Títol: **Editor de SDL amb VISIO**

Volum: **1**

Alumne: **Bernat Riera Comajoan**

Director/Ponent: **Pau Fonseca Casas**

Departament: **EIO**

Data: **18 – 6 – 2009**

GESTIÓ ACADÈMICA-FIB
ENTRADA
19 JUN. 2009

Índex

ÍNDEX 5

INTRODUCCIÓ 6

Presentació *6*
Motivació *6*
Objectius *7*

TECNOLOGIES EMPRADES 9

XML *9*
Microsoft Office *11*
Microsoft Visio® *12*
UML *13*
VB for Applications *15*
Llenguatge SDL *16*

PLANIFICACIÓ 21

ANÀLISI 25

Requisits Funcionals *25*
Requisits No Funcionals *26*
Llenguatge de Programació *27*

ESPECIFICACIÓ 28

Funció d'Exportació d'un Diagrama a XML-SDL *28*
Funció d'Importació d'un Diagrama a partir d'XML-SDL *29*

DISSENY I IMPLEMENTACIÓ 30

Entorn Visio *30*
Shape *31*
MSXML2.DOMDocument *32*
Recursió *33*
Funció d'Exportació d'un Diagrama a XML-SDL *34*
Funció d'Importació d'un Diagrama a partir d'XML-SDL *46*
Limitacions *53*

CONCLUSIONS 55

TREBALLS FUTURS 56

Tasques Pendents *56*
Perfeccionament *58*
Adaptabilitat a Altres Llenguatges *58*

BIBLIOGRAFIA 60

ÍNDEX D'IMATGES 62

Introducció

Presentació

El projecte que es desenvolupa en les següents pàgines tracta sobre la creació d'una aplicació d'importació i exportació a SDL (basat en XML) des de Microsoft Visio®.

Aquesta serà una eina molt útil per a tots aquells estudiants i creadors que treballen actualment amb el simulador SDLPS[1], ja que permetrà estalviar un volum important de feina alhora d'entrar les dades al simulador.

Aquesta idea ha sorgit de la proposta impulsada per en Pau Fonseca, director d'aquest projecte de fi de carrera i professor del departament d'Estadística i Investigació Operativa.

Motivació

En el simulador SDLPS, la definició dels models de simulació es fa a partir del llenguatge SDL. Aquest és representat emprant XML, descrit posteriorment a l'apartat de tecnologies emprades.

Fins ara, la redacció del fitxer XML que representa la formalització del model emprant SDL s'havia de fer a mà. Amb l'eina que es desenvolupa en aquest projecte, la redacció es pot realitzar de forma automàtica mitjançant un diagrama fet en Microsoft Visio®.

1. Vegeu punt 9 de la bibliografia.

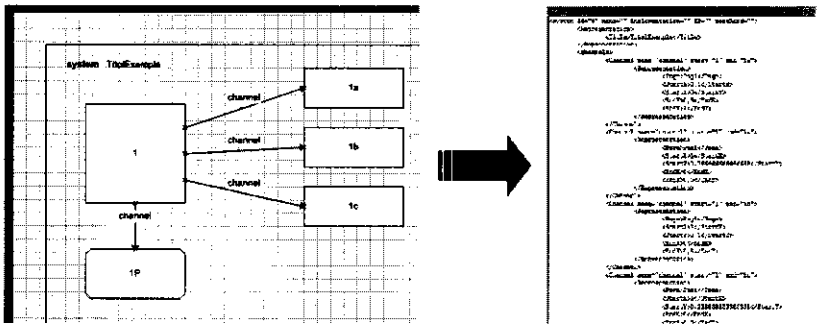
Això permet, apart d'estalviar feina, poder dissenyar un sistema tenint present sempre la perspectiva general des d'una visió il·lustrativa, que sempre és més fàcil d'analitzar que en forma de text, de codi.

El mateix procés el podem realitzar a la inversa, és a dir, recuperar un arxiu de codi i obtenir-ne la representació gràfica amb el Microsoft Visio®, facilitant així la comprovació d'un fitxer, o la ampliació, modificació d'aquest, etc.

Un motiu secundari de l'elecció d'aquest projecte és l'experiència prèvia que tenia amb el llenguatge *VB for Applications*, especialment en la realització de macros sobre programes de Microsoft Office, que m'ha permès aprofundir més en el projecte, podent-li donar alguna funcionalitat més que segurament si hagués començat de zero no hauria pogut assolir.

Objectius

Com explicàvem a l'apartat anterior, l'objectiu principal del projecte consisteix en desenvolupar una aplicació que ens permeti generar el codi XML que representi diagrames SDL partint d'un diagrama Microsoft Visio®, evitant així la redacció manual d'aquest.



1. Esquema de l'Objectiu

Aquest objectiu es pot desglossar en dues principals fites, com són primer la part funcional, i posteriorment la part gràfica.

Quan parlem de part funcional, ens referim a la capacitat d'obtenir un fitxer de codi a partir d'un diagrama que representi amb la màxima exactitud el significat, el valor teòric, de cada element i de cada connexió del diagrama, sense tenir en compte la seva posició en el dibuix ni cap altre tema relacionat amb la representació que no sigui determinant per la funció que desenvoluparà cada element dins el simulador.
La part gràfica, evidentment, és tota aquella informació que recollirem per tal de poder representar el mateix diagrama exacte en quan a representació, fet que ja inclourà la part funcional. Emmagatzemarem informació com la posició i les dimensions de cada objecte, la pàgina on es troben, els punts per on realitzem les connexions, els títols de pàgina, etc.

Una segona part, tot i que va molt lligada a la primera, és la funció inversa a l'exportació, és a dir l'importació a XML. Aquesta ens permetrà obtenir el diagrama a partir del codi.
Aquesta funció haurà de permetre representar amb exactitud un fitxer de codi creat a partir de la funció d'exportació pròpia, però també haurà de ser capaç de generar un diagrama a partir d'un fitxer de codi SDL qualsevol, tenint en compte que no té perquè contenir informació sobre la representació. Evidentment aquesta funcionalitat tindrà una sèrie de limitacions que explicarem més endavant.

Tecnologies Emprades

XML[2]

XML són les sigles de *Extensible Markup Lenguage*. Va ser desenvolupat pel World Wide Web Consortium (W3C) a partir del llenguatge GML inventat per IBM, que al 1986 seria normalitzat, donant pas al SGML.

XML no esdevé un llenguatge en sí, sinó una base per a la creació d'altres llenguatges.

Un dels punts forts de l'XML és l'extensibilitat, en el sentit de que no presenta cap problemàtica al afegir informació i etiquetes noves un cop creat el document. Un altre fet a destacar és l'estructura senzilla que permet processar i entendre fàcilment un document.

Un document XML es basa en informació nidada, delimitada per etiquetes que presenten el següent format:

<AAA> per obrir una etiqueta (On AAA és el nom que li donem a l'etiqueta)

</AAA> per tancar l'etiqueta

Dintre cada etiqueta podem obrir noves etiquetes, formant així una estructura d'arbre.

Una etiqueta pot contenir atributs, que ens permetran incorporar propietats de la següent manera:

<AAA atribut1=valor1 atribut2=valor2 … > … </AAA>

[2] Més informació a la bibliografia. Punt 1

Tot document XML ha de tenir una capçalera, on s'indicarà la versió XML que s'està utilitzant, i un sol element arrel.

```
<?xml version="1.0" encoding="utf-8"?>
```

Un factor important és la validesa d'un document XML.

XML defineix una estructura sintàctica bàsica que ha de complir qualsevol document XML, cuidant temes com que cada etiqueta tingui la corresponent homòloga de tancament, que els atributs estiguin ben definits, etc. Però no té (no pot tenir) un control sobre la validesa, ja que aquesta dependrà de l'entorn on s'utilitzi.

És per això que existeixen els documents DTD (Document Type Definition), que seran els encarregats de definir temes com per exemple quins atributs pot o no pot tenir cada node, les possibles limitacions alhora de combinar-los, quines entitats estan o no estan permeses, etc. Quan un document s'ajusta a un determinat DTD, direm que és un document vàlid.

Un element semblant són els Schemas, que tenen una funció similar als DTD però amb alguna diferencia.

Per últim anomenar també els arxius XSL, que defineixen l'estil, el format amb que serà mostrat el document XML.

```
<?xml version="1.0"?>
    <SiteVisits>
      <Country CountryName="USA">
        <TotalVisits>1346</TotalVisits>
        <LatestVisit>1/1/2000</LatestVisit>
      </Country>
      <Country CountryName="Zambia">
        <TotalVisits>42</TotalVisits>
        <LatestVisit>13/12/2008</LatestVisit>
      </Country>
    </SiteVisits>
```

2. Fragment de Codi XML

Microsoft Office[3]

Microsoft Office és un conjunt de software compost bàsicament per processadors de texts, fulls de càlcul, gestor de correu i programa per crear presentacions.

Funciona sobre plataformes Microsoft Windows i Apple Mac OS.

Esmentant algun apunt històric, Office va aparèixer el 1989, i el principal argument de venta era que unificava un seguit de programes que comprant-los amb el pack sortien més econòmics. Amb el pas dels anys les aplicacions han anat compartint funcionalitats com el corrector ortogràfic, integrador de dades OLE i el llenguatge de scripts Visual Basic per Aplicacions, que és el que prendrà més rellevància en aquest projecte.

Els principals programes que recull l'Office són els següents:

- Word: Processador de text. La primera versió va ser alliberada al 1983 i tenia com a novetat l'ús del ratolí en els processadors de text.
- Excel: Full de càlcul. Va aparèixer com a competidor del Lotus 1-2-3.
- Outlook: Administrador de informació personal i client de correu electrònic.
- Power Point: Programa per desenvolupar presentacions audiovisuals.

3. Logo Microsoft Office

[3] Més informació a la bibliografia. Punt 2

Microsoft Visio®[4]

Microsoft Visio® és un programa de dibuix vectorial que treballa sobre Windows.

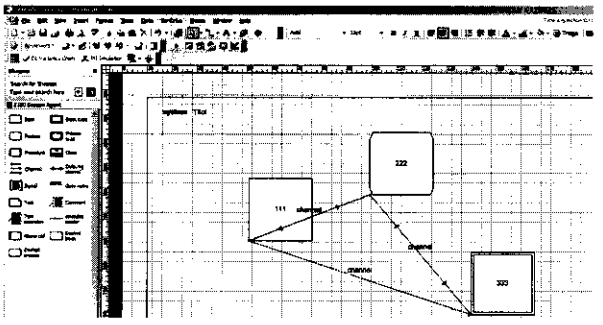
Visio va entrar a formar part de Microsoft l'any 2000, quan aquest últim va absorbir la companyia Visio.

Aquest programa permet la creació de una gran varietat de diagrames, com per exemple, diagrames de bases de dades, de flux de programes, UML, arquitectònics, etc...

Destacar també que va ser un software que apuntava a ser una aplicació de dibuix tècnic per el camp de la enginyeria i arquitectura, però al passar a Microsoft, va seguir evolucionant cap a una branca més adequada al camp dels negocis.

Microsoft Visio®, a partir de la versió 2003, automatitza la visualització de les dades al sincronitzar-se directament amb els orígens de dades per proporcionar diagrames actualitzats.

Com tots els programes pertanyents al Microsoft Office, Microsoft Visio® conté un editor de macros que és el que hem utilitzat per crear les funcions de exportació i importació XML, essència d'aquest projecte de fi de carrera.



4. Microsoft Visio®

[4] Més informació a la bibliografia. Punt 3

UML[5]

Unified Modeling Language, és el llenguatge per a modelat de sistemes de software més conegut i utilitzat actualment. Conta amb el recolzament del OMG. És un llenguatge gràfic per a visualitzar, especificar, construir i documentar un sistema.

Cal destacar que UML és un llenguatge de modelat per especificar o descriure mètodes i processos. S'utilitza per definir un sistema, per detallar i construir.

Es pot aplicar al desenvolupament de software entregant gran varietat de formes per donar suport a una metodologia de desenvolupament de software però no especifica quina metodologia o procés s'ha d'utilitzar.

UML no és comparable amb la programació estructurada sinó que tan sols es diagrama la realitat d'una utilització en un requeriment, mentre que la programació estructurada és una forma de programar com podria ser la orientació a objectes.

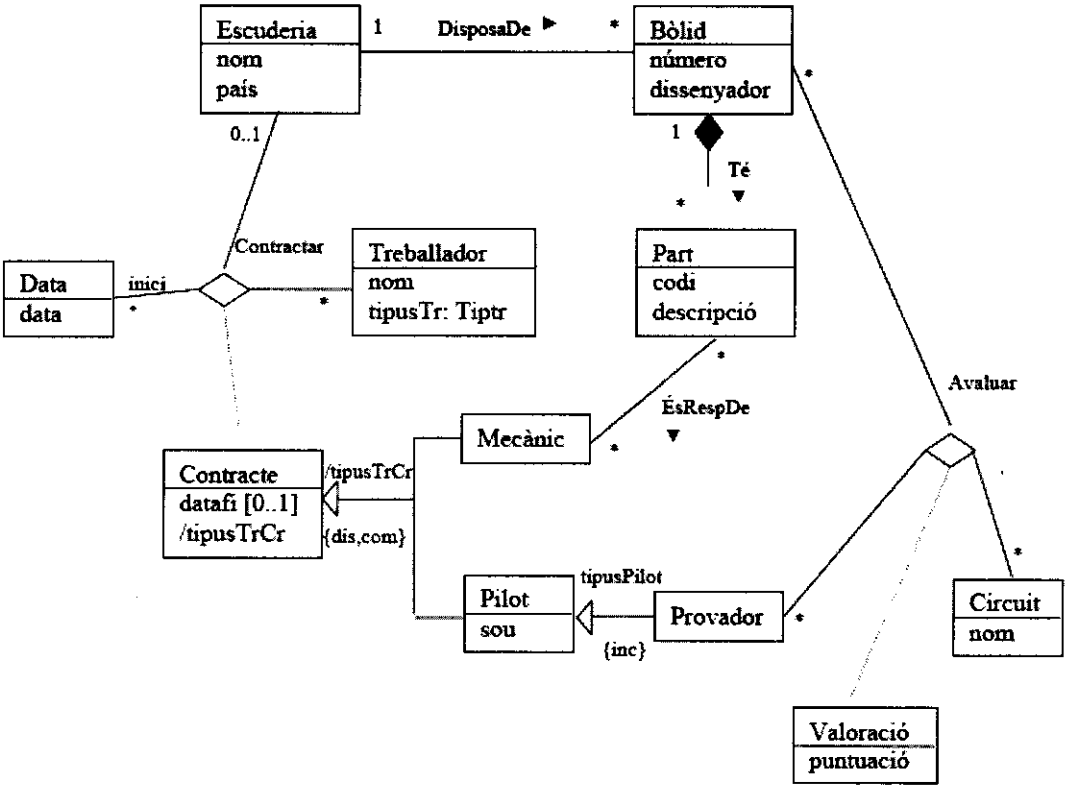
UML conta amb tres grups de diagrames diferents, que són:

- Diagrames d'Estructura
- Diagrames de Comportament
- Diagrames d'Interacció

Una de les crítiques clàssiques del llenguatge UML és que la interpretació d'un model no pot ser sempre objectiva. També presenta dificultats alhora de representar sistemes

[5] Més informació a la bibliografia. Punt 4

distribuïts (UML no permet representar factors com transmissió, serialització, persistència, etc.). No es pot, per exemple, utilitzar UML per a senyalar que un objecte és persistent o remot, o que existeix en un servidor que corre contínuament i que és compartit en varies instancies d'execució del propi sistema analitzat. No obstant, UML si permet la creació de components propis per aquest tipus de modelat.



5. Diagrama UML

VB for Applications[6]

Microsoft VbfA (Visual Basic for Applications) és el llenguatge de macros de Microsoft Visual Basic que s'utilitza per programar aplicacions Windows i que s'inclou en varies aplicacions Microsoft. VBA permet a usuaris i programadors ampliar la funcionalitat de programes de Microsoft Office com és el cas d'aquest projecte de fi de carrera. Visual Basic per a Aplicacions és un subconjunt bastant complet de Visual Basic 5.0 i 6.0

Microsoft VBA ve integrat en aplicacions de Microsoft Office com Word, Excel, Access, Power Point i Visio. Pràcticament qualsevol cosa que sigui programable en Visual Basic 5.0 o 6.0 es pot programar també dins un document d'Office, amb la limitació que el producte final no es pot compilar separadament del document (full de càlcul o base de dades) on s'ha creat; és a dir, es converteix en una macro.

La seva utilitat principal és automatitzar tasques quotidianes, així com crear aplicacions i serveis de bases de dades. Permet accedir a les funcionalitats d'un llenguatge orientat a events amb accés a la API de Windows.

```
Sub CarregarXML()

        Dim oDoc As MSXML2.DOMDocument
        Dim fSuccess As Boolean
        Dim oRoot As MSXML2.IXMLDOMNode
        Dim oCountry As MSXML2.IXMLDOMNode
        Dim oAttributes As MSXML2.IXMLDOMNamedNodeMap
        Dim oCountryName As MSXML2.IXMLDOMNode
        Dim oChildren As MSXML2.IXMLDOMNodeList
        Dim oChild As MSXML2.IXMLDOMNode
        Dim intI As Integer

        Set oDoc = New MSXML2.DOMDocument
        ' Load the  XML from disk, without validating it.
        ' for the load to finish before proceeding.
        oDoc.async = False
        oDoc.validateOnParse = False
```

6. Fragment de Codi VBfA

[6] Més informació a la bibliografia. Punt 5

Llenguatge SDL (Specification and Description Language)[7]

Els formalismes prenen protagonisme quan ens trobem limitats amb el llenguatge textual, que pot provocar ambigüitats i induir a confusions amb facilitat. Per altre banda no és un sistema de comunicació només apte per tècnics, fet que el fa útil per integrar les comunicacions entre la gent d'un equip que pugui tenir un perfil no tècnic.

L'especificació SDL està pensada per especificar el comportament de models complexes, que funcionen amb esdeveniments a temps real. Els models SDL, es composen per diferents diagrames, aquests poden estar repartits per diferents pàgines, cada pàgina ha de tenir a dalt a la part superior dreta un títol indicatiu de què és el que estem veient.

L'SDL té conceptes per a descriure comportaments i dades i estructurar grans sistemes. La descripció del comportament es basa en màquines d'estats finits ampliats que comuniquen mitjançant missatges. L'estructuració es basa en la descomposició jeràrquica i jerarquies de tipus.

Un sistema SDL consisteix en un o més agents comunicants, que alhora mantenen una relació amb l'entorn, ja sigui formant part del sistema central, o d'un *agent*, *process*, etc. En un agent hi podem trobar definicions de comportament (per màquines d'estat finites), estructures jerarquitzades, variables i processos comunicatius basats en l'intercanvi de missatges asíncrons.

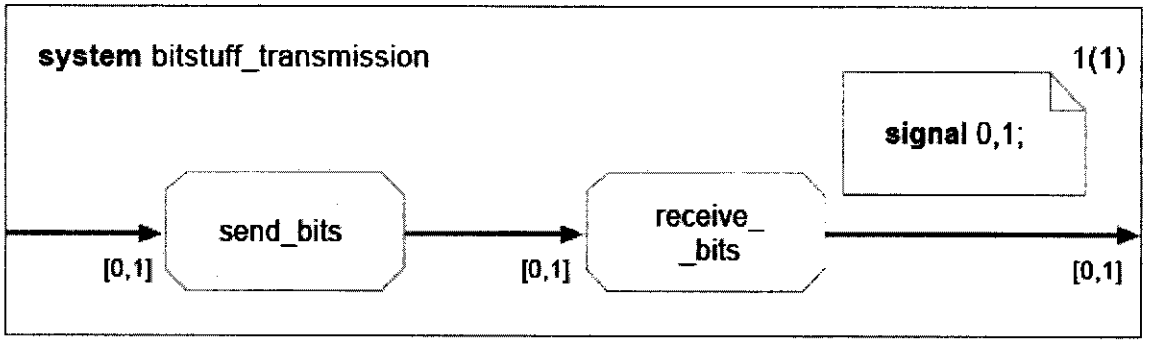
[7] Mes informació a la bibliografia. Punts 6 i següents.

El punt de partida d'un disseny d'un sistema sol ser un conjunt d'elements units per connectors, que representen la comunicació entre aquests i l'entorn. Normalment aquests elements venen representats per quadres i les connexions per línies que tenen un quadre origen i un final.

El nom que donem a aquests quadres serà el nom que definirà els respectius components, i el mateix passarà amb els canals o connectors. Aquests descriptius són abstraccions de sistemes reals.

Vegem-ne un exemple gràfic, extret també de l'article d'en Rick Reed que trobareu a la bibliografia, on es mostra simplement un diagrama de transmissió de bits.

Tenim dos processos. El primer "send_bits" rep el senyal del sistema pare (indicat amb la fletxa que surt del marc i connecta amb el procés) i llavors envia un altre senyal al segon procés "receive_bits" que alhora tanca el cicle tornant a enviar el senyal al sistema general.



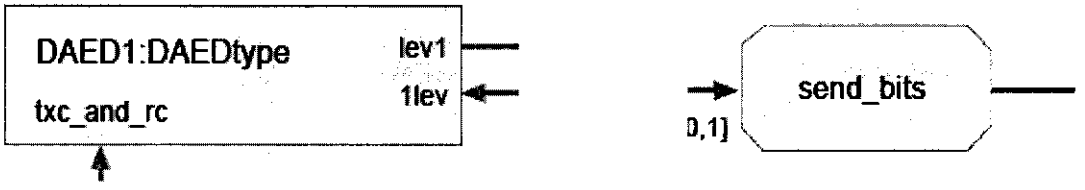
7. Transmissió de bits

Dèiem doncs, que una especificació SDL és un conjunt de diagrames. Cada diagrama pot tenir una o més pàgines, on cada una hi trobarem els següents elements:

- *Frame*: Vindria a ser el marc, que representa l'agent pare, l'agent que estem definint en aquesta pàgina.

- Capçalera: Descriu el tipus d'element que estem definint. Es troba a la cantonada superior esquerra.
- Número de Pàgina.
- Títol.

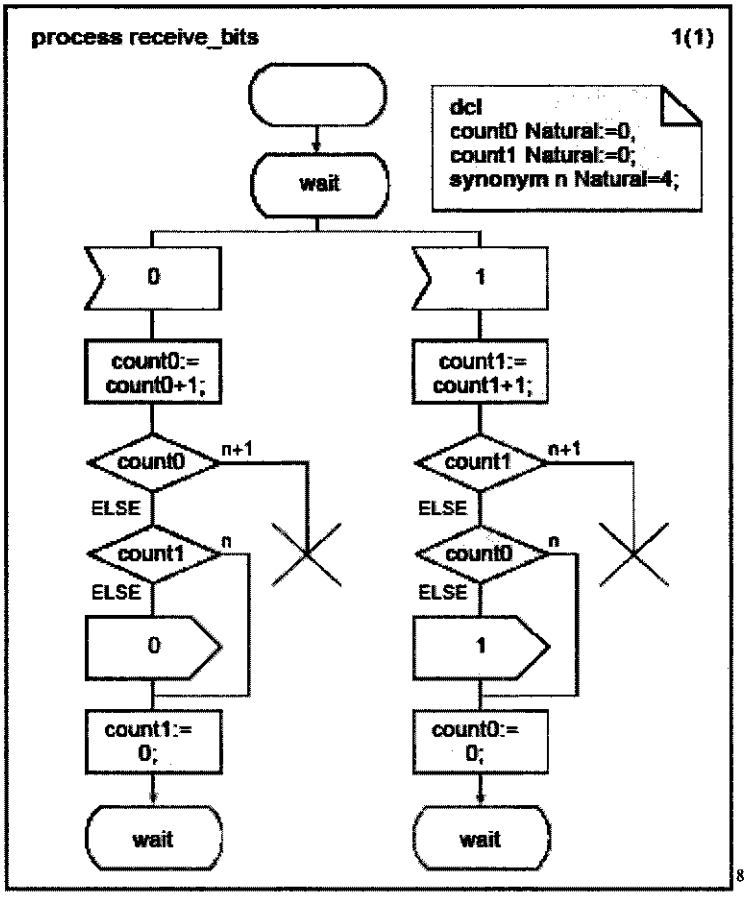
Principalment tractem amb dos tipus d'elements, que són els *Process Agents* (de símbol un rectangle amb les puntes retallades) i els *Block Agents* (de símbol un rectangle normal).



8. Block i Process respectivament

La diferencia essencial entre un *Block* (o sistema) i un *Process* és que les instancies d'un agent *Block* es comporten simultàniament i asíncronament entre sí, mentre que les instancies d'un *process* succeeixen una a una en el temps. Un *Block* pot contenir tant altres *Blocks* com *Process*, però els agents *Process* només poden contenir altres agents *Process*.

De la mateixa manera que un agent en pot contenir d'altres, també pot contenir màquines d'estat, variables i procediments. En podreu veure un exemple en el següent diagrama d'exemple.



[8]

9. Diagrama SDL

Situats ja amb el llenguatge SDL, anem a comentar una de les possibilitats de representar-lo, que és la que s'ha utilitzat per dur a terme aquest projecte.

L'eina és evidentment el llenguatge XML. És un llenguatge que encaixa molt bé amb la naturalesa d'aquest formalisme, ja que ens permet representar el fet aquest de les figures, on cada figura té les seves propietats i implementacions, i alhora ens permet definir amb exactitud les connexions entre elles.

[8] "The receive bits process as an extended finite state machine" – SDL-2000 for New Millennium Systems, Rick Reed.

Tot sistema s'inicia amb el bloc "*System*" que serà l'arrel. Dins el seu àmbit i trobarem totes les figures de que es compon.

Primerament s'informen els canals, els connectors que apareixen en el diagrama. Cada canal contindrà el nom de la *shape* inici i la *shape* final on es troba connectat. No és possible que un connector estigui desconnectat d'un o dels dos caps. Cada canal informarà també dels events que té associats (que es representen mitjançant els *Signals*).

Un cop informats els canals, trobarem cada una de les figures de dues dimensions (per diferenciar-les dels canals que tenen una sola dimensió).

Cada figura guarda la mateixa estructura que l'arrel *System* que comentàvem. En cas que contingui un altre fitxer d'implementació, contindrà la llista de canals que conté, seguida de les *shapes*, etc. Si no posseeix cap fitxer d'implementació sols presentarà la informació referent a identificador i nom, i depenent de l'origen (si el codi ve generat per la funció d'exportació o si prové d'un altre tipus de font), informació de representació. Aquesta informació de representació ens vindrà donada d'acord amb les pautes d'estructuració i condicions pròpies del sistema que hem creat per tal de poder-les recuperar a la funció d'importació.

10. Representació XML

Planificació

La planificació ens permet estructurar les tasques a realitzar en un calendari tenint en compte tots els factors que a priori ens poden condicionar la realització d'aquestes.

Un factor principal són les tasques que han d'estar completades per a poder iniciar-ne una següent. Un altre factor són els recursos que necessita cada tasca i que ens pot condicionar el fet de treballar simultaniament en dues o més tasques alhora.

La planificació ens permet controlar a cada moment si el ritme de treball que portem ens portarà a acabar la feina dins les previsions que havíem fet o en cas contrari ens informarà de la desviació que portem, del retràs acumulat que poguem portar.

Al iniciar aquest PFC es va redactar un calendari amb les tasques amb les quals creia que es podia dividir.

Es va valorar la opció d'utilitzar una aplicació de l'estil del Microsoft Project per dur a terme la planificació, però personalment crec que no aportaria cap funcionalitat ni avantatge donada la naturalesa d'aquest projecte. Un programa d'aquest estil aporta una utilitat immensa quan tractem amb tasques que depenen d'altres tasques per ser iniciades, o tasques que requereixen recursos finits (ja siguin materials o personals), llavors és quan la ment humana queda limitada alhora de preveure i analitzar desviacions. En aquest cas, al afrontar el projecte una sola persona, no tenir tasques que necessitin recursos limitats, i no tenir tampoc prioritats especials entre tasques ja que una tasca es comença quan s'ha acabat la immediatament anterior, se n'ha desestimat l'ús.

La relació de tasques és la següent, juntament amb la data en què han d'estar finalitzades:

- Creació d'una funció que guardi un diagrama Microsoft Visio® en un fitxer XML (15 de gener)

  En aquest primer pas realitzarem una macro que ens permeti consultar informació de cada element del diagrama. Seguidament guardarem aquesta informació en un document de text, seguint les pautes del llenguatge XML

- Adequar el format XML del fitxer resultant, al formalisme SDL (20 de gener)

  Les etiquetes prendran l'estil del formalisme SDL, resituarem cada dada obtinguda del diagrama i la guardarem en el seu lloc corresponent, algunes dades s'informaran com atributs, d'altres com a subnodes, etc.

- Creació d'una funció que llegeixi els nodes del document XML-SDL resultant (26 de gener)

  Aquí començarem a treballar en la funció inversa, la qual anirem treballant paral·lelament ja que una funció ens ajudarà a detectar errors de l'altre i al revés.

- A partir de la informació obtinguda de l'XML, dibuixar el diagrama (10 de febrer)

  El fet de treballar paral·lelament amb la funció de importació i la d'exportació fa que determinades problemàtiques on quedes encallat per una banda, puguis trobar-li una sortida abordant-la per la funció inversa.

- Polir detalls de les dues funcions (1 de març)

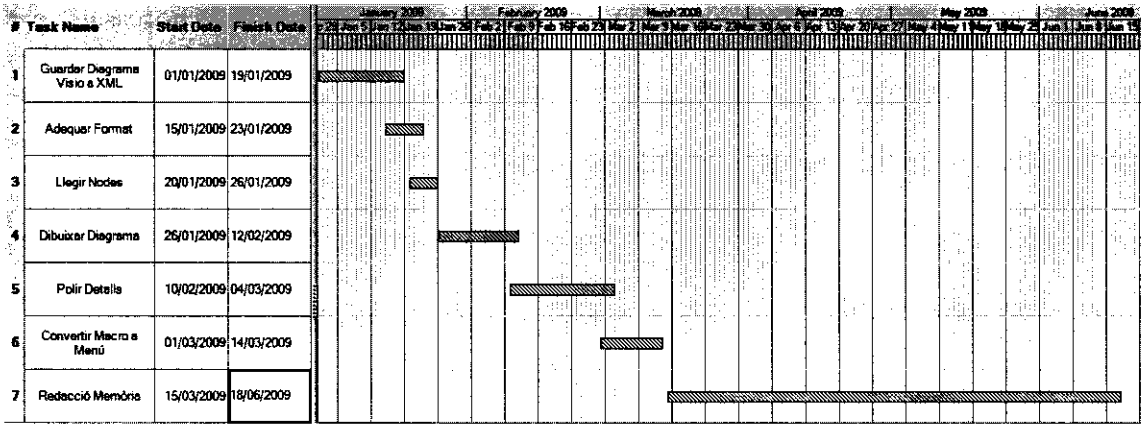
  Seguim treballant amb les dues funcions, comprovant que funcioni correctament amb totes les *shapes* possibles, tractant casos extrems, etc.

- Convertir la macro en una opció del menú Microsoft Visio® (15 de març)

  No és més que convertir les dues macros en dues opcions del menú del Sandrila o creant un menú just al costat.

- Redacció de la memòria (18 de juny)

Un cop acabada la part de programació encararem la part de redacció, ajudant-nos de les notes que hem anat prenent al llarg de tot el procés de creació.



#	Task Name	Start Date	Finish Date
1	Guardar Diagrama Visio a XML	01/01/2009	19/01/2009
2	Adequar Format	15/01/2009	23/01/2009
3	Llegir Nodes	20/01/2009	26/01/2009
4	Dibuixar Diagrama	26/01/2009	12/02/2009
5	Polir Detalls	10/02/2009	04/03/2009
6	Convertir Macro a Menú	01/03/2009	14/03/2009
7	Redacció Memòria	15/03/2009	18/06/2009

11. Diagrama de Gantt

Excepcionalment, ja que ho he pogut comprovar en la meva experiència laboral en empreses del sector informàtic, hem quadrat quasi tots els terminis que ens havíem proposat. Només resta pendent en el moment d'entregar aquesta memòria, la finalització de la penúltima tasca, que consistia en convertir la macro en una opció del menú.

Diria que aquest èxit a la planificació es deu en part als motius que expressava anteriorment en la valoració d'utilitzar un planificador de tasques.

Un altre fet a comentar es que han sorgit menys imprevistos dels que esperava, i cada un d'aquests s'ha pogut solventar dins el plaç.

A diferència d'un projecte realitzat en una empresa (o almenys hauria de ser així...) és que la dedicació prevista, ja sigui mitja jornada o una jornada completa, és molt més flexible, posem-hi un exemple: En el cas que venci el plaç i es prevegui que amb la dedicació prevista no s'hi arriba, es pot augmentar el nombre d'hores dedicades sense massa inconvenients.

Fent referència als costos.

Podríem considerar dos perfils diferenciats per dur a terme el càlcul dels costos totals del projecte.

Per una banda tindríem la figura del Cap de Projecte o director, que seria el responsable de guiar i controlar la realització de les tasques, proporcionar els recursos necessaris per solventar cada entrebanc que aparegui i mantenir un contacte periòdic amb l'analista-programador que realitzi la major part del projecte.

Aquest segon perfil d'analista-programador seria l'encarregat de realitzar la part més física del projecte. De dissenyar, especificar i implementar cada part del projecte.

Un preu més o menys usual podria ser d'uns 40€/hora l'analista-programador i d'uns 70€/hora el cap de projecte. Sempre aniria més en funció del tipus d'empresa d'on provinguin, exigències varies del projecte, etc.

Referent al número d'hores, podríem considerar unes 80 mitja jornades per part de l'analista-programador, i uns 3 jornades completes del cap de projecte.

80 jornades x 4 hores x 40 €/h = 12.800 € de l'analista-programador.

3 jornades x 8 hores x 70 €/h = 1.680 € del cap de projecte.

12.800 + 1.680 = 14.480 € d'hores de feina.

Anàlisi

En aquest apartat definirem els requisits del projecte. És una part important, molt lligada als objectius, on definirem què ha de complir el projecte, i de quina manera, per a ser vàlid, per satisfer les necessitats o resoldre la problemàtica que ens plantejàvem inicialment.

Requisits Funcionals

Actor del sistema: Usuari

- Importar un diagrama d'un XML: L'usuari ha de poder seleccionar un document XML-SDL i ha de poder visualitzar el diagrama en Microsoft Visio®. Aquesta funcionalitat pot tenir dos resultats en funció de la informació que contingui el diagrama
  a) Si el diagrama conté informació de representació, aquest quedarà representat exactament de com hagi estat guardat.
  b) Si el diagrama no conté informació de representació, Microsoft Visio® mostrarà tots els elements del diagrama però no garanteix una distribució acurada d'aquests sobre la pantalla.

- Exportar un diagrama a XML: L'usuari ha de poder guardar un diagrama en un document XML-SDL on s'hi guardarà, a més de la part funcional, totes les dades referents a representació necessàries.

Requisits No Funcionals

- Comprensió

  El primer que s'ha de tenir present és que no ens trobem davant d'una aplicació finalitzada ni molt menys, sinó que és un primer pas que compleix amb unes funcionalitats inicials, però que serà modificat i ampliat per altres estudiants i companys de professió. Detallarem aquestes ampliacions al capítol sobre treballs futurs. Anem ara a explicar com es tradueix aquest fet a l'hora de programar.

  S'ha donat prioritat a l'estructuració del codi en detreniment de l'eficiència per tal d'afavorir que un programador que hi hagi de treballar posteriorment li sigui molt més fàcil situar cada funcionalitat afavorint així la legibilitat.

  L'eficiència en aquest cas no és un requisit prioritari ja que no realitzem cap càlcul que saturi el sistema, tenim un temps de resposta "immediat" i a més estem parlant d'una funcionalitat que té una freqüència d'activació, d'ús, molt baixa, en el sentit de què l'usuari no utilitzarà aquesta funció moltes vegades en una sessió de treball.

- Depurabilitat

  Un requisit molt lligat a la comprensió de la que parlàvem en les línies anteriors. De la mateixa manera que afavorim la comprensió lectora del codi evitant combinar diferents funcionalitats en un mateix fragment de codi, en un mateix bucle per exemple, estem afavorint el procés de depuració. És a dir, qualsevol punt del codi que no treballi com desitgem és molt més detectable d'aquesta manera, afavorint també els treballs posteriors sobre aquest projecte.

- Modificabilitat

  Potser és una tercera manera d'explicar el mateix. Tant com millorem la comprensió i la depurabilitat, facilitem la modificació o substitució d'un fragment de codi alterant el mínim possible l'entorn.

  Per aquesta mateixa raó hi ha algun fragment de codi, algun càlcul o alguna variable que en aquest instant no s'està utilitzant però que és molt probable que en properes ampliacions siguin necessàries. Es per això que podem trobar fragments d'aquest tipus que facilitaran i milloraran molt l'adaptació de noves funcionalitats a l'aplicació.

Llenguatge de Programació

En aquest cas no teníem altre possibilitat en l'elecció del llenguatge de programació, ja que al haver de treballar sobre la plataforma d'edició de macros de Microsoft Office, havíem d'utilitzar VB *for applications*.

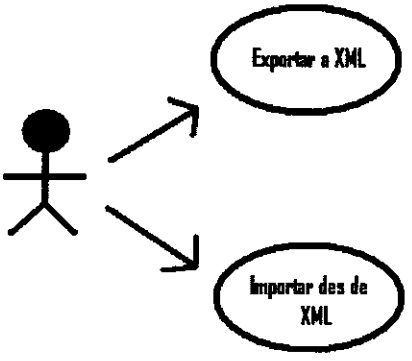
VBA és un llenguatge del que podem trobar molta informació a la xarxa. A part del seu ús extens, disposem d'una ajuda bastant útil que ens ofereix el mateix Microsoft Visio®.

12. Botó d'Ajuda

L'ajuda està enfocada a consultar el funcionament de cada funció, però no hi ha un sistema de presentar una problemàtica i que l'ajuda et suggereixi una funció. És vital conèixer la nomenclatura que utilitza Microsoft Visio® per designar cada element que participa en un diagrama.

Especificació

En aquest tipus de projecte, els casos d'ús no presenten la típica estructura on l'usuari realitza una acció i el sistema una resposta, ja que l'usuari només pot realitzar dues accions, que són importar o exportar.



13. Diagrama de Casos d'Ús

Degut això, l'especificació pren el format més d'anàlisi de les dues funcions, on detallarem tots els requisits que han de complir per a ser vàlides.

A diferència del capítol d'anàlisi, aquí ja entrarem amb característiques tècniques, pròpies dels diagrames Microsoft Visio® o del llenguatge SDL.

Funció d'Exportació d'un Diagrama a XML-SDL

El primer fet a tenir en compte és que per "redactar" aquest XML podem necessitar més d'un diagrama. Això es deu a la naturalesa del SDL, on un *block* o un procés d'un diagrama es poden explotar, definir, expandir,etc. en un altre diagrama que portarà per nom el mateix que el de la figura explotada.

Aquesta particularitat fa que quan exportem un document haguem de controlar cada figura si té o no té implementació en un altre arxiu Microsoft Visio®.

Si en té, que ho esbrinarem consultant si en el mateix directori del document arrel hi existeix un document amb el mateix nom de la figura concatenat amb l'extensió .vsd, passarem a analitzar aquest darrer, i en acabar, enganxarem el codi resultant en el lloc corresponent del XML general que estàvem treballant.

En el cas de l'exportació, no es rellevant analitzar primer els connectors o primer les *shapes*. Veurem en el cas de la importació que si ho és.

Funció d'Importació d'un Diagrama a partir d'XML-SDL

El fet més destacat d'aquesta funció és que del mateix document XML-SDL se'n poden despendre més d'un diagrama, ja que és exactament el procés invers a l'exportació. Cada *shape* que contingui informació sobre com està implementada, generarà un arxiu Microsoft Visio® apart que contindrà dita informació.

Comentat l'aspecte referent als diferents diagrames, i abans de posar-nos a implementar, hem de definir una línia d'actuació per tal que en resulti una funció ordenada i coherent. Tan bon punt ens hi posem, veiem que si anem per enganxar un connector, i no tenim ja en el diagrama les *shapes* origen i final d'aquests, no el podrem connectar. D'aquí se'n desprèn que primer haurem de situar totes les figures 2D i seguidament els connectors.

Disseny i Implementació

Entorn Visio

El primer pas a l'hora d'implementar és conèixer la notació que utilitza Microsoft Visio® per anomenar cada element, ja que per exemple, tenim diferents sistemes de connectar dues *shapes*, i cada tipus de connexió utilitza una nomenclatura diferent que hem de controlar.

Generalment utilitzarem els mètodes referents als següents objectes, que ens acompanyaran tota la implementació. Aquest són (per ordre alfabètic):

- Cell: Equivaldria a un valor d'una *shape*, per exemple l'amplada o la coordenada d'una connexió.
- Connect: Equival a una connexió dins el conjunt de connexions *Connects* que s'explica seguidament.
- Connects: Col·lecció de connexions. Comprèn totes les connexions d'una figura del diagrama.
- Document: És el que representa el diagrama. Representa cada finestra Microsoft Visio® que tinguem oberta.
- MSXML2.DOMDocument: L'utilitzarem per navegar dins el codi XML. S'explicarà més amb detall en les següents pàgines.
- Page: Representa cada pestanya d'un diagrama.
- *Shape*: Representa qualsevol figura que podem seleccionar en una diagrama, figures, grups de figures, guies, objectes importats o links, etc.

Principalment treballarem amb aquest tipus d'objectes tot i que puntualment trobarem altres referències a diferents altres objectes.

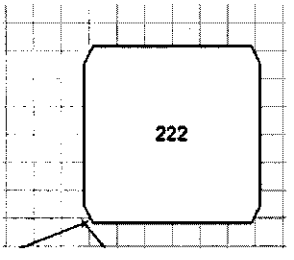
Shape

La *shape* és l'element principal d'un diagrama, i és convenient situar-lo un pèl més, abans d'entrar en el detall de les dues funcions.

Una *shape* conté un nom *.name* únic en tot el diagrama que la defineix. Si en tot el diagrama només existeix una sola *shape* d'aquell tipus, el nom coincidirà amb aquest. En el cas que n'hi hagi més d'una, les següents presentaran el següent format: tipus.X on X serà un enter cada cop diferent. Posant un exemple: Si tenim dos *blocks* en un diagrama, ens podem trobar els nom siguin *Block* i *Block*.9

Aquest *.name* l'utilitzarem per trobar-ne el tipus, però l'identificador que utilitzem nosaltres és el text *.text* que és l'string que podem informar nosaltres com a usuari, de cada *shape*.

Això és possible perquè l'SDL ens garanteix que mai dues *shapes* tindran el mateix text sinó representen la mateixa *shape*. En la següent captura de pantalla, la *shape* de tipus *process* que s'hi veu representada tindria el text "222".



14. Process 222

Cada *shape* es troba situada en un punt X,Y del diagrama. Aconseguirem aquesta informació mitjançant les cel·les (recordem que les cel·les eren propietats de les *shapes*) *pinx* i *piny* de la següent forma: *shpObj.Cells("pinx").*

L'amplada i alçada de cada una les trobarem amb les cel·les *width* i *height.*

Un punt bàsic són les connexions. Per referir-nos al conjunt de connexions de una *shape* utilitzarem l'objecte *Connects*, *shpObj.Connects*, a partir d'aquí podrem navegar per cada una i fer les consultes necessàries.

Per diferenciar les *shapes* d'una dimensió (connectors) de les de dues dimensions, utilitzarem la propietat *OneD* que és un booleà que serà cert si és 1D i fals si és 2D.

MSXML2.DOMDocument

En la funció de càrrega és imprescindible utilitzar una eina que ens permeti navegar per l'arbre XML i poder-ne adquirir la informació necessària per representar el diagrama.

Després de valorar diferents possibilitats, hem escollit aquesta que personalment crec que era la més clara i intuïtiva.

Mètodes que utilitzarem:

- DocumentElement: Definir l'arrel de l'arbre. És el punt de partida.
- NodeName: Ens retorna el nom del node, l'etiqueta.
- ChildNodes: Retorna la col·lecció de fills de un determinat node.
- NodeTypedValue: Retorna la informació que conté el node. El valor.
- Attributes.getNamedItem("X"): Retorna el valor de l'atribut X del node.

Un cop introduïts aquests mètodes principals, anem a comentar el disseny i la implementació de les dues funcions.

Recursió

Com que ens trobem davant d'un sistema de representació que permet anar baixant al detall de representació de cada figura utilitzant les mateixes eines, hem dissenyat les dues funcions utilitzant la recursivitat.

En el cas de l'exportació a XML, primer executem la funció pel diagrama central, i a mesura que anem trobant figures (*shapes*) que estan explotades en altres fitxers, seguim analitzant-les utilitzant la mateixa funció.

Com passa en la majoria de funcions recursives, tindrem una funció principal on inicialitzarem una sèrie de paràmetres, escriurem les primeres línies del document XML resultant, i llençarem la funció recursiva amb el paràmetre "System" que serà sempre el primer del document, tal com mana el formalisme SDL. Aquesta funció s'anirà cridant a si mateixa cada vegada que trobi una *shape* implementada.

Finalment, quan retorni el flux d'execució a la funció principal, tancarem el *tag System* i l'últim canal d'impressió que en aquell moment ens quedarà obert.

És vital comentar prèviament a l'anàlisi de les funcions pas a pas, la forma que utilitzarem per transmetre el codi d'una funció recursiva a la funció pare que és qui l'ha cridat.

Farem servir una jerarquia de fitxers de text. Cada funció emmagatzemarà el codi resultat en un fitxer de text, dins el directori que haurà seleccionat l'usuari. Finalitzada una recursió, el pare recollirà el codi del fitxer de text i el concatenarà amb el seu.

Això ens permetrà tenir, un cop finalitzat tot, un fitxer de text per cada figura analitzada.

En el cas d'una figura final (la que dins un context d'arbre equivaldria a una fulla) el text només contindrà les dades de representació, i en el cas de una figura diguem-n'hi intermitja (que no equivaldria a una fulla però segueix essent una part del diagrama

central) el text contindrà, apart de les dades de representació, les dades de tots les *shapes* que aquesta contingui.

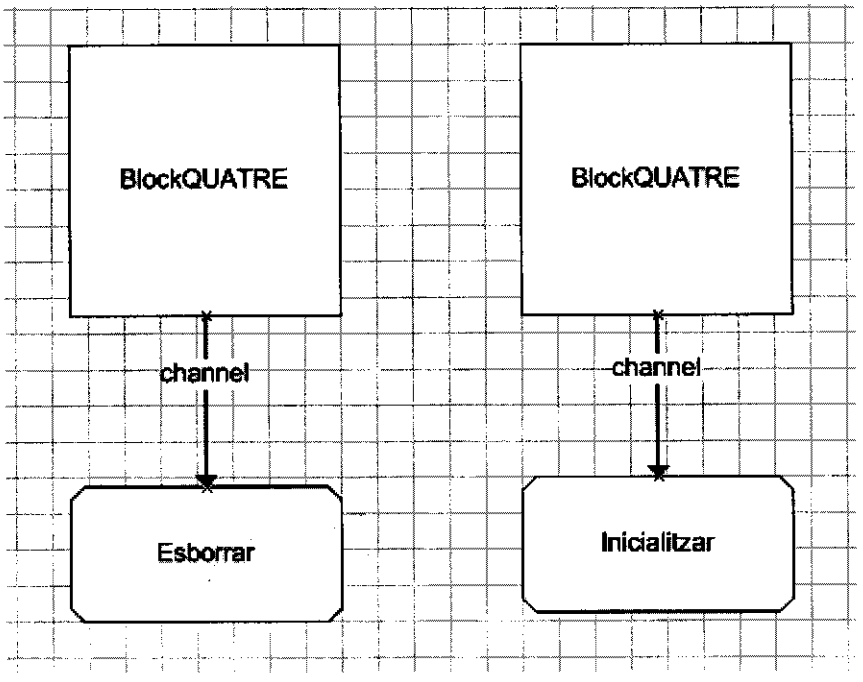
En el cas de la funció d'importació a Microsoft Visio®, també utilitzarem la recursivitat ja que necessitem una funció idèntica per tractar a un node que per tractar als nodes fills d'aquests.

Un element comú en les dues funcions és la pantalla "examinar" de Windows, que ens permetrà en un cas seleccionar el document XML a generar i en l'altre seleccionar el document Microsoft Visio® a importar.

Funció d'Exportació d'un Diagrama a XML-SDL

Ens trobem a la primera línia de codi de la funció d'exportació. És una funció principal, executada directament per l'usuari i no requereix cap paràmetre.

Inicialitzem un vector d'strings anomenat Formes que utilitzarem per guardar-hi el nom de cada *block* del diagrama. Es crea amb la intenció de tenir un control de les *shapes* repetides que puguin aparèixer en el diagrama, ja que segons el formalisme SDL, si trobem dues *shapes* amb el mateix nom, és que representen la mateixa *shape*, i per tant, només s'ha d'entrar al XML una sola vegada. Per il·lustrar-ho, en el següent diagrama vegem que tenim dos blocs iguals, per tant a l'arbre XML resultant haurem de tenir una sola instancia del BlockQUATRE que tindrà dos processos (Esborrar i Inicialitzar)



15. Shape Duplicada

Aquest vector Formes li donarem longitud 1, ja que en aquest moment desconeixem el nombre de figures que tindrem al diagrama. Cada cop que adherim una figura augmentarem en una posició aquest vector.

La següent variable que inicialitzem dins aquesta funció principal és un booleà. Aquest booleà ens indicarà si al retornar la funció recursiva hem d'afegir nou codi al XML existent. En aquest punt on es troba el projecte, aquest booleà serà sempre cert, ja que quan llancem la funció recursiva és perquè existeix un nou diagrama a exportar que haurà de ser afegit dins el codi XML existent. El booleà ha estat creat per una banda perquè per naturalesa del llenguatge VBA hem d'associar el retorn de la funció a una variable. Per l'altre, pot facilitar la feina als pròxim desenvolupadors alhora d'afegir un control d'errors o de validació del codi retornat.

Utilitzarem l'objecte *SAFRCFileDlg.FileSave* per tal d'obrir una finestra d'exploració on l'usuari pugui seleccionar el directori i anomenar el fitxer XML resultant de la funció. Se'ns permet proposar un nom i una ruta, hem suggerit "C:\XMLExport.xml".

També se'ns permet filtrar els documents a mostrar, i filtrarem per l'extensió .xml.

En funció de si l'usuari anomena un fitxer o bé cancel·la utilitzant el botó Cancel·lar, obtindrem cert o fals en el retorn. Si el resultat és fals abandonem la funció sense realitzar cap més pas.

Si cert, seguim endavant. Primerament ens guardarem el directori assenyalat en una variable que es diu Carpeta.

Sense més dilatacions creem el fitxer i li determinem el canal d'impressió, que serà el 2.

El primer que hem d'escriure al fitxer és la capçalera pròpia d'XML, que en aquest cas serà "<?xml version="1.0">"

Finalitzat això ja es quan llancem la funció recursiva. En aquesta primera recursió li passem els següents paràmetres:

- "System": És l'arrel d'on pren inici la funció. Posteriorment recuperarem el codi des del fitxer de text que portarà per nom aquest mateix .txt
- 4: És el canal d'impressió. Ens reservem el canal 3 per escriure el codi que retornarà
- Formes(): És el vector de *shapes* utilitzat per controlar els repetits
- Carpeta: És el directori de treball que ha seleccionat l'usuari

Il·lustrativament, només hem avançat una línia, però evidentment hem de tenir en compte que aquí es quan retorna la última funció recursiva i tenim tot el codi en el fitxer System.txt, és a dir que ja hem realitzat el 99% de la tasca per dir-ho d'alguna manera.

Imprimim pel canal 2 la línia inicial de *System* i obrim l'etiqueta *Representation.*

Copiem línia a línia tot el codi provinent del fitxer System.txt que es troba en el directori Carpeta.

Per acabar tanquem el *tag System*, tanquem el canal d'impressió, que era el canal 2, i finalitzem la funció.

Entrem ara a la funció recursiva, que és la que realitza la quasi totalitat de les tasques a fer.

Public Function ExpXMLRecursiu(ShapePerExplotar As String, port As Integer, Formes() As String, Carpeta)

Ja hem comentat els 4 paràmetres que ens arriben. El primer pas que farem serà obrir el fitxer de text on escriurem el codi.

Aquest fitxer tindrà per nom l'string que ens passen per paràmetre amb la variable ShapePerExplotar, el guardarem en el directori Carpeta com tots, i utilitzarem el canal passat per paràmetre per imprimir-hi. En cada recursió utilitzarem un port d'impressió i lectura diferent per evitar possibles conflictes.

En l'etapa de desenvolupament vaig definir un sistema de sangrat molt senzill que em permetia sangrar el codi resultant com si fos un lector d'XML, però sobre un fitxer de text. Això ho vaig generar amb l'idea de poder depurar un fragment de codi, ja que si aquest no compleix totes les característiques que ha de presentar un codi XML per ésser considerat com a tal, no es pot obrir amb un lector d'XML qualsevol i s'ha d'obrir com a text pla.

En aquests moments aquest sistema està desactivat, però no l'he eliminat del codi perquè pot ésser molt útil en pròximes modificacions a realitzar. En què consisteix:

Al imprimir en format XML no s'informen els salts de línia ni els tabuladors, i si llegim aquest com a text pla, ens trobem tot el codi en una línia immensa sense espais ni tabuladors ni salts de línia.

Per solventar-ho, en cada línia d'impressió vaig afegir dos tipus de variables. Una és Enterchr, i l'altre TabchrX, on X és el número de tabuladors que representa. En aquest moment aquestes variables tenen totes com a valor "" i per tant no modifiquen en absolut el text a imprimir. Només caldria per activar-les de nou, descomentar el valor que tenien, que correspon als caràcters ASCII que els representen, i així imprimir el text de manera ordenada en l'espai.

Després d'aquest senzill sistema de tabulació hem de realitzar una comprovació per saber si estem analitzant l'arrel principal *System* o ens trobem analitzant una altre *shape*, ja que en funció hem de realitzar una assignació o una altra.

Si estem analitzant *System*, el diagrama a tractar és el que tenim obert i actiu. A diferència, si es tracta d'una altre *shape*, hem d'obrir l'arxiu .vsd (Microsoft Visio®) corresponent.

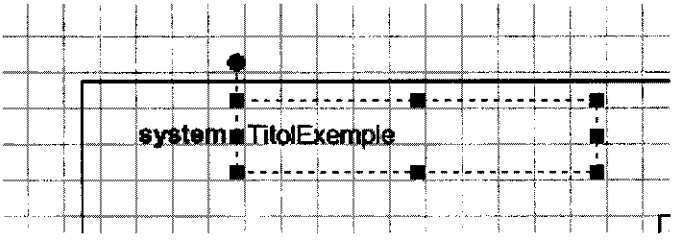
Utilitzarem l'objecte *Scripting.FileSystemObject* per comprovar si existeix el fitxer d'implementació de la *shape*. En aquest punt només el creem. L'utilitzarem unes línies més endavant.

Si recordem el que em comentat quan parlàvem de la funció principal, hem obert el *tag representation*, però no l'hem tancat, perquè necessitem informar-hi el títol, i obtenir aquesta informació des de la funció pare no era possible. Per tant, realitzarem un bucle per totes les *shapes* del diagrama per trobar la *shape Title* i obtenir-ne el valor.Ens podem trobar en el cas que el diagrama tingui més d'una pàgina, si és així ens quedarem

només amb el títol de la primera pàgina. En properes implementacions es podria ampliar aquesta funcionalitat i guardar el títol de cada pàgina.

Per controlar que el títol sigui el de la primera pàgina utilitzarem la propietat Índex de l'objecte *Page*. La primera pàgina tindrà índex 1.

Un cop trobat ens guardarem aquest valor a la variable varTitol i seguidament l'imprimirem.



16. Títol

Un cop imprès el títol ja podrem tancar també l'etiqueta *representation*.

```
<Representation>
        <Title>TitolExemple</Title>
</Representation>
```

17. Representació Títol a XML

Entrem a analitzar les figures d'una dimensió, o sigui, els canals.

Obrim l'etiqueta i realitzem un bucle on només entraran les *shapes* que tinguin el booleà OneD a cert. Sempre que parlem de bucle de *shapes*, ens referim a un bucle de *shapes* dins un bucle de pàgines, ja que com hem comentat amb anterioritat, podem tenir diagrames que ocupin més d'una pàgina.

La primera informació que guardarem de cada *shape* és el tipus. Si és un *Channel*, un *Delaying Channel*, etc. Per obtenir aquesta informació hem de retallar tota la informació que puguem trobar després d'un possible punt "." Mitjançant la propietat .name

Guardarem la informació referent a la localització de la *shape*. Hi ha un punt destacat a comentar en el cas de la coordenada X.

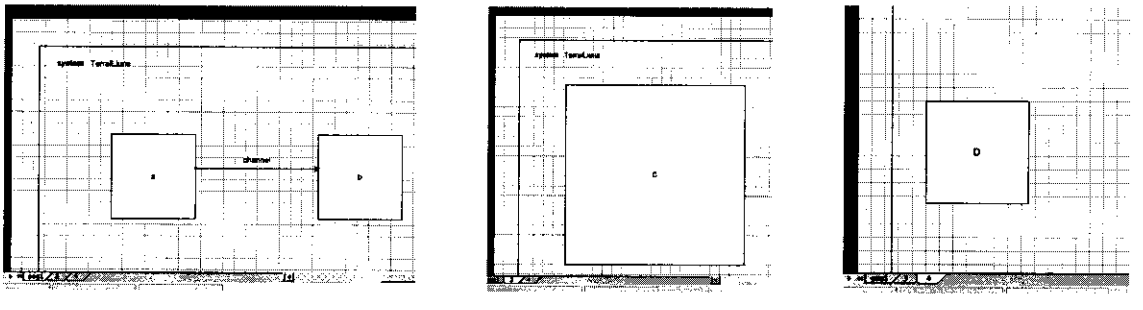
En el moment de dissenyar aquesta aplicació no disposàvem de la possibilitat de crear noves pàgines en un diagrama; sí és possible afegir pàgines en un diagrama Microsoft Visio®, però no en un diagrama Microsoft Visio® SDL. Degut això varem dissenyar un succedani per intentar representar el diagrama en diferents pàgines, i que alhora comprometés el mínim la possible modificació d'aquesta problemàtica.

Aquest sistema el que fa és modificar la coordenada X de les *shapes* que no estan a la pagina principal per tal que es dibuixin a la dreta de les altres *shapes* simulant tenir les pàgines col·locades consecutivament una a la dreta de l'altre. Vegem-ne un exemple:

Tenim un diagrama que ocupa 3 pàgines. Buscarem una representació semblant a la que obtenim col·locant les tres pàgines una al costat de l'altre tal com ho fem a l'exemple.

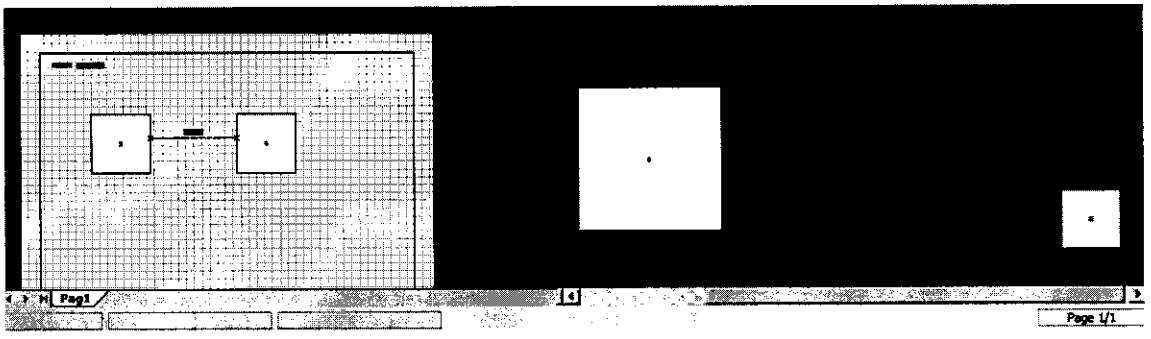
Per aconseguir-ho haurem d'afegir un desplaçament a la coordenada X en funció del número de pàgina en que es trobi cada figura.

Un bon desplaçament podrien ser 10 inxes, per tant multiplicarem l'índex de la pagina per 10 i li sumarem al desplaçament que ja tenia. De fet a l'índex de la pàgina li haurem de restar 1 abans de multiplicar, ja que si és la pàgina 1 no volem que ens afegeixi desplaçamanent, per tant seria (1 – 1) * 10 = 0. En el cas de la pagina 2, seria (2 – 1) * 10 = 10, obtindríem el desplaçament de 10 inxes desitjat per les *shapes* de la pàgina 2.



18. Diagrama de Tres Pàgines

El resultat d'exportar i llavors importar aquest diagrama seria el següent:



19. Resultat Unificat

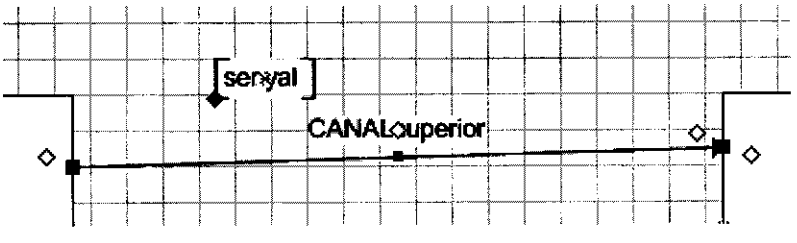
Que les *shapes* es dibuixin fora del marc establert no té cap influència en el funcionament.

Una virtut d'aquest sistema és que no modifica en absolut la funció d'importació, i per tant facilita la modificació d'aquest fragment estalviant feina als propers desenvolupadors. La funció d'importació llegirà que la coordenada X és tal, i l'enganxarà sense més ni menys a on li indiquem. Seguidament n'obtenim l'alçada i l'amplada.

Anem a tractar les connexions. Necessitem saber quines *shapes* uneix el canal que estem analitzant. Farem un recorregut pel conjunt *Connects* del canal, que si no té *Signals* adherits, serà de dos elements.

Utilitzem el mètode *.FromPart* per determinar quin tipus de connexió hi ha entre el canal i la *shape*. Aquí tenim un conjunt de condicionals (if's) que ens permeten anomenar d'una manera més pròxima el tipus de connexió.

El primer dels tres tipus que controlarem, són els *Control Points*.



20. Control Points

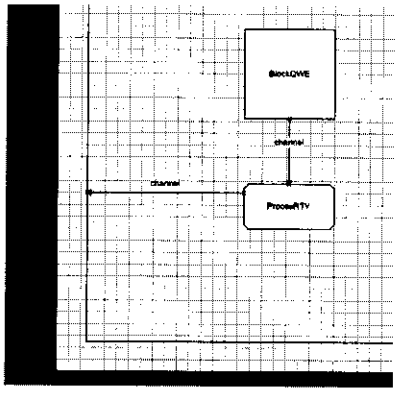
Els *Control Points* són aquests rombes de color groc si estan desconnectats, o de color vermell si estan connectats, que ens permeten detectar si el canal té *Signals* enganxats o no.

En el retorn del mètode que anomenàvem, reconeixerem la connexió per *Control Points* quan aquest sigui *visControlPoint* . Fent una resta d'strings obtindrem el punts exacte pel que es realitza la connexió. Detectat quin dels punts és el que està connectat aconseguirem les coordenades on es produeix la connexió. Cal dividir cada coordenada entre l'amplada o l'alçada de la *shape* Signal ja que hem d'expressar aquesta mesura en percentatge respecte l'amplada o alçada.

Emmagatzemem les dues mesures en dues variables, *GlueX* i *GlueY*, que ja tractarem després.

Analitzem ara el tipus de connexió origen (*begin*). En aquest punt hem de tenir en compte una excepció, que és la detecció d'aquella connexió entre una *shape* qualsevol del diagrama i el pare, o sigui la connexió amb la *shape* pare que s'està definint.

Aquesta connexió s'il·lustra connectant el canal al marc de la fulla, que és la *shape Frame.*



21. Shape Frame

L'únic que haurem de fer per resoldre aquesta excepció és controlar que si la *shape* connectada és *Frame*, a l'XML s'informi la *shape* pare, que portarà per nom el valor del títol de la pàgina, i com que aquest el tenim guardat a la variable varTitol, aplicarem l'assignació varText = varTitol. En el cas de no ser *Frame* la *shape* connectada, ens guardarem el nom, utilitzant *.text*, de la figura esmentada. Aquest fet el tindrem en compte tant en el control de la *shape* origen com en el de la *shape* final que vindrà a continuació.

En els tres tipus de connexió dividirem les coordenades entre l'amplada i alçada totals per poder expressar aquestes mesures en tan per cent.

El darrer tipus de connexió és el que ens determinarà la *shape* final, que només diferirà de la *shape* inicial en el retorn del mètode *.FromPart*, que enlloc de "*begin*" serà "*end*".

Després de l'anàlisi de les connexions venen les impressions. Iniciarem el connector amb l'etiqueta corresponent on hi haurà el tipus, la *shape* origen i la *shape* final, i seguidament les dades de representació (pàgina i coordenades en %).

En el cas que aquest canal tingui un *Signal* connectat, aquí és on imprimirem les dades necessàries d'aquest. Començarem amb el *tag Event* i el nom del *Signal*, i posteriorment les dades de representació, que són número del punt i coordenades de situació.

Un cop finalitzat el bucle clourem l'etiqueta que engloba els canals ("</channels>").

Anem a les *shapes* 2D.

Un element que amb els connectors no teníem és l'identificador numèric de les *shapes*. Segons el formalisme SDL, cada *shape* del document resultat té un identificador enter que comença del 1 i es va incrementant unitàriament.

Un cop més fem un bucle general de *shapes*. Abans de passar el control de si són 1D o 2D ens cal esbrinar el tipus de *shape* que són, ja que hi ha *shapes* de dues dimensions

que tampoc ens interessa que entrin el bucle. Aquestes són les *shapes* pròpies d'una fulla en blanc d'un document Microsoft Visio® SDL que no tenen un paper dins el diagrama i per tant no s'han d'informar al simulador. Són quatre:

- *Agent_type*: Informa del tipus de *shape* que s'està definint
- *Pages*: És l'indicador de pàgines de la part superior dreta de la fulla
- *Title*: És el títol de la pàgina
- *Frame*: És el marge de la fulla

A l'inici del bucle inicialitzarem el booleà Repetit a *false* que utilitzarem pel control de les *shapes* repetides.

Guardem, de la mateixa manera que fèiem amb als connectors, les coordenades de posició, amplada i alçada. Tenim el mateix codi per controlar les connexions de tots els tipus possibles, però en aquest moment no l'estem utilitzant. És molt probable que en futures ampliacions es vulguin controlar les connexions entre *shapes* 2D que no utilitzin connectors 1D. En aquest futur moment serà útil aquest fragment de codi.

Seguidament afegim el control de les *shapes* repetides. Creem un identificador concatenant el tipus de *shape* i el nom. Comprovem si aquest es troba ja dins el bucle, i si no hi és, li afegim. Si hi és, posem el booleà Repetit a cert. Cada cop que afegim un *block* al vector augmentarem la capacitat d'aquest en una posició, així evitem reservar memòria innecessàriament.

Procedim a comprovar, sempre i quan l'indicador Repetit sigui fals, si la *shape* que estem estudiant té implementació en un arxiu. Només poden tenir implementació externa els blocs i els processos.

Aquí és on utilitzem l'objecte *FileSystemObject* per preguntar si existeix un arxiu amb el nom de la *shape* .vsd en el directori Carpeta.

D'aquest doble condicional if (primer si la *shape* és *Block* o *Process*, i després si existeix el fitxer) se'n desprenen 3 opcions:

1. És *block* o *process* i té implementació.
2. És *block* o *process* i no té implementació.
3. No és *block* ni *process*.

En el primer cas, imprimirem la primera línia, informant l'atribut "*implementation*" amb valor "". Just després incrementarem l'identificador per deixar-lo apunt per la següent *shape*.

Incrementem també el port i llancem la funció recursiva amb els paràmetres següents:

- shpObj.Text: El nom de la *shape* que analitzarem.
- Int(PortSeguent): El port d'impressió.
- Formes(): El vector de repetits.
- Carpeta: El directori de treball.

Quan retorna el flux d'execució tenim el nou document Microsoft Visio® obert i l'hem de tancar ja que la recursió és un tema transparent a l'usuari i no ens interessa que quedin tots els documents entremitjos utilitzats oberts.

Realitzem les impressions corresponents i deixem el *tag representation* obert perquè la funció filla pugui informar-hi el *Title*.

Copiem el text provinent de la recursió a l'XML actual. Com sempre copiem línia a línia fins al final del document (EOF).

En el segon cas, l'atribut implementation prendrà com a valor "not defined". Incrementarem igualment l'identificador i realitzarem les impressions, que aquesta vegada si que clourem el *tag representation* ja que no hi escriurà cap funció filla darrera.

En el tercer cas, realitzarem exactament el mateix que en el segon. La única diferencia és que ens hem estalviat la pregunta de si tenia o no implementació ja que era impossible que en tingués.

Finalment tanquem l'etiqueta de la *shape* i tornem a dalt del bucle per analitzar la següent figura o en el cas de ser la darrera, finalitzarem la funció d'exportació tancant el port que utilitzàvem per imprimir.

Funció d'Importació d'un Diagrama a partir d'XML-SDL

Ens disposem a analitzar pas a pas la funció d'importació que ens permet recuperar un diagrama a partir d'un fitxer XML. Anàlogament, tindrem una funció principal bastant senzilla que ens llençarà la funció recursiva.

Utilitzarem la classe *MSXML2.DOMDocument* per moure'ns a través de l'XML. Ja l'hem introduït en un capítol anterior.

El primer pas és crear un objecte d'aquesta classe que l'anomenarem oDoc i que representarà el document XML que tindrem entre mans. Desestimarem l'opció de validar el codi en temps de lectura.

Igual que l'altre funció, crearem un objecte per què l'usuari seleccioni el document a carregar. Per dur-ho a terme utilitzarem l'objecte *UserAccounts.CommonDialog*.

Filtrarem els documents seleccionables amb l'extensió .xml.

Dins un condicional que detectarà si l'usuari ha seleccionat un document i no ha sortit mitjançant el botó cancel·lar, obrirem el document seleccionat i en guardarem també el directori a la variable Carpeta. Fets aquests passos, llancem la funció recursiva on li passarem un sol paràmetre.

- Arrel: És el node arrel de l'arbre XML que hem de llegir.

Function CarregarRec(oRoot)

Iniciarem la funció recursiva amb un condicional que ens permetrà saber si estem analitzant *System* (la primera crida) o si estem analitzant un altre sub arbre.

En cas d'estar tractant *System* el document Microsoft Visio® a utilitzar és l'actiu. En cas de tractar amb un sub arbre, serà necessari crear un altre document Microsoft Visio®.

En aquest pas li indiquem el tipus de diagrama a utilitzar, que és el SanDriLa.vst, ja que estem utilitzant el *plug-in* SanDriLa que ens afegeix les quatre *shapes* que ha de tenir tot diagrama SDL.

La segona comprovació és esbrinar si es tracta d'un *Process* o d'un *Block.* Ho descobrirem senzillament consultant el nom del node arrel que hem rebut com a paràmetre, ja que correspon amb el tipus de *shape* que serà. En funció de quin tipus tractem, seleccionarem un *Stencil* o un altre. Un *Stencil* és una llibreria de *shapes*.

Seguidament venen un seguit d'inicialitzacions de variables que ens permetran més o menys solucionar una problemàtica important d'aquest PFC.

Si bé la funció de càrrega presenta pocs problemes alhora de carregar un XML generat per la seva funció complementària, objectiu d'aquest projecte, ens hem trobat amb bastants traves alhora de carregar un XML generat per altres vies. Un dels principals problemes és que ens pot arribar un document XML que segueixi el formalisme SDL

però que no contingui dades de representació, o que no contingui les dades de representació que esperàvem.

En aquest tram definim unes coordenades i unes dimensions que si la *shape* a dibuixar ja té informades, sobreescriurà i no afectaran en absolut. Però si la *shape* a analitzar no les sobreescriu, no quedaran com a nul·les, i podrem dibuixar i situar la *shape* en una posició evitant un error de la funció al desconèixer aquests valors.

Inicialitzem també un títol que ens serà útil alhora de depurar el codi en possibles millores més endavant.

Iniciem el bucle on enganxarem tots les figures 2D.

Si recordem l'estructura XML,

```
<?xml version="1.0"?>
<system id="0" name="" implementation="" IP="" portRead="">
        <Representation>
                <Title>TitolExemple4</Title>
        </Representation>
        <channels>
                <Channel name="1to3" start="BLO1" end="BLO3">
                        <Representation>
                                <Page>Pag1</Page>
                                <StartX>0,5</StartX>
                                <StartY>0</StartY>
                                <EndX>0,5</EndX>
                                <EndY>1</EndY>
                        </Representation>
                </Channel>
                <Channel name="1to2" start="BLO1" end="BLO2">
                        <Representation>
                                <Page>Pag1</Page>
                                <StartX>1</StartX>
                                <StartY>0,5</StartY>
                                <EndX>0</EndX>
                                <EndY>0,5</EndY>
                        </Representation>
                </Channel>
        </channels>
        <Block id=" 1 " name="BLO1" implementation="not defined" IP="" portRead="">
                <Representation>
                        <Page>Pag1</Page>
                        <LocationX>1,1811</LocationX>
                        <LocationY>10,5512</LocationY>
                        <ShapeWidth>001,1811</ShapeWidth>
```

22. Estructura XML

Veurem que trobem a la mateixa alçada l'etiqueta *Representation*, l'etiqueta *Channels* i l'etiqueta *Block*. Aquests tres etiquetes tenen significats diferents i han de ser tractades diferents.

Primerament analitzem *Representation* per enganxar el títol.

Tal com està definit l'standard SDL, aquest pas no caldria ja que el títol sempre ha de coincidir amb el nom del fitxer, i per tant no necessitaríem informar-lo dins el global *Representation*. Personalment he cregut encertat deixar les portes obertes a possibles ampliacions en aquesta vessant, ja fos permetent canviar el títol de les pàgines, afegir distincions en els títols de diferents pàgines. A més aquesta estructuració permetrà fàcilment als propers desenvolupadors, la possibilitat de guardar informació de representació de la pàgina al tenir ja reservat i implementat aquest espai dins el document XML.

Tornant a la representació del títol en el codi, un cop hem recuperat el valor d'aquest de l'XML, realitzarem un bucle de les *shapes* existents en el document (que només n'hi haurà quatre, que seran les que venen ja de sèrie amb el document per utilitzar el *plug-in* SanDriLa) per trobar la *shape* Title i informar-la.

En el pròxim condicional evitarem que entri qualsevol node que no sigui una *shape* convencional 2D.

Ens trobem l'assignació posx = posx + 2. La funcionalitat d'aquesta instrucció ve d'acord amb les coordenades i dimensions que havíem inicialitzat per si ens trobàvem amb un document que no contingués dades de representació. Doncs bé, aquesta instrucció, que es durà a terme cada vegada que analitzem una *shape*, és la que ens farà dibuixar les *shapes* desproveïdes de representació una al costat de l'altre.

Generalment en un document, o totes les *shapes* tindran informació de representació o no en tindrà cap, fet que fa que no limitem aquesta instrucció només a les *shapes* que no tenen aquesta informació.

Per cada *shape* memoritzarem les coordenades i les dimensions en variables. Farem al mateix amb el nom, que es troba dins l'atribut *name* i no dins el global de representació. Substituïm els guions baixos per espais en l'*string* nom, mesura que havíem adoptat per evitat datar l'XML amb noms compostos que podrien ocasionar imprevistos.

Seleccionem el tipus de *shape* a enganxar i l'afegim al diagrama mitjançant el mètode *.Drop* que requereix com a paràmetres la *shape* i les dues coordenades on es situarà la figura. Informem el nom i les dimensions.

Hi ha tres tipus de figures (dins l'SDL) al Microsoft Visio® que no accepten unes dimensions imposades. Aquestes són el *Text*, el *Signal* i el *Comment*. Caldrà doncs que les evitem sempre abans de realitzar un redimensionament de *shape*.

Si la *shape* està explotada en un altre arxiu Microsoft Visio®, l'atribut implementation serà diferent de "*not defined*". Si aquest és diferent, llançarem la funció recursiva per poder generar el document Microsoft Visio® corresponent on s'explotarà la *shape*. Ho controlarem amb un condicional on cridarem la funció en cas de valor diferent de "*not defined*" i no farem res en cas de coincidir.

Aquí s'acaba el bucle de les figures 2D, anem ara al bucle dels canals (1D).

Repetim el bucle de *shapes* on només deixarem passar en el cas que varshape sigui "*channels*".

Channels és l'etiqueta que engloba tots els canals del diagrama, per això hem de desplegar un segon bucle que analitzi els fills d'aquest, ja que el tag *Channels* en sí no ens aporta cap informació.

Per cada canal en guardem el nom, la *shape* origen i la *shape* final. Aquestes tres dades provenen dels atributs del canal, que recuperem utilitzant el mètode *.Attributes.getNamedItem("X").Text* on X és el nom de l'atribut.

El tipus de la *shape* l'extraiem del nom del node, i igual que fèiem en el cas de les *shapes* 2D, substituïm els guions baixos per espais per obtenir exactament la cadena de caràcters que representa el tipus.

Enganxem el connector,

Set ShapePerEng = Application.Documents.Item(varDoc).Masters.ItemU(TipusShape)

varDoc és l'*stencil* que parlàvem abans, i TipusShape és el tipus de connector a enganxar.

Les coordenades que posem per defecte son (1,1), però no tenen cap rellevància ja que al connectar després amb les figures que correspongui, el connector quedarà posicionat en funció del seu origen i final.

Afegim el nom que tenim guardat en la variable que porta aquest mateix nom.

El codi que segueix serà l'encarregat de connectar el canal.

Trobem el rectangle origen realitzant una cerca de figures i preguntant si la *shape* que estem analitzant té el mateix nom que el valor de l'atribut Sta. En aquest moment afegim un condicional que ens detecti si la *shape* origen és la *shape* pare, o sigui si el nom és el títol de la pàgina. En aquest cas, com que la manera d'il·lustrar aquesta connexió és connectant al lateral, enllaçarem el connector amb el marc de la pàgina, que és la *shape Frame*.

El mateix codi es donarà per la *shape* final. En aquest moment tenim les dues *shapes* que necessitem per realitzar la connexió.

Inicialitzem les variables a zero per si no hi hagués el *tag representation*. En el cas que no hi hagués aquesta etiqueta, totes les connexions es farien per la cantonada inferior esquerra de les *shapes*.

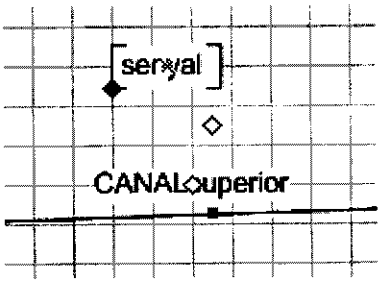
Recuperem les coordenades de les dues connexions, inicial i final, del canal en qüestió. Creem les cel·les necessàries, que vindrien a ser els punts, i connectem, ara sí, les dues terminacions. Utilitzem el mètode *.GlueToPos* que ens permet indicar la posició en percentatge i d'aquesta manera solventem el problema que teníem en el cas de no tenir *representation*.

Connectem els Signals. Recordem que el *Signal* ja es troba enganxat al diagrama, tan sols queda lligar-lo. Busquem el node de nom "event" per recuperar el nom del *Signal*, després el buscarem entre les *shapes* del diagrama.

La condició de cerca és que coincideixi el nom i que la *shape* sigui de tipus *Signal*. Un cop trobada la figura, igualem l'iterador al total de *shapes* perquè ja no ens interessa entrar més al bucle.

Següent pas, extreure de l'XML la informació de quin *Control Point* hem d'utilitzar i en quin punt es fa la unió. Són dades que tenim dins el *tag representation*.

Utilitzarem el mètode *.CellsSRC(visSectionControls, NumPunt, visCtlX)* per editar el punt de connexió.



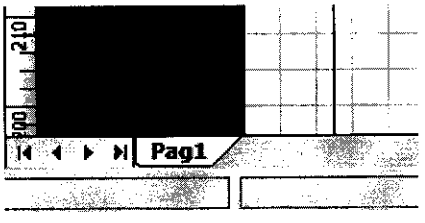
23. Diferents Control Points

Com podem veure en l'exemple de la imatge, tenim dos punts de control disponibles per realitzar la connexió, per això li passem com a paràmetre el número de punt.

Arribats al final de la funció, si estem tractant *System* com a node arrel, no hem de fer cap més acció, però en el cas d'estar tractant qualsevol altre node arrel, estarem treballant sobre un nou document que hem de guardar i tancar.

L'última instrucció es deu a la naturalesa del llenguatge i les funcions, on hem de retornar almenys un paràmetre. Retornarem un booleà a cert, que més endavant es pot evolucionar en un control d'errors més exhaustiu.

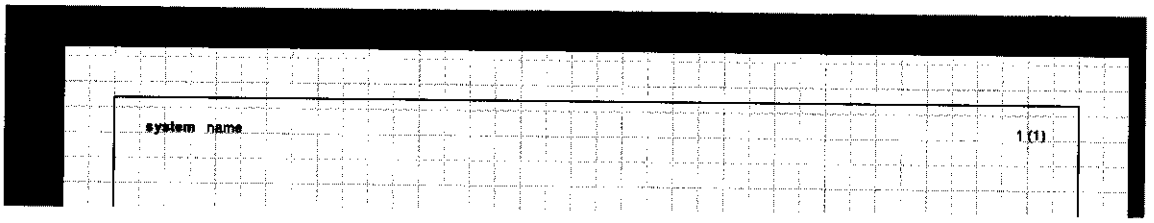
Limitacions



24. Paginació

La principal limitació que ens hem trobat ha set la impossibilitat d'afegir noves pàgines en un document Microsoft Visio® utilitzant el *plug-in* SanDriLa.

Si que és possible afegir noves pàgines però aquestes no tenen el format SanDriLa, amb les *shapes* inicials que ha de tenir, i l'*stencil* corresponent.



25. Shapes Inicials Pròpies del SanDriLa

Quan es disposi de la instrucció per a fer-ho, caldrà substituir el fragment de codi on multipliquem l'índex de pàgina per una determinada distancia amb la fi de simular la concatenació d'aquestes; per una simple lectura de la pàgina, que ja hem deixat informada en el document XML resultant.

La funció d'importació no es veu afectada per aquesta limitació ja que l'entrebanc rau en la creació però no en la lectura de les pàgines.

Un altre element que hagués afavorit l'eficiència, hagués set disposar d'una taula de referencia per accedir a cada *shape* sense haver de realitzar una cerca. Un taula de *hash* per exemple.

Conclusions

No es tracta d'un projecte dedicat a la investigació, amb uns objectius principals de conclusions, però si que de tota aquesta feina realitzada n'han sortit alguns punts a expressar.

Potser ha estat que la meva experiència en el sector informàtic, des d'un punt de vista laboral, estava més encarada a resoldre projectes de consultoria des d'una altre mirada, però he trobat interessant el fet de desenvolupar una eina que des d'una vista d'ocell es pot considerar una part molt petita de tot l'entramat. És a dir, estem parlant d'una aplicació que ens permet facilitar el procés d'entrar informació en un simulador, que està integrat per moltes altres petits peces i que alhora no deixa de ser una eina més.

Amb això no vull dir res més que ha estat un procés enriquidor, positiu i esperem que el màxim d'útil possible que m'ha fet veure des d'un altre punt de vista el procés de creació i desenvolupament d'aplicacions que desconeixia.

Treballs futurs

Hem dividit aquest capítol en dos principals punts, per una banda les funcionalitats pendents d'acabar o petites problemàtiques que no s'han pogut resoldre. Per l'altre banda hi ha una part de perfeccionament que dotarà de major funcionalitat aquesta eina.

Tasques Pendents

En aquest PFC hem dissenyat un lector de XML des de zero, i això fa que hagin quedat punts per resoldre. Un d'ells són els comentaris (<!-- -->). L'aplicació no entén els comentaris com a tal i intentarà entendre'ls com una *shape* més, donant un error.

L'etiqueta de representació s'ha d'anomenar exactament "*Representation*". Produiran errors altres etiquetes semblants com "*Representations*".

```
<Representation>
        <Page>Pag1</Page>
        <StartX>0,5</StartX>
        <StartY>0</StartY>
        <EndX>0,5</EndX>
        <EndY>1</EndY>
</Representation>
```

26. Tag Representation

Un altre entrebanc que no està contemplat és la doble nidació de la informació de representació, és a dir, tenir un node "*Representations*" que alhora tingui un sub node "*Representation*".

```
<representations>
  <!--value permet especificar la (
  representació serà vàlida.-->
  <representation value="empty">
    <mesh>"..\\Server.wrl"</mesh>
    <position></position>
  </representation>
</representations>
```

27. Doble Nidació

Gran part de les hores realitzades en el disseny i l'implementació han estat dedicades en l'explotació de *shapes* de tipus *Block*, deixant més de banda les *shapes* de tipus *Process*. Han quedat pendents de solventar algunes representacions d'elements referents a aquests, com les declaracions.

```
<process id="1" name="PEnvironment" in
  <!--PQueue declarations.-->
  <DCLS>
    <DCL name="Creation" type="int" va
  </DCLS>
```

28. Declaracions dins de Process

Tal com treballa ara la funció, entén que el *process* té una *shape* de tipus DCLS, que evidentment no existeix perquè es tracta d'una declaració.

Exactament la mateixa problemàtica tenim amb el global "*Start*" que trobem dins els *process*.

Per acabar, comentar que no ha set possible implementar un control d'errors i ho hem d'anotar com a tasca pendent. Un solució senzilla seria muntar un sistema d'excepcions, que evitin que l'usuari xoqui amb missatges d'error inintel·ligibles a l'ull no tècnic, i simplement mostri un error formal per pantalla en el cas, per exemple, que intentem

carregar un XML no vàlid, o intentem exportar a XML un diagrama no vàlid des del punt de vista del SanDriLa.

Perfeccionament

En aquest punt comentarem les millores que proposaria per aquesta aplicació amb la finalitat de donar-li més facilitat d'ús de cara l'usuari.

Una millora seria la creació d'un menú que s'adaptés al menú ja existent del Microsoft Visio®.

Alhora seria interessant la creació d'un instal·lador que creés tots aquests punts d'una manera automàtica i deixés l'entorn de treball perfectament adequat per treballar.

Adaptabilitat a Altres Llenguatges

En aquest capítol ens referim a les modificacions que hauríem de realitzar al codi per tal d'exportar els diagrames mitjançant un altre format o un altre formalisme.

Generalment dins el codi trobarem les instruccions referents a impressió d'informació d'una manera el més compacte possible, de manera que primer guardem tota la informació necessària, i llavors la imprimim tota de cop. Això facilitarà pròximes modificacions sobre aquestes impressions.

L'XML no és més que englobar la informació en etiquetes, inicial i final, de manera que no quedi cap dada no referenciada. Per imprimir en un altre format només caldrà canviar aquests fragments, ja que tota la part important d'obtenció de la informació no requerirà canvis.

La funció d'importació si que patiria més modificacions, ja que molta part del codi es deu a la cerca de la informació dins l'arbre XML. Les modificacions dependrien molt del format amb que ens vinguessin les dades.

Un cop recuperades les dades, tota l'altre part de codi romandria igual.

Bibliografia

1. XML:

http://msdn.microsoft.com/ca-es/xml/default(en-us).aspx
http://www.spanish-translator-services.com/espanol/t/infoset.htm

2. Microsoft Office

http://office.microsoft.com/es-es/products/FX100487413082.aspx?pid=CL100571083082

3. Microsoft Visio®

http://www.microsoft.com/latam/office/visio/prodinfo/overview.mspx

Fòrum:
http://social.technet.microsoft.com/Forums/en-US/visioes/threads

Informació sobre punts de connexió:
http://www.design-drawing.com/visio/floatin.htm
http://www.tech-archive.net/Archive/Visio/microsoft.public.visio.general/2004-05/0034.html

Tutorials:
http://office.microsoft.com/training/

4. UML

http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado
http://www.uml.org/

5. VBfA

http://msdn.microsoft.com/en-us/isv/bb190538.aspx

Macros Exemple sobre Visio:
http://visio.mvps.org/VBA.htm

Tractament XML amb VBfA:
http://msdn.microsoft.com/en-us/library/aa163921(office.10).aspx

6. SDL a wikipedia:

http://en.wikipedia.org/wiki/Specification_and_Description_Language

7. Article referent al SDL:

Telektronikk 4.2000
SDL-2000 for New Millenium Systems, de Rick Reed.

8. Sandrila SDL:

http://www.sdl.sandrila.co.uk/

9. Winter Simulation Conference 2008 Final Abstracts:

http://wintersim.org/abstracts08/POS.htm#fonsecaicasasp84590

Poster:
http://www-eio.upc.es/~pau/cms/cmsfiles/Papers/2008%20-%20WSC%20(poster).pdf

Índex d'Imatges

1. Esquema de l'Objectiu 7
2. Fragment de Codi XML 10
3. Logo Microsoft Office 11
4. Microsoft Visio® 12
5. Diagrama UML 14
6. Fragment de Codi VBfA 15
7. Transmissió de bits 17
8. Block i Process respectivament 18
9. Diagrama SDL 19
10. Representació XML 20
11. Diagrama de Gantt 23
12. Botó d'Ajuda 27
13. Diagrama de Casos d'Ús 28
14. Process 222 31
15. Shape Duplicada 35
16. Títol 39
17. Representació Títol a XML 39
18. Diagrama de Tres Pàgines 40
19. Resultat Unificat 41
20. Control Points 42
21. Shape Frame 42
22. Estructura XML 48
23. Diferents Control Points 53
24. Paginació 53
25. Shapes Inicials Pròpies del SanDriLa 54
26. Tag Representation 56
27. Doble Nidació 57
28. Declaracions dins de Process 57