cel·lulars (propietats).

15.6 Millores de contingut

15.6.1 Objectius

L'objectiu es aplicar millores al model que incrementin la intel·ligència dels agents o l'ampliïn incloent nous conceptes.

15.6.2 Estat Inicial

L'abast del model inicial era prou ampli i complet per ser un model preliminar. Tot i això seria desitjable que cada cop que sigui revisat s'ampliessin els conceptes que engloba.

15.6.3 Desenvolupament

- S'ha millorat el funcionament dels següents aspectes:
- Gestió del *Stock* familiar.
- Funció de càlcul d'aportacions dels productes.
- Funció de càlcul de salut.
- Funció de naixement/mort.
- Càlcul de recursos obtinguts per les propietats.

15.6.4 Conclusions

Tot i no ser una tasca troncal del projecte, vaig trobar interessant el seu desenvolupament, ja que potser és una feina més lleugera i entretinguda que el disseny pur del model. Al incloure conceptes interdisciplinaris es fa més interessant.

Cal tenir en compte que el model continua sent un prototipus i ha d'evolucionar amb el temps. No és fàcil ja que per modelar una ciutat necessitaríem quasi infinits factors a tenir en compte. La idea és que pas a pas s'incloguin nous conceptes segons rellevància.

16 Planificació

16.1 L'estimació d'hores

El projecte civiSDL necessitava una bona planificació per acotar els períodes de feina correctament. No és fàcil ja que el projecte no és gens típic i no tenia la mateixa facilitat per estimar el temps que si fos un projecte de software més convencional.

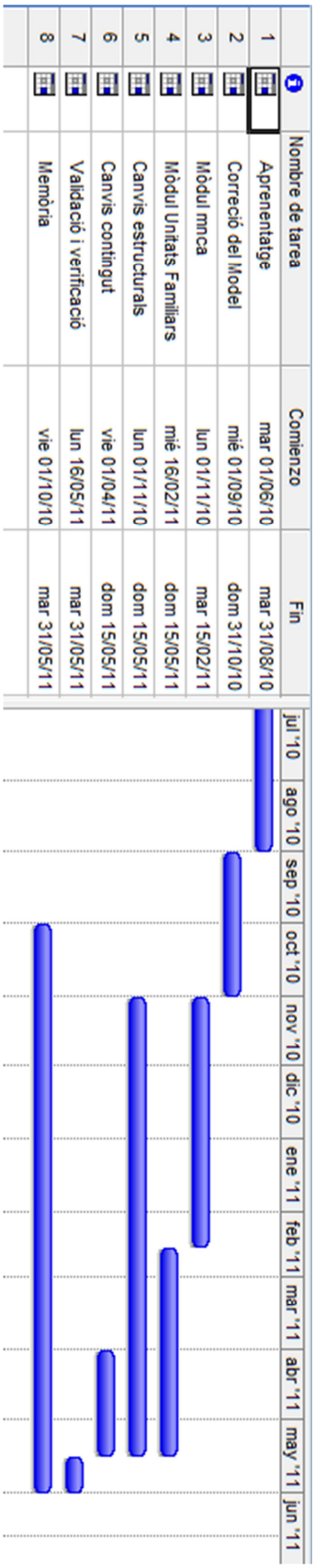
D'altra banda s'afegeix la dificultat d'acotar molt bé el que es faria al projecte. Dins el marc d'un PFC òbviament no podem fer un model final, sino l'evolució d'un prototipus amb cert sentit, tot i que encara molt ampliable. Un model de simulació d'aquestes característiques pot créixer notablement.

Addicionalment, el fet de que s'experimenti amb metodologies de disseny fins ara poc explorades i s'utilitzi una eina en desenvolupament com és el SDLPS, fan que l'estimació temporal sigui més incerta.

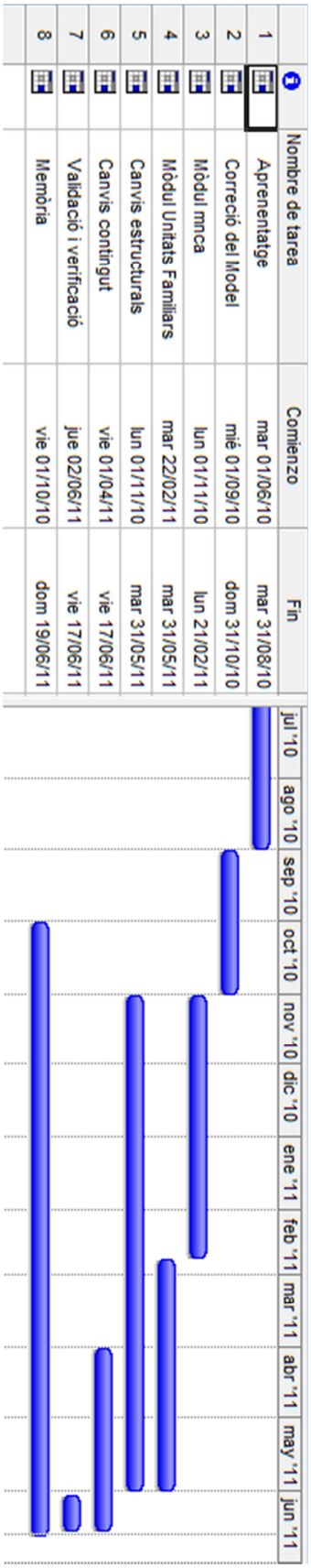
16.2 Diagrames de Gantt

El projecte s'inicia a l'estiu de 2010. La idea inicial era entregar-lo a inicis de 2011, però degut a les dificultats que porta el projecte i a que les hores diàries de treball disminuïen per motius laborals, es va fixar com a data final d'entrega al juny del 2011.

A continuació trobareu un diagrama de Gantt que mostra la planificació per etapes del projecte. Aquesta planificació va sofrir desviacions degut a retards a la fase d'implementació. El segon diagrama de Gantt que es mostra reflexa l'evolució de la feina total.



	Nombre de tarea	Comienzo	Fin
1	Aprenentatge	mar 01/06/10	mar 31/08/10
2	Correcció del Model	mié 01/09/10	dom 31/10/10
3	Mòdul mnca	lun 01/11/10	mar 15/02/11
4	Mòdul Unitats Familiars	mié 16/02/11	dom 15/05/11
5	Canvis estructurals	lun 01/11/10	dom 15/05/11
6	Canvis contingut	vie 01/04/11	dom 15/05/11
7	Validació i verificació	lun 16/05/11	mar 31/05/11
8	Memòria	vie 01/10/10	mar 31/05/11



	Nombre de tarea	Comienzo	Fin
1	Aprenentatge	mar 01/06/10	mar 31/08/10
2	Correció del Model	mié 01/09/10	dom 31/10/10
3	Mòdul mnca	lun 01/11/10	lun 21/02/11
4	Mòdul Unitats Familiars	mar 22/02/11	mar 31/05/11
5	Canvis estructurals	lun 01/11/10	mar 31/05/11
6	Canvis contingut	vie 01/04/11	vie 17/06/11
7	Validació i verificació	jue 02/06/11	vie 17/06/11
8	Memòria	vie 01/10/10	dom 19/06/11

16.3 Costos

El desenvolupament del projecte implica uns costos en la compra de software específic. En la taula següent es detallen:

Software	*Preu*
Microsoft Windows 7	122 €
Microsoft Office 2010	109 €
Sandrila plug-in	23 €
SDLPS	0 €
Total	254 €

A continuació s'especifiquen els costos per hora de treball. Es divideixen segons les etapes del cicle de desenvolupament de software, ja que el preu per hora varia:

Fase	*Hores*	*Preu per hora*	*Cost*
Documentació	45	30 €	1.350 €
Disseny	340	50 €	17.000 €
Implementació	310	50 €	3.100 €
Test	90	50 €	90 €
Presentació	20	50 €	80 €
Total			21.620 €

El cost total del projecte és de **21620 €** i es desenvolupa en un total de **805 hores**.

17 Verificació

La verificació és el procediment que determina si el model és funcional. Es té en compte que tots els components del model estan construïts i units correctament. Per tant el model pot iniciar i terminar una execució sense errors.

Per fer la verificació utilitzarem la tècnica del **Disseny Experimental.** És un procediment que ens permet avaluar el comportament del model segons varis experiments amb diverses variables.

Concretament, utilitzarem el Disseny Experimental 2k. En aquest disseny s'avaluen les diferents combinacions de k factors, on cadascun té 2 nivells. Per cada combinació analitzem l'impacte sobre alguna variable mesurable.

Per fer aquesta verificació al model civiSDL utilitzarem tres variables (k=3). La següent taula reflexa els factors, juntament amb els valors mínim i màxim que es contemplaran:

Variable	Mínim (-)	Màxim (+)
Nombre inicial de famílies	1	6
Nombre de cel·les de terreny	25	100
Anys d'execució	1	10

La variable de resposta que estudiarem serà el temps en finalitzar la simulació.

El procediment consisteix en llençar execucions amb totes les possibles combinacions entre els factors. Com **k= 3**, tindrem 8 possibles combinacions. La taula que reflexa els resultats serà d'aquesta forma:

Experiment	#Families	#Propietats	#Anys	Temps
1	-	-	-	45 s
2	-	-	+	7 min
3	-	+	-	2 min
4	-	+	+	15 min
5	+	-	-	2 min
6	+	-	+	20 min
7	+	+	-	5 min
8	+	+	+	30 min

S'ha pogut verificar que totes les execucions terminen i no generen errors. Com a comprovacions addicionals destaquem que:

- Generen correctament els fitxers de sortida.
- La traça de simulació generada per SDLPS no conté errors.
- La traça de simulació indica que tots el *signals* enviats han sigut rebuts sempre per algun estat i cap s'ha perdut per una incorrecta gestió dels processos.

18 Validació

El procés de validació consisteix en comprovar que el model construït és una bona representació del sistema real. Per fer-ho utilitzem variables de resposta escollides al model i analitzem si els valors s'adapten a la realitat.

Una validació pot ser un procés iteratiu que inclou correccions i cal·libració del model. És una fase que pot ser tan extensa com la implementació del model degut al volum de feina que porta.

Dins els objectius d'aquest projecte no entra una fase extensa de validació (es podria emmarcar dins un futur PFC). Tot i això farem una validació funcional per comprovar la coherència de la implementació del model amb diversos escenaris.

A continuació s'exposen els escenaris provats, juntament amb un breu anàlisi dels resultats obtinguts. Totes les proves estan fetes en base a una graella de 10x10, amb un total de 100 cel·les de terreny.

18.1 Escenari 1

Nombre de Unitats Familiars	Anys
1	10

El primer escenari és el més simple, només simula una família.

Ens interessarà sobretot observar els factors que són més independents de la interrelació entre agents familiars, com pot ser la salut.

18.1.1 Demografia

En aquest període neixen 5 fills y moren 2 membres. S'observa que la probabilitat de naixement i mort és massa alta i es calibren els càlculs per fer-los més reals. Després dels canvis, neixen 3 fills i n'hi ha una mort.

No apareixen noves famílies ja que no es poden casar entre fills de la mateixa.

18.1.2 Salut

Podem veure com el nivell de salut dels membres de la família es manté en bon estat si la alimentació es la correcta. És curiós observar com si baixem la quantitat de diners inicials dels qual disposa la UF, la salut cada cop es veu més perjudicada ja que han de racionar aliments.

18.1.3 Mercat

Inicialment trobo un error al fer la compra i la venda de productes. La UF venia més del que tenia en *stock* i comprava més del que necessitava.

Vaig procedir a corregir aquest error i la UF va passar a tenir el comportament esperat:

- Compra dels productes que no produeix i necessita
- Venda dels productes que produeix

Aquest punt era important de validar i es va comprovar que la UF no comprava unitats del producte sobre el que treballava. De la mateixa forma venia el que produïa la seva parcel·la més els excedents de *stock* que té assignats al inici.

18.1.4 Finances

Els preus dels productes que no produeix la UF es disparen, ja que no n'hi ha famílies que el produeixen però la única UF que n'hi ha al sistema si que els demanda.

18.1.5 Propietats i producció

La UF comença produint blat i rep correctament els beneficis. Podem observar que cada cop en són més degut a la experiència que va acumulant en el treball d'aquest producte.

Com que només n'hi ha una UF i hem dit que la resta de preus de productes es dispara, la UF té tendència a canviar molt de producte (ho fa fins 5 cops). Podem considerar que és normal degut a la situació "particular" que simulem.

De totes formes apliquem més pes al valor de la experiència alhora de decidir si canviem a produir un altre producte.

18.2 Escenari 2

Ara introduirem més d'una família. Així podrem veure com interactuen i si fins i tot poden generar noves famílies. Comentem a continuació els resultats més destacables segons el tipus.

Nombre de Unitats Familiars	Anys
2	10

S'ha observat que s'han generat dues noves famílies. La família 1 tenia 2 fills i la 2 també uns altres dos. S'han casat entre ells i han generat dos famílies noves.

Les noves famílies han tingut fills, això constata que la creació de famílies funciona correctament, ja que s'adapten al sistema com les inicials.

Pel que fa a la feina, les noves Unitats Familiars comencen amb el producte amb el que més experiència acumulen.

En global, encara veiem molts canvis de producte i preus que es disparen, degut a que al sistema n'hi ha productes que encara no tenen una família especialitzada.

18.3 Escenari 3

Per últim llençarem un escenari on inicialment existeixen 6 famílies, on cada família produeix un producte dels que tenim al model.

Nombre de Unitats Familiars	Anys
6	10

Després d'aquest experiment es pot observar que la formació de noves famílies continua funcionant, així com els naixements i les morts.

Pel que fa als productes, és curiós observar com les famílies que tenien assignats d'inici productes que no són de primera necessitat tenen tendència a canviar per d'altres que sí ho són, i es mantenen més inestables ja que segons el mercat es mou, tornen a canviar a l'antic degut a l'experiència que tenen acumulada.

Això es degut a que a fer les compres, una família sempre opta per agafar productes en ordre de prioritat vital. Si no té per tot, els productes de luxe es compraran menys. Això en èpoques de "crisi econòmica" de les famílies, causa que els productors de luxe o segona necessitat baixin.

Les famílies que comencen amb productes bàsics es solen mantenir estables en la seva producció.

19 Conclusions

Podem classificar les conclusions del treball fet i el resultat final segons el punt de vista des del qual valorem el projecte: si valorem el desenvolupament del model en sí o les millores provocades en el paradigma de treball amb models SDL.

19.1 Evolució del Model

S'ha aconseguit un prototipus que modela adequadament el petit món que es pretenia representar. Considero que és flexible i amb marge de creixement degut a l'arquitectura emprada.

Tot i que en la fase final d'implementació no s'ha pogut treballar amb instàncies de Unitats Familiars i Membres degut a problemes tècnics, el model està especificat i preparat per fer aquest canvi de forma relativament ràpida.

La implantació de l'autòmat cel·lular ha sigut un gran avanç que ens ha ajudat en el model actual, però que serà molt explotable en versions futures.

En definitiva, el model resultat és molt satisfactori, ja que constitueix un prototipus funcional i amb capacitat de evolucionar i millorar.

19.2 El model com eina d'experimentació

El model proposat no resultava gens fàcil de dissenyar i implementar ja que pretenia construir un model basat en SDL multi agent i amb un autòmat cel·lular incorporat. Això feia que l'acoblament entre els components no fos trivial.

Durant tot el procés de desenvolupament del projecte ens hem trobat amb errors al software SDLPS. És completament normal degut a que està en actual desenvolupament. A això li sumem que en aquest projecte pretenia fer dissenys fins ara poc treballats.

Aquest fet va propiciar que alhora el SDLPS pogués millorar quan notificava els errors al director del PFC. De forma paral·lela, es va aconseguir treballar un model de simulació i experimentar amb un mètode de treball que servirà per crear-ne nous models de camps d'estudi variats.

20 Treball futur

Aquest projecte continua sent un prototipus d'un model de simulació que podrà créixer considerablement, ja que els límits del món a modelar són molt amplis.

De totes formes no només consisteix en afegir coneixement al model o ampliar els seus límits, sino també de consolidar la base tecnològica sobre la qual està construït, aplicant canvis de disseny a l'arquitectura del model per tal de fer-lo més estable, que s'entengui millor i sigui més flexible al canvi o incorporació de nou coneixement.

Des de el meu punt de vista, destaco dos punts per on es podria millorar el model per evolucionar el model un pas més:

20.1 Conversió completa a SDL

El primer pas en un treball futur seria implantar l'ús de l'operació CREATE per famílies i membres. Haurà de ser un pas senzill, ja que l'especificació està feta i els processos que ara simulen tots els agents i membres estan implementats de forma que amb uns petits canvis passin a ser processos preparats per ser instanciables.

Durant la realització d'aquest PFC s'ha aconseguit eliminar gran part del codi extern en C i implementar la funcionalitat associada dins els diagrames SDL. També hem comentat que era inevitable mantenir part del codi C, sobretot per guardar en estructures complexes algunes dades relacionades amb els agents.

Segons maduri el SDLPS i es puguin crear instàncies d'agents de tots els tipus, seria interessant aprofitar la potència del SDL com a llenguatge orientat a objectes i crear agents que representin les estructures complexes, ja que de fet són objectes.

Aquesta idea podrà fer-se realitat quan en un futur els agents SDLPS puguin tenir llistes com a tipus de variable (ara només accepten tipus simples). D'aquesta forma es podrà tenir navegabilitat entre objectes (agents) quan es tracta de relacions 1..n, guardant a les llistes els identificadors de les entitats associades.

20.2 Reforçament de la intel·ligència dels agents

Serà important que a cada iteració del projecte s'incrementi en la mida del possible la intel·ligència i el coneixement dels agents del sistema, ampliant els límits del món representat al model.

Des de el meu punt de vista un dels punts més prioritaris a millorar seria l'algoritme de decisió de les famílies, com pot ser en l'àmbit de compra/venda de propietats o producció de productes.

21 Bibliografia

Clark Labs. *IDRISI.* 2009. http://www.idrisi.com/ (últim accés: 26 / 11 / 2009).

«DARPA, U.S. Army looking for social computing technology.» *Homeland Security NewsWire*, 2009.

Fonseca i Casas, P. *Formalismes III, SDL.* sense data.

—. «SDL distributed simulator.» *Winter Simulation Conference 2008.* Miami: INFORMS, 2008.

—. «SDL, A Graphical Language Useful to Describe Social Simulation Models.» *Proceedings of the 2nd Workshop on Social Simulation and Artificial Societies Analysis (SSASA'08). Social Simulation and Artificial Societies Analysis.* Barcelona, 2008.

Fonseca i Casas, Pau, i Josep Casanovas. «Simplifying Gis Data Use Inside Discrete Event Simulation Model Through m:n-ac Cellular Automaton.» Editat per Chiara Brianco, Claudia Frydman, Antonio Guasch i Piera Miquel Angel. *Environmental Modeling and Simulation Symposium.* Marsella. - FRANCE, 2005. 7-15.

Fonseca i Casas, Pau, Màxim Colls, i Josep Casanovas. «Representing Fibonacci function through cellular automata using Specification and Description Language.» *Procediings of the 2010 Summer Simulation Multiconference.* 2010.

—. «Towards a representation of environmenal models using specification and description language.» *International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management.* València, 2010.

—. «Using Specification and Description Language to define and implement discrete simulation models.» *Procediings of the 2010 Summer Simulation Multiconference.* 2010.

Rello, M. «Lemúria: Simulació d'una Ciutat Medieval.» 2009.

«Wikipedia.» 2011. http://es.wikipedia.org/wiki/Juego_de_la_vida (últim accés: 2011).