

[1 SDL SUBSET Utilitzat 1](#_Toc413168249)

[1.1 CARACTERÍSTIQUES DEL SUBSET 1](#_Toc413168250)

[1.2 REPRESENTACIÓ D’UN MNCAK EN SDL 1](#_Toc413168251)

[1.2.1 Representació de l’MNCAK 1](#_Toc413168252)

[1.2.2 Representació de les capes (1) 1](#_Toc413168253)

[1.2.3 Representació de les capes (2) 1](#_Toc413168254)

[1.2.4 Implementació de la lògica de l’mnca 1](#_Toc413168255)

[2 Regenerar un model 1](#_Toc413168256)

[3 Estructura del model 1](#_Toc413168257)

[3.1 6ESTRUCTURA DE CARPETES 1](#_Toc413168258)

[3.2 CARACTERÍSTIQUES DELS DIAGRAMES 1](#_Toc413168259)

[4 MANUAL D’ÚS DE L’SDLPS 1](#_Toc413168260)

[4.1 ESTRUCTURA DE CARPETES 1](#_Toc413168261)

[4.2 INSTRUCCIONS PAS A PAS 1](#_Toc413168262)

[4.2.1 Carregar el model .sdlps en el programa SDLPS 1](#_Toc413168263)

[4.2.2 Parametritzar el model 1](#_Toc413168264)

[4.2.3 Definir una representació 3D del model. 1](#_Toc413168265)

[4.2.4 Crear el codi, compilar i enllaçar la *dll*. 1](#_Toc413168266)

[4.2.5 Configurar la simulació 1](#_Toc413168267)

[4.2.6 Executar la simulació 1](#_Toc413168268)

[4.3 EXECUCIÓ D'UN EXPERIMENT 1](#_Toc413168269)

[4.3.1 Resultat de l'experimentació 1](#_Toc413168270)

[4.3.2 Mode SaaS 1](#_Toc413168271)

[4.4 UTILITZACIÓ DE CODI EXTERN: SDLCodeExternal 1](#_Toc413168272)

[4.5 IMPLEMENTACIÓ D’UN *PROCEDURE* A L’SDLPS 1](#_Toc413168273)

[4.6 PROCESSOS EXTERNS 1](#_Toc413168274)

[4.7 PROCEDURES INTEGRATS A L’SDLPS 1](#_Toc413168275)

[4.7.1 Animation procedures 1](#_Toc413168276)

[4.7.2 Statisticas information retrieval procedures 1](#_Toc413168277)

[4.7.3 Miscelania 1](#_Toc413168278)

[4.8 Utilitats per debuggar 1](#_Toc413168279)

[4.9 CARACTERÍSTIQUES I FUNCIONALITATS DE L’SDLPS 1](#_Toc413168280)

[5 Manual d’ús de l’EYE DX 1](#_Toc413168281)

[5.1 REPRESENTACIÓ DE FITXERS WAVEFRONT OBJ 1](#_Toc413168282)

[5.2 REPRESENTACIÓ DE FITXERS IDRISI 1](#_Toc413168283)

[5.3 EVENTS DE REPRESENTACIÓ DE L’EYEDX 1](#_Toc413168284)

[5.3.1 EVENT EYE\_Report 1](#_Toc413168285)

[5.3.2 EVENT EYE\_AnimTo 1](#_Toc413168286)

[5.3.3 EVENT EYE\_RemoveAt 1](#_Toc413168287)

[5.3.4 EVENT EYE\_SetState 1](#_Toc413168288)

[5.4 PROBLEMES CONEGUTS 1](#_Toc413168289)

[6 Bibliografía 1](#_Toc413168290)

[7 Annexes 1](#_Toc413168291)

[7.1 MODEL FILE (.sdlps) 1](#_Toc413168292)

[7.2 REPRESENTATION FILE (.eye) 1](#_Toc413168293)

[7.3 SDL PROCESS File (.sdlprocess) 1](#_Toc413168294)

[7.4 PARAMETRIZATION File (.param) 1](#_Toc413168295)

[7.5 TRACE FILE (.trace) 1](#_Toc413168296)

[7.6 Eye main options 1](#_Toc413168297)

[7.6.1 Càmera: Moviments 1](#_Toc413168298)

[7.6.2 Càmera: Opcions i configuració 1](#_Toc413168299)

# SDL SUBSET Utilitzat

## CARACTERÍSTIQUES DEL SUBSET

Per tal de poder representar qualsevol tipus de model mitjançant SDL s’ha utilitzat una variant (*subset*) amb les següents característiques:

* **Ús de codi C++** en lloc del pseudocodi que utilitza SDL. Això afecta als agents *Task*, *DCL*, *Procedure*, *Decision* i *Output* (*Text Extension*)
* Especificació del destí a on s’enviarà una senyal. D’aquesta manera l’agent Output es complementarà:
  + Mitjançant la cadena **TO** seguida del Procés al qual es vol enviar la senyal. Als exemples, en el primer cas s’enviarà la senyal cap al Procés amb identificador ‘Block\_Process’, en el segon exemple la senyal s’enviarà al mateix Procés i al tercer exemple la senyal s’enviarà al Procés que tingui com identificador el contingut de la variable *stringVar*.

Event TO ‘Block\_Process’

Event TO SELF

Event TO stringVar

* + Mitjançant la cadena **VIA** seguida del canal pel qual es vol enviar la senyal. El canal es pot especificar literalment mitjançant el seu nom entre cometes simples o bé utilitzant una variable char\*.

Event VIA ‘channel\_1’

Event VIA stringVar

* **Extensió de l’agent *Output*** mitjançant la forma (Visio Shape) *Text Extension.* La extensió serveix per completar l’enviament de l’event especificant el valor de variables associades a l’event:

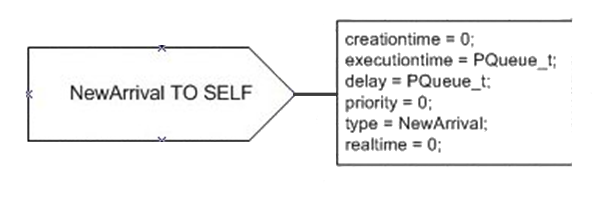


Figura - Exemple d’utilització del *text extension*

Les variables pròpies de l’event que es podran modificar seran:

* + double delay; //indica el temps que tarda a arribar al destí
  + int priority; //prioritat en cas d’arribada de dos events simultàniament

Una descripció més profunda d’aquesta extensió es pot trobar a:

(Fonseca, SDL, A Graphical Language Useful to Describe Social Simulation Models., 2008)*.*

## REPRESENTACIÓ D’UN MNCAK EN SDL

### Representació de l’MNCAK

Un mnCAk és una generalització d’un autòmat cel·lular on *m* és el nombre de capes, *n* les dimensions de cada capa i *k* el nombre de capes principals.

Per representar un mnCAk s’utilitzarà una *Shape* de tipus *Block* on s’afegirà el prefix “MNCA\_” davant el nom del bloc. Així si volem definir un mnCAk anomenat “MA”:



Figura – Exemple de Block

La definició de l’mnca estarà dins del document que implementi el block. Aquest document tindrà el mateix nom que l’mnca (mnca\_ma.vsdx).

Es pot trobar informació relacionada amb aquest capítol a:

(Fonseca, Colls, & Josep, Representing Fibonacci function through cellular automata using Specification and Description Language, 2010)

(Fonseca, Colls, & Casanovas, A novel model to predict a slab avalanche configuration using m:n-CAk cellular automata, 2011)

(Fonseca, Colls, & Casanovas, Towards a representation of environmenal models using specification and description language, 2010)

### Representació de les capes (1)

Atenció: aquesta versió està en desús tot i que es segueix mantenint. Es recomana utilitzar l’especificada a la següent secció (1.2.3).

Es diferencien dos tipus de capes: capes principals i secundàries. Les capes principals són aquelles que es modifiquen al transcurs de la simulació. Les secundàries, en canvi, no modifiquen el seu contingut.

Les **capes principals** es declaren mitjançant blocs que estan dins el block que hereda de mnca.

Les **capes secundàries** es declaren dins el requadre DCL utilitzant el prefix “layer\_”.

En el cas dels dos tipus de capes, cada capa està lligada a un fitxer raster en format IDRISI. Aquests fitxers es trobaran dins la carpeta /data/ i tindran el mateix nom que la capa (menystenint el prefix).

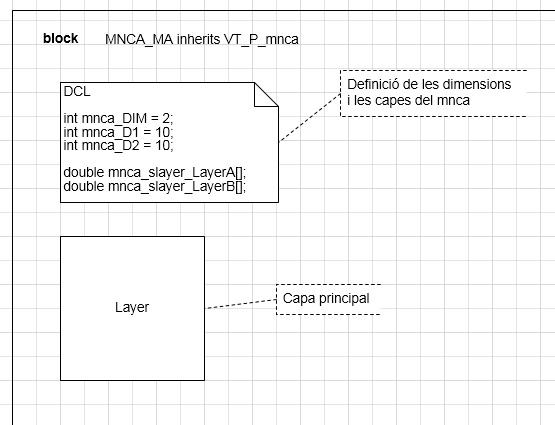
En cas contrari, si no hi ha un fitxer IDRISI associat, s’haurà d’especificat el nombre de dimensions i el tamany de cada dimensió mitjançant les DCL “mnca\_DIM”, “mnca\_D1”, “mnca\_D2”, etc.

Figura – Exemple de definició d’un mnca

### Representació de les capes (2)

En aquesta representació d’un mnca no hi ha distincions entre tipus de capes.

Si es disposa d’almenys un fitxer ràster de tipus IDRISI que defineixi les dimensions de la capa, no cal declarar les variables *mnca\_DIM, mnca\_D1, mnca\_D2* per dimensionar l’mnca Figura 3. Els fitxers es trobaran dins una carpeta anomenada /data/ al mateix directori a on es troba el fitxer .sdlps.

En l’exemple de la Figura 4, l’mnca té un total de 8 capes, 4 de les quals contenen els valors definits als fitxers ràster especificats. Les 4 capes restants (*path, speed, time, force*) tindran inicialment un valor per defecte de *0*.

En aquest tipus de representació, el block *Block\_State* no es representa com una capa sinó que conté la lògica de l’mnca que modificarà el contingut de les capes.

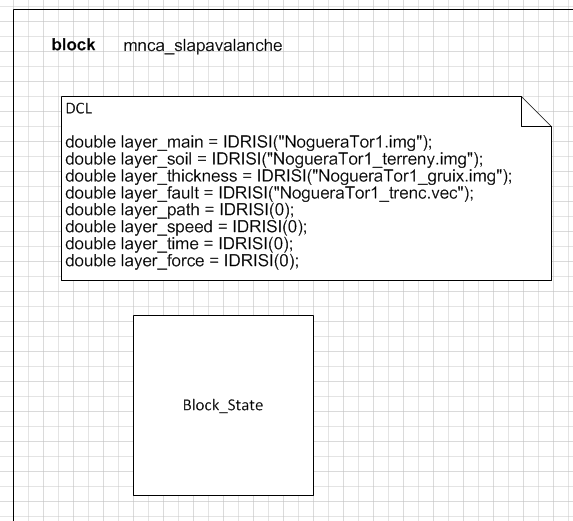


Figura - Exemple 2 de definició d'un mnca

### Implementació de la lògica de l’mnca

La lògica o motor (*engine*) que governa el funcionament d’un mnca es defineix en un agent de tipus *Process*. Per això, és necessari disposar d’eines que permetin interactuar amb les capes de l’mnca. Aquesta interacció es fa mitjançant *Procedures.*

A continuació es defineixen els *Procedures* bàsics per poder implementar la lògica de l’mnca:

#### mncaGetCurrentCell(int nCell)

Aquest procedure té un paràmetre de sortida (nCell) el qual se li assigna el número de cel·la actual.

#### mncaGetCellValue(char\* strKey, int nCell, double dValue)

Retorna el valor d’una cel·la i capa determinada. Té dos valors d’entrada (strKey, nCell) i un de sortida (dValue).

#### mncaSetCellValue(char\* strKey, int nCell, double dValue)

Aquest procedure assigna un valor a una cel·la i capa determinada. Els tres valors són d’entrada.

#### mncaGetMooreVicinity(char\* strKey, int nCell, double dValueNW, double dValueN, double dValueNE, double dValueE, double dValueSE, double dValueS, double dValueSW, double dValueW)

Retorna els valors de totes les cel·les veïnes (considerant veïnatge de moore). Té dos valors d’entrada (strKey, nCell) i vuit de sortida.

#### mncaGetNeighborID(int nDir, int nCell)

Retorna el número de cel·la d’una cel·la veïna a partir d’una direcció donada. El conveni de direccions és el següent:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

El procediment habitual a l’hora d’establir els mecanismes que comuniquen les capes i la lògica del model és el següent:

* + Per una banda hi ha la **Funció de Nucli** que és la responsable d’actualitzar els valors de les capes. Aquesta funció s’implementa mitjançant un *Procedure* que s’alimenta dels *Procedures bàsics* descrits prèviament.
  + Per altra banda hi ha la **Funció de Veïnatge** que s’ocupa de consultar la informació de les capes necessària perquè el model funcioni. Igual que la funció de nucli, utilitzarà els *Procedures bàsics.*

A continuació s’il·lustra un exemple del que podria ser una funció de nucli.

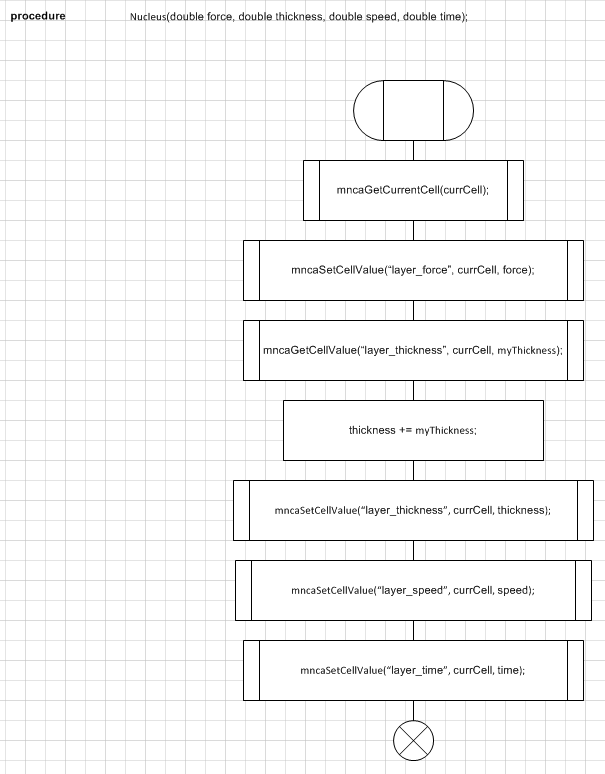


Figura - Exemple d'un *Procedure* que implementa una Funció de Nucli

# Regenerar un model

És necessari tenir instal·lat el MinGW i en la variable PATH de sistema tenir inclòs el directori “mingw\_path \_installed \bin”.

En la carpeta doc del directori on es treballa, i on es troben els fitxers necessaris per a fer la simulació, ha d’haver-hi els fitxers de Visio que contingui guardats en format xml dibuix “vsdx”.

Carregar el model fer les modificacions que es desitgin amb la plataforma Visio i guardar-los o sobreescriure amb el format que s'ha dit en el paràgraf anterior "vsdx" i prémer el botó **Regenerate** per a desar els canvis i que es regenerin els fitxers “.sdlps”. S’ha de tenir en compte que es sobreescriuran.

El fitxer .sdlps principal es troba en l’arrel del directori, tota la resta es troben en la carpeta model (els blocks, procedures i/o process ) que defineixin els diferents blocs en altres fitxers de Visio.

Regenerar el codi c i la DLL i ja es poden llançar les simulacions.

# Estructura del model

El model està compost per un conjunt de fitxers .sdlps (xml) que genera SDLPS a partir dels diagrames *"vsdx"* que estàn explicats en l'apartat anterior i que genera el programa Visio. Per a generar els fitxers sdlps sencillament s'obre el fitxer arrel del model.

Els diagrames SDL es representen amb l’ajut del Microsoft Visio i el *plug-in* SanDriLa (<http://www.sandrila.co.uk/>) o bé amb una plantilla que es troba associada amb la plataforma, que permeten la utilització de les formes estàndards de l’SDL.

## 6ESTRUCTURA DE CARPETES

Si volem generar el fitxer .sdlps a partir dels diagrames SDL (fitxers .vsdx) caldrà tenir els fitxers correctament localitzats.

En l’exemple següent es mostra el cas que vulguem exportar el model anomenat GG2:

/GG2/

GG2.sdlps

/doc/

GG2.vsdx

NomBlock1.vsdx

NomProcess1.vsdx

NomProcedure1.vsdx

...

La carpeta que conté els diagrames del model ha de tenir el nom del propi model, en aquest cas /GG2/. Dins del directori del model hi haurà la carpeta /doc/ que contindrà tots els arxius dels diagrames SDL. El nom dels fitxers ha de correspondre amb el nom de l’agent que representa i el fitxer arrel ha de tenir el mateix nom que el directori en aquest exemple GG2.

## CARACTERÍSTIQUES DELS DIAGRAMES

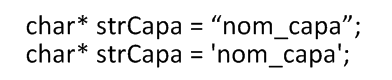
Per a que els diagrames SDL es puguin reconèixer s'han de seguir una sèrie de regles que són les següents:

Perquè el *plug-in* funcioni correctament s’han de complir una sèrie de característiques:

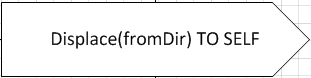
* + En el diagrama *Process*: cada estat ha d’estar definit en una sola pestanya diferent. Un *Process* amb 3 estats ha de tenir 3 pestanyes.



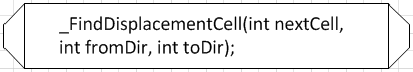
* + Els char\* literals han d’estar entre cometes dobles. Aquestes cometes dobles han de ser les pròpies d’Office. Sinó, s’utilitzaran cometes simples:

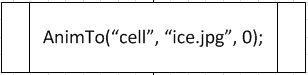


* + Per enviar una senyal al propi *Process* s’utilitza el text “TO SELF” a continuació del nom de la senyal i els possibles paràmetres d’usuari:



* + Tant la declaració com la crida dels *Procedures* acaben amb “punt i coma”:





# MANUAL D’ÚS DE L’SDLPS

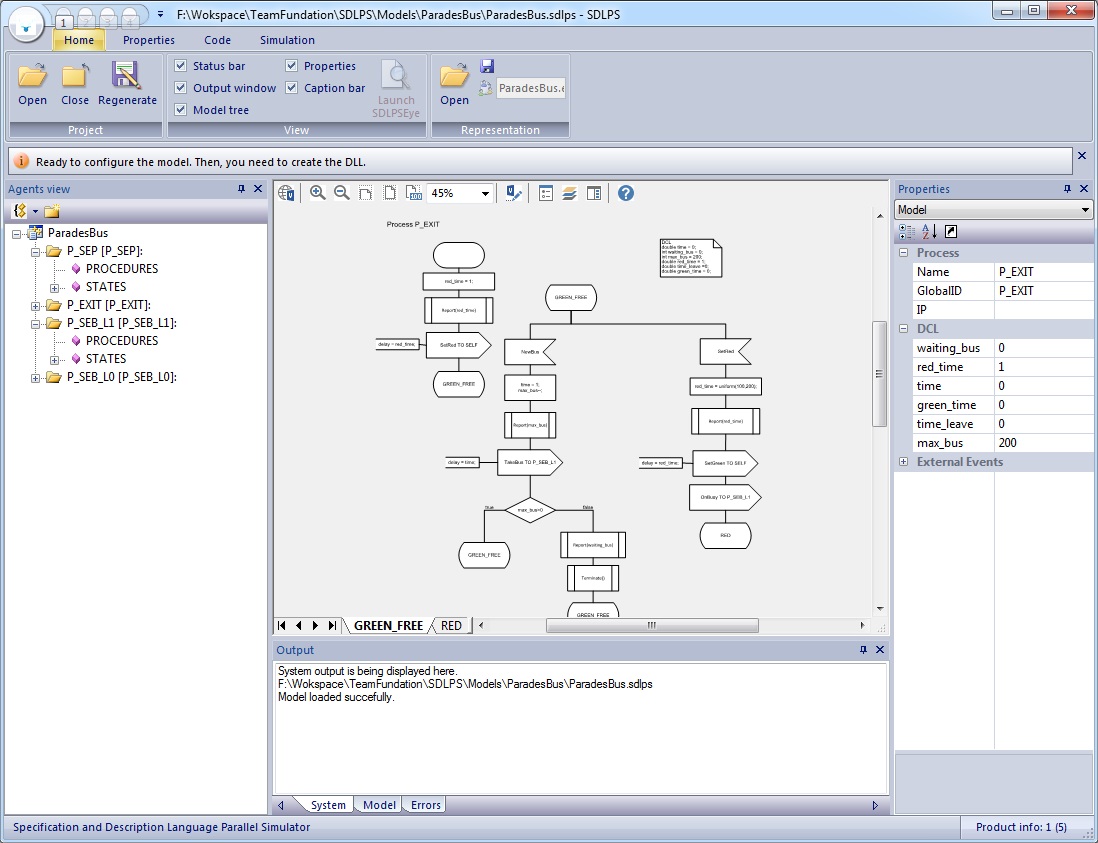


Figura – Captura de pantalla del simulador SDLPS

Per a realitzar una simulació primer cal disposar de l’arxiu .sdlps que conté la estructura i la lògica del model.

## ESTRUCTURA DE CARPETES

A la carpeta on es troba el model (fitxer .sdlps) s’hi troben altres fitxers relacionats com ara parametritzacions (configuracions) del model, fitxers de representació, traces de simulació i diagrames del model, continuem amb l'exemple de GG2.

/GG2/

GG2.sdlps (arxiu generat)

GG2.eye (representació per defecte)

GG2.param (parametrització per defecte també s'ha de generar)

/doc/

GG2.vsdx (diagrama de system)

NomBlock1.vsdx

NomProcess1.vsdx

NomProcedure1.vsdx

...

/params/ (parametritzacions)

Param1.param

...

/model/ (arxius generats associats als diferents diagrames)

NomBlock1.sdlps

NomProcess1.sdlps

...

/traces/ (conté totes les traces que s'han generat de les simulacions realitzades)

GG2 (data).trace

GG2 (data2).trace

Aquests arxius estan classificats en carpetes relatives a l’arxiu del model (.sdlps).

És important que aquesta jerarquia de directoris es conservi per tal que el simulador SDLPS funcioni correctament. Si es vol definir una parametrització i/o representació per defecte, els arxius que ho defineixin es trobaran a la mateixa carpeta que el model i tindran el mateix nom que aquest.

## INSTRUCCIONS PAS A PAS

### Carregar el model .sdlps en el programa SDLPS

Per carregar un model es pot aconseguir de dues formes diferents la primera és l'explicada en els apartats anteriors i la segona és carregar directament el fitxer .sdlps del model, que s'ha generat en anterioritat.

Si no hi ha cap error durant la càrrega apareixerà l’estructura dels agents en el panell *Agents View*.

### Parametritzar el model

El procés de parametrització del model consisteix en aplicar una parametrització concreta. Aquest procés es pot ometre si volem executar la parametrització per defecte, o sigui la definida en el model.

Per aplicar una parametrització es pot fer manualment modificant els valors de les variables DCL en el panell *Properties* o important un fitxer de parametrització (.param) des de la pestanya *Properties.* Per modificar manualment el valor de les variables DCL d’un *Process* concret primer cal navegar per l’arbre *Agents View* i seleccionar un element de tipus *Process*.

Un com s’hagi acabat de parametritzar el model, aquesta parametrització es pot exportar des de la pestanya *Properties*. Des del mateix panell, es podran importar parametritzacions ja existents.

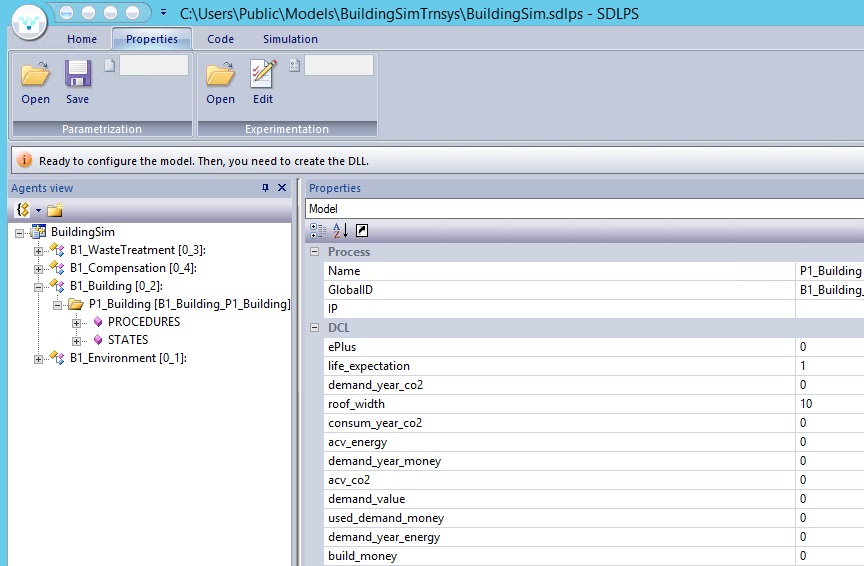


Figura – A la part superior, es mostra la pestanya *Properties* des d’on es pot importar/exportar parametritzacions i representacions. A la part inferior esquerra, el panell *Agents View* amb l’estructura del model. A la part inferior dreta, el panell *Properties* d’es d’on es poden modificar les variables DCL i la representació d’un Process.

### Definir una representació 3D del model.

De manera similar es poden crear/exportar/importar fitxers de representació. Per modificar la representació d’un *Agent Process* primer cal seleccionar-lo a l’arbre *Agents View* i seguidament, en el panell Properties canviar la opció del desplegable a *Representation.* Un cop seleccionada la opció del desplegable, es podran definir les diferents representacions del *Process* per a cada estat.

De la mateixa manera que amb les parametritzacions, les representacions es poden Importar i Exportar des de la pestanya *Properties.*

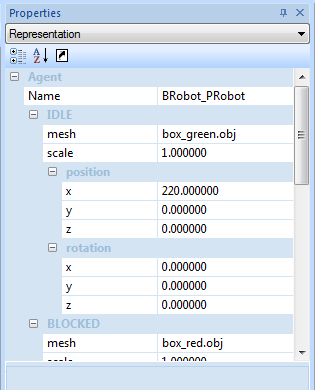


Figura – Panell *Properties* editant la representació del *Process ‘BRobot\_PRobot’*

### Crear el codi, compilar i enllaçar la *dll*.

Un canvi en la parametrització o representació del model implica generar de nou la DLL que executa el model. Aquest procés es realitza des de la pestanya *Code* mitjançant el botó: *Create, Compile and Link DLL*.

### Configurar la simulació

Abans d’inicialitzar la simulació es pot definir el final de la simulació. A la pestanya *Simulation* es troba el botó *Simulation time*. Des del menú que apareix, podem definir com s’acabarà la simulació: per temps, nombre d’iteracions o mai.

### Executar la simulació

Un cop s’ha acabat tot el procés previ a la simulació es podrà inicialitzar i executar. L’execució del model pot ser pas a pas o contínua. Per definir quin tipus d’execució es vol, abans d’inicialitzar la simulació es marcarà si es vol una simulació *step by step* o no. Un cop s’hagi configurat la simulació és moment d’inicialitzar-la (*Initialize*) i posteriorment executar-la (*Run*).

L’execució de la simulació genera una traça que es guarda en un fitxer extern (.trace) dins la carpeta traces relativa al fitxer del model.

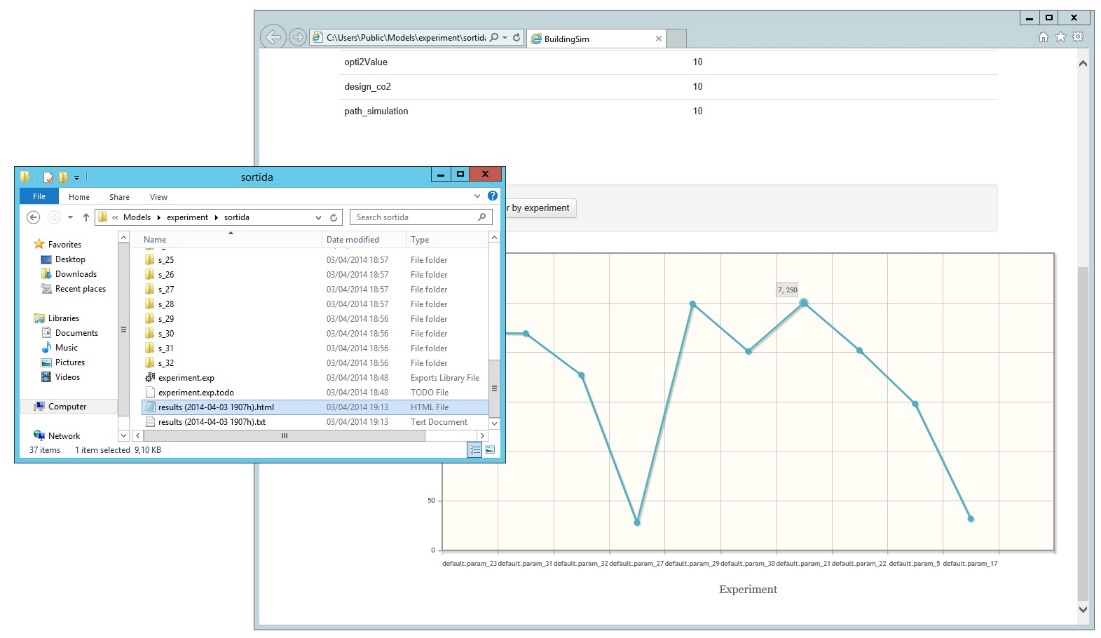
## EXECUCIÓ D'UN EXPERIMENT

En alguns casos es voldrà executar no solament una simulació, com s'ha explicat en l'últim punt de l'apartat anterior, sinó un conjunt d'elles, amb diferents dades d'entrada, és anomenat experimentació, els passos per a dur-ho a terme són els següents :

1. Primerament s'han de plantejar i definir les diferents permutacions que es volen simular. Crear un fitxer .perm que les contingui i desar-lo amb el nom de permutation.perm. Aquest fitxer .perm està explicat en detall a l'annex amb el nom de Documentació del fitxer de permutacions.
2. Generar amb l'aplicació SDLPS una parametrització associada al model i copiar-la al mateix directori que el fitxer de permutacions.
3. Utilitzant l'IDFPlugin generar cada una de les carpetes associades a cada simulació de l'experimentació. Executant el plugin, via comanda, amb els següents paràmetres, directori arrel del fitxer experiment.perm (també ha d'estar el fitxer default.param) i el directori de sortida, en l'últim directori és on es crearà un directori per a cada una de les permutacions amb el nom s\_\* on \* será un número enter entre 1 i el número de permutacions.
4. Obrir amb SDLPS la experimentació que ha creat el plugin amb el nom de experiment.exp, en cas que es vulgui executar una optimització de l'experiment és possible en la funcionalitat d'Edit on és necessari afegir les variables les quals es vol optimitzar, si es volen maximitzar o minimitzar a més si es vol associar a alguna restricció.
5. La primera vegada que s'executa l'experimentació és necessari generar una DLL per a cada simulació de l'experiment, en la pestanya Code es pitja a Create DLL la qual farà aquestes DLLs, en cas que no s'hagi generat el codi amb anterioritat també és necessari generar el codi, compilar-lo i "Linkar-lo".
6. Finalment en la pestanya Simulation es configura els paràmetres de simulació a Configuration, s'escull el tipus de finalització del model i l'algoritme d'optimització que es vol amb els paràmetres associats. Posteriorment s'inicialitza amb el botó de Initialize i es llença l'experimentació amb el botó Experiment.

### Resultat de l'experimentació

En la carpeta on es troben tots els directoris de les permutacions en quan s'ha acabat l'execució es generen dos fitxers un HTML i un txt, el segon és una traça amb cada un dels reports que s'ha fet en totes les simulacions, el primer conté les mateixes dades però el qual es pot obrir amb un explorador i veure algunes gràfiques associades amb els resultats.



### Mode SaaS

Aquest mode consisteix en monitoritzar un directori i executar una experimentació en el moment que aquest directori monitoritzat es crea o apareix un fitxer .exp, llavors es llença aquest experiment automàticament, quan acaba segueix monitoritzant amb la mateixa funció. Per a poder monitoritzar és necessari obrir un model, i seguir els passos 1,2,3 i 5 a l'inici d'aquest capítol, si es vol es pot associar a la monitorització una optimització, prement amb anterioritat a la opció de Configure, finalment es pitja a Folder Monitoring i es selecciona la carpeta escollida.

## UTILITZACIÓ DE CODI EXTERN: SDLCodeExternal

Per tal de simplificar la definició dels models és possible utilitzar rutines implementades en codi C en un fitxer extern. El codi ha d’estar implementat en un fitxer .c i ha d’anar acompanyat d’un fitxer de capçalera amb el mateix nom, p. ex: *fitxer1.c* i *fitxer1.h*. Cada funció implementada al fitxer .c ha d’estar declarada al corresponent fitxer de capçalera.

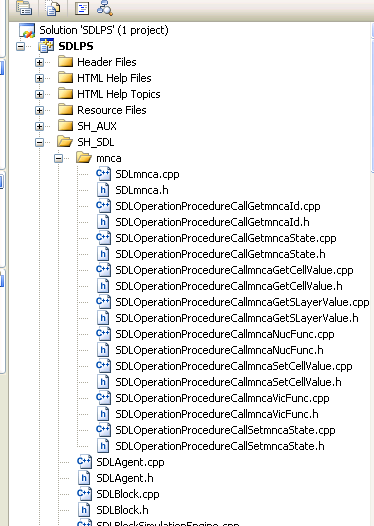
Cada parella de fitxers ha d’estar inclosa en un fitxer anomenat SDLCodeExternal.h que és el fitxer que inclou per defecte el simulador SDLPS. Per exemple, el contingut del fitxer SDLCodeExternal.h podria ser:

#include “fitxer1.h”

#include “fitxer2.h”

seguit de tots els *includes* que es desitgin.

## IMPLEMENTACIÓ D’UN *PROCEDURE* A L’SDLPS

Molts cops ens interessarà implementar un *Procedure* directament a l’SDLPS. Per això haurem de crear una subclasse de la classe *CSDLOperationProcedureCall* i reimplementar el mètode *Execute()*.

Un cas pot ser la implementació de les classes *CSDLOperationProcedureCallmncaNucFunc* i *CSDLOperationProcedureCallmncaVicFunc* que implementen les funcions de nucli i veïnatge respectivament.

Sempre que s’afegeix la implementació d’un *Procedure* cal modificar el mètode *LoadOperationProcedureCall* de la classe *CSDLLoader*. Cal indicar que si a l’XML s’especifica que el Procedure té una implementació (atribut *implementation*) aquest està implementat en una classe de l’SDLPS:

Figura – Estructura de carpetes dels fitxers que implementen els *Procedures*.

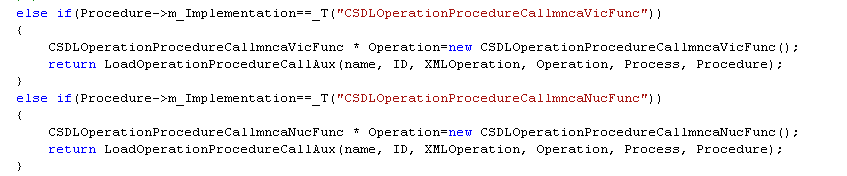


Figura – Part del codi c++ en el fitxer SDLLoader.cpp on es carrega el *Procedure* implementat.

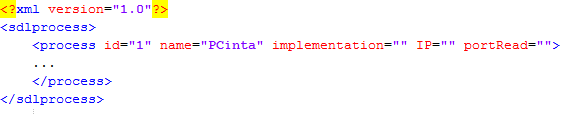
## PROCESSOS EXTERNS

Es pot donar el cas de voler aprofitar un agent ja implementat. El cas més comú és el dels Processos (agent *Process*) ja que aquest conté la lògica d’un mecanisme que possiblement es voldrà reaprofitar.

Aquesta opció simplifica molt els arxius .sdlps i fa que aquests es puguin separar en parts. Tot seguit hi ha un exemple:



Aquest fragment d’un fitxer .sdlps defineix un *Block*. A l’exemple però, el *Process* que aquest bloc conté no està implementat en el mateix fitxer sinó a l’arxiu “CINTA.sdlprocess”. A continuació es mostra el contingut d’aquest fitxer:



Tal i com es veu, aquest fitxer conté la implementació del procés. Cal tenir en compte que les variables DCL definides en el fitxer arrel (el primer) sobreescriuran el contingut definit al fitxer .sdlprocess. Això permet parametritzar els processos definits en un fitxer extern.

L’estructura de carpetes que s’ha de mantenir per utilitzar processos externs és la següent:

/GG2/

GG2.sdlps

/doc/

GG2.vsdx (diagrama de system)

NomBlock1.vsdx

NomProcess1.vsdx

NomProcedure1.vsdx

...

/SDLCodeExternal/

SDLCodeExternal.h (conté tots els includes dels fitxers de capçaleres)

Codi1.h (capçaleres de les funcions)

Codi1.c (codi de les funcions)

...

D’aquesta manera el simulador SDLPS trobarà la implementació dels processos externs i mostrarà els diagrames SDL que els representen.

## PROCEDURES INTEGRATS A L’SDLPS

L’SDLPS té diversos *Procedures* implementats per tal de definir de manera completa els diagrames SDL.

Els *Procedures* relacionats amb els mnca estan descrits a la secció 1.2.4 (Implementació de la lògica de l’mnca).

Altres *Procedures* són*:*

### Animation procedures

#### AnimTo

**AnimTo(char\* strID, char\* meshFileName, double dDelay, double dPosX, double dPosY)**

Aquest procedure s’utilitza per representar el model. Representa l’objecte amb ID *strID* amb una malla definida per *meshFileName*. La representació es posicionarà mitjançant dPosX i dPosY i es pot retardar mitjançant un *delay.*

Si no existeix l’objecte amb identificador *strID,* se’n crearà un de nou.

Nota: Els *Processes i Blocks* s’identifiquen afegint recursivament el nom de l’agent pare fins arribar al system: exemple: “Block1\_SubBlock1\_Process1”

**AnimTo(char\* strID, char\* textureFileName, double dDelay)**

En el cas d’una cel·la, les representacions es realitzen mitjançant textures. El primer paràmetre ha de valdre “cell”, el segon defineix la textura que pendrà la cel·la (arxius .jpg, .png i .dds) i el tercer el *delay* de la representació.

#### RemoveAt

**RemoveAt(char\* strID)**

El *Procedure* elimina l’objecte amb identificador *strID*.

### Statisticas information retrieval procedures

#### Report

**Report(double Variable);**

Escriu a la traça un event <Eye\_Report> especificant el nom de la variable, tipus, valor, agent al que pertany i temps d’execució. Aquest *Procedure* és molt útil per debugar l’aplicació o per recollir estadístics.

#### ReportArray

**ReportArray(char\* desc, double Variable);**

Escriu a la traça un event <Eye\_Report> especificant el nom de la variable, tipus, valor, agent al que pertany i temps d’execució. A diferència del anterior ara el paràmetre que espera es un array. Hem de tenir dos paràmetres, el primer es un array que conté els descriptors, el segon es un array que conté els valors. Està pensat per recuperar informació estadística del model en forma de series temporals.

### Miscelania

#### Execute

**Execute(char\* program\_path)**

Executes a program that is on the “program\_path”.

#### GetAgentClock

**GetAgentClock (double Variable);**

Obté el temps associat al agent procés que el crida.

#### Terminate

**Terminate**

To end the simulation in the current PROCESS.

#### GetAgentSender

**GetSender(char\* PId);**

Obté el PId del AGENT que ens ha enviat la senyal.

#### SetAgentPIdToQueue

**SetAgentPIdToQueue(char\* PId)**

Afegeix a una cua de char\* el PId del element. Es útil quan volem guardar peticions que es fan a un AGENT.

#### GetAgentPIdFromQueue

**GetAgentPIdFromQueue(char\* PId)**

Obté de la cua de PId el que està al principi, l’elimina de la cua.

#### GetAgentPIdQueueNum

**GetAgentPIdQueueNum(int num)**

Returns the number of agents PId that are stored on the queue.

#### OpenFile

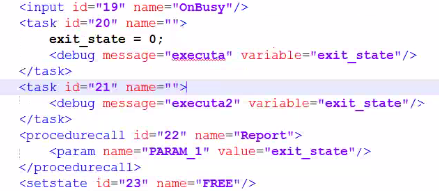
#### GetModelPath

**GetModelPath(char\* path)**

Obtains the path of the corrent model.

## Utilitats per debuggar

Per depurar les aplicacions podem emprar el **task**, afegint el camp debug directament sobre el fitxer xml generat per SDLPS com es mostra en la següent figura:



SDLPS dona un missatge en la finestra de error en la que es mostra el missatge “executa” i el valor de la variable “exit\_state”.

## CARACTERÍSTIQUES I FUNCIONALITATS DE L’SDLPS

Característiques reportades i funcionalitats pendents de l’SDLPS:

* + Les **declaracions dins els Procedures** no s’utilitzen. Dins un Procedure, quan s’utilitza una variable, primer es consulta si és un paràmetre i després si és una variable de procés.
  + Quan es **compila una DLL** que ja ha estat compilada una vegada (ja existeix el fitxer .o) el missatge que mostra l’SDLPS no és correcte: s’ha de comprovar que el fitxer .o tingui la data de creació correcte ja que l’SDLPS únicament comprova si el fitxer existeix o no per considerar que s’ha compilat correctament.
  + Tots els estats (en el marc del diagrames de PROCESSOS) han de estar al inici o al final en els diagrames de processos. No podem tenir un estat que estigui a la meitat d’un procés.

# Manual d’ús de l’EYE DX

L’*EyeDX* o *SDLPSEye* és un client per l’SDLPS que interpreta i representa en un espai virtual la traça que genera el simulador.

Pot representar l’espai 3D mitjançant:

* + Un fitxer de malla: fitxer .obj (WaveFront) i .img (IDRISI).
  + Un fitxer de traça (.trace).
  + Una connexió amb el simulador SDLPS.

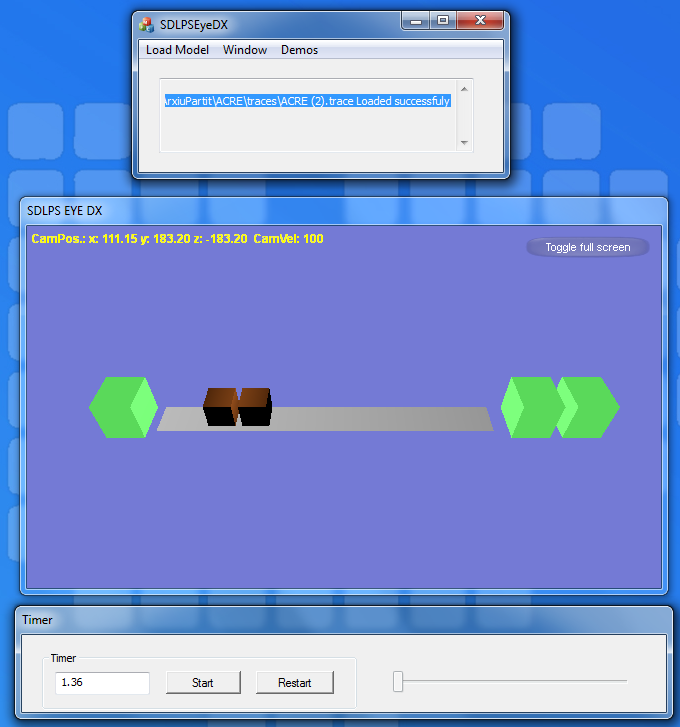
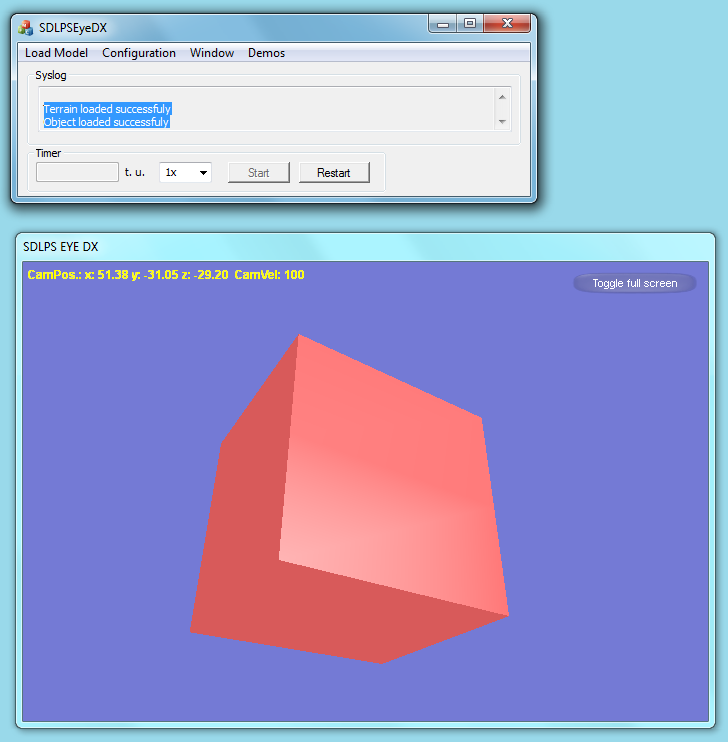


Figura – Captura de pantalla del programa EYEDX on es veu representat un model en 3D

## REPRESENTACIÓ DE FITXERS WAVEFRONT OBJ

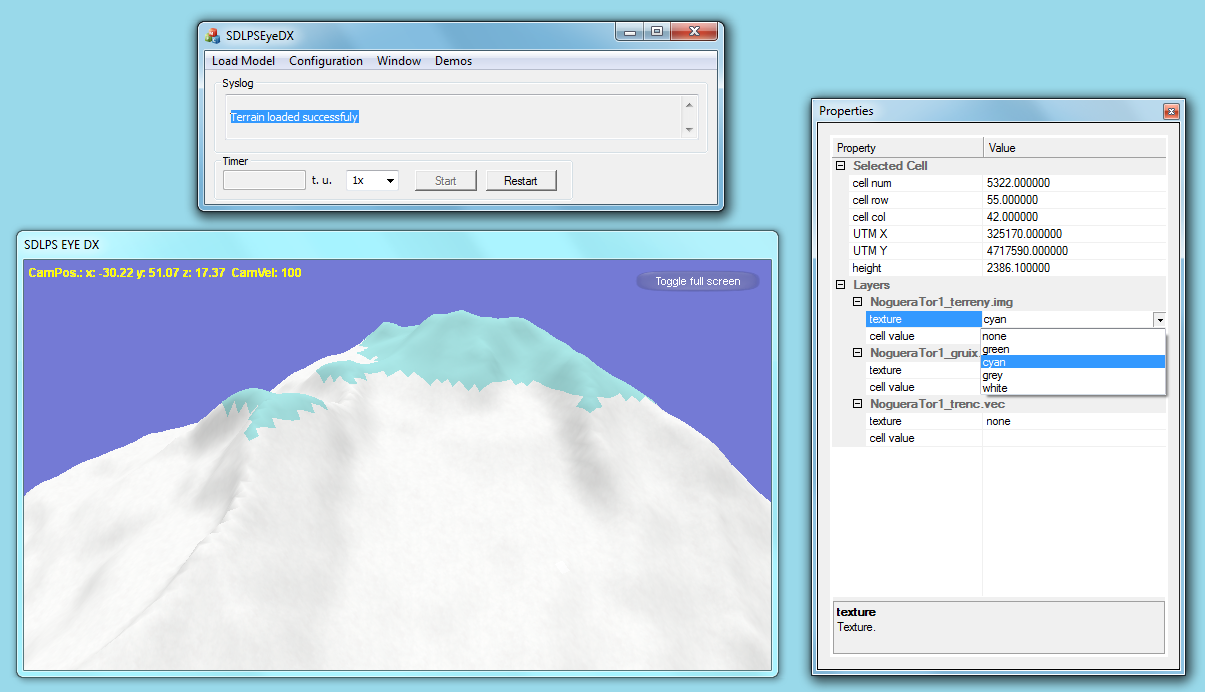
Des del menú *Load Model > From File* es pot representar un fitxer en format *Wavefront Object \*.obj.* Un cop s’ha carregat el fitxeres pot moure la càmera mitjançant el teclat i el mouse. El manual de la interfície es troba a l’Annex 2.



## REPRESENTACIÓ DE FITXERS IDRISI

Mitjançant el menú *Load Model > From File* es poden carregar models digitals de terreny en format IDRISI. El visor crea virtualment el terreny i detecta si hi ha altres capes dins la carpeta que conté el fitxer base (fitxer d’altures). Les capes addicionals que s’hagin trobat es poden representar amb diferents textures des del panell *Properties.*

Es pot obtenir informació d’una cel·la mitjanant el cursor. Clicant amb el botó dret sobre una cel·la recuperarem la informació de les capes així com també les coordenades UTM del punt.



## EVENTS DE REPRESENTACIÓ DE L’EYEDX

### EVENT EYE\_Report

Aquest event reporta el valor d’una variable concreta d’un *Process*.

Serveix per debugar l’aplicació o per visualitzar un valor d’interès pel simulador.

Exemple:

<EYE\_Report xTime=”” agent="">

<variable name="" type="" value=""></variable>

</EYE\_Report>

### EVENT EYE\_AnimTo

Aquest event permet crear una animació d’un Agent.

Si l’agent que s’especifica no existeix es crearà una nova representació al punt/rotació/escalat especificat.

Si l’agent ja existeix, es crearà una animació lineal cap al punt/rotació/escalat especificat. En cas que la malla de representació canviï, l’animació no serà lineal sinó que serà sobtada.

Exemple:

<EYE\_AnimTo xTime=”1.0” agent=”BRobot\_PRobot”>

<mesh scale=”1”>blue\_box.obj</mesh>

<pos x=”0” y=”0” z=”0” />

<rot x=”0” y=”0” z=”0” />

</EYE\_AnimTo>

### EVENT EYE\_RemoveAt

L’event RemoveAt elimina un Agent de la representació.

Exemple:

<EYE\_RemoveAt xTime=”2.5” agent=”BRobot\_PRobot”> />

### EVENT EYE\_SetState

Aquest event canvia l’estat d’un agents i per tant la seva representació. La representació de cada estat està especificada abans del tag <Events>, dins de <ModelInfo>. Per a més informació consultar la secció “Tipus de Fitxers”.

Exemple:

<EYE\_SetState xTime=”5.0” agent="BADP\_PADP" state="ACTIVE" />

En el cas que l’agent que està canviant d’estat es tracti d’un mnCA, cal especificar també el numero de cel·la:

<EYE\_SetState xTime=”5.0” agent="BLayer\_PLayer" state="EMPTY" cell=”23” />

## PROBLEMES CONEGUTS

* + A vegades la càmera no funciona correctament si el temps no corre.
  + En mode *Debug* salta un ASSERT al consultar el valor d’una cel·la del terreny (Panell Properties).

# Bibliografía

Fonseca, P. (2008). SDL, A Graphical Language Useful to Describe Social Simulation Models. *SSASA.*

Fonseca, P. (2008). SDLPS, a SDL distributed simulator. *WinterSim.*

Fonseca, P. (2010). Using Specification and Description Language to define and implement discrete simulation models. *UPCcommons* .

Fonseca, P. (2011). Using specification and description language to represent users’ profiles in OMNET++ simulations.

Fonseca, P., Colls, M., & Casanovas, J. (2011). A novel model to predict a slab avalanche configuration using m:n-CAk cellular automata. *Computers, Environment and Urban Systems* .

Fonseca, P., Colls, M., & Casanovas, J. (2010). Towards a representation of environmenal models using specification and description language.

Fonseca, P., Colls, M., & Josep, C. (2010). Representing Fibonacci function through cellular automata using Specification and Description Language. *SumerSim.*

# Annexes

## MODEL FILE (.sdlps)

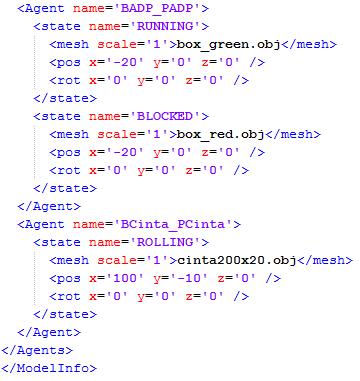
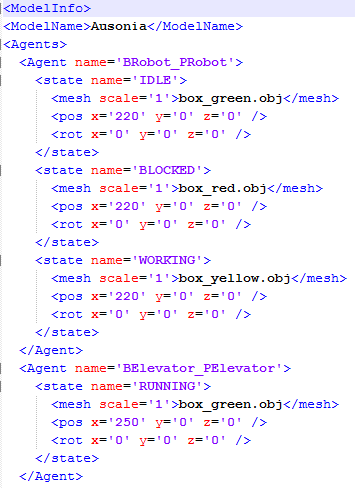
Aquest fitxer conté la estructura de Blocs i Processos i les seves connexions (canals). Aquest fitxer es pot generar manualment o mitjançant el *plug-in* de *Microsoft Visio*.



## REPRESENTATION FILE (.eye)

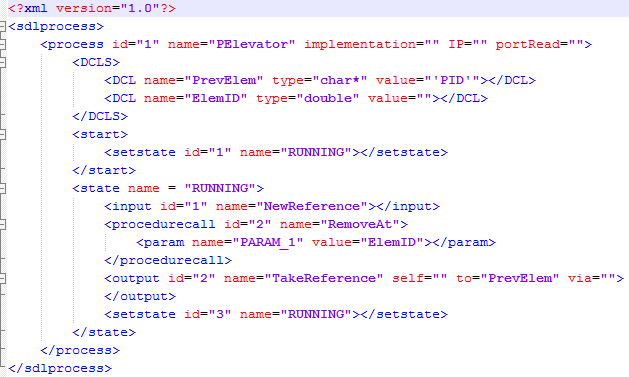
Aquest fitxer conté la informació referent a la representació del model dins un espai 3D. Cada agent que vol ser representat tindrà definida una representació per a cada un dels seus estats. La informació de la representació consta de:

* Escalat i fitxer de la malla (format *Wavefront* *Object* File \*.obj)
* Posició x, y, z
* Rotació x, y, z



## SDL PROCESS File (.sdlprocess)

Un *Process* pot estar implementat en un fitxer extern d’aquest tipus. D’aquesta manera es pot construir una llibreria d’objectes i reutilitzar-los en diferents models.



## PARAMETRIZATION File (.param)

Aquest fitxer conté les parametritzacions d’un model concret. Una parametrització concreta pot ser la configuració d’una execució d’un joc d’experiments. S’especifiquen valors concrets de variables DCL que caracteritzen el model i Events externs que s’executaran a la simulació.

Exemple:

<Parametrizations>

<DCLS>

<DCL agent=”BRobot\_PRobot” name=”PackagingTime” type=”double” value=”3” />

</DCLS>

<ExternalEvents>

<ExternalEvent agent=”BRobot\_PRobot” type=”BreakDown” xTime=”4” />

<ExternalEvent agent=”BRobot\_PRobot” type=”Fix” xTime=”15” />

</ExternalEvants>

</Parametrizations>

## TRACE FILE (.trace)

Aquest fitxer conté la sortida de l’execució del simulador SDLPS.

Es composa de dues parts diferenciades:

* La primera, encapçalada pel *tag* ***ModelInfo*** conté la informació del model així com el nom, la seva representació (.*eye*), tipus d’execució, parametrització, etc.
* La segona part està formada per una seqüència **d’Events**. D’aquests en tenim de dos classes:
  + Events de l’execució del model. (*input, output, setstate*)
  + Events de representació (Eye\_AnimTo, Eye\_Remove, Eye\_SetState, Eye\_Report)



## Eye main options

### Càmera: Moviments

* + A i ←: Desplaçament a l’esquerra.
  + S i ↓: Desplaçament cap enrere.
  + D i →: Desplaçament cap a la dreta.
  + W i ↑: Desplaçament cap a endavant.
  + Q: Desplaçament cap amunt.
  + E: Desplaçament cap avall.
  + Botó esquerre del *mouse*: Rotació

### Càmera: Opcions i configuració

* + + : Augmenta la velocitat del desplaçament.
  + - : redueix la velocitat de desplaçament.