Sistemas Informáticos
Curso 2002-03

Traductor de especificaciones de StateFlow - Simulink a EdROOM

Alejandro González Triviño
David M. García Cortés
José Miguel Ilzarbe

Dirigido por:
Prof. Jesús Manuel De la Cruz García
Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

AUTORIZACIÓN

Los alumnos responsables de este proyecto autorizan a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Fmdo.

Alejandro González Triviño David M. García Cortés José Miguel Ilzarbe

ÍNDICE

1. PRESENTACIÓN DEL PROYECTO

Resumen del Proyecto

El objetivo del Proyecto es desarrollar un traductor que convierta los archivos generados por StateFlow de Matlab Simulink en archivos utilizados por el editor de tecnología ROOM EdROOM para generar código en tiempo real.
Para realizar este trabajo hemos necesitado estudiar a fondo ambos sistemas de edición y los formatos de sus archivos.
El resultado es el traductor Sim2Ed, construido de forma parecida a un compilador.
Sim2Ed consta de un analizador que extrae los tokens necesarios del archivo fuente y los guarda en una estructura y de un generador de código que produce el archivo destino.

Abstract of the Project

Project's Objetive is the researching of a translator which became files generated by StateFlow from Matlab Simulink into files used by the ROOM technology editor EdROOM to generate real time code.
In order to do this job we have needed to make a hard study of both edition systems and the format of their files.
The result is the Sim2Ed translator, built as a compiler.
Sim2Ed is made of an analyzer that takes the needed tokens from the source file and keep them into an estructure and a code generator that produces the destiny file.

2. PLAN DE FASE

Para llevar a cabo este proyecto hemos dividido el tiempo disponible en tres fases: Investigación, Diseño e Implementación y Pruebas.

Análisis

Dedicamos los dos primeros meses de trabajo a documentarnos y estudiar tanto el funcionamiento como la estructura de los sistemas y ficheros con los que íbamos a trabajar ya que nos eran desconocidos.
En el apartado 3 se detallará la investigación realizada.

Diseño

A resultas de la investigación previa y con el requisito de realizar un diseño orientado a objetos se hizo la selección de clases que se iba a necesitar para guardar toda la información extraída del archivo fuente y que sería necesaria para crear el archivo destino.
En el apartado 4 se expone el diseño realizado.

Implementación y Pruebas

Llegado el momento de implementar los diseños obtenidos en la fase anterior nos decidimos por utilizar el lenguaje Java por ser el lenguaje en el que el equipo de desarrollo tenía más experiencia.
El entorno elegido fue el J Builder de Borland.
El apartado 5 incluye la documentación de las diferentes clases implementadas. Dicha documentación se escribió mediante el uso de Javadoc.

3. ANÁLISIS

Este es el estudio realizado de los dos sistemas con los que vamos a interactuar en el proyecto.
Nos interesamos en el funcionamiento de los editores gráficos y muy fundamentalmente en el código de archivos que generaba cada programa y que nuestro programa debe traducir.
A continuación detallamos las características de StateFlow y de EdROOM.

3.1 Matlab Simulink Stateflow

Matlab es un entorno técnico de altas prestaciones para cálculo numérico. Integra análisis numérico, cálculo matricial, proceso de señales y la posibilidad de visualizar en gráficos todos estos cálculos.
Una de las librerías de Matlab es Simulink.

Simulink es un sistema para realizar simulaciones dinámicas de sistemas. Incluye su propio entorno gráfico. Puede manejar sistemas lineales, no-lineales, en tiempo continuo, en tiempo discreto, multivariables...
Además permite el uso de las herramientas de Matlab para la visualización y el análisis de los resultados de las simulaciones.

StateFlow es un elemento de Simulink que permite introducir autómatas para coordinación, supervisión y paso de datos en modelo Simulink.Su editor gráfico permite realizar los modelos.
El comportamiento de cada máquina simulada mediante StateFlow está basado en los diagramas de Harel.

La máquina pasa por diferentes estados conectados mediante transiciones que se disparan mediante la sucesión de eventos y la producción de datos.

Cada modelo Simulink tiene una única máquina y puede contener varios Charts o Gráficos. Cada bloque StateFlow corresponde a un Gráfico.
El bloque se comunica con los otros bloques de Simulink y puede comunicarse con fuentes externas a Simulink (datos, eventos y código de usuario).

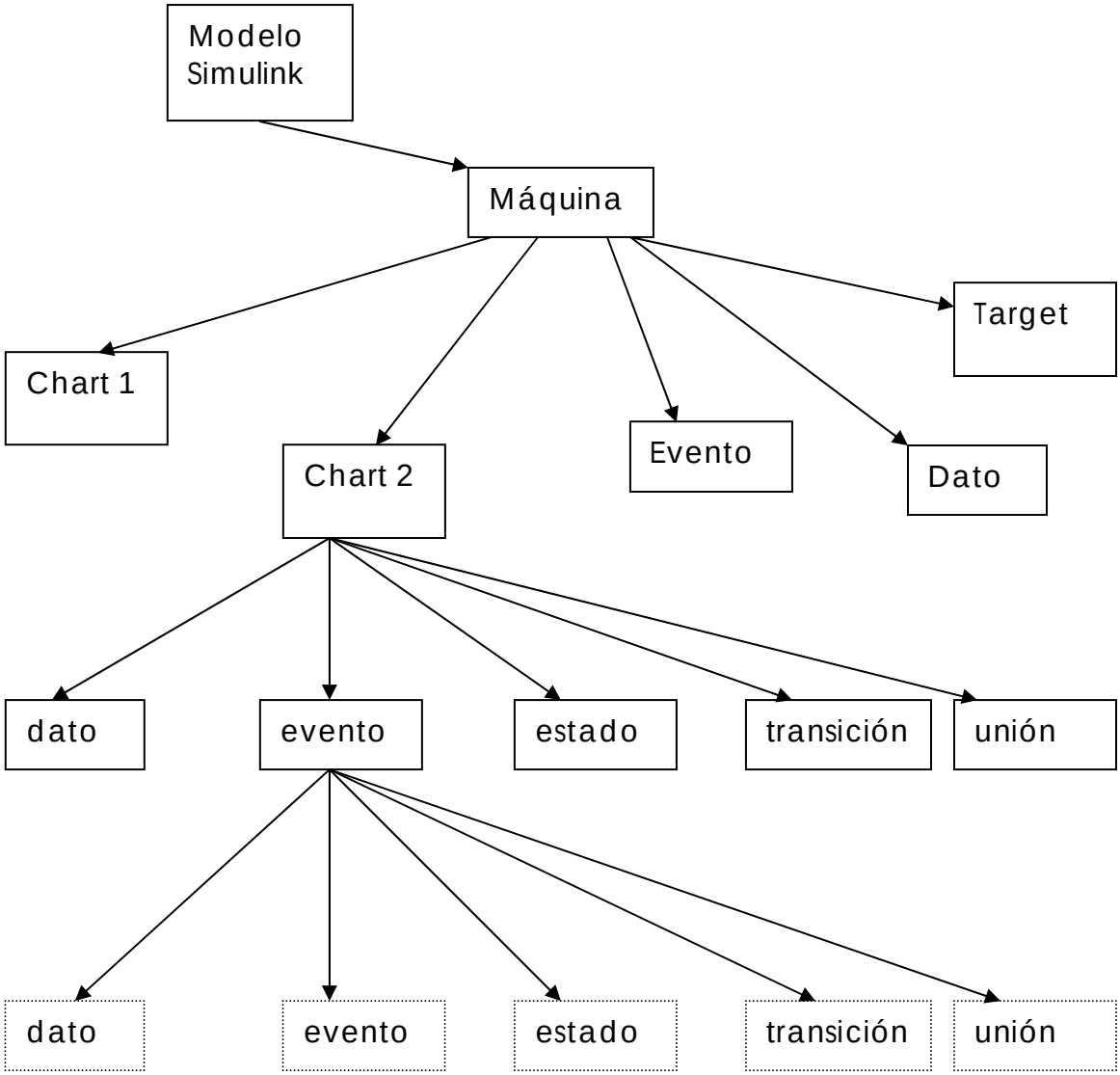
El Gráfico consiste en una combinación de objetos gráficos (estados, transiciones y uniones históricas y de conexión) y no gráficos (eventos, datos y target).

Toda la información de todos los objetos de StateFlow se guardan en un “diccionario de datos”.

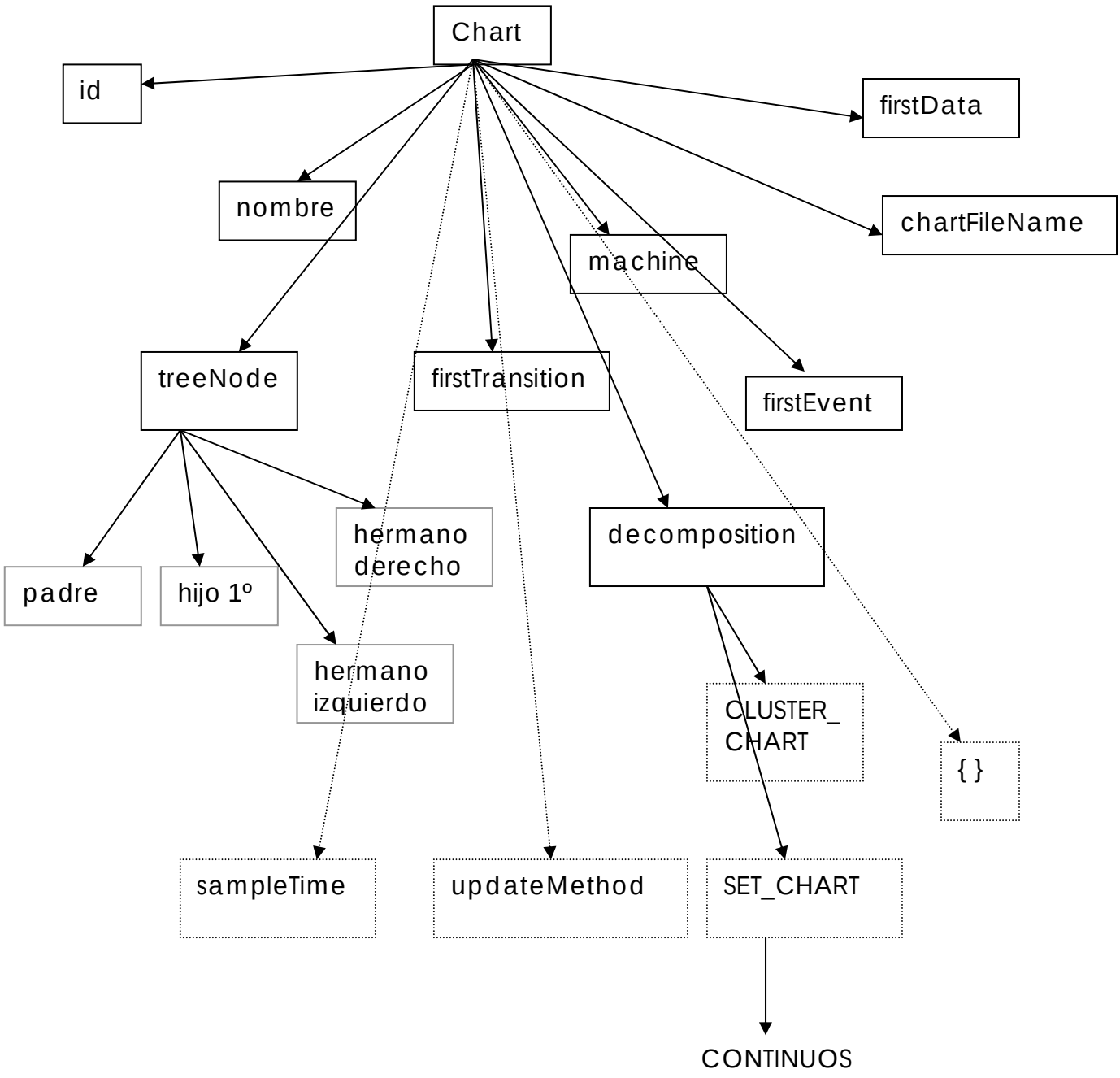
Este diccionario de datos será nuestro objetivo a la hora de analizar un archivo de SateFlow.

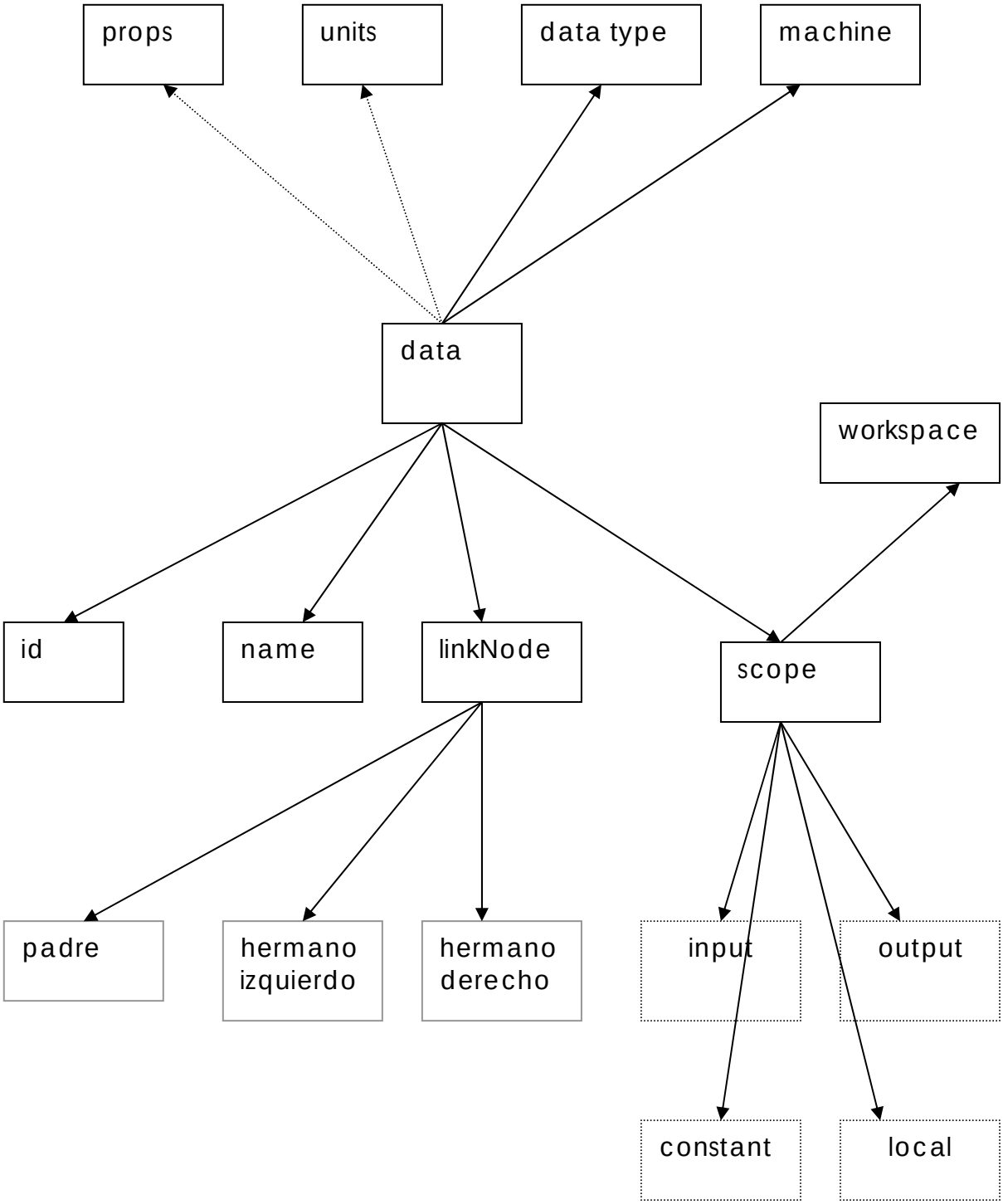
Los objetos se conocen en la jerarquía desde el nodo en que son creados hacia abajo.

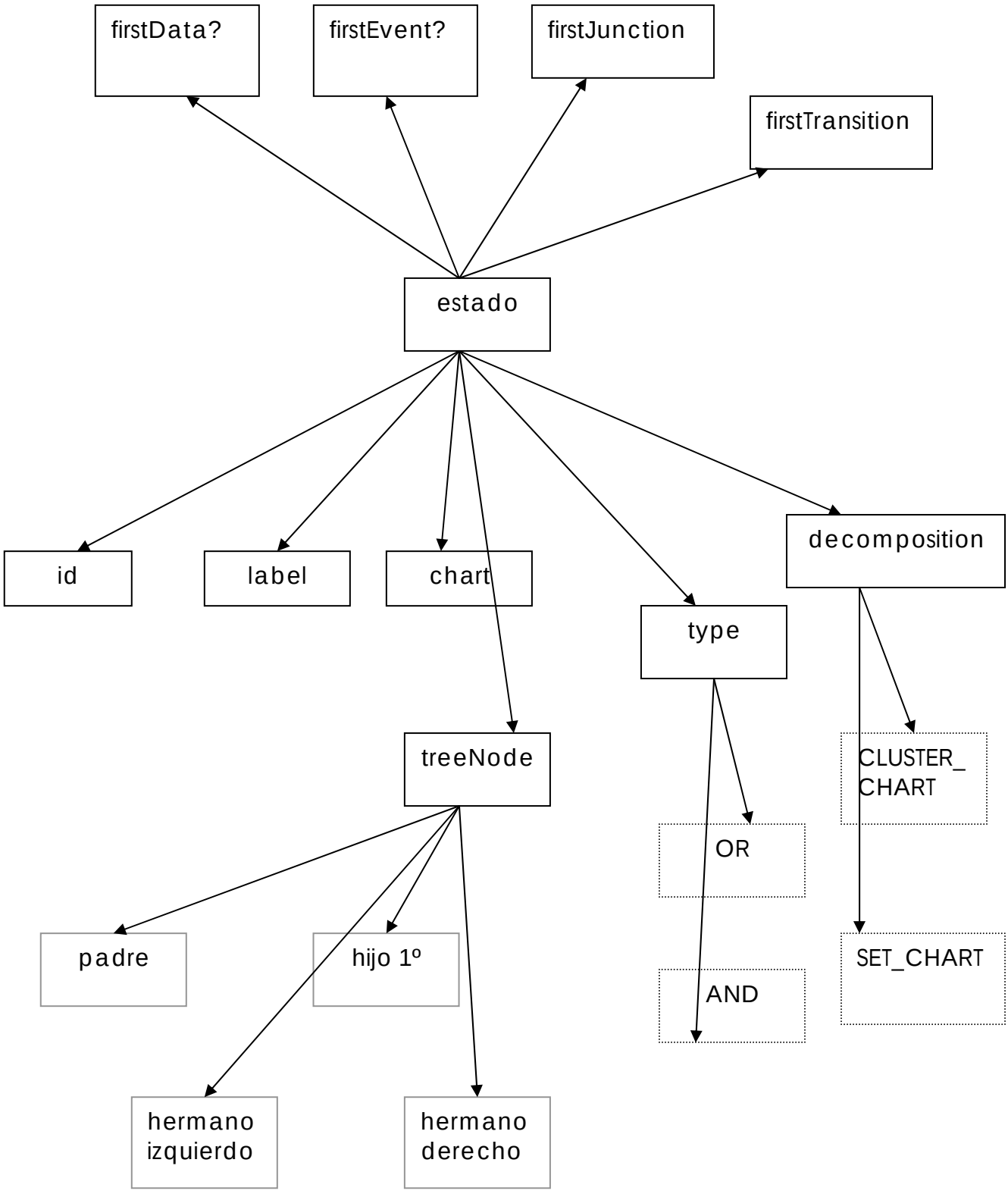
Los eventos y los datos pueden ser locales al bloque StateFlow o pasarse y recibirse de Simulink o de otras fuentes externas.
Estructura de un modelo Simulink con StateFlow

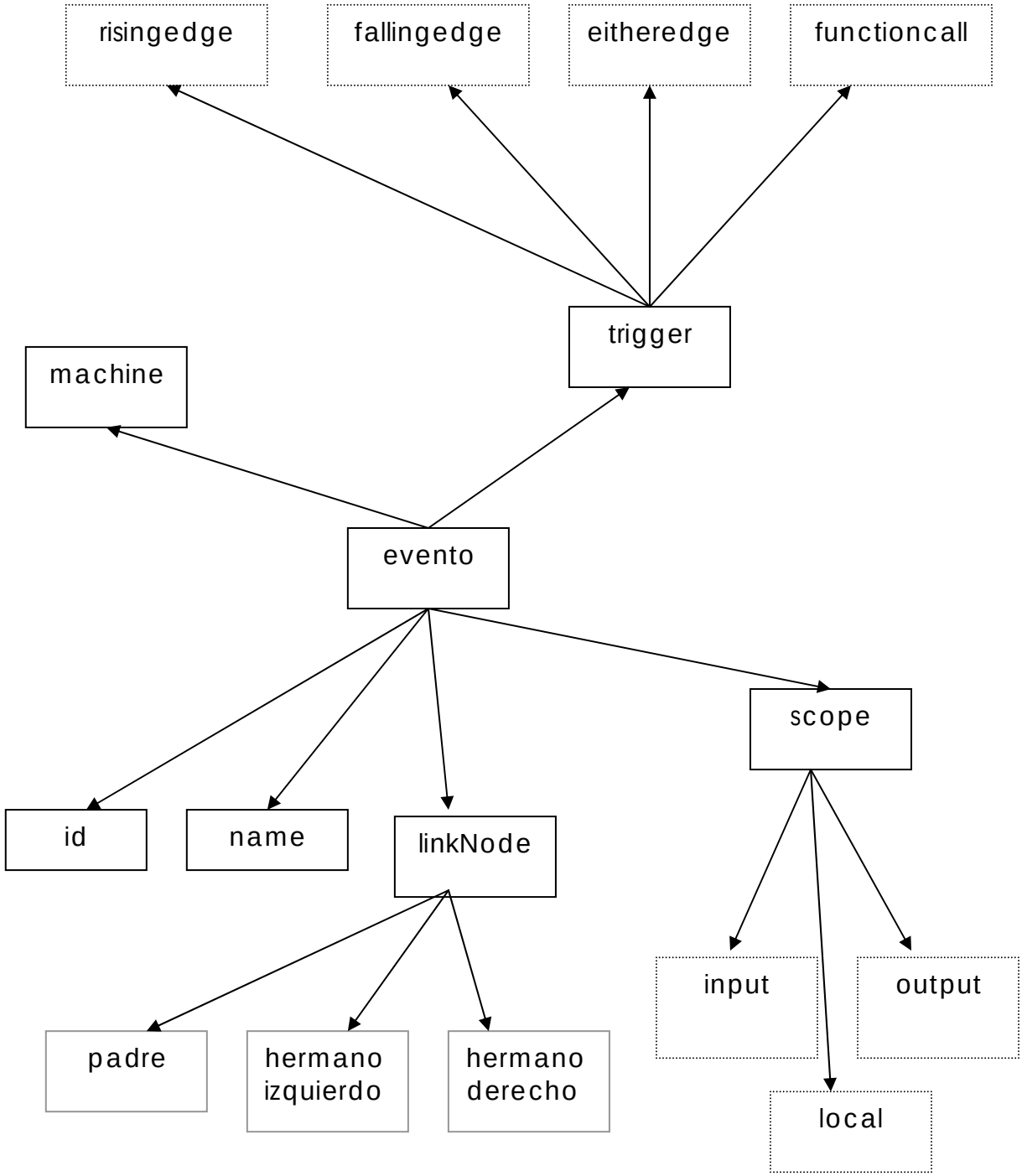


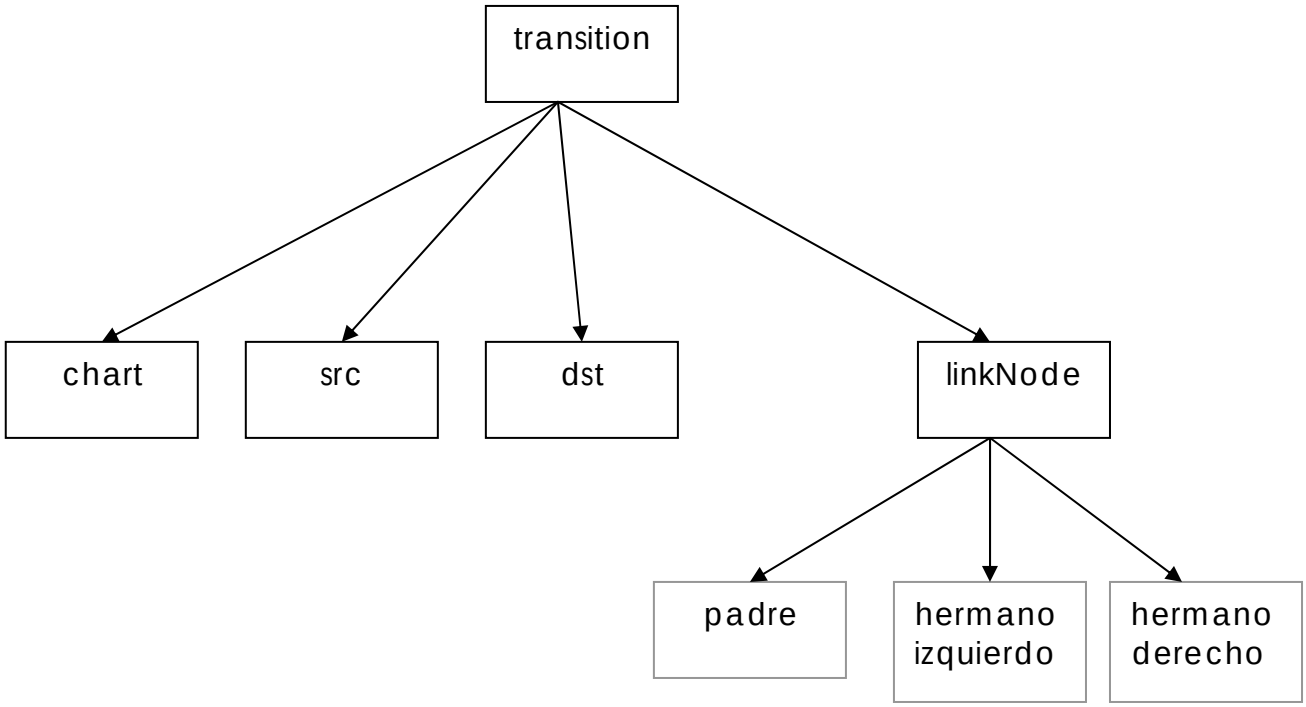
La estructura de la información mantenida por el diccionario para cada uno de los elementos es la que se indica a continuación.

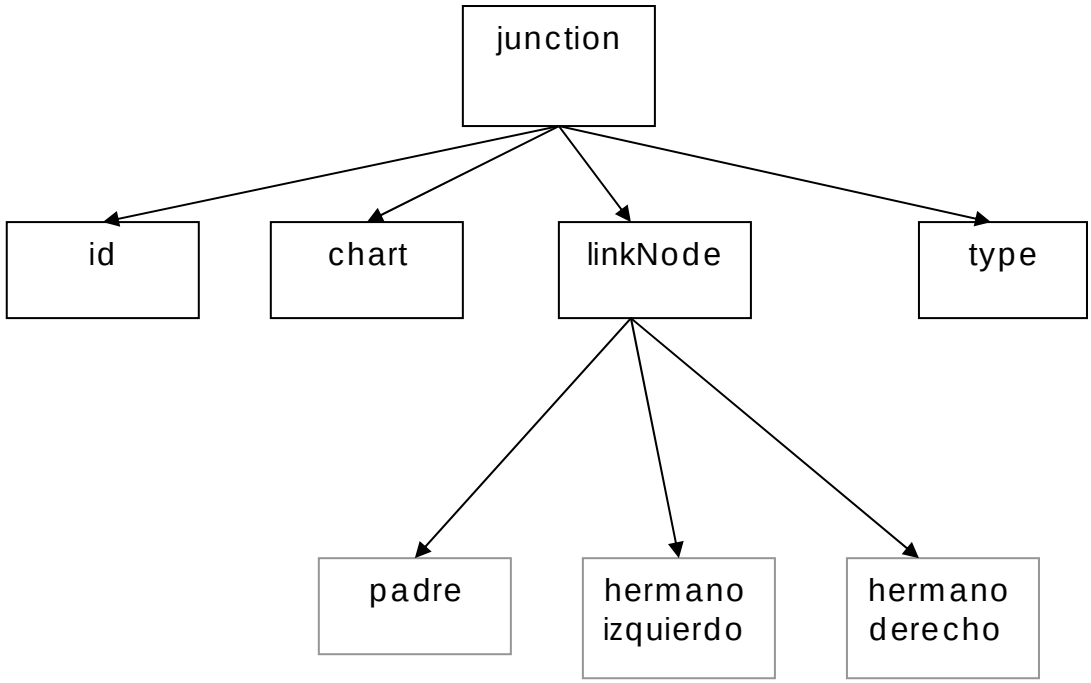



props
units
data type
machine
data
workspace
id
name
linkNode
scope
padre
hermano izquierdo
hermano derecho
input
output
constant
local


firstData?
firstEvent?
firstJunction
firstTransition
estado
id
label
chart
treeNode
padre
hijo 1º
hermano izquierdo
hermano derecho
type
OR
AND
decomposition
CLUSTER_ CHART
SET_CHART


risingedge
fallingedge
eitheredge
functioncall
trigger
machine
evento
scope
id
name
linkNode
input
output
padre
hermano izquierdo
hermano derecho
local


transition
chart
src
dst
linkNode
padre
hermano izquierdo
hermano derecho


junction
id
chart
linkNode
type
padre
hermano izquierdo
hermano derecho

El traductor deberá ir reconociendo las estructuras descritas en los anteriores diagramas e ir almacenándolas para ir realizando la correspondencia con ROOM.
Las palabras clave que nos interesan son:

- Machine: A partir de esta entrada se empieza a describir en un archivo de StateFlow el comportamiento de la máquina.

- Chart: Una máquina puede incluir uno o varios Chart. Cada Chart es un diagrama de comportamiento de un autómata. Sim2Ed sólo traducirá un Chart cada vez.

- State: El comportamiento del autómata se describe mediante Estados (States) que pueden incluir acciones a realizar a su entrada o a su salida. Un Estado puede tener dentro de si a otros Estados internos que constituyan autómatas internos. Además se puede modelar un autómata que tenga varios Estados activados al mismo tiempo. Con Sim2Ed los Estados deberán ser excluyentes, sólo habrá un Estado activado en un instante de tiempo.

- Transition: Las Transiciones conectan los diferentes estados para regular el comportamiento del autómata. Están disparadas por una sucesión de eventos. Cada Transición puede llevar asociada una condición de guarda que deberá cumplirse al realizar la Transición y una acción para realizar durante el cambio de Estados.

- Junction: Una unión une dos o más ramas de Transiciones que confluyen a un mismo Estado.

- Event: Un Evento es una señal que dispara un aTRansición y por lo tanto provoca un cambio en el Estado en que se encuentre el autómata. Puede ser externo o interno (Local) y de llegada (Input from Simulink) o de salida (output from Simulink).

- Data: Se trata de una señal que además transporta un dato para el autómata. También pueden ser internos (Local_Data) o externos de llegada (Input_Data) o de salida (Output_Data).

- History: El punto de Historial recuerda el último estado donde se encontraba el autómata cuando éste entró en estados de un autómata interno.

A continuación se muestra un archivo StateFlow de ejemplo con las palabras clave resaltadas.
Sólo se incluye a partir de la definición de la máquina de estados. El resto de la información no nos es necesaria y por lo tanto no se procesa.

Un ejemplo de StateFlow: Chart1.mdl

```
# Finite State Machines
# -- File save dated 22-Nov-2002 07:23:36
#
#    Stateflow Version 2.0 (R11) dated Jan 21 1999, 19:12:04
#
#

Stateflow {

machine {
id                                                              1
name      "chart1"
created   "22-Nov-2002 06:35:17"
firstTarget                                                     13
sfVersion  20011061
}

chart {
id                                                              2
name      "On_off"
windowPosition                                                  [189
64.5 365.25 342.75]
viewLimits                                                      [0
321.75 0 306.75]
screen                                                          [1 1
800 600 1.333333333333333]
treeNode                                                        [0 3 0
0]
firstTransition                                                 5
visible                                                         1
machine                                                         1
decomposition         CLUSTER_CHART
firstEvent                                                      8
firstData                                                       9
chartFileNumber                                                 1
}

state {
id                                                              3
labelString    "Power_on\n"
"en:\n"
"cuenta++\n"
"salida=1"
position
                                                                [80.260
45523104342 68.25000000000017 90 60]
fontSize                                                        10
chart                                                           2
```

```
treeNode                                                    [2 0 0
4]
type        OR_STATE
decomposition  CLUSTER_STATE
}

state {
id                                                          4
labelString    "Power_off\n"
"en:\n"
"cuenta++\n"
"salida=0"
position
                                                           [166.25
184.5 90 60]
fontSize                                                    10
chart                                                       2
treeNode                                                    [2 0 3
0]
type        OR_STATE
decomposition  CLUSTER_STATE
}

transition {
id                                                          5
labelString  "{cuenta=0}"
labelPosition[129.2547134238309 41.25000000000028 48 13.5]
fontSize                                                    10
src {
intersection                                               [0 0 0 0
129.7604552310434 24.00000000000017]
}
dst {
id                                                          3
intersection                                                [1 0 -
1 0.5833333333333334 132.7604552310434 68.25000000000017]
}
midPoint
                                                           [131.04
06393355885 37.15299455963847]
chart                                                       2
linkNode                                                    [2 0
6]
dataLimits
                                                           [129.76
04552310434 135.1604552310434 24.00000000000019
68.25000000000017]
}

transition {
id                                                          6
```

```
labelString  "Switch"
labelPosition[219.704993108786 128.844388247945 30.75 13.5]
fontSize                                                    10
src {
id                                                          4
intersection                                                [1 0 -
1 0.7583333333333333 234.5 184.5]
}
dst {
id                                                          3
intersection                                                [2 1 0
0.75 170.2604552310434 113.2500000000002]
}
midPoint
                                                            [226.64
00002746076 150.6297423494484]
chart                                                       2
linkNode                                                    [2 5
7]
dataLimits
                                                            [170.26
04552310434 234.5000000000001 110.8500000000002 184.5]
}

transition {
id                                                          7
labelString  "Switch"
labelPosition[160.4550265041131 141.6816520372935 30.75 13.5]
fontSize                                                    10
src {
id                                                          3
intersection                                                [3 0 1
0.5083333333333333 124.5104552310434 128.2500000000002]
}
dst {
id                                                          4
intersection                                                [1 0 -
1 0.2666666666666667 190.25 184.5]
}
midPoint
                                                            [142.76
3912191188 147.696153815318]
chart                                                       2
linkNode                                                    [2 6
0]
dataLimits
                                                            [124.51
04552310432 192.65 128.2500000000003 184.5]
}

event {
```

```
id                                                    8
name       "Switch"
linkNode   [2 0 0]
scope      INPUT_EVENT
trigger    RISING_EDGE_EVENT
machine                                               1
}

data {
id                                                    9
name       "cuenta"
linkNode   [2 0 10]
scope      OUTPUT_DATA
machine                                               1
dataType
                                                    "double"
}

data {
id                                                    10
name       "entrada"
linkNode   [2 9 11]
scope      INPUT_DATA
machine                                               1
dataType
                                                    "double"
}

data {
id                                                    11
name       "salida"
linkNode   [2 10 0]
scope      OUTPUT_DATA
machine                                               1
dataType
                                                    "double"
units                                                 "sec"
props {
range {
minimum                                               "0"
maximum                                               "10"
}
initialValue                                          "5"
}
}

instance {
id                                                    12
name
                                                    "On_off"
machine                                               1
```

```
chart                                                    2
}

target {
id                                                       13
name                                                     "sfun"
description
                                                         "Default
Simulink S-Function Target."
machine                                                  1
linkNode                                                 [1 0
0]
}
}
```

Como se puede observar en el ejemplo hay estructuras que pueden repetirse todas las veces que se necesiten, tales como los eventos, los datos y las transiciones. Otros sólo aparecen una vez en cada archivo como son machine y chart y otros pueden no aparecer dependiendo de las elecciones del usuario de StateFlow, como son las Junction.

Habrá que estar atentos y guardar también ciertos parámetros pertenecientes a los objetos gráficos (eventos, transiciones, uniones y puntos de historial) y adaptarlos para que el editor gráfico de EdROOM pueda reconstruir los diagramas.

La manera de recorrer todos los objetos por la jerarquía es el identificador numérico “id” que acompaña a la declaración de toda estructura.

3.2 EdROOM

EdROOM es una herramienta gráfica desarrollada por el grupo de Ingeniería de Sistemas y Automática del Departamento de Arquitectura de Computadores y Automática de la Facultad de Físicas de la Universidad Complutense de Madrid.
Usa la tecnología ROOM (Real-time Object Oriented Machine) para generar código en tiempo real para realizar la simulación de sistemas.
Es un entorno para modelar diseños orientados a objetos y simular sistemas en tiempo real.
Permite el prototipado rápido de sistemas distribuidos, dirigidos por eventos usando comunicación síncrona o asíncrona, y el desarrollo de implementaciones eficientes para ejecutar en plataformas de tiempo real.

ROOM usa los siguientes conceptos:

- Actores: Los componentes estructurados se llaman actores y son la unidad principal de diseño. Los actores pueden ser concurrentes, se comunican entre ellos mediante el paso de mensajes, están encapsulados, tienen una estructura y un comportamiento.

- Mensajes: Los actores interactúan con otros actores mediante mensajes. Cada actor tiene una seria de mensajes a los que puede responder. Un actor es invisible a otros actores, sólo pueden ver su interfaz a través de la que se comunican. Otros actores envían mensajes a un actor para que éste cumpla con las funciones de las que es responsable.

- Clases Actor: Lo que se diseña es una clase actor que sirve como base para crear actores de ese tipo específico. Se dice que dos o más actores del mismo tipo usados en n diseño son referencias a la misma clase actor. Una clase actor especifica una estructura y un comportamiento, así como los mensajes que pueden ser enviados y recibidos.

- Herencia: Hay una jerarquía de herencia de clases. Las subclases pueden heredar de las superclases atributos como la estructura, el comportamiento entre otras.

- Estructura de un actor: Un actor puede hacer referencia a otras clases actor. Puede estar compuesto a partir de otros actores. La descomposición de un actor se describe por su estructura. La estructura se define en la especificación de la clase actor y captura las comunicaciones y las relaciones de contenido entre los sistemas componentes.

- Comportamiento de un actor: Los componentes de un sistema deben tener una manera de reaccionar a los eventos del sistema. Tienen que tener también una forma de comunicarse con otros componentes dentro o fuera del sistema. Todos los actores pueden tener un comportamiento, que define como el actor responde a los mensajes entrantes. El comportamiento se especifica mediante una Máquina Finita de Estados. Cuando un actor recibe un mensaje puede ocurrir una transición que hace que la Máquina de Estados realice una acción específica y posiblemente haga un cambio de estado. El comportamiento especifica acciones que deben ser realizadas cuando se produce una transición. Las acciones pueden incluir el envío de mensajes, realizar cálculos, cambiar los valores de las variables locales y acceder a servicios de alguna capa más baja.

- Puertos: Los actores se comunican e interactúan entre ellos a través de los puertos. Un puerto es una referencia a una clase protocolo que define la serie de mensajes que un puerto puede enviar y recibir. Los puertos están conectados externamente a la interfaz de la estructura de un actor o internamente para comunicaciones internas del actor.

- Datos: Los datos se usan en el comportamiento de un actor. Una clase dato define un tipo de datos y las operaciones válidas sobre él. Los datos pueden ser enviados y recibidos por los actores usando mensajes. El dato del mensaje es procesado por el comportamiento del actor que lo recibe. Los datos, como los actores y los protocolos se especifican mediante clases.

Nuestro objetivo es obtener una clase actor a partir del chart de StateFlow.

El comportamiento de dicho actor lo especificaremos según la máquina de estados descrita en el chart.

Habrá que definir los mensajes que puede enviar y recibir el actor a partir de los datos y eventos de Simulink.

En cuanto a los puertos y protocolos, no hay objetos similares en StateFlow. La idea es que el usuario pueda definir mediante comentarios los puertos que quiere que tenga su actor y los protocolos que usen. Si no lo hace Sim2Ed irá asignando puertos y protocolos sistemáticamente. Formalizaremos esta idea más adelante.

Cada modelo de EdROOM sigue esta estructura:

Directorio del Modelo
    Archivo general.FMR
    Archivo de enumeración de clases Actor: ClasACtor.EGR
    Archivo de enumeración de clases protocolo: ClasProt.EGR

Archivo de enumeración de clases dato: ClasDato.EGR

Directorio ClasesActor

Archivos de las clases actores: ClaseActor_i_.EGR

Directorio ClasesProtocolo

Archivos de las clases protocolo: ClaseProtocolo_i_.EGR

Directorio ClasesDato

Archivos de las clases dato: ClaseDato_i_.H

3.3 Resultados y primera aproximación.

Después de estudiar ambos sistemas tenemos una idea general de la forma de ir realizando la traducción. Surgen puntos importantes que hay que resaltar. Algunos serán necesarios para el diseño del sistema y otros tiene que tenerlos en cuenta el usuario final.

- Aunque los dos sistemas basan sus máquinas de estados en diagramas de Harel la evaluación de estos modelos para detectar el estado siguiente se hace de forma diferente: StateFlow evalúa el contexto desde el exterior hacia el interior y EdROOM lo hace al revés, del más interno al más externo. Esto debe tenerlo en cuenta el usuario de Sim2Ed ya que aunque él diseñe usando StateFlow su modelo lo evaluará EdROOM.(*Ver Capítulo 6 sobre Recomendaciones de Uso*).

- Diferente sistema de comunicación. Hay que estudiar la correspondencia entre Eventos y Datos y Mensajes (con o sin Datos).

- La manera en que el usuario de Sim2Ed indicará los puertos y protocolos que formarán parte de su modelo.

- Correspondencia entre los parámetros de ROOM y los que disponemos de Stateflow.

- En una primera fase de diseño e implementación se ha acordado no permitir la inclusión de estados dentro de un mismo estado. Una vez que el sistema funcione sin permitir recursión interna se añadirá esta funcionalidad.

A continuación se ofrece la estructura de los archivos de EdROOM y la correspondencia de los parámetros que componen una máquina ROOM con una máquina StateFlow. Se detalla el lugar del archivo fuente donde obtener dichos parámetros necesarios.

Correspondencia StateFlow – EdROOM

ÍNDICE

1. ESTRUCTURA DE ARCHIVOS ROOM

Todo modelo ROOM debe crearse mediante el siguiente sistema de directorios:

Directorio del Modelo

- Archivo General.FMR
- ClasActor.EGR
- ClasProt.EGR
- ClasDato.EGR

Clases Actor

ClaseActor0.EGR
ClaseActor1.EGR
…

Clases Protocolo

ClaseProtocolo0.EGR
ClaseProtocolo1.EGR
…

Clases Datos

ClaseDato0.H
ClaseDato1.H
…

2. ARCHIVO GENERAL

<***nombreModelo***>.FMR = Machine → name

Estructura:

[Versión Modelo]
0
[Nombre Modelo]
<***nombreModelo***>
[Modelo Asociados]
0
[Bibliotecas Asociadas]
0
[Maquinas del Modelo]
1
[Versión Maquina]
0
Maquina Principal

1
1

3. ARCHIVO ENUMERACIÓN CLASES ACTOR

ClasActor.EGR

NombreClaseActor = Chart → name

Estructura:

[Versión Lista Actores]
0
<***nombreModelo***>
<***nombreClaseActor_0***>
…
<***nombreClaseActor_n***>
//FINACTORES

4. ARCHIVO ENUMERACIÓN CLASES PROTOCOLO

ClasProt.EGR

Estructura:

[Versión Lista Protocolos]
0
<***nombreClaseProtocolo_0***>
…
<***nombreClaseProtocolo_n***>
//FINPROTOCOLOS

5. ARCHIVO ENUMERACIÓN CLASES DATO

ClasDato.EGR

NombreClaseDato = Data → name

Estructura:

[Versión Lista Datos]
0
<***nombreClaseDato_0***>
…
<***nombreClaseDato_n***>
//FINDATOS

6. ARCHIVO CLASE ACTOR Y MÁQUINA DE ESTADOS

<***NombreClaseActor_i***>.EGR

Parámetros:

<***nombreTipoParam***>	Nombre del tipo del parámetro de inicialización. NULL si no lo hay.
<***CreateParam***>	Valores para el constructor del parámetros. Ocupan una línea. Será vacía si no necesitan o el tipo es NULL.
<***DescripParam***>	Descripción de los valores que necesita el constructor del parámetro. Ocupa 1 línea que puede estar en blanco.
<***TamañoPila***>	1024
<***NumPuertos***>	Número de puertos que forman la interfaz de la clase actor.
<***NombrePuerto***>	Nombre de un puerto.

Estructura:

[Versión Actor SW]
0
[Estructura]
[Versión Estructura]
2
[Parámetro]
<***nombreTipoParam***>
<***CreateParam***>
<***DescripParam***>
[FinParametro]
[Stack]
<***TamañoPila***>
[FinStack]
[Puertos]
<***NumPuertos***>
[Versión Puerto]
0
<***NombrePuerto***>
1
1
0
[Versión Grafico Puerto]
0
93
44
<***NombreProtocolo***>
[FinPuertos]
[ActoresComponente]
0
[FinActoresComponente]
[Uniones]
0
[FinUniones]

[NumTimers]
0
[FinEstructura]
[Comportamiento]
[Versión Comportamiento]
1
0
[***Descripción Máquina de Estados***]
[FinComportamiento]

MÁQUINA DE ESTADOS

Parámetros:

<***NombreContexto***> = Top
<***nombreContextoPadre***> = NULL
<***numSubestados***> El número de estados de la máquina.
<***numTransiciones***> El número de transiciones de la máquina.
<***numVariablesConstantes***>
<***numPools***> = 0
<***numMetodos***> Son las funciones.
<***numModulos***> = 0
<***numSubestadosConMaquina***> Número de estados compuestos.

Estructura:

[Maquina de Estados]
[Versión Maquina de Estados]
0
<***NombreContexto***>
<***nombreContextoPadre***>
[Subestados]
<***numSubestados***>
[Versión Subestados]
0
[***Descripción Subestado0|Descripción Subestadon***] Ver 6.1 Estados
[Puntos Elección]
0
[Versión Puntos Elección]
0
[Transiciones]
<***numTransiciones***>
[Versión Transición]
0
[***DescripcionTransicion0|DescripcionTransicionn***] Ver 6.2 Transiciones
[VariablesConstantes]
<***numVariablesConstantes***>
[***DescripcionVaryCons0|DescripcionVaryConsn***]
[Pools]
<***numPools***>
[Metodos]

[***DescripcionMetodo0*|*DecsripcionMetodon***]
[Modulos]
<***numModulos***>
[MaquinasSubEstados]
<***numSubestadosConMaquina***> *Al principio será 0
[***DescripcionMaquinaSub0*|*DescripcionMaquinaSubn***]
[FinMaquinasSubEstados]
[Fin Maquina de Estados]

6.1. Estados

Parámetros

<***nombreSubestado***>	=	State → labelstring
<***AccionEntrySub***>	=	State → labelstring → "entry"
<***AccionExitSub***>	=	State → labelstring → "exit"
<***AccionNoValidMsg***>	=	
<***Xpos***>	=	State → position → dato 1º
<***Ypos***>	=	State → position → dato 2º
<***Ancho***>	=	State → position → dato 3º
<***Alto***>	=	State → position → dato 4º

Estructura:

<***nombreSubestado***>
<***AccionEntrySub***>
<***AccionExitSub***>
<***AccionNoValidMsg***>
1
[ObjetoGrafico]
[Versión Objeto Grafico]
0
<Xpos> <Ypos> <Ancho> <Alto>
[Fin Objeto Grafico]

6.2. Transiciones

Parámetros

<***nombreTrans***>	=	Transition → labelstring
<***nombreOrigen***>	=	ROOMBorde si es borde | Transition → src → id → state → labelstring
<***TipoOrigen***>	=	OSubEst si es Subestado | OSubEstI si es Subestado Inicial * | OBorde si es borde | OPEntrada si es pto. de entrada ¿?
<***nombreDestino***>	=	ROOMBorde si es borde | Transition → dst → id → state → labelstring

<***TipoDestino***>	=	DSubEst si es Subestado | DBorde si es borde | DPSalida si espto. De salida ¿?
<***Accion***>	=	Labelstring a partir del símbolo "\" si la hay
<***Puerto***>	=	Puerto por el que se recibe la señal. Veremos como la especificamos en Simulink.
<***Señal***>	=	Identificador de la señal. Si se comunica con un Input/Output Simulink puede ser el propio Labelstring.
<***Guarda***>	=	Condición de guarda. Si viene aparece en el Labelstring como condición.
<***XOrigen***>	=	5º dato de Transition → scr
<***YOrigen***>	=	6º dato de Transition → scr
<***XDestino***>	=	5º dato de Transition → dst
<***YDestino***>	=	6º dato de Transition → dst
<***TipoLinea***>	=	

```
      if (((destinoX – origenX) >=0)&&((destinoY – origenY) >=0))
TipoLinea = 0;
      if (((destinoX – origenX) >=0)&&((destinoY – origenY) <=0))
TipoLinea = 1;
      if (((destinoX – origenX) <=0)&&((destinoY – origenY) >=0))
TipoLinea = 2;
      if (((destinoX – origenX) <=0)&&((destinoY – origenY) <=0))
TipoLinea = 3;
      if ((destinoY = origenY) && ( origenX < destinoX))
TipoLinea = 4;
      if ((destinoY = origenY) && ( origenX > destinos))
TipoLinea = 5;
      if ((destinoX = origenX) && (origenY < destinoY))
TipoLinea = 6;
      if ((destinoX = origenX) && (origenY > destinoY))
TipoLinea = 7;
```

Estructura:

<***nombreTrans***>
<***nombreOrigen***>
<***TipoOrigen***>
<***nombreDestino***>
<***TipoDestino***>
<***Accion***>
[Trigger]
<***Puerto***>
<***Señal***>
<***Guarda***>
[Grafico Transición]
[Versión Grafico Transición]
0
<***XOrigen***> <***YOrigen***> <***XDestino***> <***YDestino***> <***TipoLinea***>
[FinGraficoTransicion]

* La transición Inicial de Simulink hay que convertirla en el Estado Inicial de EdROOM ¡¡¡OJO!!!!

6.3. Junctions y Estados Especiales

6.4. Variables y Constantes

Parámetros

<***NombreVariable***>	=	Data → name
<***TipoVariable***>	=	Data → datatype
<***ValorInicial***>	=	Data → initialValue
<***VarConst***>	=	Variable/Constante
<***DimArray***>	=	0 si no es Array, si es Array la dimensión

Estructura

[Versión VarYConst]
0
<***NombreVariable***>
<***TipoVariable***>
<***ValorInicial***>
<***VarConst***>
<***DimArray***>

7. Archivo Protocolo

ClaseProtocolo_i.EGR

Parámetros

<***numMensajeIn***>
<***numMensajeOut***>

Estructura

[Versión Protocolo]
1
0
0
<***numMensajeIn***>
[***DescripciónMensaje0*** **|** ***DescripciónMensajen***]
<***numMensajeOut***>
[***DescripciónMensaje0*** **|** ***DescripciónMensajen***]

7.1 Descripción Mensajes

Parámetros

<***nombreSeñal***>	=	Event → labelstring | Data → name
<***nombreClaseDato***>	=	Data → dataType
<***pathClaseDato***>	=	
<***esSincrono***>	=	0, siempre es asíncrono

Estructura

[Versión Mensaje]
0
[Versión Señal]
0
<***nombreSeñal***>
[Versión Dato]
0
<***nombreClaseDato***>
<***pathClaseDato***>
<***esSincrono***>

8. Archivos de las Clases Dato

ClaseDatoi.H

No tienen formato. Son archivos .h que contienen la declaración de la clase dato correspondiente.

Correspondencia Eventos – Mensajes

StateFlow

- Eventos
  - Input from Simulink
  - Output from Simulink
  - Local
- Datos
  - Input from Simulink
  - Output from Simulink
  - Local

ROOM

- Mensajes : Contienen señal + <dato>
  - Entrada

  - Salida
- Datos: Información contenida en un mensaje.
  - Entrada
  - Salida
- Variables: Información interna a una tarea o actor
- Señales en las Transiciones : primer campo de un mensaje.

Realizamos la siguiente asignación:

- Eventos Input / Output : Son mensajes sin dato, sólo con una señal.
- Datos Input / Output: Son mensajes que además incluyen información de un dato.
- Datos Locales : Son las Variables internas de la máquina ROOM.
- Eventos Locales: No tienen correspondencia en ROOM. Con cada mensaje una tarea o actor comunica algo a otra tarea o actor. Un evento local no implica el paso de información entre tareas sino el paso de un estado a otro dentro de una misma tarea. Por ello, un evento local sería código que se ejecutaría dentro de una tarea sin pasar ningún tipo de mensaje. Se debería transformar en una función que realiza un cambio de estado y/o unas acciones. (*Ver Capítulo 6 sobre Recomendaciones de Uso.*)

En ROOM cada mensaje debe llevar un nombre, que es el de una señal, y después, de forma optativa, puede llevar un dato. Cada mensaje se identifica por el nombre de la señal y el puerto por el que se recibe.

Una transición o varias pueden ser disparadas por un mismo tipo de mensaje, es decir, con el mismo nombre de señal. No se puede asignar el mismo nombre a una transición y a una señal pues puede ocurrir que haya varias transiciones con el mismo nombre, lo que no es correcto.

Por su parte, una transición se puede realizar por distintos tipos de mensajes dependiendo de las circunstancias, luego no se puede asignar de forma general y directamente a una señal.

Definición de Puertos y Protocolos

Vamos a ver la forma en que se ha decidido realizar la definición de Puertos y los Protocolos en su paso de Simulink a EdROOM.

1. EVENTOS Y DATOS.

El usuario de Simulink tendrá la posibilidad de asociar a cada dato o evento de entrada / salida su puerto correspondiente.
La manera de hacerlo será la siguiente:

En el campo *"Description"* del dato o evento se incluirá como comentarios:

#PRTO <nombre puerto> donde:

: marca que indicará la aparición de un identificador.
PRTO : identificador que nos indicará un Puerto
<nombre puerto> : nombre del Puerto dado por el Usuario.

Si no se indica el campo #PRTO <nombre puerto> se asignará un puerto por defecto que consistirá en **_PRTO_nº consecutivo**.

Ejemplos:

Dato / Evento: Start
Description: #PRTO Gate → Puerto Asociado: Gate

Dato / Evento: Stop
Description: → Puerto Asociado: _PRTO_1

Dato / Evento: Pause
Description: → Puerto Asociado: _PRTO_2

Siguiendo este método, si el usuario no introduce nombres de puertos el sistema asignará un puerto por cada dato y evento del sistema.

2. PROTOCOLOS.

Cada puerto llevará asociado un protocolo salvo que el conjunto de entrada / salida de ese puerto sea un subconjunto de un protocolo ya existente con **nombre** en cuyo caso se le asignará el nombre ya existente.

La forma de nombrar los Protocolos será la siguiente:

Si existe el nombre del Puerto dado por el Usuario el nombre del Protocolo será el mismo nombre precedido de PRTCL, esto es:

PRTCL + **<nombre puerto>**

Si el Puerto está nombrado por defecto por Sim2Ed el nombre del Protocolo será:

PRTCL_PRTO_nº

Ejemplos:

Dato / Evento: Start Description: #PRTO Gate	→	Puerto Asociado: Gate Protocolo: PRTCLGate
Dato / Evento: Stop Description:	→	Puerto Asociado: _PRTO_1 Protocolo: PRTCL_PRTO_1
Dato / Evento: Pause Description:	→	Puerto Asociado: _PRTO_2 Protocolo: PRTCL_PRTO_1

Como se puede ver en el ejemplo, si el dato / evento Stop es del mismo tipo que el dato / evento Pause ambos seguirán un mismo Protocolo.

Una vez que se tienen claros todos estos conceptos podemos pasar al apartado del Diseño de la aplicación.

4. DISEÑO

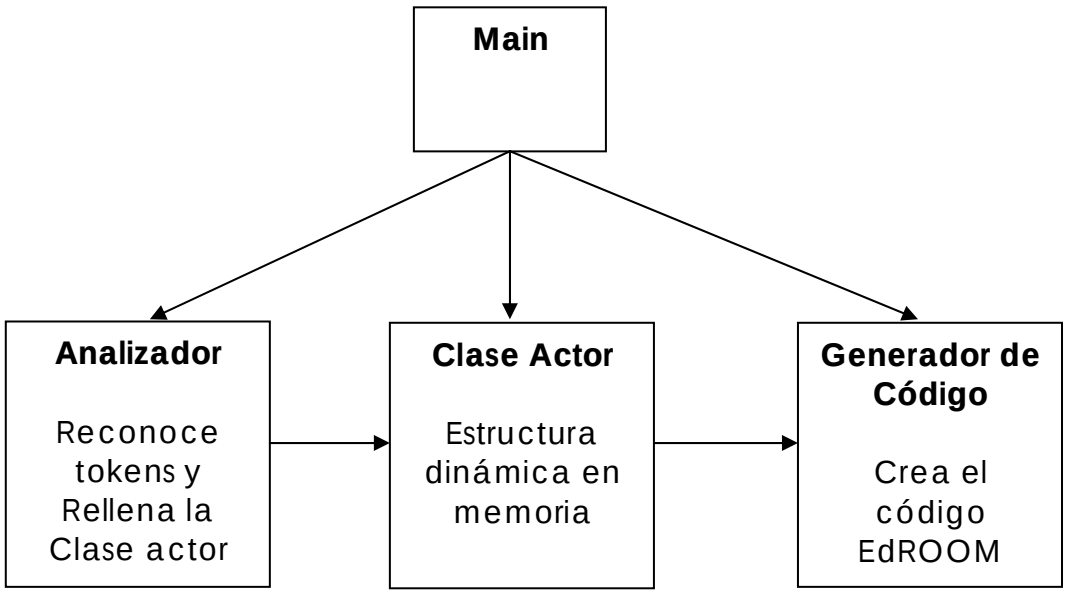
El sistema se basará en la creación de una estructura dinámica en memoria que guarde todos los objetos tanto gráficos (estados, transiciones, puertos) como no gráficos (mensajes, protocolos) necesarios para hacer la traducción.

Esta estructura la recogerá un analizador que recorrerá el archivo fuente de StateFlow y que reconocerá los tokens expuestos anteriormente. El analizador rellenará la estructura y una vez finalizado el análisis de la fuente se pasará la estructura al generador de código.

El generador de código irá escribiendo los diferentes directorios y archivos con la información existente en la estructura.

Este diseño es similar al usado por los compiladores actuales, donde se hace un análisis sintáctico, uno semántico y se guarda la información en tablas de símbolos. En este caso la estructura dinámica es nuestra tabla de símbolos.
La parte que nos ahorramos en este problema es el corrector de errores ya que los archivos generados por StateFlow siempre serán correctos. El único que puede cometer errores es el usuario a la hora de introducir los nombres e los puertos, pero si los comete el sistema los subsana nombrándolos él mismo.

Gráficamente:



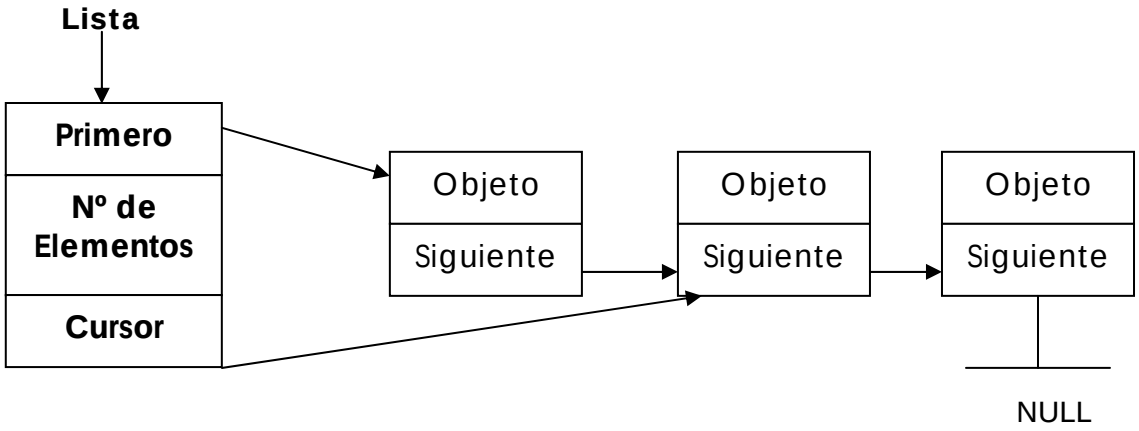
4.1 Estructura en Memoria.

La base para guardar los tokens son las listas enlazadas dinámicamente. Usando la metodología de la Programación Orientada a Objetos cada elemento diferente de los ya estudiados será un objeto con sus atributos y sus métodos de acceso a ellos.

La clase principal será la Clase Actor que contendrá los nombres del modelo y punteros a diferentes listas con los tokens reconocidos: estados, transiciones, protocolos, mensajes, etc...

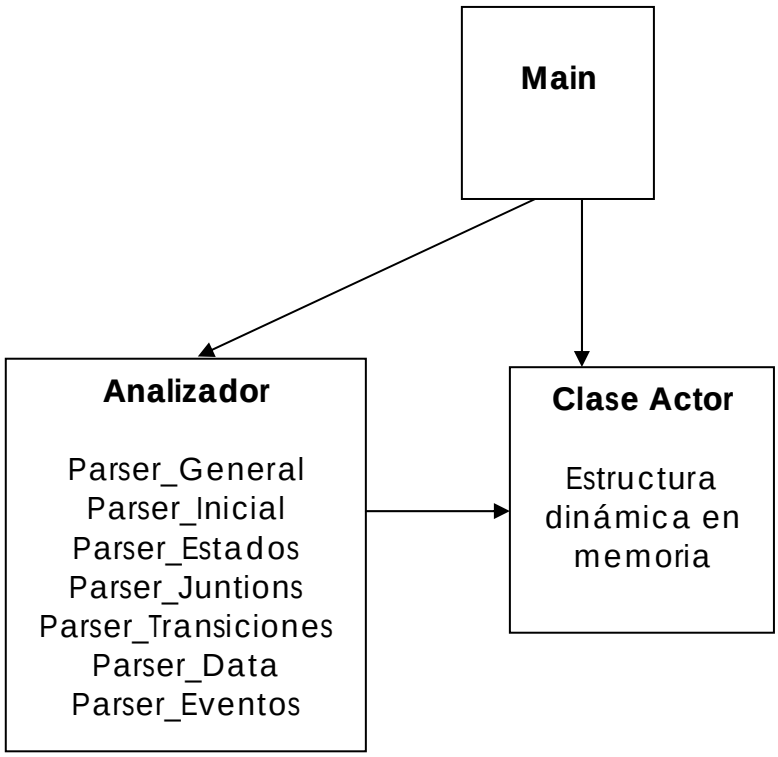
Las lista serán similares y estarán ordenadas por el orden de llegada e inserción de los objetos.
Tendrán también métodos para acceder a cada objeto, para escribirse por pantalla, modificar sus atributos y llevarán siempre la cuenta del número de elementos en la lista.
Contarán con un puntero “cursor” para recorrer la lista a la hora de generar el código.

Modelo de Lista:



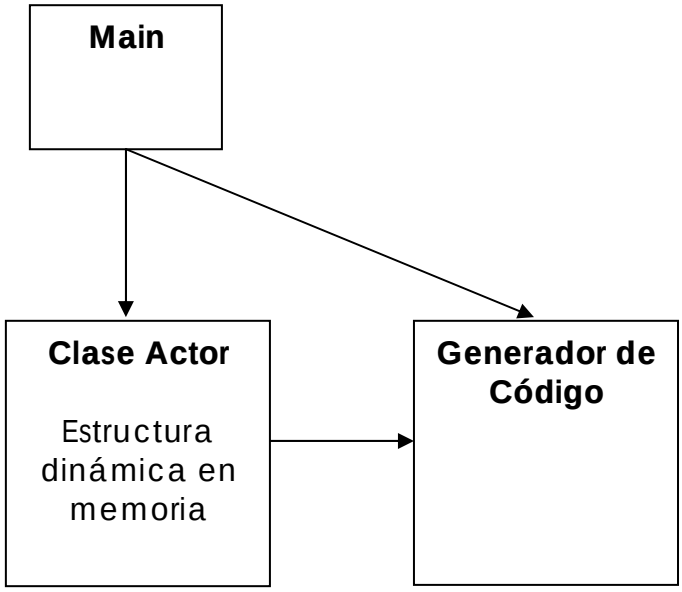
4.2 Analizador.

Al iniciar una traducción se creará una estructura vacía en memoria.
Esta estructura se pasa a los analizadores que la irán rellenando con los tokens reconocidos.
Hemos preferido dividir los analizadores en varios “parsers” para que cada uno detecte e inserte en la estructura un objeto determinado.
La otra posibilidad era realizar el análisis secuencialmente desde un mismo “parser” pero esto nos parecía menos modular y “orientado a objetos”.



4.3 Generador de Código.

El generador de Código se encargará de recorrer la estructura e ir generando los archivos correspondientes.
De nuevo hay un generador para cada objeto que se encarga de todo la información relacionada con esos tokens.



Además de las clases que implementan la estructura en memoria (20 en total), las que implementan los parsers (7) y la que implementa el generador de código (1) existe también la clase que implementará el interfaz gráfico (clase GUI) y la que gestionará el tratamiento de los eventos que se produzcan en esa interfaz (clase Orden) completando las 30 clases componentes del proyecto. Estas 2 últimas están implementadas conforme a la típica aplicación que aprovecha las ventajas del AWT de Java. Luego las comentaremos más a fondo.

5. IMPLEMENTACIÓN Y PRUEBAS

5.1 Estructura en Memoria.

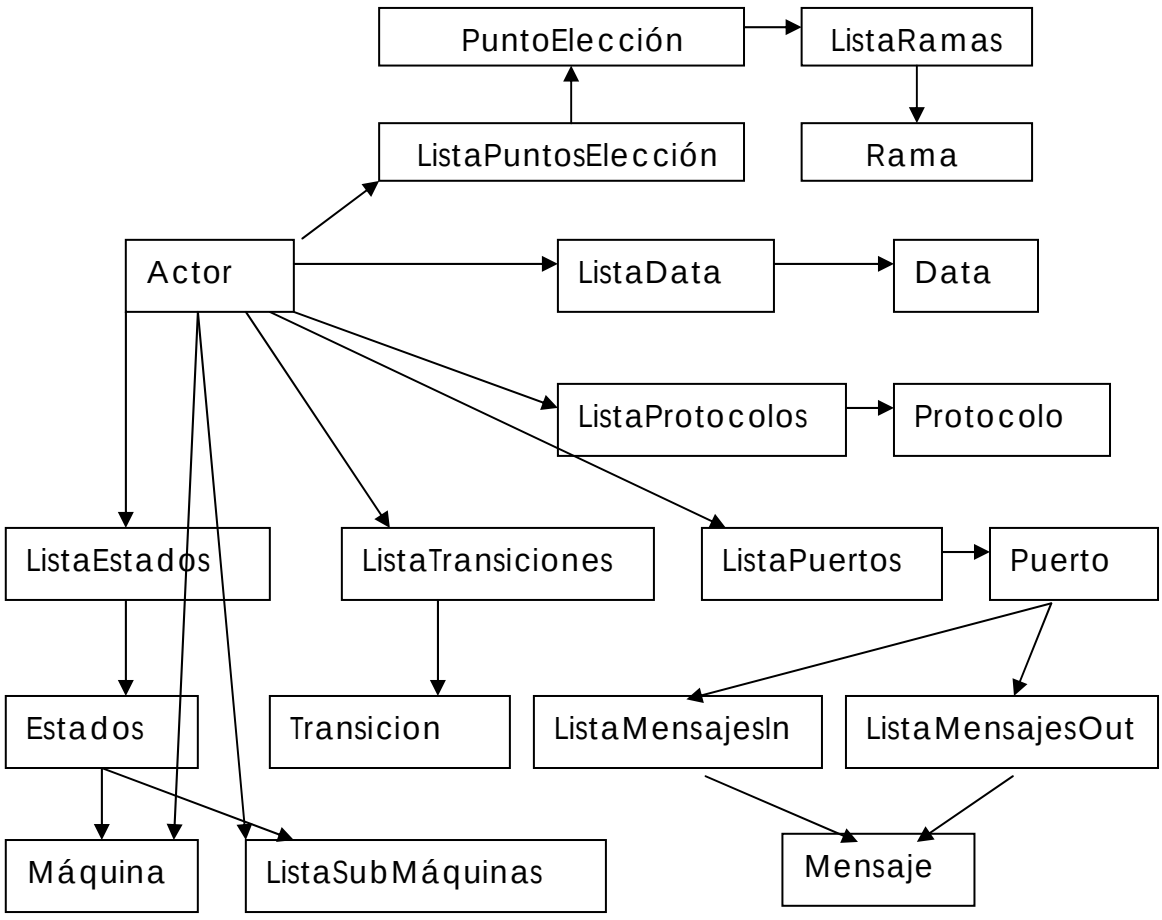
Esta son las clases resultantes a la hora de implementar nuestra estructura para guardar los tokens:

Clases:

1. Actor
2. ListaEstados
3. Estado
4. ListaTransiciones
5. Transición
6. ListaDatos
7. Dato
8. ListaPuntosElección
9. PuntoElección
10. ListaRamas
11. Rama
12. ListaPuertos
13. Puerto
14. ListaMensajesIn
15. ListaMensajesOut
16. Mensaje
17. ListaProtocolos
18. Protocolo
19. Lista SubMáquinas
20. Submáquina

El objeto de tipo Actor contendrá una liata de cada tipo de objetos que se irá rellenando dinámicamente a medida que vamos obteniendo del texto _.mdl de StateFlow los campos y tokens que nos interesan, que son aquellos necesarios para generar luego cada estructura EdROOM equivalente en el código definitivo.

Esta será su organización jerárquica, mostrada gráficamente:


PuntoElección
ListaRamas
ListaPuntosElección
Rama
Actor
ListaData
Data
ListaProtocolos
Protocolo
ListaEstados
ListaTransiciones
ListaPuertos
Puerto
Estados
Transicion
ListaMensajesIn
ListaMensajesOut
Máquina
ListaSubMáquinas
Mensaje

5.2 Analizador.

Esta son las clases resultantes a la hora de implementar nuestra estructura para guardar los tokens:

Clases:

1. ParserGeneral
2. ParserInicial
3. ParserEstados
4. ParserJunctions
5. ParserTransiciones
6. ParserEventos
7. ParserDatos

El primero de ellos se encarga de tomar el texto .mdl del modelo StateFlow a traducir, saltarse toda la zona referente a detalles de Simulink hasta llegar a la descripción de la máquina finita de estados, y llamar sucesivamente a los parsers que sean necesarios según los tipos de objetos StateFlow que se vaya encontrando en el texto .mdl.
Primero llamará al ParserInicial que será el encargado de extraer del inicio de esa descripción (zonas 'machine' y 'chart') el nombre del modelo y del actor. Cada vez que se llama a un parser éste analiza y procesa todos los objetos de la zona que le corresponde y devuelve el tipo de parser que se deberá usar a continuación según sea lo que se encuentre al finalizar su zona. Por ejemplo, el ParserInicial al acabar su zona se encontrará la palabra 'state',que significa que el siguiente objeto del modelo StateFlow será un estado. El ParserInicial devolverá el token 'state' al encontrárselo al finalizar su zona, y el ParserGeneral sabrá entonces que el siguiente parser que debe emplear será el ParserEstados, que, al situarse todos los objetos del mismo tipo siempre juntos, analizará todos los estados siguientes hasta llegar a una 'junction' o 'transition' etc. y devolverá ese token a su vez al ParserGeneral, que continuará empleando el parser correspondiente, y así sucesivamente hasta procesar todas las zonas del modelo.

Cada parser en concreto funciona básicamente distinguiendo casos, cuando detecta el token disparador de cada subcampo del objeto en cuestión analiza lo siguiente para extraer el parámetro correspondiente a ese subcampo. Por ejemplo, una vez encontrado el token 'state' y llamado el ParserEstados, éste ira procesando token a token comparando cada uno con los posibles campos que nos interesen ('name','labelstring', etc). Cuando detecte alguno de ellos (en este caso detectará primeramente el campo 'name') se ejecutará el código correspondiente a ese caso, que normalmente será seguir analizando la línea hasta captar el token o conjunto de tokens que representen a ese campo, en este caso el nombre del estado. Una vez obtenido sólo nos falta almacenarlo en el campo de la estructura que le corresponda, en este caso almacenarlo como el nombre del estado que se esta extrayendo. Para facilitar esta tarea iremos

obteniendo todos los campos que nos interesen del objeto en cuestión (en nuestro caso un estado) y al obtener el último (justo en ese momento) los compondremos todos creando un objeto de ese tipo (de tipo Estado en este ejemplo) con los campos obtenidos colocados como parámetros de su constructor,y finalmente insertaremos ese objeto en la lista de objetos de ese tipo que posee el Actor. Dado que los campos de cada objeto en un modelo StateFlow no son fijos, al menos todos, se puede dar la posibilidad de no saber si en ese objeto encontraremos ese último campo o no, por lo que si ese campo puede ser opcional deberemos asegurarnos de construir el objeto y de insertarlo una vez obtenidos los parámetros que tenga, por lo que en este caso ya no haremos la creación e inserción en el código de obtención de ese último campo (pues puede no existir) sino al empezar a procesar el objeto del mismo tipo siguiente, o en caso de ser el siguiente ya de otro tipo, al salir de ese parser.
Estas son resumidamente las 2 modalidades de creación e inserción del los parámetros del objeto extraído, con lo que todos los parsers seguirán esos principios para obtener y almacenar en forma de objetos (Estados, Transiciones, PuntosElección, etc.) todos los parámetros que necesita para la generación del código equivalente en EdROOM.

La equivalencia entre un modelo StateFlow y uno EdROOM no es obvia, de hecho se requieren modificaciones, creaciones extras y eliminaciones de objetos para construir dicha equivalencia. En los parsers será también donde se detecte la necesidad de esas acciones anteriores. Por ejemplo, en EdROOM hay siempre estado inicial, en StateFlow lo que hay son transiciones iniciales en tal caso,por lo tanto el parser que detecte una transición inicial (ParserTransiciones en este caso) deberá mandar también la creación de un nuevo estado inicial en EdROOM. Como este ejemplo hay muchos, y serán los parsers los que tengan que detectarlos y decidir que transformaciones emplear si son necesarias. Esto no deja de ser más tratamiento de casos. Otro caso es por ejemplo el hecho de que una transición que atraviese paredes de estados en EdROOM debe estar segmentada, sin embargo en StateFlow no es necesario, así que cada vez que el parser se encuentre una transición de este tipo deberá encargar las acciones oportunas para que esa transición se transforme en varios segmentos de transición equivalentes aceptados ya por la gramática de EdROOM. Esta última tarea por cierto no es desde luego nada trivial.
Otro caso más es la conversión en mensajes EdROOM de eventos y datos StateFlow, con la detección o generación de puertos correspondiente, creación, reconocimiento, asociación o fusión de protocolos a esos puertos, etc., etc., etc. En resumen un sin fin de funciones de transformación, reescalados, distinción de multitud de casos, consultas y operaciones entre parámetros para generar otros parámetros etc., pero dentro de las pautas de comportamiento que hemos señalado.

5.3 Generador de Código.

¿Qué hace concretamente el generador de código?
El generador de código se encarga de construir todos los directorios y archivos del modelo EdROOM correspondientes al modelo resultado de la traducción. Cada archivo se construirá conforme a la gramática propia del lenguaje destino. El generador 'sólo' tendrá que colocar cada uno de los parámetros obtenidos en el procesado del modelo StateFlow en el sitio que le corresponda dentro de la gramática EdROOM propia del objeto al que pertenecen, generando ordenadamente tantos objetos EdROOM de cierto tipo como objetos de ese tipo se hayan encontrado en el parseado. O sea, se recorre la estructura de objetos construida en la lectura generando el código EdROOM correspondiente a cada objeto de la estructura. ¿Cómo se hace esto? insertando cada parámetro de cada objeto de la estructura en el 'hueco' correspondiente a ese parámetro dentro de la gramática que impone EdROOM para ese objeto.

Para este cometido se implementan multitud de funciones que generan directorios y archivos y vierten en estos últimos segmentos de código fijos consultando parámetros de la estructura cuando hace falta para verterlos también, y esto se hace para cada tipo de objeto de una manera distinta y las veces correspondientes al número de objetos que exista de ese tipo, por lo que habrá distintas funciones para generar el código correspondiente a las distintas partes de la gramática. Con lo cual seguiremos teniendo una función general que llamará de forma jerárquica a todas las demás para generar el modelo completo.

Al estar permitido el anidamiento de estados se tiene que contemplar también la posibilidad de generaciones continuas y distintas de las descripciones propias del comportamiento de cada contexto (o submáquina), por lo que la estructura está paramétricamente estructurada, que no estructuralmente, pues sería menos eficiente su construcción. Es decir, a pesar de ser una jerarquía estructurada linealmente (pues la estructura no es un árbol, sino una serie de listas de objetos) cada objeto tiene también como parámetro el contexto al que pertenece. Todos los contextos se generarán ordenadamente y con ellos los objetos que se encuentren inmediatamente en ese contexto (que no más profundo). El parámetro de un objeto que indica el contexto al que pertenece en ese modelo lo obtendremos en el parseado, leyendo su estado padre en el campo 'linknode' del objeto.

Respecto a las clases GUI y Orden sólo decir que la primera implementa el interfaz explicado en la sección 7 y la segunda es más código de distinción de casos para tratar los distintos tipos de eventos que se pueden producir en nuestra aplicación (pulsación de cada botón, elección de menús, etc.), ejecutando como respuesta el código pertinente al evento producido.

Al final el cómetido de las distintas clases se verá reflejado en el informe de la traducción, ofrecido tanto por pantalla como en el archivo info.txt. Un proceso de traducción típico realizado por nuestra aplicación podría ser por ejemplo el siguiente:

Archivo encontrado.
* PROCESANDO ARCHIVO: sf_carousel(puertos).mdl.
* Procesando zona inicial.
* Zona inicial procesada con éxito.
* Procesando zona de estados.
* Zona de estados procesada con éxito.
* Procesando zona de puntos de elección.
* Zona de puntos de elección procesada con éxito.
* Procesando zona de transiciones.
Generado nuevo estado inicial 1001.
Insertado en el texto estado inicial 1001.
Generado nuevo estado inicial 1002.
Insertado en el texto estado inicial 1002.
Transformada en rama de punto de eleccion la transición 39.
Generado nuevo estado inicial 1003.
Insertado en el texto estado inicial 1003.
(Accion de transicion demasiado larga para marcarla en pantalla pero correctamente almacenada).
(Accion de transicion demasiado larga para marcarla en pantalla pero correctamente almacenada).
Transformada en rama de punto de eleccion la transición 44.
Generado nuevo estado inicial 1004.
Insertado en el texto estado inicial 1004.
Transformada en rama de punto de eleccion la transición 48.
Generado nuevo estado inicial 1005.
Insertado en el texto estado inicial 1005.
(Accion de transicion demasiado larga para marcarla en pantalla pero correctamente almacenada).
Generado nuevo estado inicial 1006.
Insertado en el texto estado inicial 1006.
* Zona de transiciones procesada con éxito.
* Procesando zona de eventos.
Generado nuevo puerto _PRTO_1.
Generado nuevo puerto _PRTO_2.
Desechado evento local por no tener correspondencia en EdRoom.
Desechado evento local por no tener correspondencia en EdRoom.
Desechado evento local por no tener correspondencia en EdRoom.
Desechado evento local por no tener correspondencia en EdRoom.
* Zona de eventos procesada con éxito.
* Procesando zona de datos.
Generado nuevo puerto _PRTO_3.
Desechado dato Max_speed por no ser ni de entrada ni de salida ni local.(tipos WORKSPACE_DATA, etc).
Desechado dato Duration por no ser ni de entrada ni de salida ni local.(tipos WORKSPACE_DATA, etc).
* Zona de datos procesada con éxito.
* ARCHIVO STATEFLOW PROCESADO CON ÉXITO.
* CREANDO LISTA DE PROTOCOLOS PROVISIONAL.
* FUSIONANDO PROTOCOLOS EQUIVALENTES.
* REESCALANDO MODELO GRÁFICO.
* GENERANDO DIRECTORIO DEL MODELO EDROOM.

```
* GENERANDO ARCHIVO GENERAL (.FMR) DEL ACTOR EDROOM.
* GENERANDO ARCHIVO ClasActor.EGR EDROOM.
* GENERANDO ARCHIVO ClasProt.EGR EDROOM.
* GENERANDO DIRECTORIO DE CLASES ACTOR EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DEL ACTOR EDROOM.
Generando Maquina de Estados del Actor.
        -> incluyendo estado Power_on.
        -> incluyendo estado Power_off.
        -> incluyendo estado Switch_init.
        -> incluyendo estado I.
        -> incluyendo transicion Switch_on(Trans id 29).
        -> incluyendo transicion Switch_off(Trans id 41).
        -> incluyendo transicion Switch_off(Trans id 43).
        -> incluyendo transicion Switch_on(Trans id 46).
        -> incluyendo transicion (Trans id 53).
        -> incluyendo dato Time.
        -> incluyendo dato tempo.
        -> incluyendo dato theta0.
    Generando Maquina de Estados del estado Power_on.
        -> incluyendo estado Timer.
        -> incluyendo estado Ride.
        -> incluyendo estado Video.
        -> incluyendo estado Audio.
    Generando Maquina de Estados del estado Timer.
        -> incluyendo estado Disabled.
        -> incluyendo estado Counting.
        -> incluyendo estado I.
        -> incluyendo transicion (Trans id 30).
        -> incluyendo transicion exit(On)(Trans id 34).
        -> incluyendo transicion (Trans id 36).
        -> incluyendo transicion Start_button(Trans id 37).
    Generando Maquina de Estados del estado Counting.
        -> incluyendo punto de eleccion 27.
        -> incluyendo transicion time_base(Trans id 49).
    Generando Maquina de Estados del estado Ride.
        -> incluyendo estado Stopped.
        -> incluyendo estado On.
        -> incluyendo estado I.
        -> incluyendo transicion (Trans id 28).
        -> incluyendo transicion (Trans id 31).
        -> incluyendo transicion Start_button(Trans id 32).
        -> incluyendo transicion Ride_done(Trans id 33).
        -> incluyendo transicion Emergency_stop(Trans id 35).
    Generando Maquina de Estados del estado Video.
        -> incluyendo estado panic.
        -> incluyendo estado push_start.
    Generando Maquina de Estados del estado panic.
        -> incluyendo estado edown.
        -> incluyendo estado eup.
        -> incluyendo estado I.
        -> incluyendo transicion emergency(Trans id 38).
```

```
        -> incluyendo transicion (Trans id 40).
        -> incluyendo transicion (Trans id 42).
    Generando Maquina de Estados del estado push_start.
        -> incluyendo estado sup.
        -> incluyendo estado sdown.
        -> incluyendo estado I.
        -> incluyendo transicion start(Trans id 50).
        -> incluyendo transicion (Trans id 51).
        -> incluyendo transicion (Trans id 52).
    Generando Maquina de Estados del estado Audio.
        -> incluyendo estado no_unix_sound.
        -> incluyendo estado metronome.
        -> incluyendo estado I.
        -> incluyendo punto de eleccion 24.
        -> incluyendo transicion (Trans id 45).
    Generando Maquina de Estados del estado metronome.
        -> incluyendo punto de eleccion 25.
        -> incluyendo punto de eleccion 26.
        -> incluyendo transicion (Trans id 47).
    Generando Maquina de Estados del estado Power_off.
        -> incluyendo estado start_switch_init.
        -> incluyendo estado emergency_switch_init.
* GENERANDO DIRECTORIO DE CLASES PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO ARCHIVO DE DESCRIPCION DE PROTOCOLO EDROOM.
* GENERANDO DIRECTORIO DE CLASES DATO EDROOM.
* GENERADA TODA LA ESTRUCTURA DE DIRECTORIOS Y ARCHIVOS DEL
MODELO EDROOM.
                    >>> FIN DE LA TRADUCCION <<<

( Detalles en esta ventana o en archivo de información e incidencias del
proceso: INFO.txt )
```

6. RECOMENDACIONES DE USO

El usuario debe tener presente que aunque use el editor gráfico de Simulink su simulación será interpretada y ejecutada por EdROOM. Debido a ciertas incompatibilidades intrínsecas en la construcción de ambos sistemas deberá respetar las reglas siguientes para obtener una traducción fiable.

- Stateflow evalúa la obtención del siguiente estado desde el contexto más exterior al más interior.
  ROOM evalúa desde el interior hacia el exterior.
  Por tanto, en su modelo a simular hay que tener en cuenta que la evaluación se hará de este último modo.
  Hay que evitar todo lo posible la ambigüedad en el modelo.

- No usar Eventos Locales en modelo en Stateflow ya que éstos no tienen correspondencia en ROOM y por el momento no se ha decidido una forma eficiente de implementarlos.

- Cada Transición debe ir asociada a un Evento que la dispare.

- Sería recomendable definir los Puertos y Protocolos de los Datos y Eventos. Para ello ver el documento “Puertos y Protocolos”.
  Si no se quieren definir el sistema los asignará por defecto.

- No usar Conditions-Actions para definir las Transiciones.

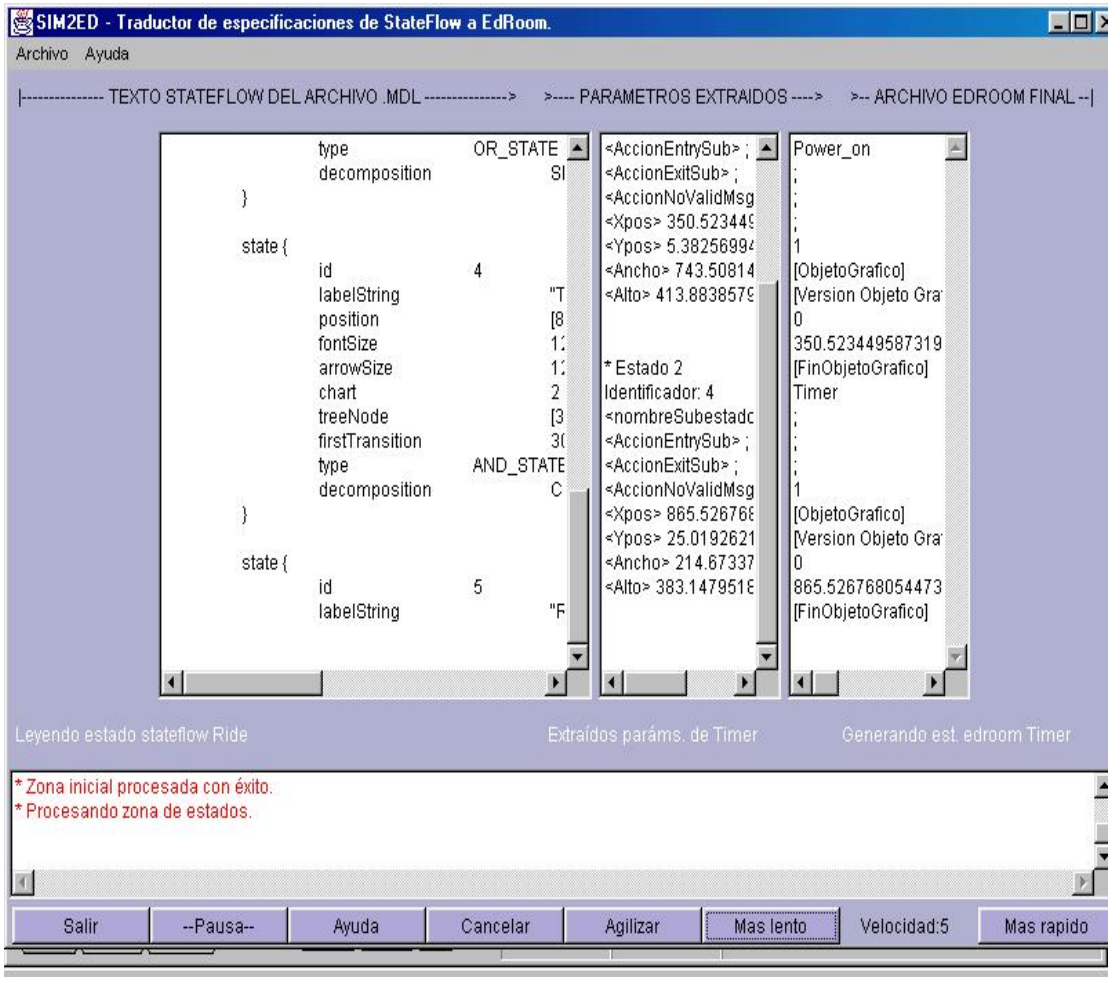
7. MANUAL DE USUARIO

En este apartado realizaremos una breve explicación acerca del funcionamiento de Sim2Ed.
Esta ayuda puede ser consultada durante la ejecución de la aplicación pulsando el botón de “Ayuda”.

1. **Presentación en Pantalla.**
2. **Comenzar una Traducción.**
3. **Modos de Traducción.**
4. **Opciones de los diversos Botones.**
5. **Advertencias.**

1. PRESENTACIÓN EN PANTALLA.

Al ejecutar Sim2Ed se accederá a la pantalla principal.En la parte superior se encuentra el menú que permite comenzar una nueva traducción, salir del programa y acceder a la ayuda.



La primera vez que se ejecute Sim2Ed se presentará automáticamente el cuadro de diálogo para comenzar una nueva traducción.

La parte central de la pantalla se divide en tres zonas, la de la izquierda indica la lectura de tokens que se va realizando del archivo fuente de StateFlow. El centro indica los parámetros necesarios par ala traducción que se van extrayendo de la fuente. Por último, el cuadro de la derecha muestra la generación del archivo EdROOM correspondiente.
Debajo de cada cuadro aparecen mensajes de texto que informarán del estado de la traducción.

El cuadro de texto inferior realiza un informe detallado del curso de la traducción que además se guarda en un archivo de texto para posteriores consultas.

En el borde inferior de la pantalla se encuentran los botones de opciones que se comentarán más adelante.

2. COMENZAR UNA TRADUCCIÓN.

Tanto si ejecutamos la aplicación por vez primera como si seleccionamos la opción

Archivo --> Nueva Traducción

aparecerá el cuadro de diálogo donde selecionar el archivo de StateFlow .mdl que queramos traducir.
A continuaciñon deberemos seleccionar el modo de traducción pulsando en el botón correspondiente.

3. MODOS DE TRADUCCIÓN.

Sim2Ed ofrece dos modos de traducción : Traducción Visual y Traducción Inmediata.

En el modo Traducción Visual la aplicación irá informando al usuario a través de la interfaz de pantalla ya comentada del desarrollo del proceso.

El modo Traducción Inmediata presenta de forma más rápida un cuadro de texto con el resultado y la generación de archivos de la traducción.

4. OPCIONES DE LOS DIVERSOS BOTONES.

La botonera del borde inferior de la pantalla permite realizar las acciones siguientes:

- Salir: Cierra la aplicación y suspende toda actividad en curso.
Se consigue el mismo efecto seleccionando Archivo --> Salir

- Pausa: Congela la traducción en el punto en el que se encuentre.Al pausar el proceso el botón cambia a "Continuar". Pulsándolo el proceso continúa a partir del punto en el que se detuvo.

- Ayuda: Permite acceder a este archivo de ayuda.
  Se consigue el mismo efecto seleccionando Ayuda en el menú.

- Cancelar: Interrumpe la traducción en curso sin posibilidad de reanudar el proceso.

- Agilizar: Cambia al modo de Traducción Inmediata.

- Más Lento: Reduce la velocidad de traducción.

- Velocidad: Indicador de la velocidad de traducción.(valor entre 1 y 10) (si el valos es menor que 10 se mostrarán marcados los campos de interés y tokens extraídos en cada momento)

- Más Rápido: Aumenta la velocidad de traducción.

5. ADVERTENCIAS.

Para obtener una buena traducción de las especificaciones de StateFlow el usuario debe tener en cuenta el Capítulo 6 del manual del Proyecto Sim2Ed. Allí se resumen las condiciones que deben cumplir los archivos fuente.

8. PALABRAS CLAVE

A continuación detallamos una lista de palabras clave para entender el contexto en el que se enmarca este proyecto.

- **Actor** **18**
- **Comportamiento** **5**
- **Evento** **5-6-12**
- **Mensaje** **18**
- **Máquina de Estados** **12-19**
- **Protocolo** **19-20-32**
- **Puerto** **19-31**
- **Tiempo Real** **3**

9. BIBLIOGRAFÍA

Bibliografía consultada durante la realización del proyecto Sim2Ed:

- **MATLAB User's Guide. 1992. The MathWorks, Inc.**
- **Using Simulink. 1999. The MathWorks, Inc.**
- **StateFlow User's Guide. 1999. The MathWorks, Inc.**
- **Object Time User guide. 1998. Object Time Limited.**

10. APÉNDICE – JAVADOC-

Documentación de las clases que componen el sistema Sim2Ed generada mediante la aplicación Javadoc disponible en J Builder.

Se describe cada clase, sus atributos ,los métodos que se ofrece para su manejo así como las relaciones de herencia entre clases.