Sistemas Informáticos
Curso 05-06

Extensión de una herramienta para testear propiedades temporales y probabilísticas

María Cuesta Rubio
Laura García Martín
Estefanía Pérez Jiménez

Dirigido por:
Prof. Manuel Núñez García
Departamento de Sistemas Informáticos y Programación

Facultad de Informática

Universidad Complutense de Madrid

Autorización

Los abajo firmantes, Dª María Cuesta Rubio, Dª Laura García Martín y Dª Estefanía Pérez Jiménez autorizamos 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.

María Cuesta Rubio Laura García Martín Estefanía Pérez Jiménez

Índice

1.- Resumen

El proyecto aquí presentado ha consistido en la extensión de una herramienta para el testeo de sistemas de aspectos cualitativos.

Esta herramienta permite tanto el testeo de especificaciones de propiedades funcionales, como especificaciones de propiedades no funcionales. Para ser más preciso, un ejemplo de propiedad no funcional sería el tiempo.

El proyecto está basado en la implementación de una máquina de estados finita para la comprobación de la conformidad entre especificaciones e implementaciones, y en la ampliación de la misma para permitir, como se ha comentado anteriormente, el testeo de propiedades no funcionales a través del desarrollo de un sistema temporizado por un lado (tiempo fijo), y un sistema con tiempos estocásticos por otro (tiempo variable).

Abstract

The project here mentioned has consisted on the extension of a tool for testing systems of qualitative features.

This tool allows both, testing specifications of functional properties, as well as the test of specifications of non functional properties. To be more precise, an example of non functional property is the time.

The project is based on the implementation of a finite states machine for checking the agreement between specifications and implementations, and its extension to allow, as it was previously mentioned, the testing of non functional properties by means of the development of a timed system on the one hand, and a system with stochastic times on the other.

2.- Palabras clave

Las palabras clave que más fácilmente pueden encontrarse o con mayor frecuencia a lo largo del desarrollo del proyecto son:

- **Especificación**
- **Implementación**
- **Derivación**
- **Test**
- **Conforme**
- **Input**
- **Output**

3.- Desarrollo del proyecto. Contenido

Como ya se ha detallado en el resumen del proyecto, éste ha consistido en la extensión de un sistema para testing de aspectos cuantitativos.

El desarrollo del proyecto ha consistido básicamente en dos partes:

- Ampliación y mejoras de la funcionalidad de la herramienta.
- Creación de una nueva interfaz gráfica, que hiciera más intuitivo y sencillo el uso de la herramienta.

Ambas partes se han desarrollado simultáneamente.

A continuación se detalla el funcionamiento del sistema, incluyendo tanto su funcionalidad anterior, como la explicación correspondiente a las mejoras realizadas.

3.1.- Propiedades funcionales

3.1.1.- Máquina de Mealy. Generalidades

La **estructura general** de una máquina de Mealy, es la siguiente:
Mealy = (S, Ent, Sal, tran, q0), siendo:

- S: conjunto de estados
- Ent: alfabeto de entrada
- Sal: alfabeto de salida
- tran: S x Ent x Sal → S, función de transición
- q0 ∈ S, estado inicial
- *Ent* ∩*Sal* =Ø

y sigue la siguiente **semántica procedimental**:

- Estado inicial q0
- Estado s $\in$ S
- Recibe entrada: i $\in$ Ent y genera salida o $\in$ Sal
- Transita nuevo estado: p = tran(q, i, o)

Para la máquina de Mealy que se ha implementado en este proyecto, y dado el objetivo del mismo, la característica fundamental que se ha seguido como máxima es:

"Una implementación se dice conforme con su especificación si la implementación no muestra outputs (salidas) no esperadas".

Ejemplo 1: Especificación genérica

Sea la siguiente especificación, tras la que la ejecución de una serie de transiciones, se espera obtener como respuesta a la entrada i3 un conjunto definido, tal y como se muestra a continuación:
Especificación: tras i1/o1, i2/o2, i3 / {o4, o5, o6}

La diferencia entre una implementación conforme y una implementación no conforme se muestra a continuación:

- Implementación conforme: tras i1/o1, i2/o2, i3 $\rightarrow$ {o4, o5, o6}
- Implementación no conforme: tras i1/o1, i2/o2, i3 $\rightarrow$ o7

  La salida o7 no está contemplada como una de las posibles outputs al recibir como input i3

Ejemplo 2: Máquina de Coca-Cola

Sea la siguiente especificación para el funcionamiento de una maquina de Coca-cola:

Especificación:

Insertar moneda / oír ruido moneda caer; pulsar botón / iluminar botón; esperar / {sale coca cola, luz producto agotado}

Implementación conforme:

Insertar moneda / oír ruido moneda caer; pulsar botón /iluminar botón; esperar / sale coca cola

El resultado obtenido en la implementación se encontraba entre los posibles de la especificación.

Implementación no conforme: tendríamos, que tras la

Insertar moneda / oír ruido moneda caer; pulsar botón /iluminar botón; esperar / sale fanta de naranja como se muestra a continuación:

3.1.2.- Proceso de testeo. Derivación de tests

Para lograr el objetivo de este proyecto fue necesario desarrollar un proceso de testeo sobre especificaciones. Para ello se realizaba un recorrido por el esquema (máquina de Mealy) obteniendo todos los posibles tests de la especificación.

Ahora bien, un test no es más que un conjunto de entradas y salidas, así como un conjunto de transiciones y salidas esperadas.

Los tests tienen como origen una especificación, es decir, son derivados a partir de una especificación y suobjetivo es la comprobación de la conformidad de la implementación respecto de la especificación. Esta comprobación se lleva a cabo mediante la aplicación de los tests derivados a las implementaciones.

Para realizar este proceso de derivación, ha sido necesario implementar el siguiente algoritmo de derivación:

Input: M = (S, I, O, δ, sin).

Output: T = (S', I, O, δ', s_0, S_I, S_O, S_F , S_P , ζ).

Initialization:

– S' := {s_0}, δ' := S_I := S_O := S_F := S_P := ζ := .

– S_{aux} := {(s_{in}, ξ_0, s_0)}.

Inductive Cases: Apply one of the following two possibilities until Saux = ∅ .

1. If (s^M, ξ, s^T.) ∈ S_{aux} **then** perform the following steps:

(a) S_{aux} := S_{aux} − {(s^M, ξ, s^T)}.

(b) S_P := S_P ∪ {s^T }; ζ(s^T) := ξ.

2. If $S_{aux} = \{(s^M, \xi, s^T)\}$ is a unitary set **and** there exists $i \in I$ such that $out(s^M, i) \neq \emptyset$ **then** perform the following steps:

(a) $S_{aux} := \emptyset$.

(b) Choose i such that $out(s^M, i) \neq \emptyset$.

(c) Create a fresh state $s' \notin S'$ and perform $S' := S' \cup \{s'\}$.

(d) $S_I := S_I \cup \{s^T\}$; $S_O := S_O \cup \{s'\}$; $\delta' := \delta' \cup \{(s^T, i, s')\}$.

(e) For each $o \notin out(s^M, i)$ do

– Create a fresh state $s'' \notin S'$ and perform $S' := S' \cup \{s''\}$.

– $S_F := S_F \cup \{s''\}$; $\delta' := \delta' \cup \{(s', o, s'')\}$.

(f) For each $o \in out(s^M, i)$ do

– Create a fresh state $s'' \notin S'$ and perform $S' := S' \cup \{s''\}$.

– $\delta' := \delta' \cup \{(s', o, s'')\}$.

– $s_1^M := after(s^M, i, o)$.

– Let $(s^M, i, o, \xi_1, s_1^M) \in \delta$. $S_{aux} := S_{aux} \cup \{(s_1^M, \xi + \xi_1, s'')\}$.

3.2.- Propiedades no funcionales

3.2.1.- Sistema temporizado

Este sistema está basado en la máquina de Mealy expuesta anteriormente, es decir, en un sistema de propiedades funcionales, añadiéndole aspectos no funcionales a las transiciones definidas como puede ser el tiempo. En este caso se trata de un tiempo fijo introducido por el usuario.

Ahora bien, al añadir una propiedad no funcional a la especificación:

"Se dice que una implementación es conforme a una especificación no sólo si la implementación hace lo que debe, si no que lo hace en el tiempo que debe".

Por lo que nos encontramos con la obtención de éxitos ligados a tiempos y a la obtención de los valores esperados.

Ejemplo: Máquina de Coca-Cola

Sea la siguiente especificación para el funcionamiento de una maquina de Coca-cola:

Especificación:

Insertar moneda / oír ruido moneda caer **(2seg)**; pulsar botón / iluminar botón **(1 seg)**; esperar / {sale coca cola **(3seg)**, luz producto agotado **(1 seg)**}

Total: sale coca cola **(6 seg) ;** luz producto agotado **(4 seg)**

Implementación conforme:

Insertar moneda / oír ruido moneda caer; pulsar botón / iluminar botón; esperar / sale coca cola **(≤ 6 segundos)**

Implementación no conforme:

Insertar moneda / oír ruido moneda caer; pulsar botón / iluminar botón; esperar / {sale Fanta de naranja , sale coca cola **(>6 seg)**, luz producto agotado **(>4 seg)**}

2 seg **1 segundo** **3 seg** **> 6 seg**

3.2.2.- Sistema con tiempos estocásticos

Este sistema está basado, como el caso anterior, en la máquina de Mealy expuesta previamente, es decir, en un sistema de propiedades funcionales añadiéndole como propiedad no funcional tiempos variables definidos u obtenidos mediante la aplicación de distintas funciones de distribución.

Es decir, en este caso el atributo tiempo no es una propiedad que se pueda determinar a priori y de manera fija, sino que dependerá de una variable aleatoria pudiendo darse el caso (de hecho es lo normal) que un mismo test, tarde distintos tiempos.

De esta manera, para poder determinar si una implementación es conforme con la especificaciónes necesaria la aplicación reiterada de los tests sobre dicha implementación, así como la creación de contrastes de hipótesis que sirvan de base para la determinación de la conformidad.

Por tanto nos encontramos con que:

"Se dice que una implementación es conforme a una especificación no sólo si la implementación hace lo que debe, sino si además:
El contraste de hipótesis determina que los resultados obtenidos son lo suficientemente similares a los esperados por la función de distribución".

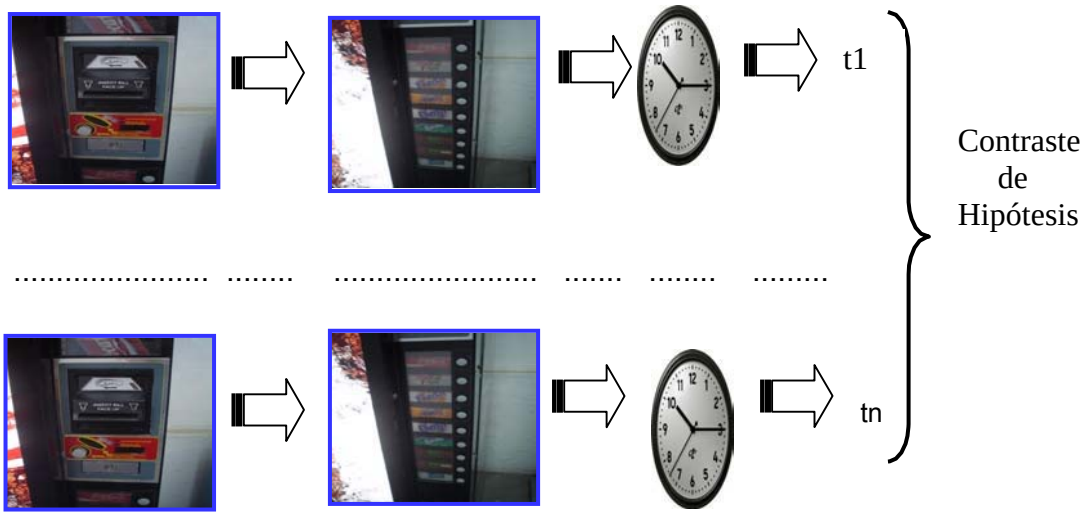
Ejemplo: Máquina de Coca-Cola

Sea la siguiente especificación para el funcionamiento de una maquina de Coca-cola:

Especificación:

Insertar moneda / oír ruido moneda caer **(x seg)**; pulsar botón / iluminar botón **(y seg)**; esperar / {sale coca cola **(z1 seg)**, luz producto agotado **(z2 seg)**}

Y en cuanto a los tests, tenemos:



El grado de confianza es un factor que puede determinar el usuario. Para ello hemos implementado tres tipos de funciones de distribución:

- Uniforme
- Dirac
- Exponencial

4.- Estructura de la implementación

La herramienta que se ha implementado como proyecto y que aquí se expone, es una aplicación Java con su correspondiente estructura de clases.

En este apartado se pretende reflejar dicha estructura con el objetivo de que posibles programadores que deseen ampliar el sistema dispongan de la información detallada para tal fin.

No obstante, es importante resaltar que no se pretende exponer aquí el código implementado o la explicación del mismo puesto que para tal finalidad está disponible el Javadoc generado por la aplicación.

4.1.- Estructura general

La estructura general de la aplicación consta de una estructura de paquetes como a continuación se muestra y con la siguiente información:

- **Paquete Maquina_Mealy.Estado**: contiene la implementación referida a los estados que constituyen las especificaciones y las implementaciones representadas como máquina de Mealy

- **Paquete Maquina_Mealy.Functions**: contiene la implementación referida a las distintas funciones de distribución consideradas en la aplicación usadas para los sistemas con tiempos estocásticos.

- **Paquete Maquina_Mealy.GUI**: contiene la implementación de todas las componentes del interfaz de usuario utilizadas por la aplicación.

- **Paquete Maquina_Mealy.Mealy**: contiene el código de las especificaciones y de las implementaciones expresadas como máquina de Mealy.

❖ **Paquete Maquina_Mealy.Tests**: contiene la implementación referida a los tests resultantes de la derivación de las especificaciones.

❖ **Paquete Maquina_Mealy.Transicion**: contiene la implementación referida a las transiciones de la maquina Mealy para las especificaciones e implementaciones.

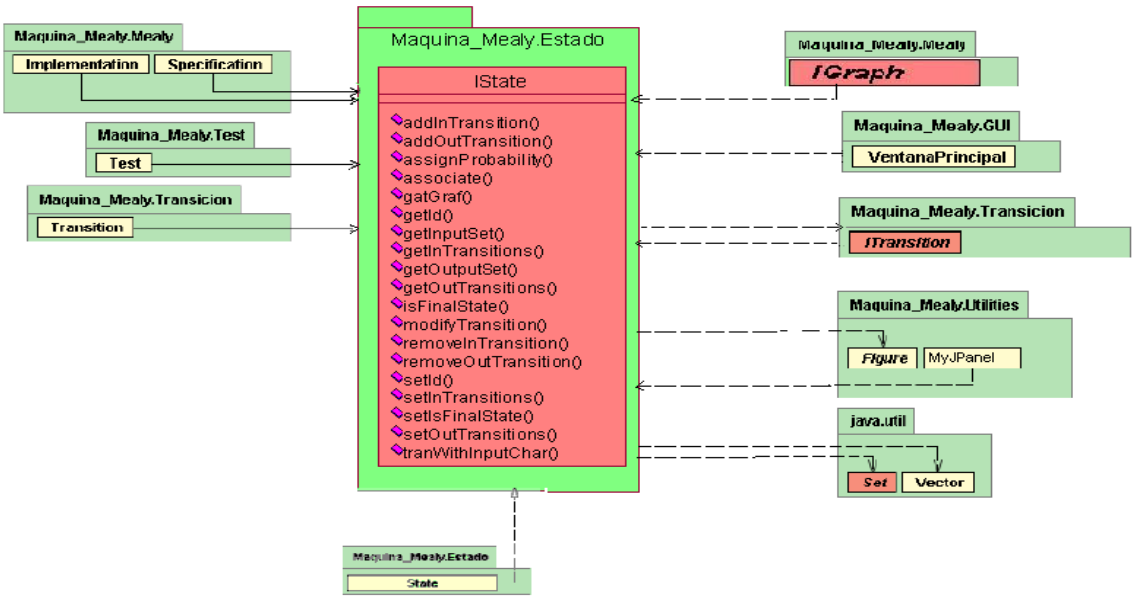
❖ **Paquete Maquina_Mealy.Utilities**: contiene la implementación de distintas utilidades necesarias para aportar la funcionalidad adecuada a la herramienta desarrollada.

4.1.1.- Paquete Maquina_Mealy.Estado

Este paquete está constituido por un interfaz "IState.java" y una clase "State.java" que lo implementa. Su principal función es la declaración de los métodos relacionados no sólo con la definición de las características propias de un estado, tales como, la declaración de los accesores y mutadores de sus atributos, si no de aquellos métodos requeridos para la definición de las transiciones asociadas al mismo.

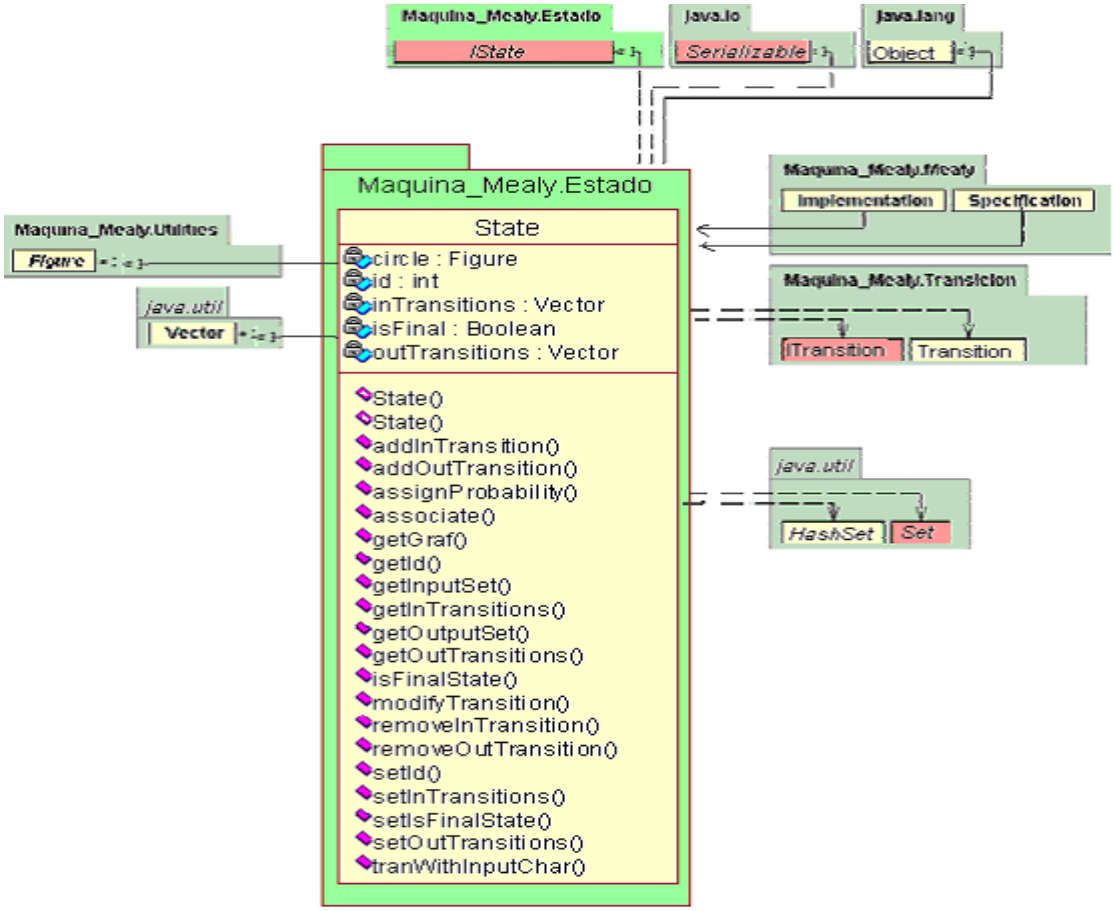
Se muestra a continuación los diagramas UML correspondientes a ambos elementos:

Interfaz "IState.java"



Clase "State.java"

Implementa los métodos declarados en su interfaz.

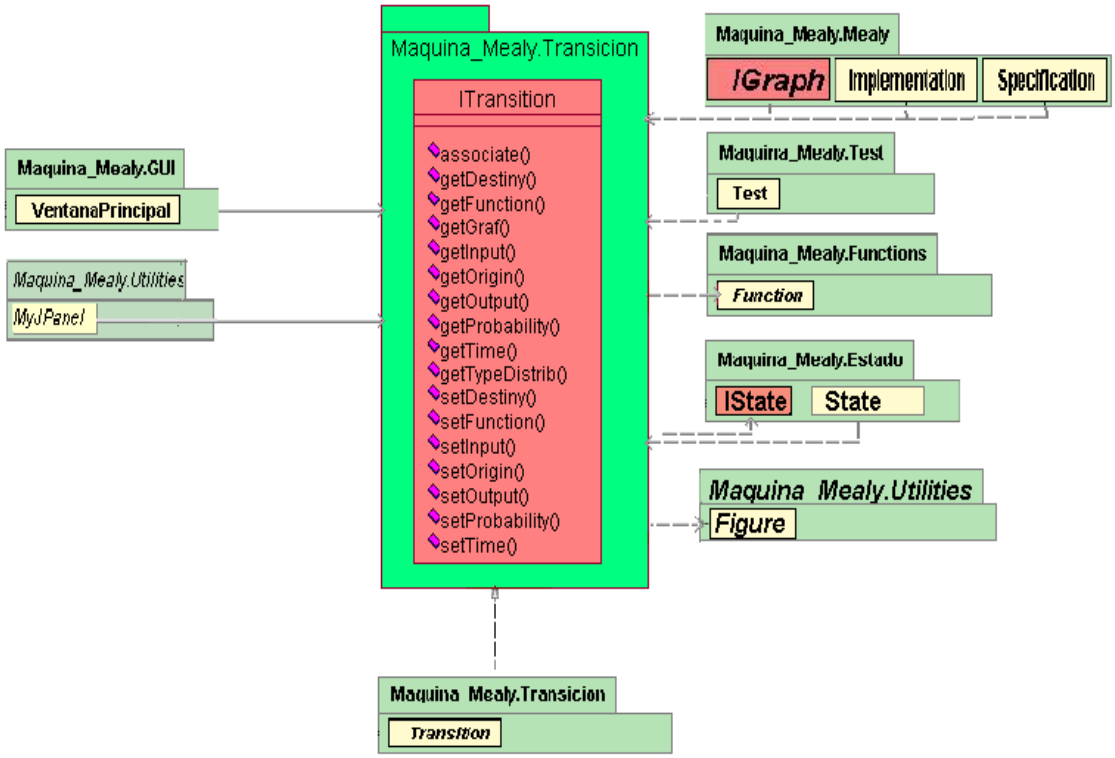


4.1.2.- Paquete Maquina_Mealy.Transicion

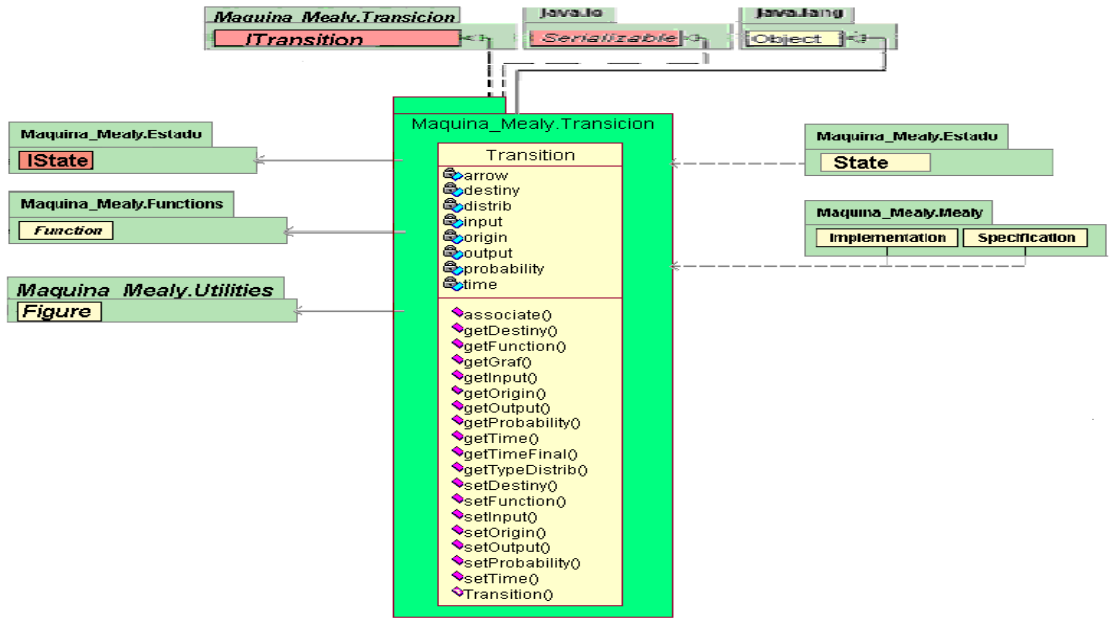
Este paquete está constituido por un interfaz "ITransition.java" y una clase "Transition.java" que lo implementa. La misión de estas clases es definir las transiciones entre estados en la máquina Mealy. Además estas transiciones llevarán asociada la probabilidad de la transición entre los estados que comunica, así como los datos referentes a tiempo, en caso de que la implementación o especificación se definan con propiedades no funcionales.

Se muestra a continuación los diagramas UML correspondientes a ambos elementos:

Interfaz “ITransition.java”



Clase “Transition.java”



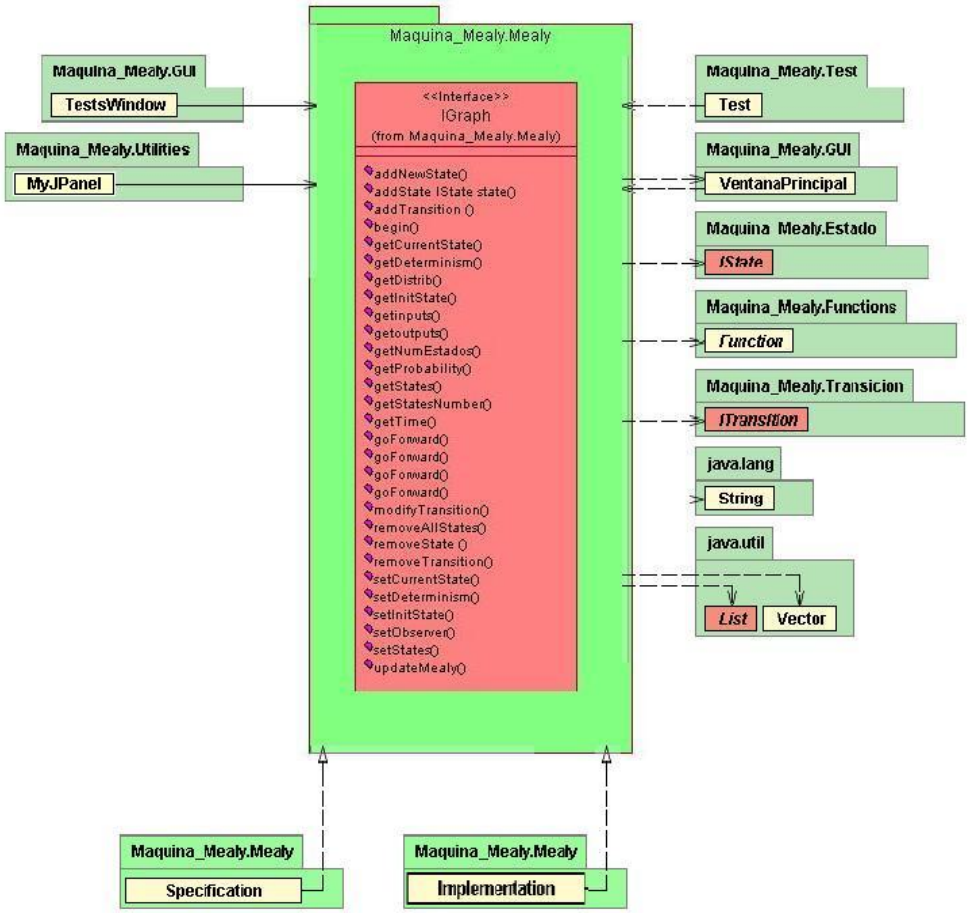
Implementa los métodos declarados en su correspondiente interfaz.

4.1.3.- Paquete Maquina_Mealy.Mealy

Este paquete está constituido por la interfaz “IGraph.java”, así como por las dos clases que la implementan, “Specification.java” e “Implementation.java”.

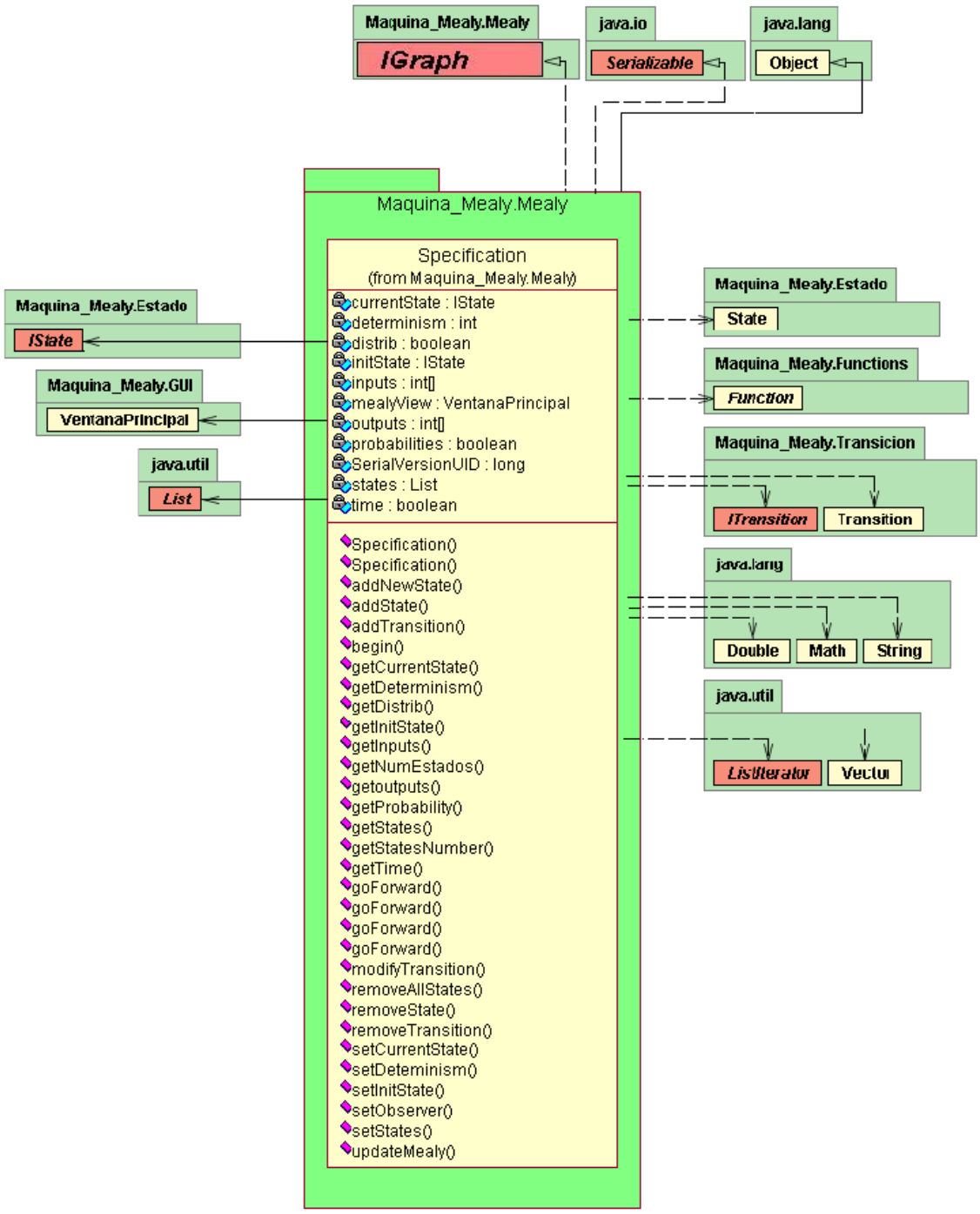
Se muestra a continuación los diagramas UML correspondientes a todos los elementos:

Interfaz “IGraph.java”



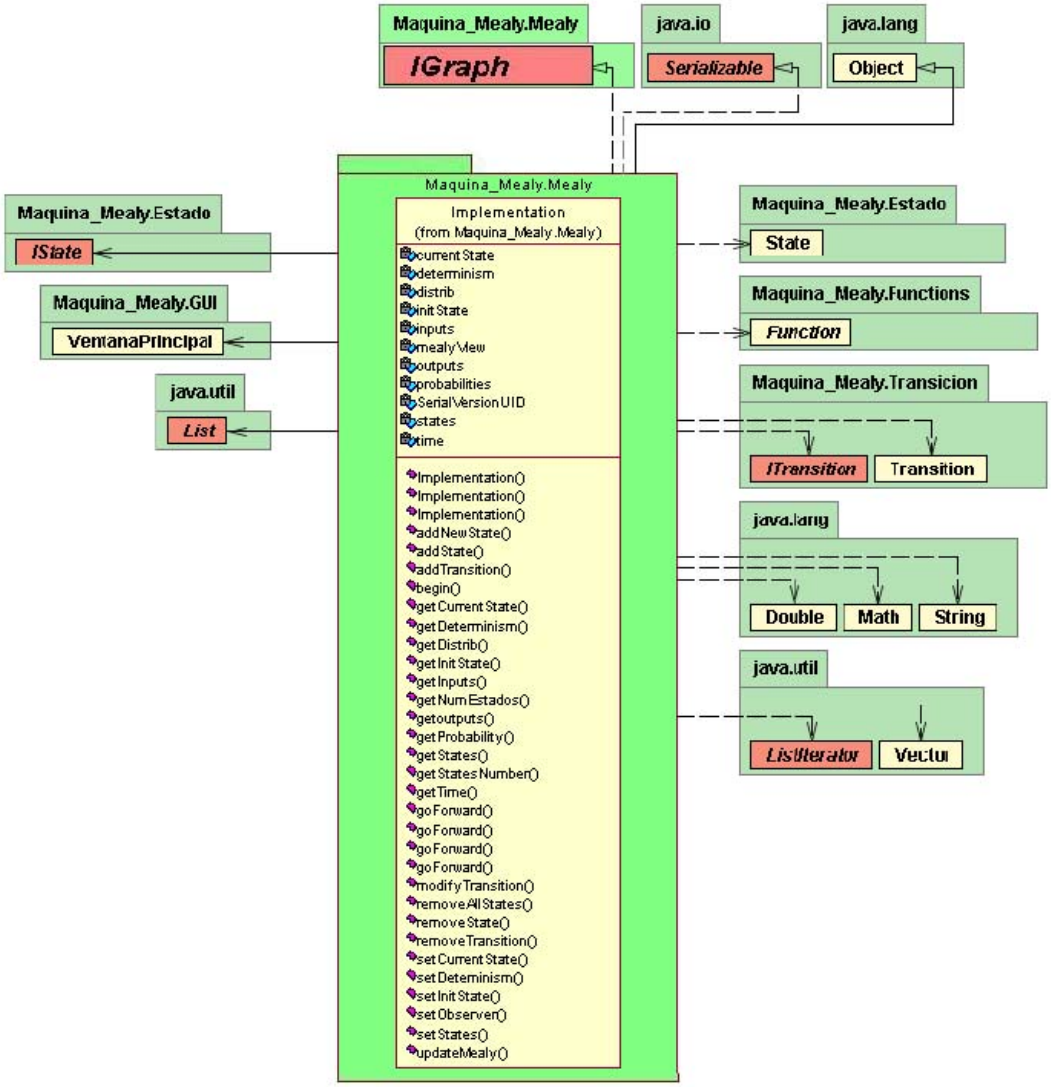
Clase “Specification.java”

Implementa el Interfaz IGraph para la realización de las especificaciones



Clase “Implementation.java”

Implementa el Interfaz IGraph para la realización de las implementaciones



4.1.4.- Paquete Maquina_Mealy.Function

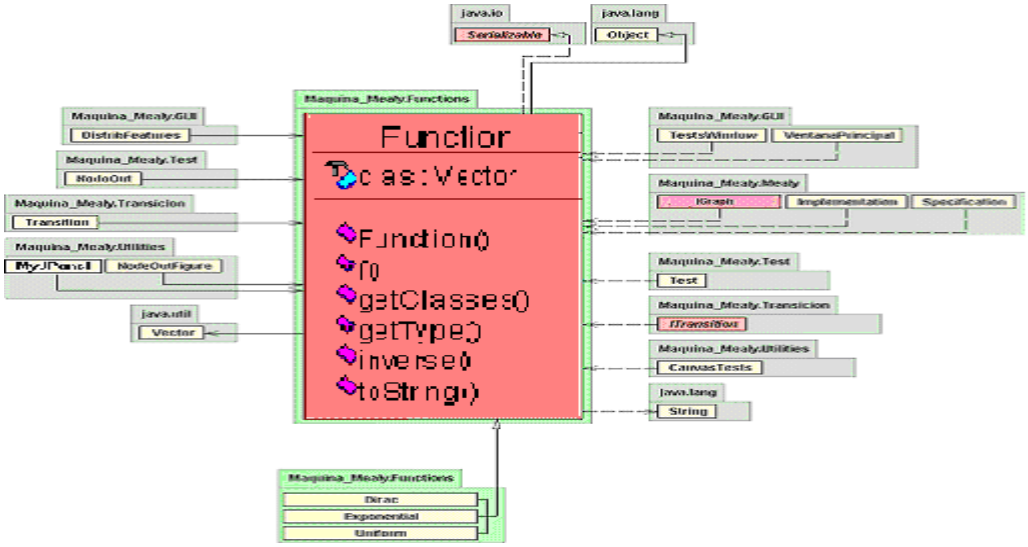
Este paquete está constituido por cuatro clases Java que implementan las funciones de distribución desarrolladas para la creación de sistemas con tiempos estocásticos. Es uno de los paquetes con mayor interés en cuanto a posibles ampliaciones futuras, pues será aquí donde nuevas funciones de distribución deberían ser implementadas debido a la existencia, como se mostrará a continuación, de una clase perteneciente a este paquete que servirá como plantillas para dichas adiciones.

Estas clases representan las funciones necesarias para aportar a las especificaciones e implementaciones una propiedad no funcional como es el tiempo de una manera variable, es decir, dicho tiempo se calculará en función de la aplicación de una función u otra.

Mostramos a continuación los diagramas UML de las clases pertenecientes a este paquete.

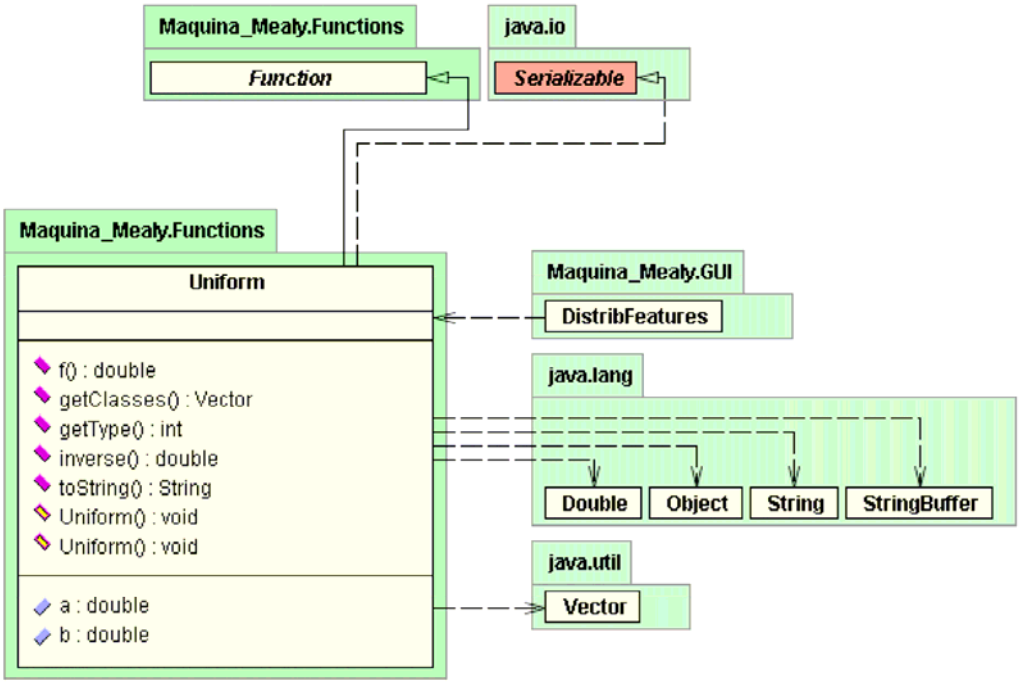
Clase “Function.java”

Esta es una clase abstracta que sirve como plantilla para la creación de clases que deriven de ella y que representen distintas funciones de distribución que puedan ser aplicadas a especificaciones e implementaciones con propiedades no funcionales como ocurre en los sistemas con tiempos estocásticos que aquí se desarrollan.



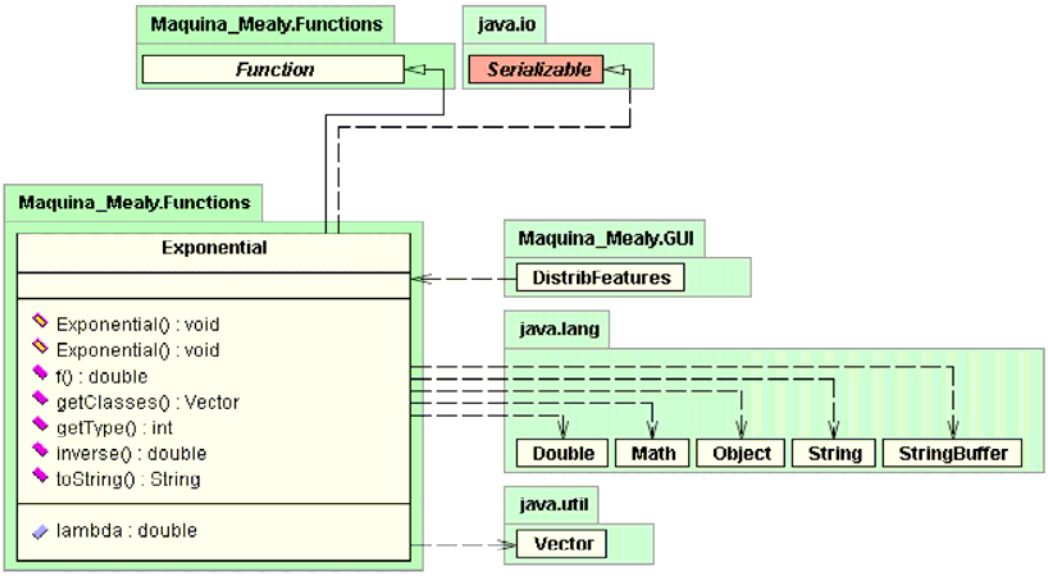
Clase “Uniform.java”

Define la función uniforme.



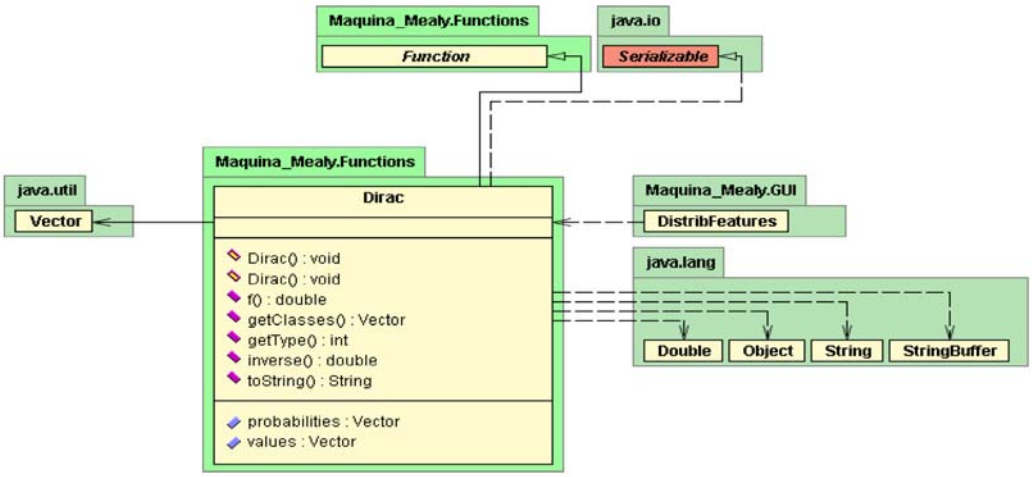
Clase “Exponential.java”

Define la función exponencial.



Clase "Dirac.java"

Define la función Dirac o función de saltos.



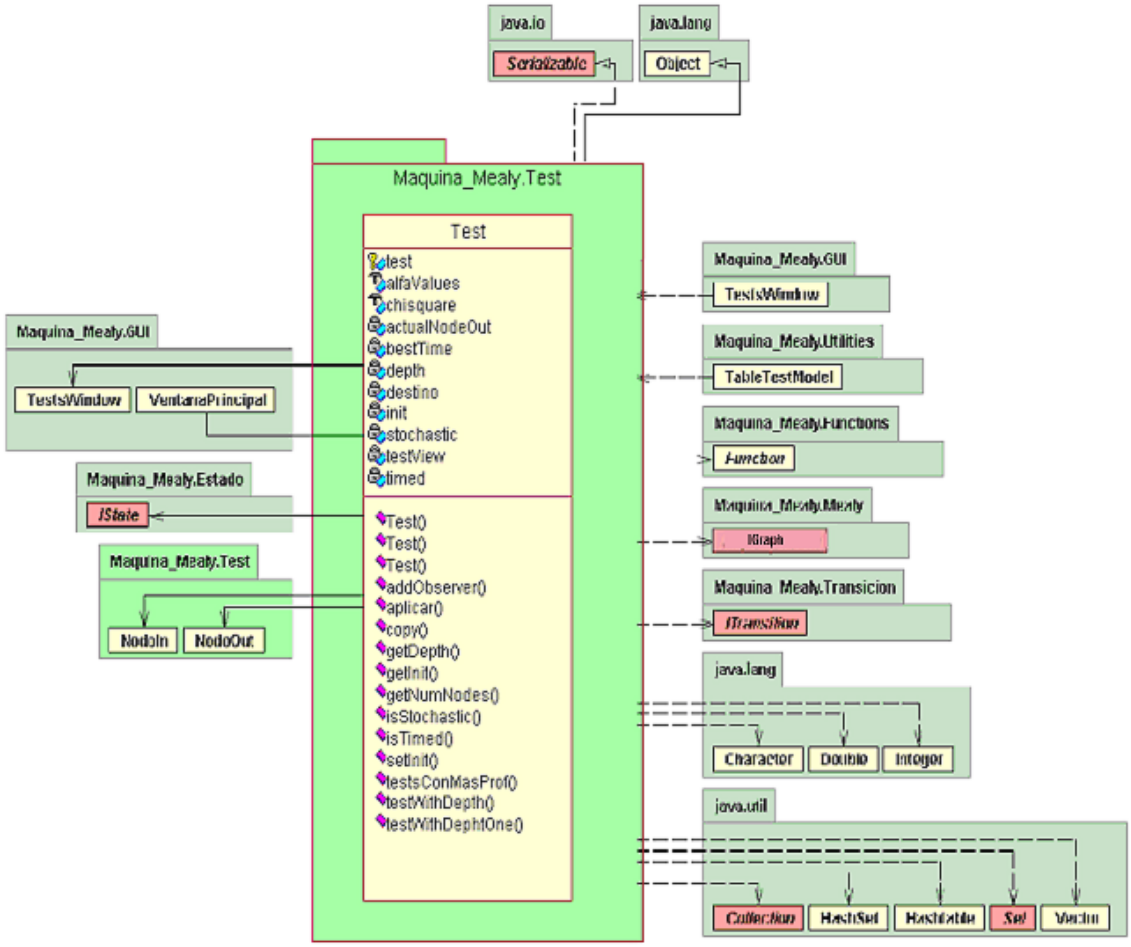
4.1.5.- Paquete Maquina_Mealy.Tests

Este paquete está constituido por las clases mediante las cuales se ha implementado los tests resultantes de la derivación de las especificaciones.

A continuación se muestra los diagramas UML correspondientes a dichas clases.

Clase "Test.java"

En esta clase se encuentran los métodos necesarios para la representación de los tests resultado de la derivación. Es en esta clase donde se realiza la derivación, es decir, donde se lleva a cabo el algoritmo de derivación seguido, para que partiendo de una especificación previamente creada se construyan los tests que serán aplicados a continuación a la implementación correspondiente.

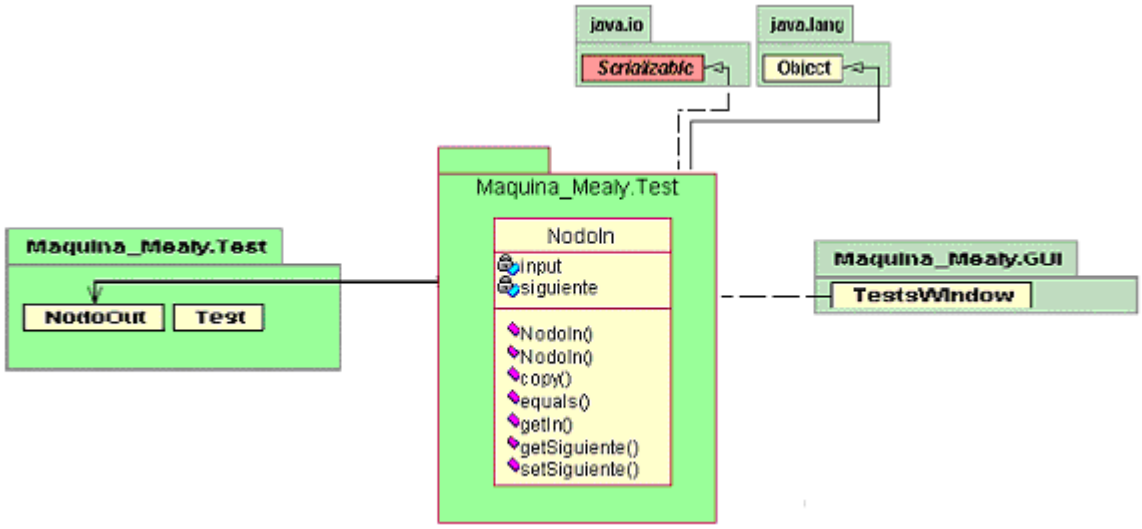


Este algoritmo se basa en el método *"testWithDepth"* que comienza creando todos los test posibles de profundidad uno mediante el método "*testWithDepthOne*", tras el cual se crean todos los tests de distintas profundidades hasta llegar a la profundidad pedida por el usuario

Podemos destacar que en los casos de especificaciones no deterministas se podrá observar en los tests todas las posibles outputs ante una determinada input

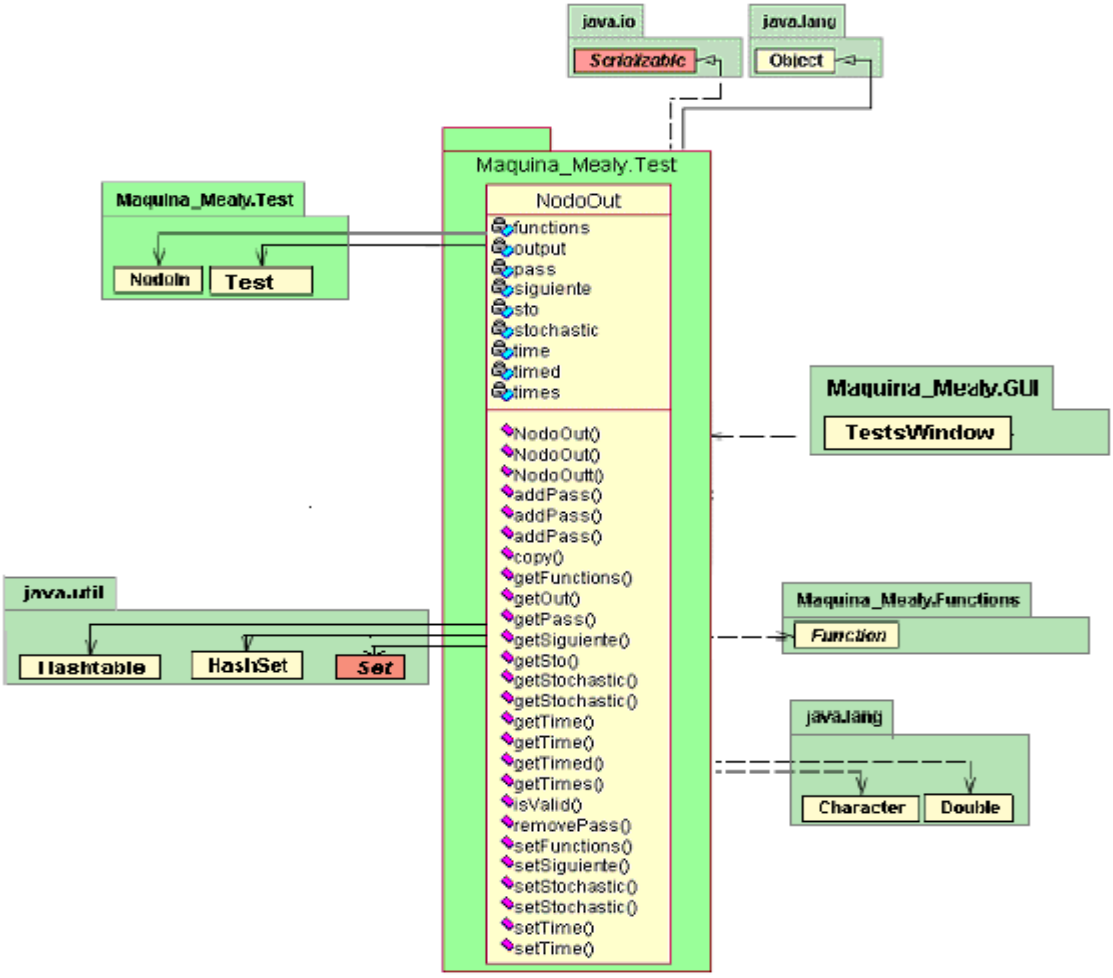
Clase "NodoIn.java"

Esta clase representa un nodo de entrada del test, es decir, una entrada de una transición de la especificación.



Clase "NodoOut.java"

Esta clase representa un nodo de salida del test.



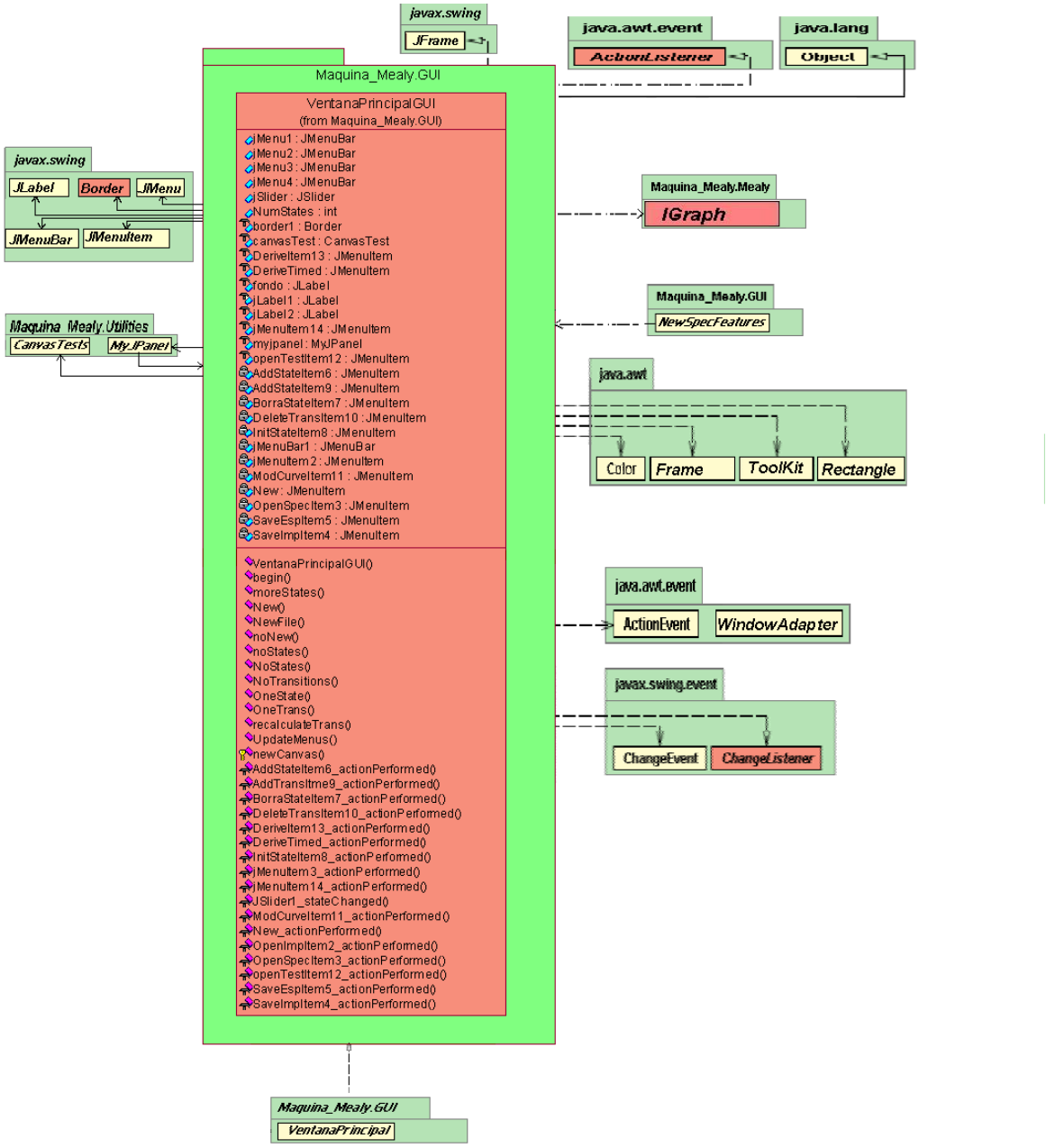
4.1.6.- Paquete Maquina_Mealy.GUI

Este paquete está constituido por todas aquellas clases que representan interfaces gráficas de usuario para la interacción con el mismo.

Se muestran a continuación los diagramas UML asociados a dichas clases.

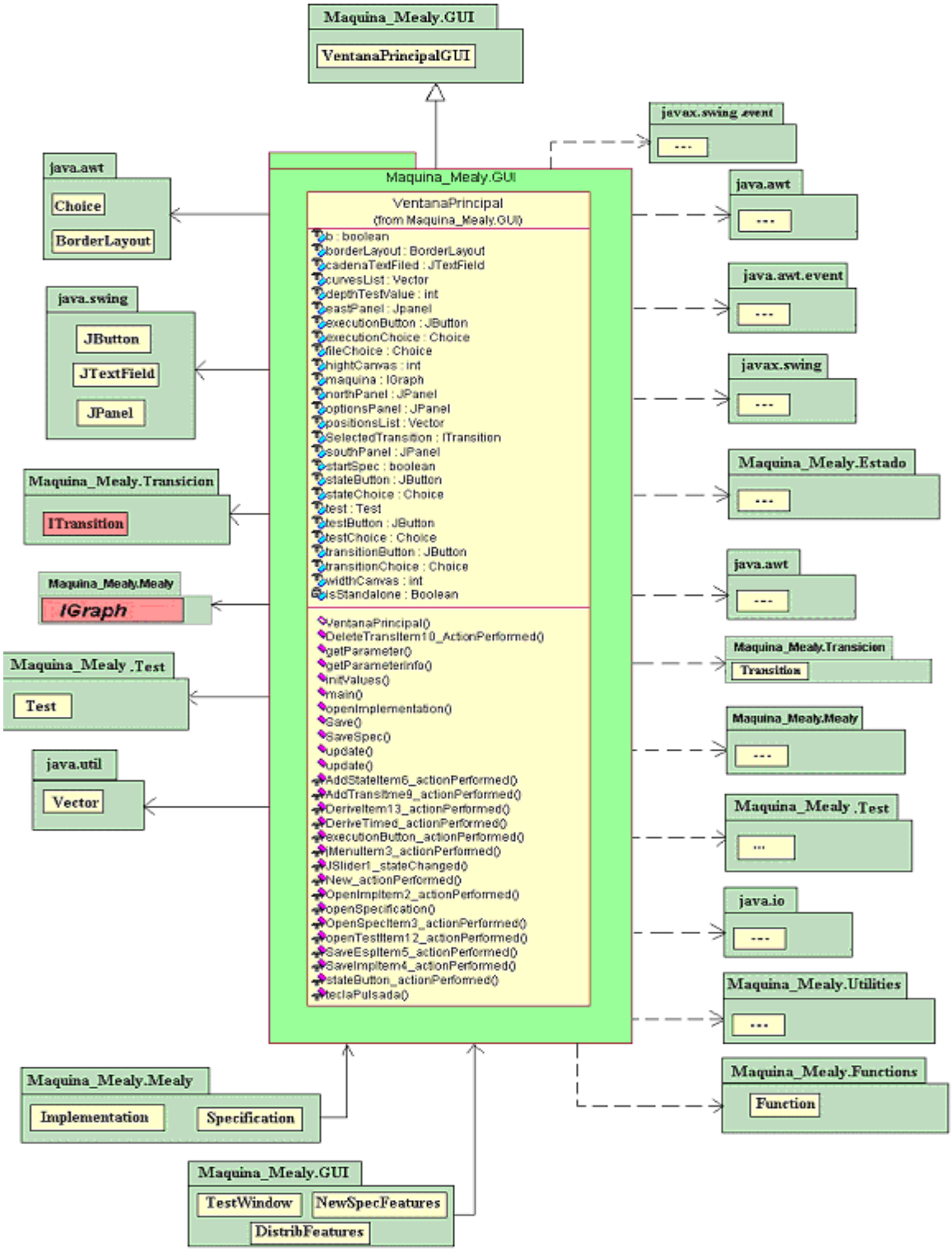
Clase "VentanaPrincipalGUI.java"

Es la interfaz principal de la aplicación y el punto de entrada a la misma. Es desde esta ventana desde donde se tiene acceso a todas las opciones que la herramienta soporta.



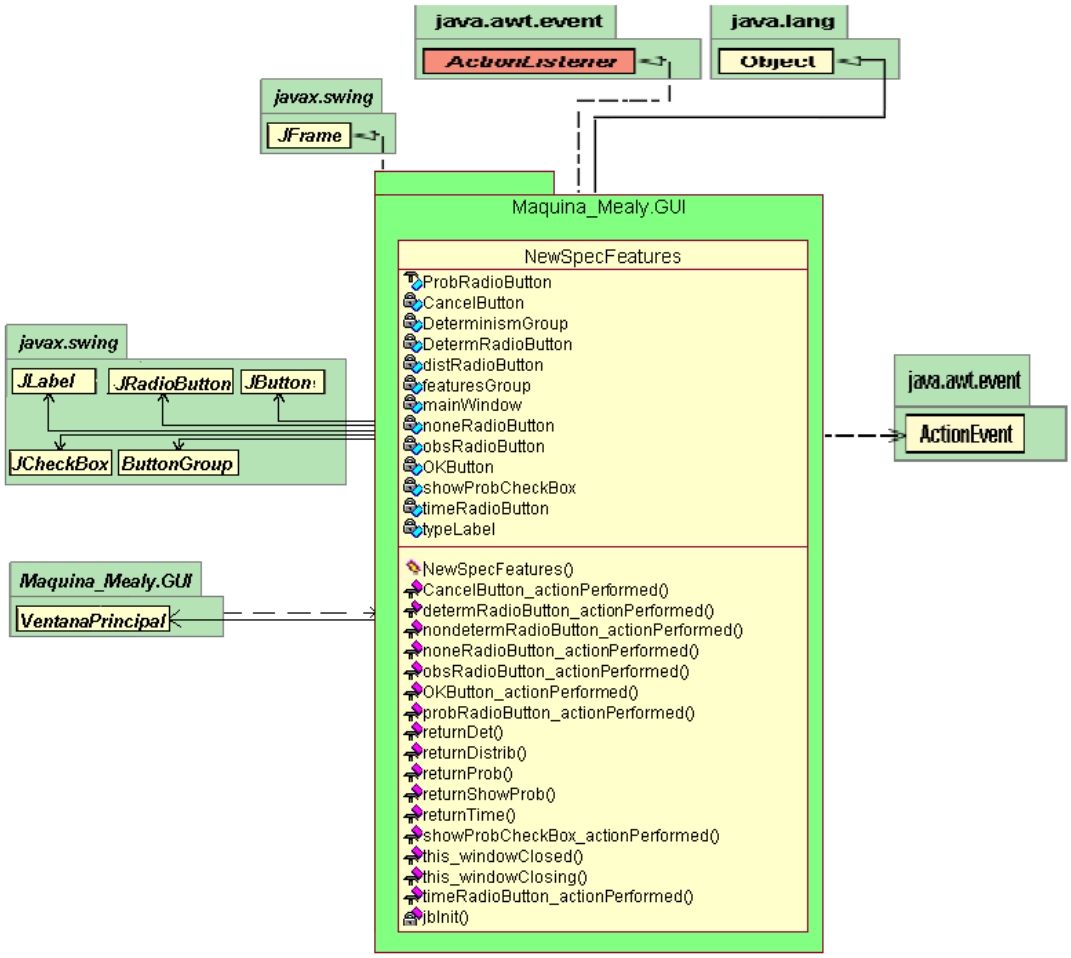
Clase “VentanaPrincipal.java”

Implementa la interfaz anterior.



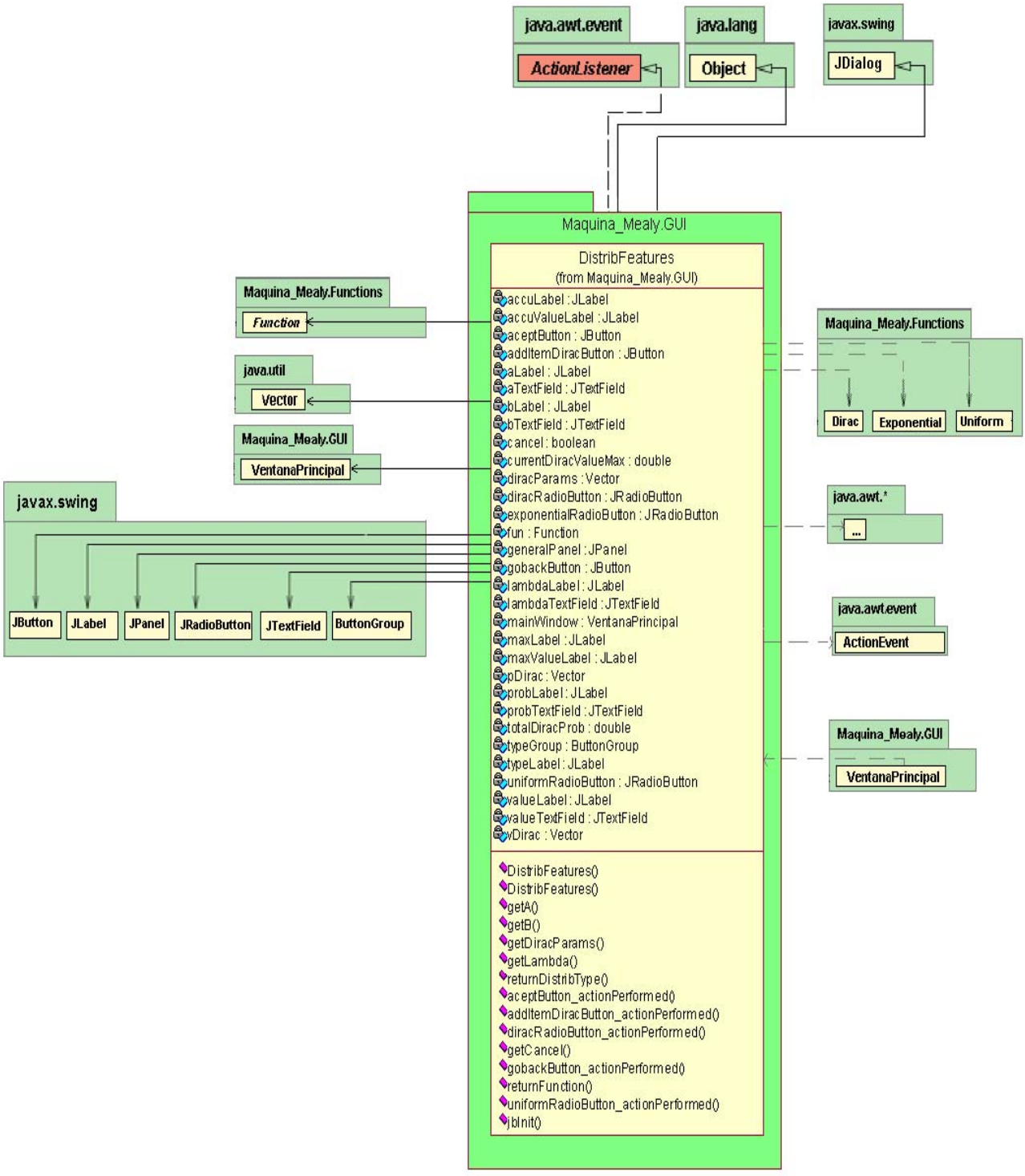
Clase "NewSpecFeatures.java"

Esta clase es la que implementa la interfaz para la introducción por parte del usuario del tipo de especificación o implementación que va a ser creada así como las características de la misma.



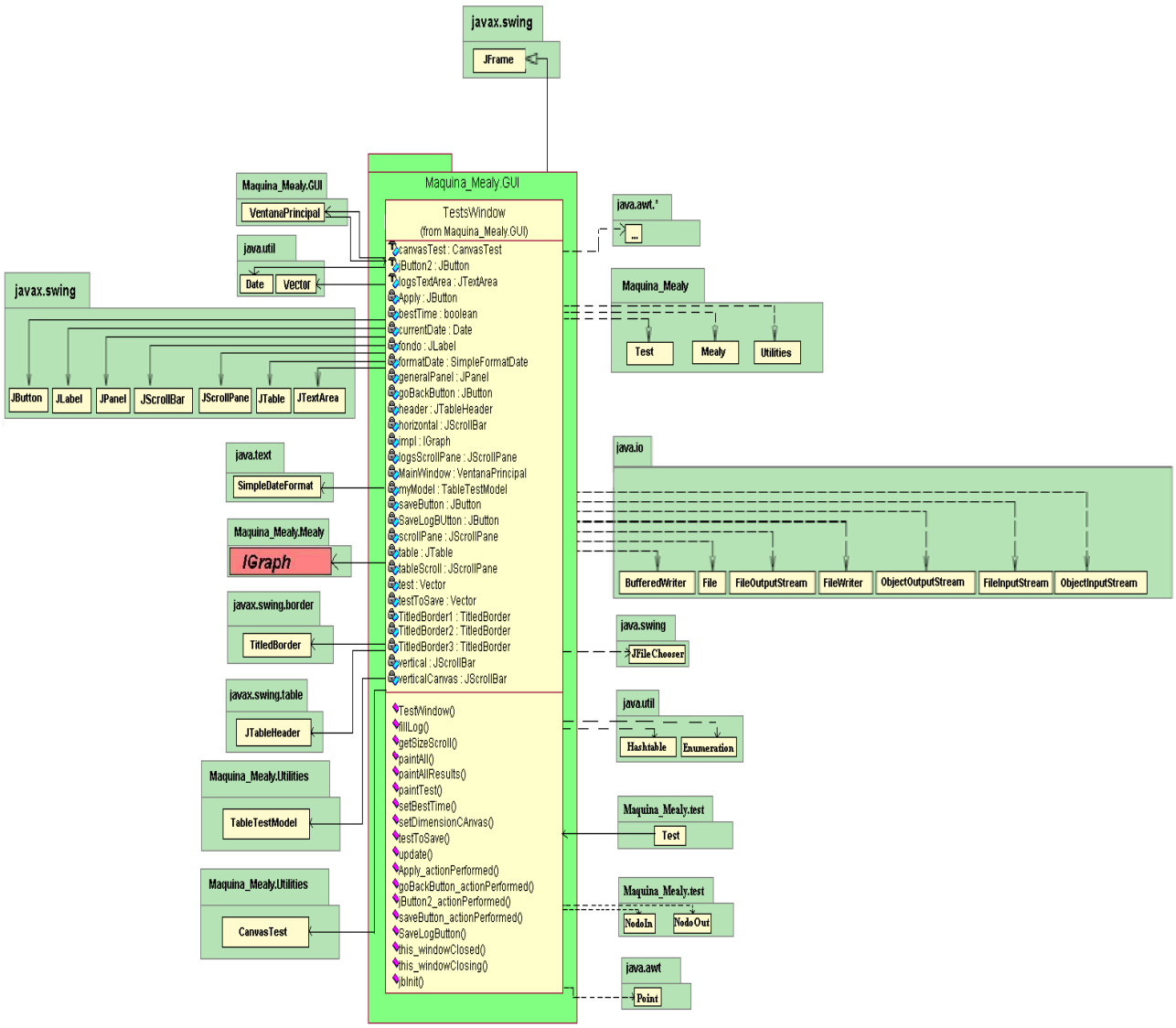
Clase "DistribFeatures.java"

Esta es la clase en la que se implementa la interfaz para que el usuario seleccione la función de distribución que va a utilizar para el cálculo del tiempo de las transiciones correspondientes a la máquina Mealy que vaya a crear. Será utilizada en aquellos casos sen los que se haya decidido diseñar un sistema con tiempos estocásticos.



Clase “TestsWindow.java”

En esta clase está implementada la interfaz que representa la visualización de los tests generados tras la derivación. Es en esta interfaz donde se ofrece al usuario, la posibilidad de aplicar los tests a una implementación para la comprobación de la conformidad de ésta última con la especificación derivada.



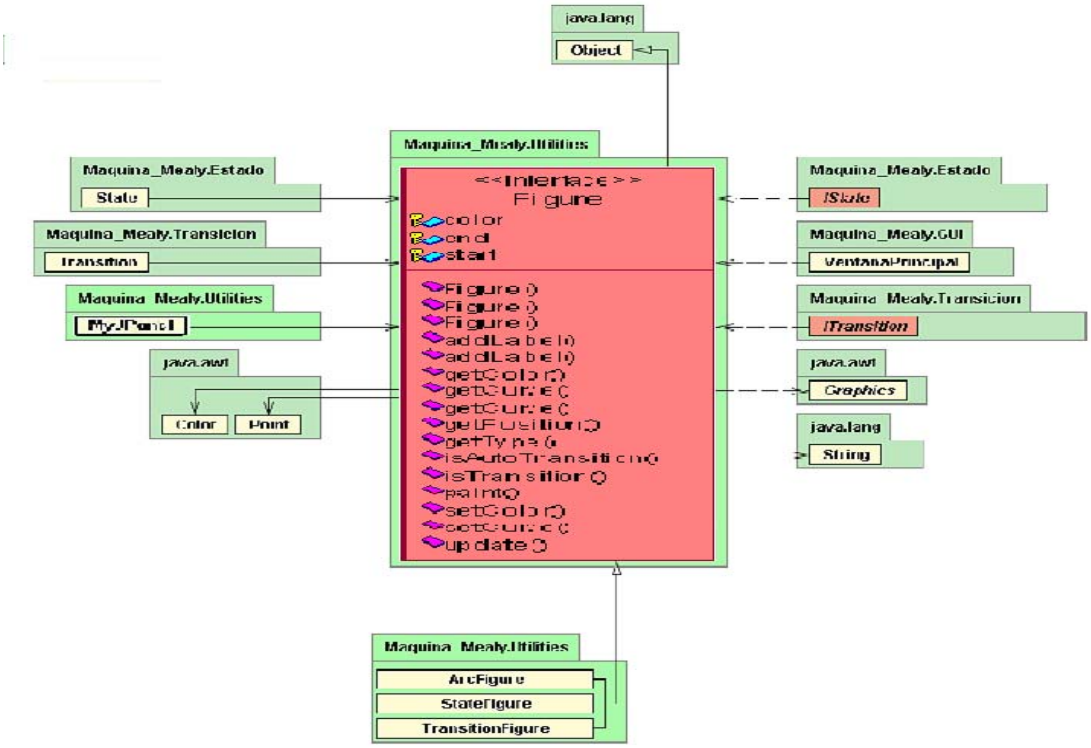
4.1.7.- Paquete Maquina_Mealy.Utilities

Este paquete está integrado por una serie de clases necesarias para la representación gráfica de los elementos de los que consta la herramienta, es decir, para la representación gráfica tanto de las máquinas de Mealy para las especificaciones e implementaciones como de los tests. Así mismo se ha implementado en este paquete clases auxiliares utilizadas por la herramienta para su correcto funcionamiento.

Se muestran a continuación los diagramas UML asociados a dichas clases.

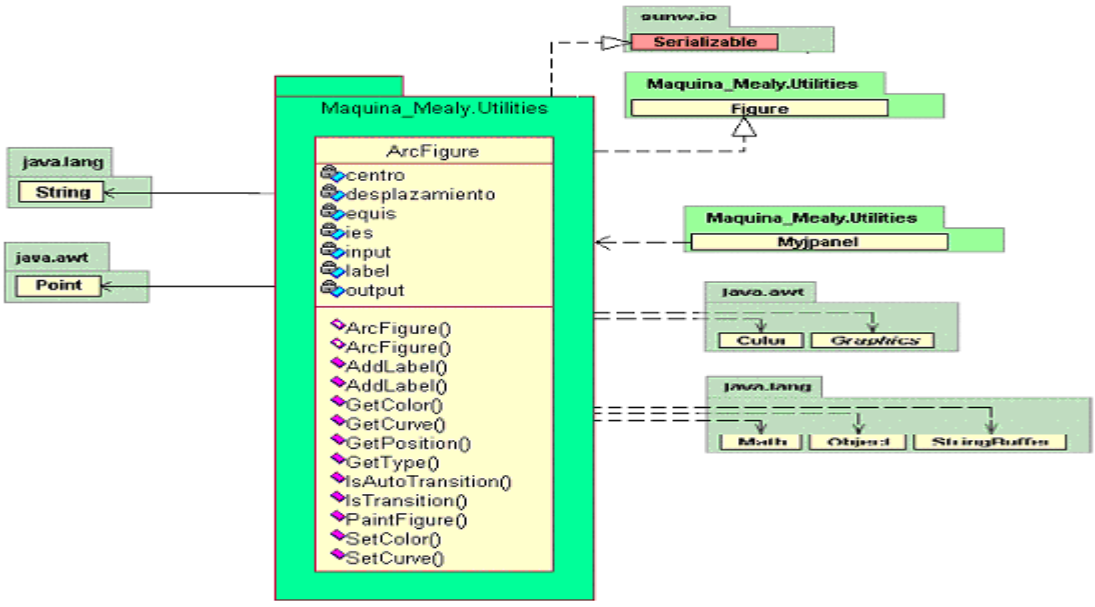
Clase “Figure.java”

Esta clase es una clase abstracta que sirve como plantilla para los distintos tipos de figuras que se van a dibujar en la aplicación para la creación de esquemas (especificación e implementación) , concretamente, para la creación de figuras que representen: estados, transiciones y auto transiciones.



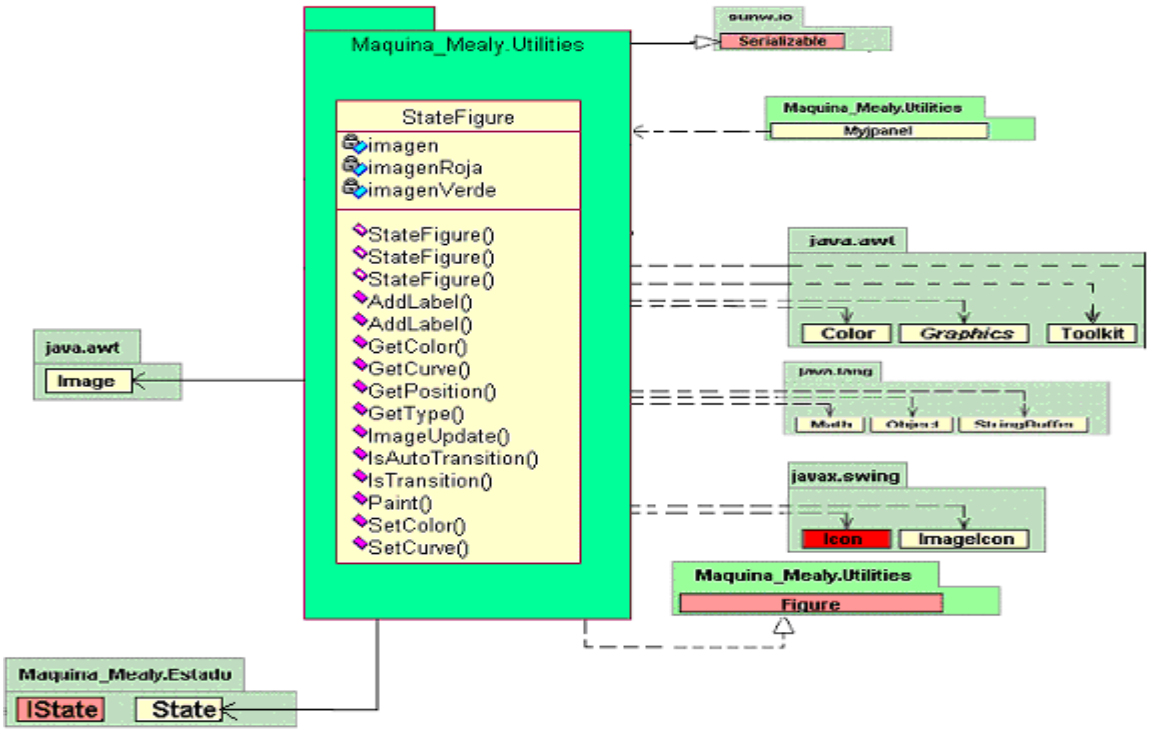
Clase “ArcFigure.java”

Representa gráficamente una auto-transición



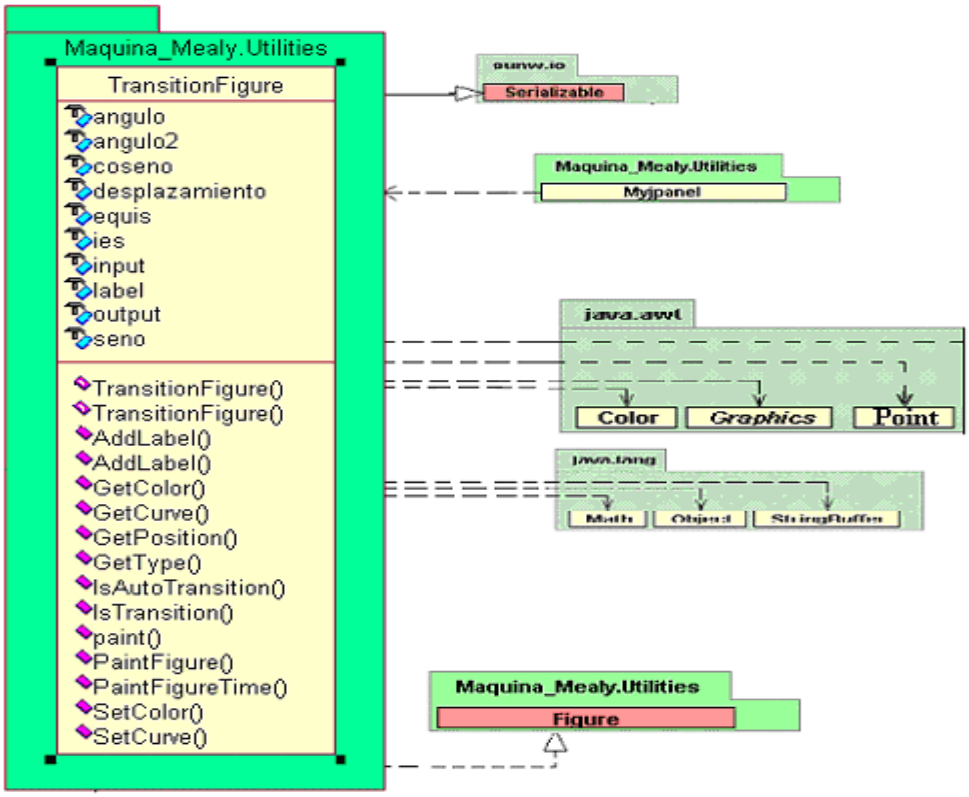
Clase “StateFigure.java”:

Representación gráfica para los estados



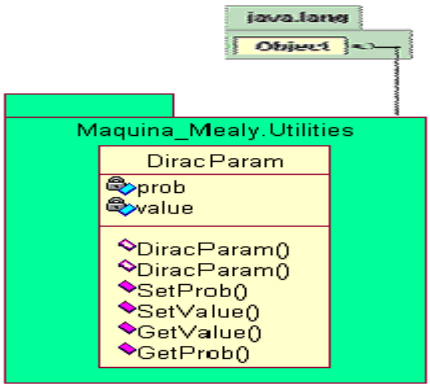
Clase “TransitionFigure.java”

Clase implementada para la representación gráfica de las transiciones.



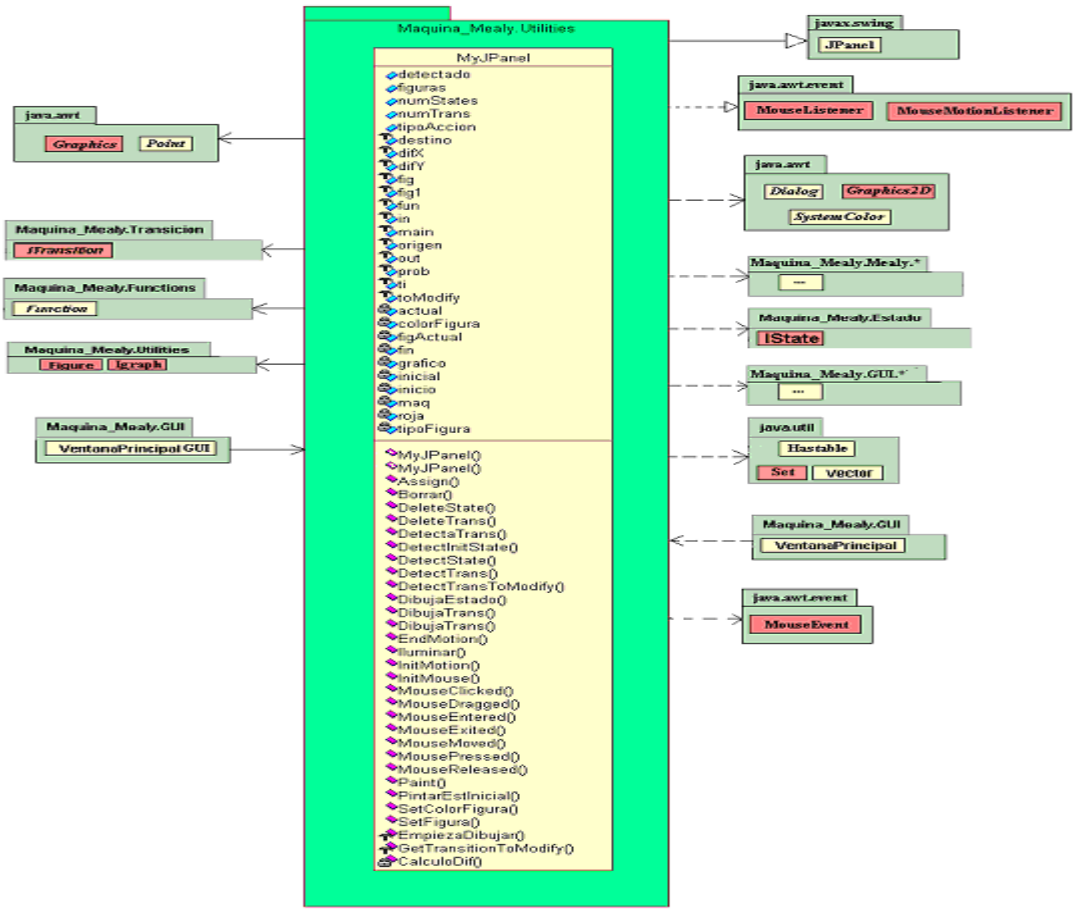
Clase "DiracParam.java"

Clase implementada para la manipulación de los parámetros de la función de distribución Dirac.



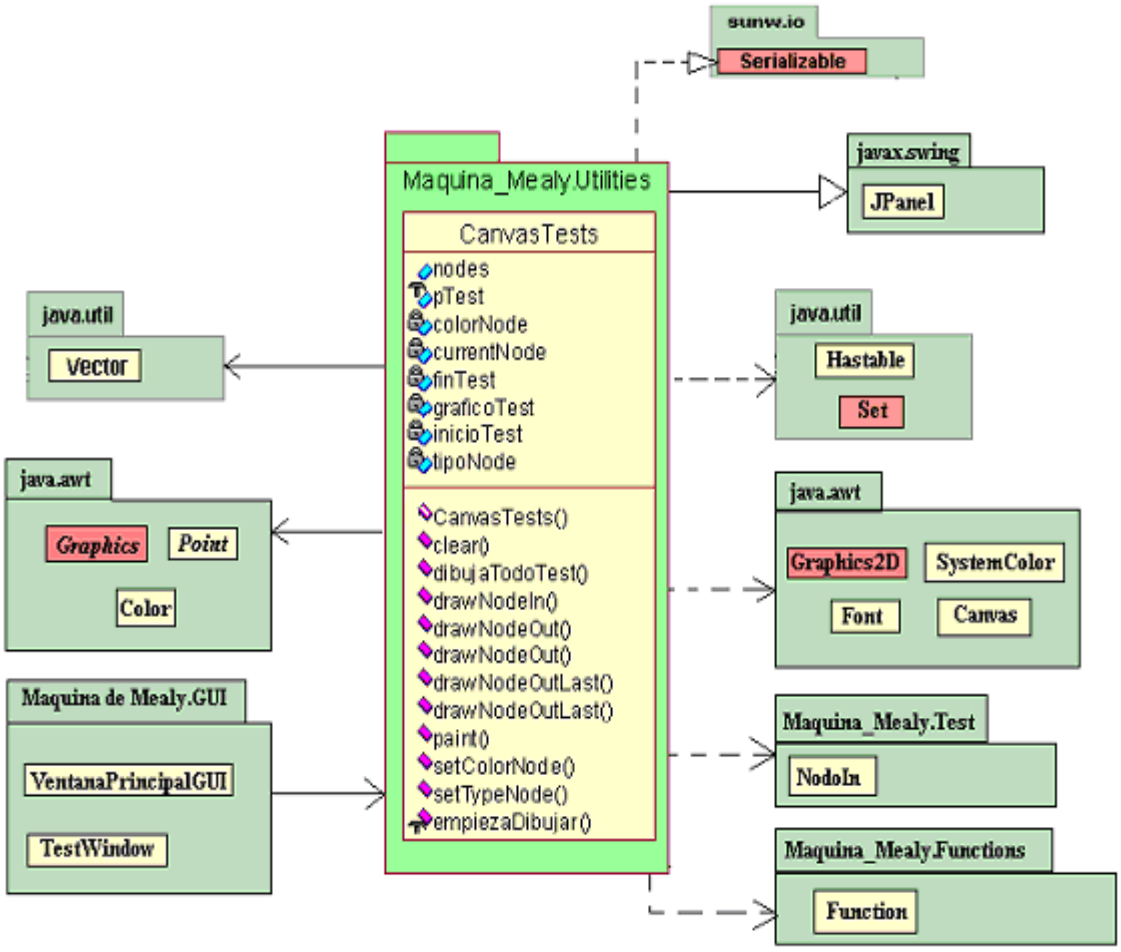
Clase "MyJPanel.java"

Esta clase implementa todos los métodos necesarios para representar los esquemas (implementación, especificación) en el panel de edición de la ventana principal.



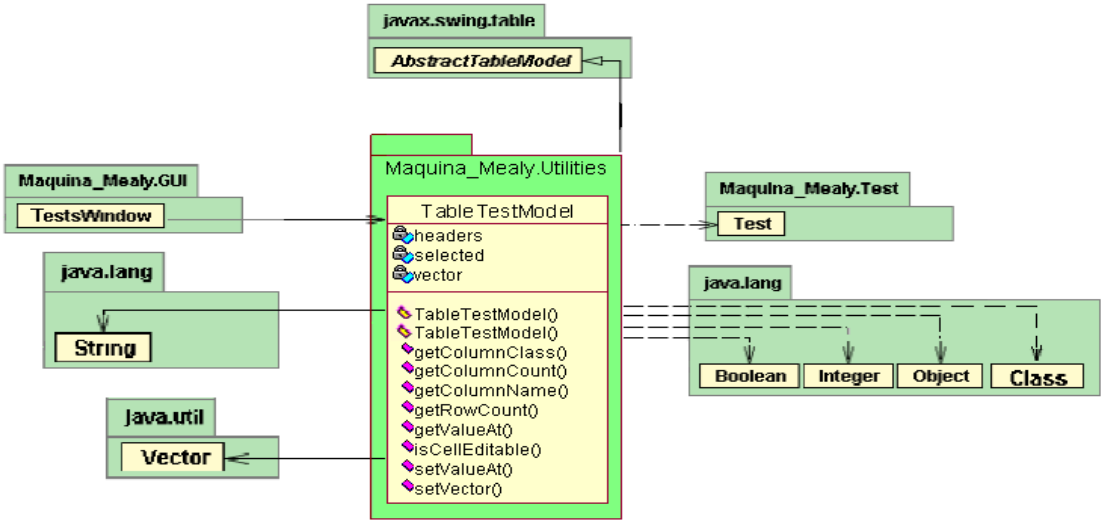
Clase "CanvasTests.java"

Esta clase implementa todos los métodos necesarios para la representación gráfica en el panel de edición asociado a la visualización de tests, de los tests que hayan sido derivados.



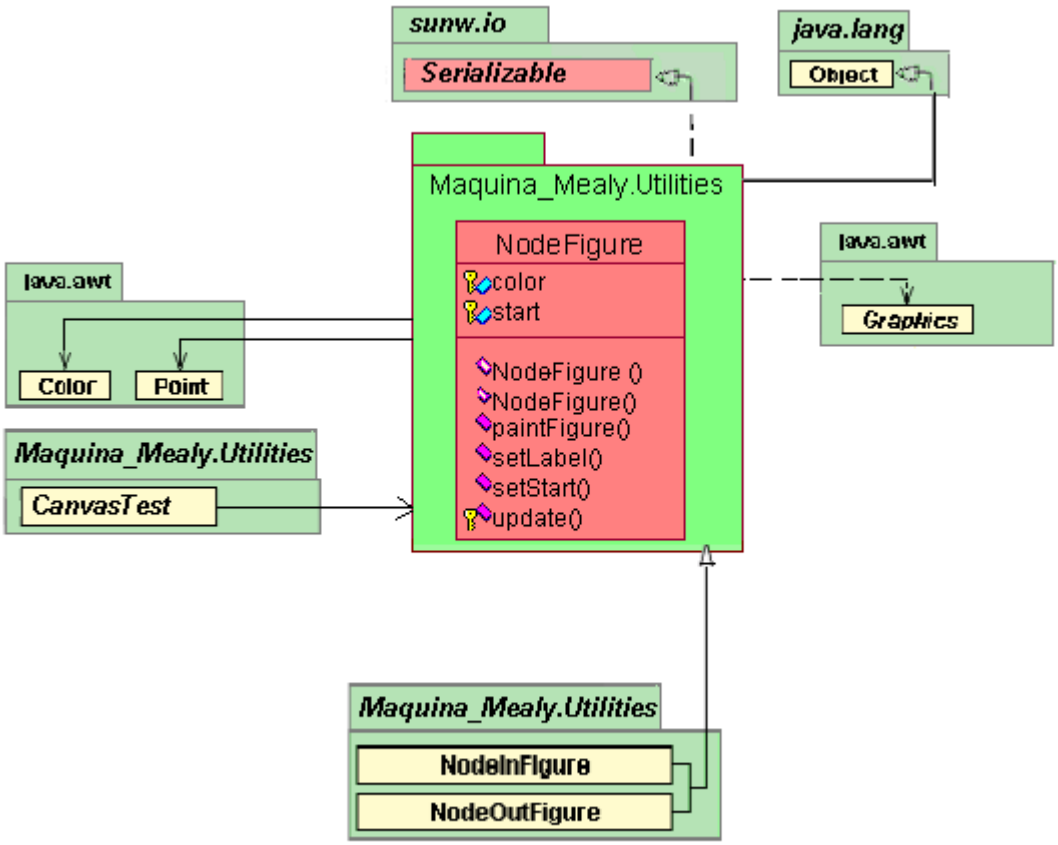
Clase "TableTestModel.java"

Esta clase implementa el modelo de tabla utilizado en la interfaz de visualización de tests, y en la que se almacenan algunas de las características de los tests que se han obtenido como resultado de la derivación.



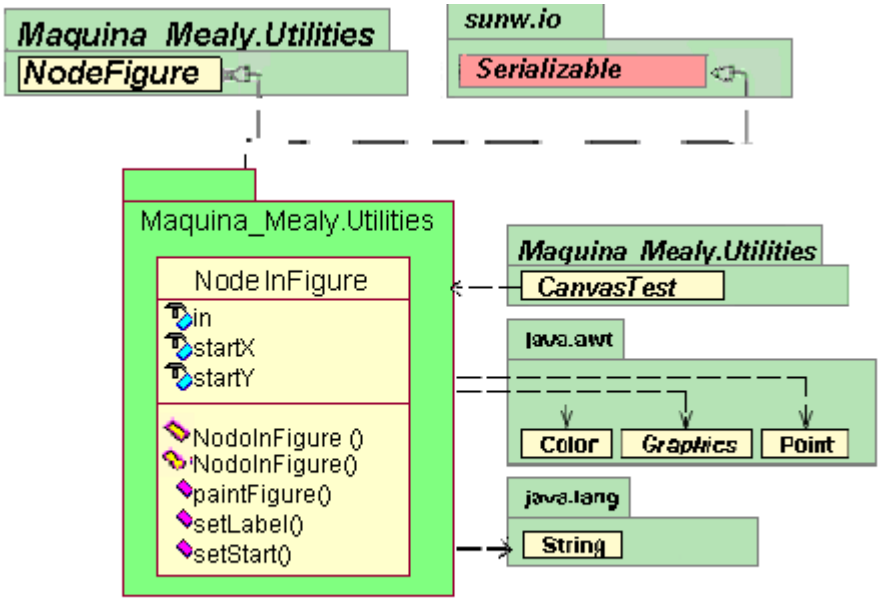
Clase "NodeFigure.java"

Esta clase es una clase abstracta que sirve como plantilla para los distintos tipos de figuras que se van a dibujar en la aplicación para la visualización de los tests, concretamente, para la creación de figuras que representen: nodos de entrada y nodos de salida.



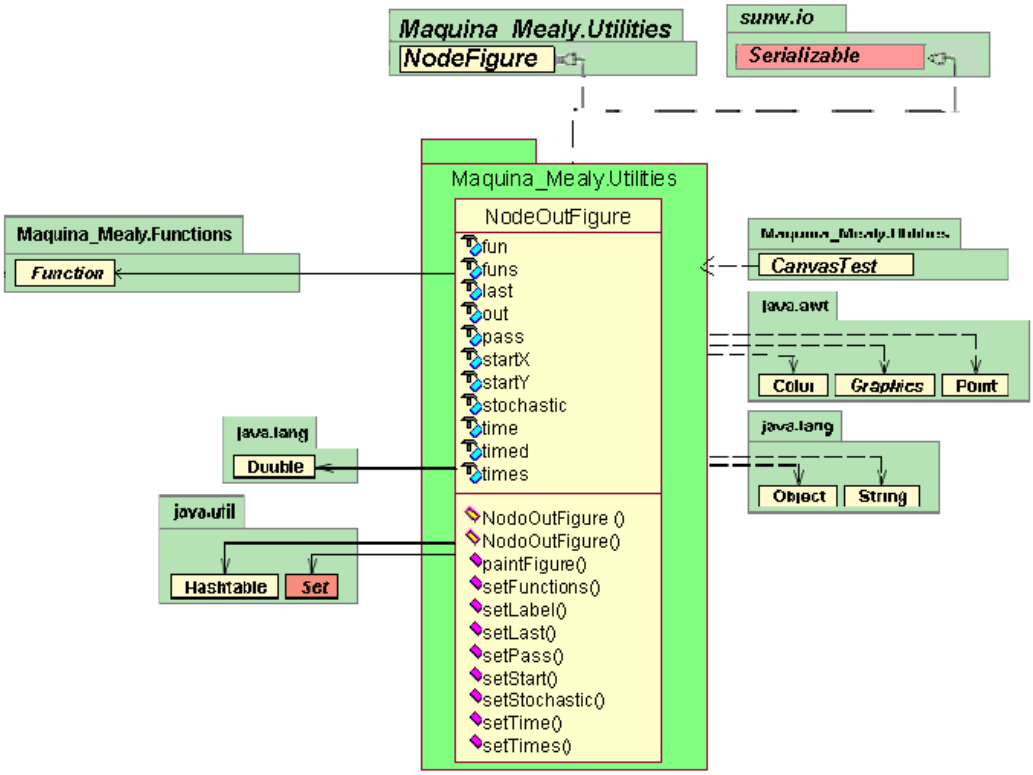
Clase "NodeInFigure.java"

Representa gráficamente los nodos de entrada en los tests.



Clase "NodeOutFigure.java"

Representa gráficamente los nodos de salida en los tests.



5.- Manual de Usuario

5.1.- Introducción

A lo largo de este manual se exponen las distintas opciones de las que consta la herramienta aquí presentada, así como los pasos necesarios a seguir para la ejecución de las mismas.

Pretende ser por tanto, una guía que permita a cualquier usuario beneficiarse de las características proporcionadas por el sistema implementado de manera sencilla, y que el manejo de la aplicación no resulte un impedimimento para lograr los objetivos buscados: testear propiedades cuantitativas.

No obstante, se ha realizado una implementación y se ha creado una estructura de clases pensando en el usuario final de la aplicación, de tal forma que de una manera rápida e intuitiva pueda manejar la herramienta y descubrir en poco tiempo y sin esfuerzo los beneficios del sistema.

5.2.- Requisitos previos necesarios

Para la ejecución de la herramienta y el correcto funcionamiento de la misma es necesario disponer previamente de:

- Una Máquina Virtual de Java (versión jdk1.4.2 o superior).

- Eclipse 7.0 o JBuilder 7 o superior: en caso de no disponer de este software puede realizarse la ejecución de la herramienta a través de un .bat, con el siguiente contenido:

5.3.- Instalación y configuración

La implementación de la aplicación así como sus componentes se encuentran almacenados en un archivo llamado “TestingTool.zip”, cuyo contenido es el siguiente:

- Implementación: contiene la implementación de la aplicación.
- Proyecto: contiene un proyecto realizado con JBuilder, con la siguiente estructura:
  - Classes: contiene las clases compiladas de la aplicación.
  - Src: contiene los fuentes (ficheros .java) de la aplicación.
  - Proyecto.jpx: permite la apertura con JBuilder de la aplicación.
  - Javadoc: contiene el Javadoc generado por la aplicación.

5.4.- Ejecución de la herramienta. Funcionamiento

A continuación se va a detallar el funcionamiento de la herramienta y cómo navegar por las distintas opciones que ésta proporciona.

La aplicación se encuentra dividida en 2 partes muy diferenciadas:

- Por un lado, se tiene un EDITOR DE ESQUEMAS que nos permite crear un nuevo esquema como máquina de Mealy ó modificar numerosas opciones gráficas.
- Por otro lado, se tiene una pantalla de DERIVACIÓN DE ESPECIFICACIONES Y APLICACIÓN DE TESTS.

La secuencia básica que un usuario realiza cuando usa el programa por primera vez sería:

1º- Crear una especificación (con las opciones que se prefieran en cada caso).

2º- Guardar la especificación.

3º- Crear una implementación (con las opciones que se prefieran en cada caso).

4º- Guardar la implementación.

5º- Abrir la especificación guardada en el paso 2.

6º- Derivar la especificación.

7º- Aplicarle la implementación guardada en el paso 4.

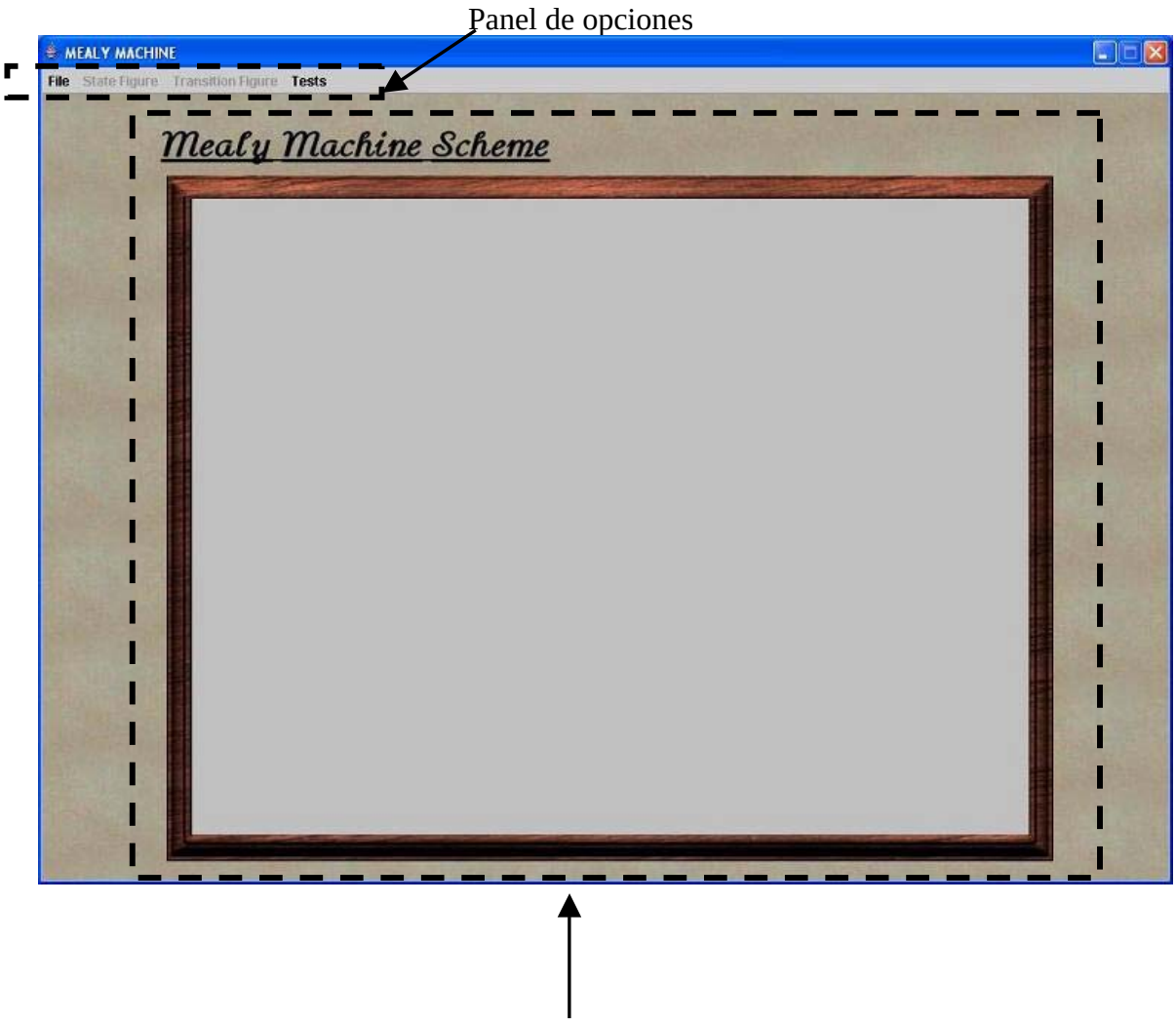
8º- Ver resultados.

No obstante, no es necesario que se realicen todos los pasos en este orden y la herramienta permite usar especificaciones, implementaciones y tests guardados, de modo que puedan obviarse determinados pasos.

Tras la ejecución del programa, aparece en pantalla la siguiente imagen que se corresponde con el EDITOR DE ESQUEMAS, por lo que será el primero que pasemos a detallar.

5.4.1.-Editor de esquemas.

El usuario puede querer crear un esquema propio, abrir uno existente o modificarlo. Para cualquiera de estas opciones, debe situarse sobre la ventana del editor de esquemas que ofrece toda esta funcionalidad.



En la parte superior de la pantalla aparece el menú de acciones de esta pantalla.

Inicialmente las únicas opciones habilitadas en el menú File son

- crear una nueva especificación (Véase el apartado 5.4.1.1 “Crear una especificación”),
- abrir una especificación o una implementación existentes (Véase el apartado 5.4.1.3)
- salir del programa.

En el menú Test se permite abrir un test que ha sido guardado anteriormente por si directamente el usuario desea pasar a la pantalla de derivación y aplicación de tests. Los menús correspondientes a los estados y las

transiciones están deshabilitados, ya que primero tenemos que indicar al programa que tipo de esquema (implementación o especificación) queremos crear (Véase el apartado 5.1.4.1- “Crear una especificación”).

Una vez cargada una especificación, el usuario puede modificar el esquema que este creando a su gusto. Véanse los apartados 5.4.1.1,5.4.1.2

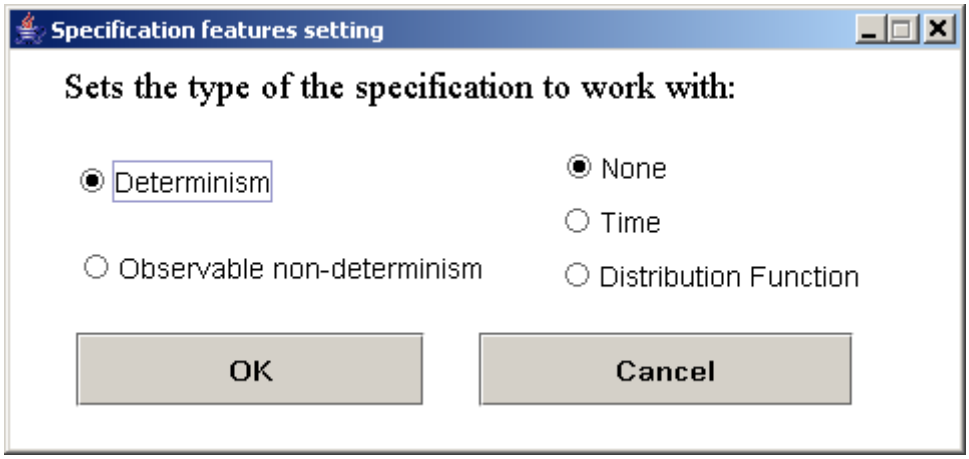
5.4.1.1.- Crear una especificación ó implementación

En este apartado se va a explicar cuáles son los pasos necesarios para la creación de una nueva especificación o implementación.
Como se verá más adelante, la diferencia entre “implementación” y “especificación” es puramente conceptual a la hora de representarlas sobre el editor de esquemas, se verá que ambas se representan del mismo modo y es prácticamente imposible deducir de un esquema ya dado si es una implementación o una especificación a simple vista.
La manera que el usuario tendrá de diferenciar ambos tipos de esquema, es la extensión del archivo en el que se guarda la representación (.impl para implementaciones o .espec para especificaciones).

Nuestra herramienta está diseñada para distintos marcos formales de actuación, y ofrece por tanto distintas opciones en la creación, permitiendo seleccionar entre distintas características en función del objetivo que se pretenda obtener y del estudio que se pretenda realizar.
De esta forma, hay que tener en cuenta que la posterior navegación por la aplicación dependerá del tipo de especificación que se haya decidido implementar.

Para crear un nuevo esquema, se seleccionará FILE->New como sigue:

Tras esto, aparecerá una ventana nueva (Specification Features Setting) como la siguiente:



En esta ventana seleccionamos el tipo de esquema que vamos a realizar, que puede ser uno de los siguientes:

- ❖ Determinista.
- ❖ No determinista.

Y dentro de cada uno de ellos, tenemos:

- ❖ Con tiempo.
- ❖ Con función de distribución.
- ❖ Sin características adicionales.

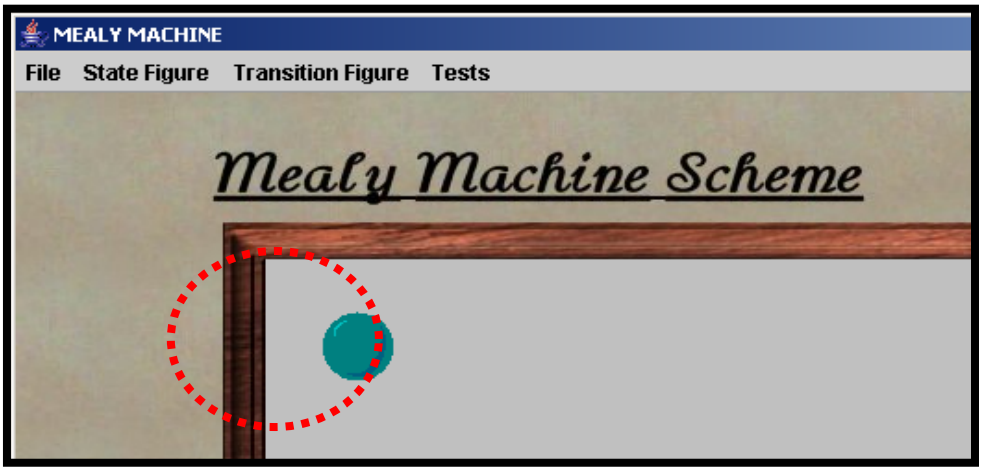
Una vez que se hayan concluido estos pasos, se volverá a la ventana principal de la aplicación en la que puede observarse que se ha habilitado el menú “State Figure”.

Podremos pasar entonces a añadir los estados y transiciones que formarán parte del esquema recién creado.(Véase apartados 5.4.1.1.1 y 5.4.1.1.2)

5.4.1.1.1- Crear estados

Para añadir estados al esquema, basta con seleccionar del menú STATE FIGURE->Add State.

Tras lo cual aparecerá un círculo verde en el editor de esquemas, que representará el estado.

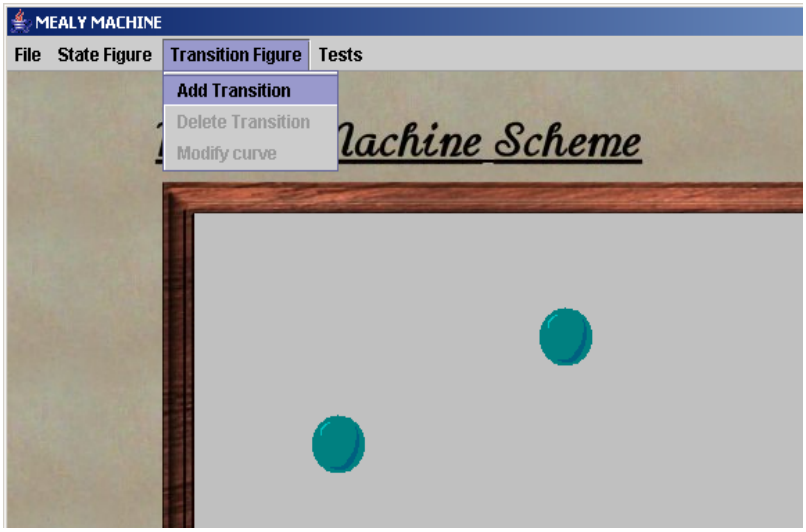


Existe la posibilidad de desplazar dicha figura por el panel para obtener así una distribución de los estados creados que permitan una visualización más clara de la especificación creada por parte del usuario. Para ello basta con mantener pulsado el botón izquierdo del ratón sobre la figura y arrastrarlo hasta la posición deseada.

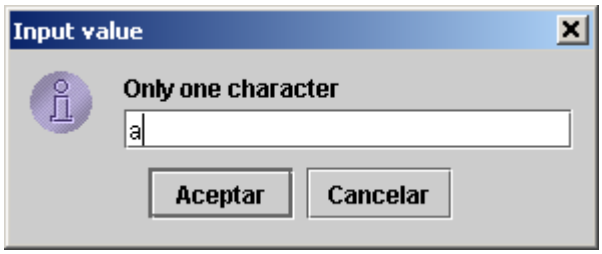
Una vez que se ha añadido un estado a la especificación, se habilitará el menú “Transition Options” que permitirá la creación de transiciones (Véase apartado 5.4.1.1.2).

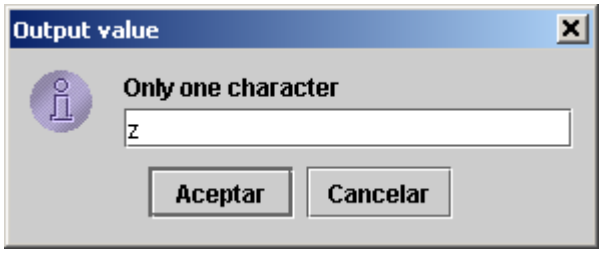
5.4.1.1.2.- Crear transiciones

Para añadir una transición al esquema, basta con seleccionar del menú TRANSITION_FIGURE->Add Transition



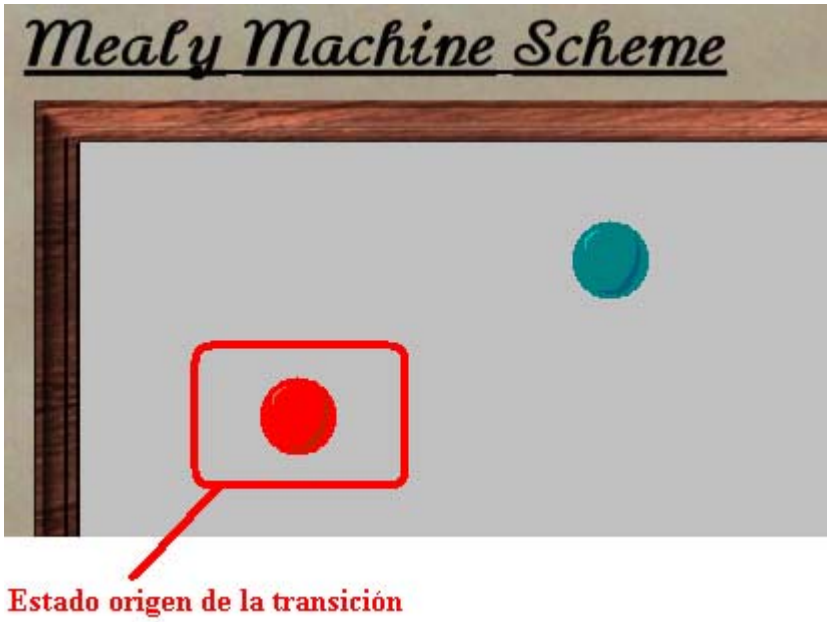
Para el esquema que estamos creando, se pedirá al usuario que se introduzcan los valores para el carácter de entrada y de salida de la transición (sólo un carácter está permitido). Si el carácter introducido para la input está ya definido como output o viceversa saldrá una ventana informando del error.



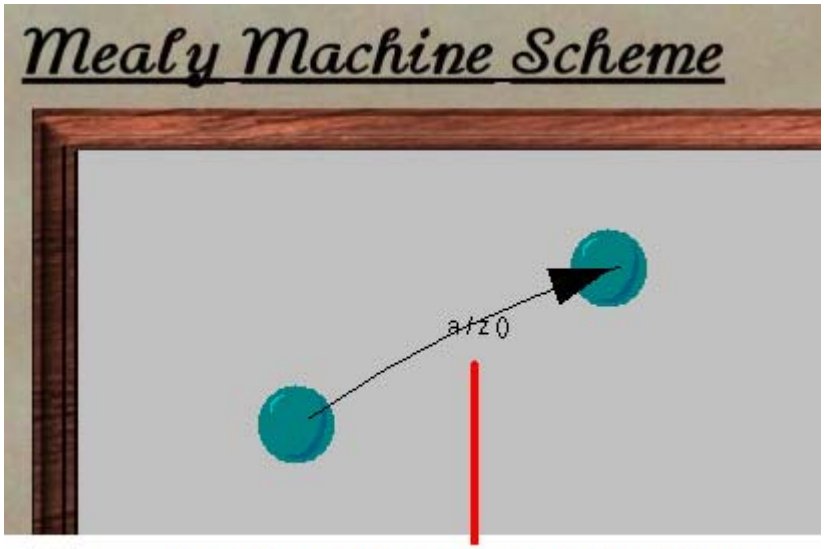


Una vez introducidos dichos parámetros se tendrá que especificar cuáles van a ser los estados origen y destino de la transición.

Para seleccionar ambos bastará con hacer click sobre los estados que se deseen de entre los que han sido ya añadidos a la especificación. Al seleccionar el estado origen de la transición, el color de éste cambia a rojo para indicar su condición de estado origen.



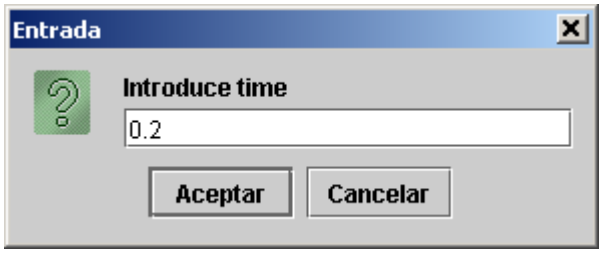
Seleccionamos el estado destino de la transición y se nos muestra la nueva transición que habíamos especificado. Además, el estado origen de la transición recupera el color original.



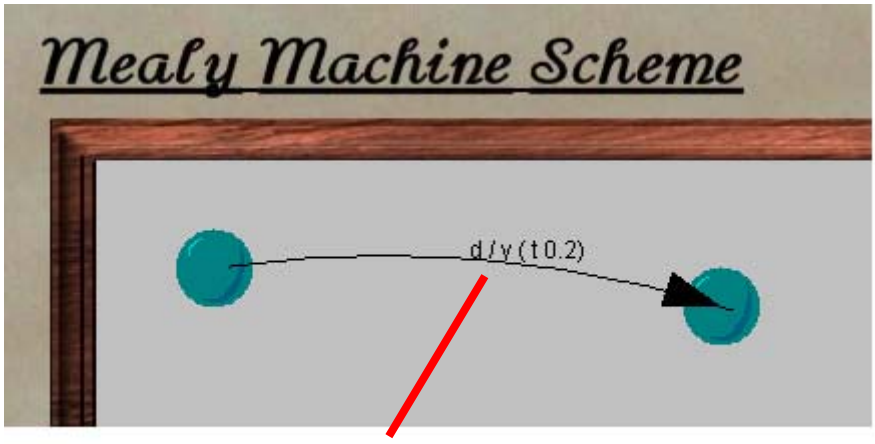
Nueva transición de entrada "a" y salida "z"

Si se eligió un esquema temporal:

Tras introducir los caracteres de entrada y salida de la transición, aparecerá una nueva ventana para que se introduzca el tiempo que se desea asociar a esa transición.



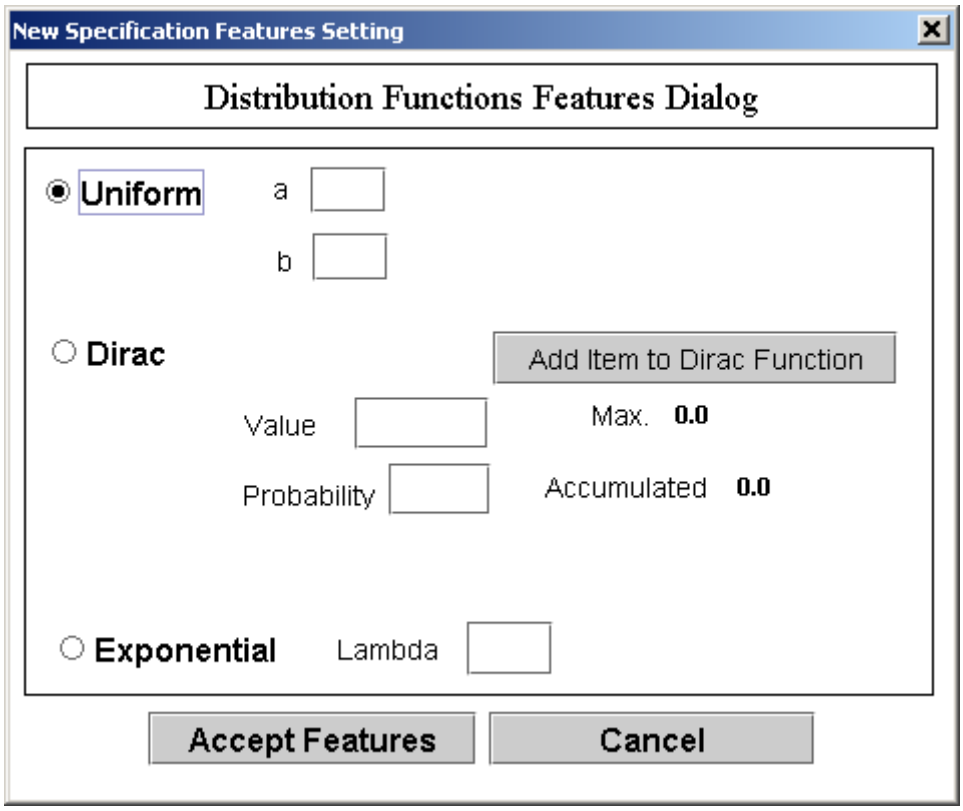
Y gráficamente la representación será:



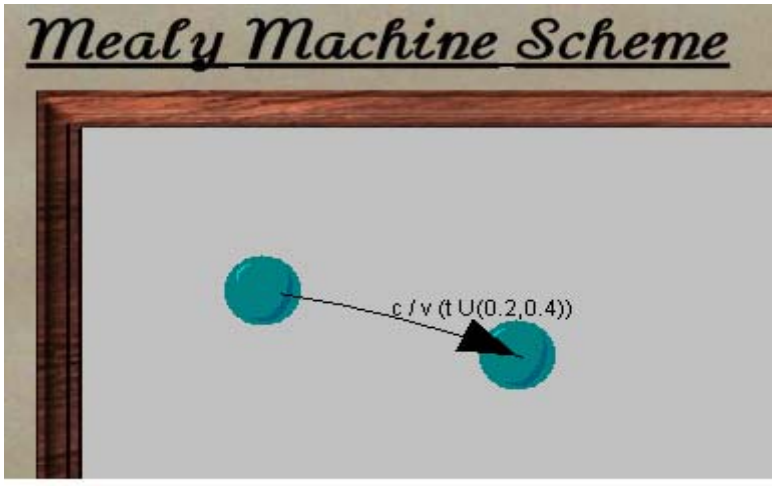
Transicion de entrada "d"y salida "y", que consume 0.2 de tiempo

Si se eligió un esquema con funciones de distribución asociadas:

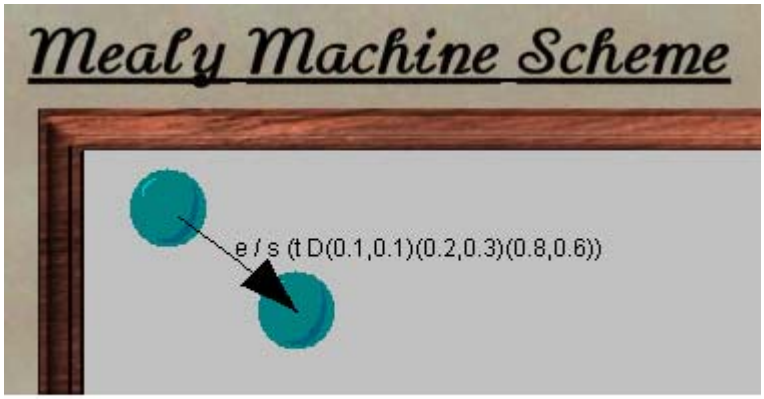
Tras introducir los caracteres de entrada y salida, aparecerá una nueva ventana para que se introduzcan los parámetros de la distribución que queremos asociar a la transición.



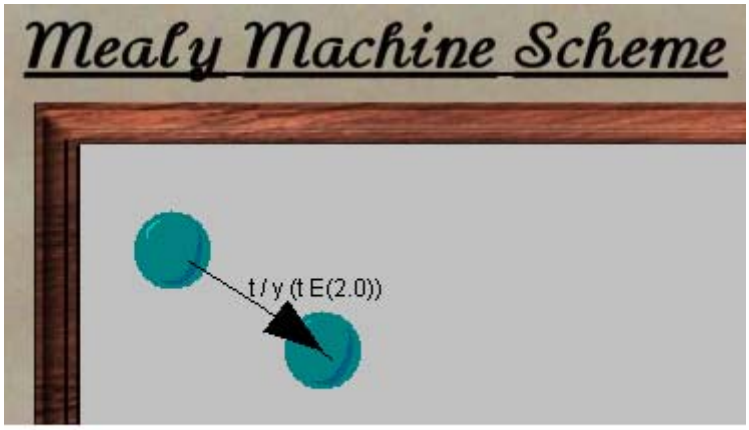
El usuario puede elegir entre asociar una función uniforme, una función de Dirac o una exponencial a la transición que se está creando. En cualquiera de los casos, se deben introducir parámetros correctos de acuerdo a las especificaciones de estas funciones.



Transición asociada a funcion Uniforme



Transición asociada a una función delta de Dirac

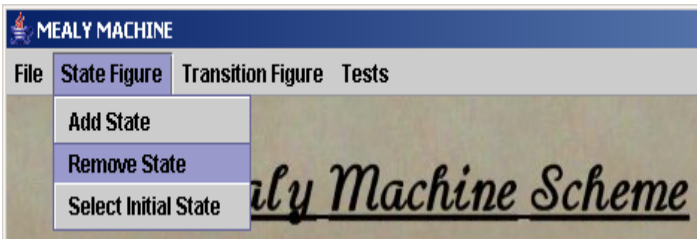


Transición asociada a una exponencial

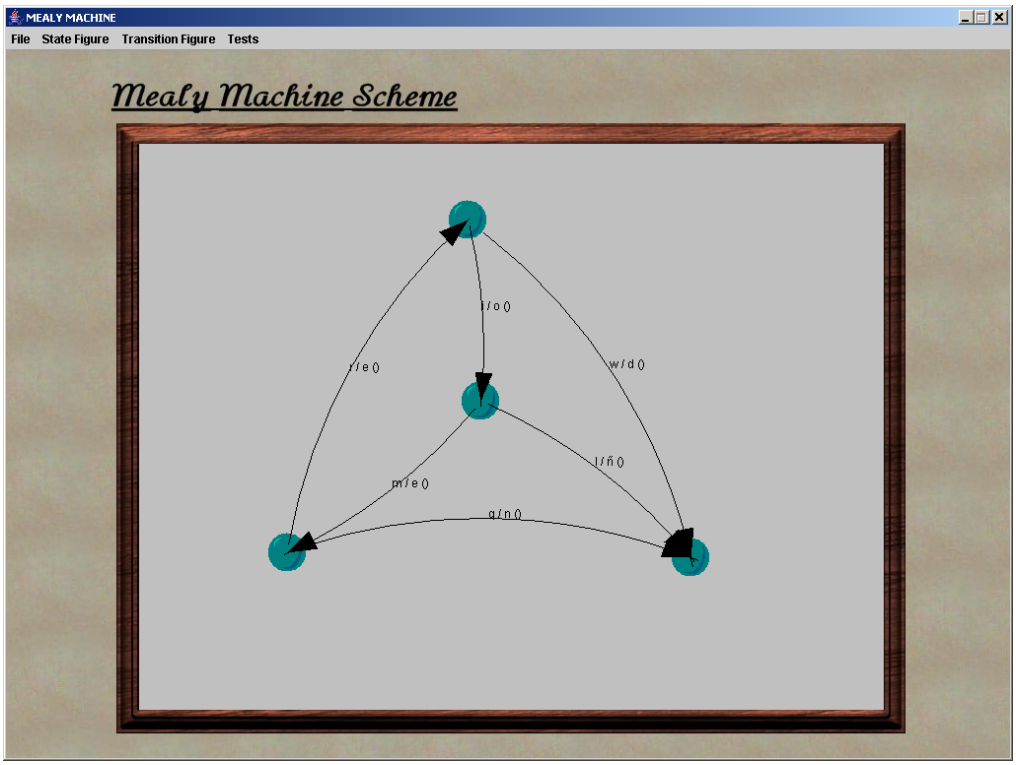
5.4.1.1.3- Eliminar estados

Durante la implementación de la herramienta se ha tenido en cuenta la posibilidad de que el usuario desee eliminar alguno de los estados previamente creados, bien porque hayan sido añadidos por equivocación o bien porque se quiera realizar modificaciones sobre especificaciones o implementaciones existentes para nuevas pruebas.

Para llevar a cabo esta acción, se selecciona en el menú STATE_FIGURE>Remove_State.

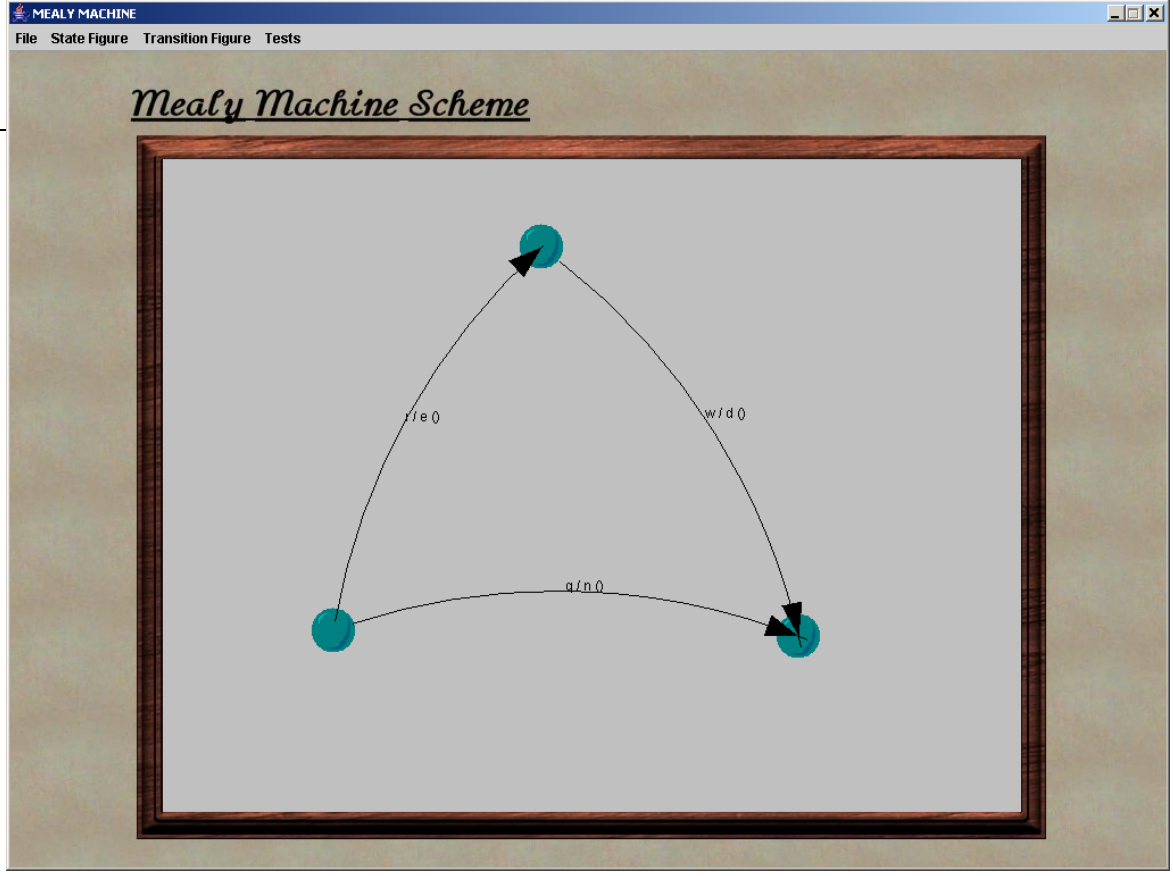


A continuación se deberá pulsar sobre el estado que se desea eliminar. Hay que tener en cuenta que la eliminación de un estado supone la eliminación de las transiciones que entran o salen de estado.



Esquema determinista del que deseamos eliminar el estado central

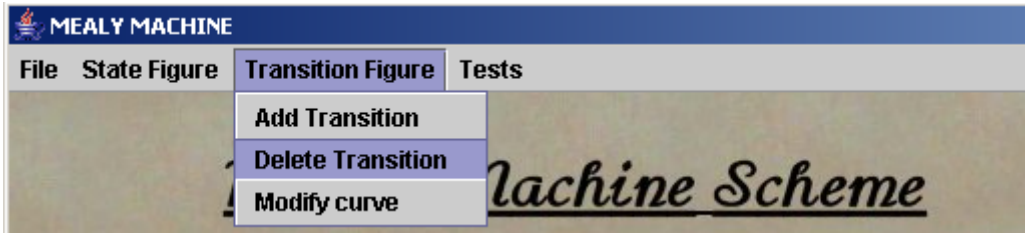
La eliminación de estados es análoga para todos los tipos de esquema.



Esquema determinista del que hemos eliminado el estado central

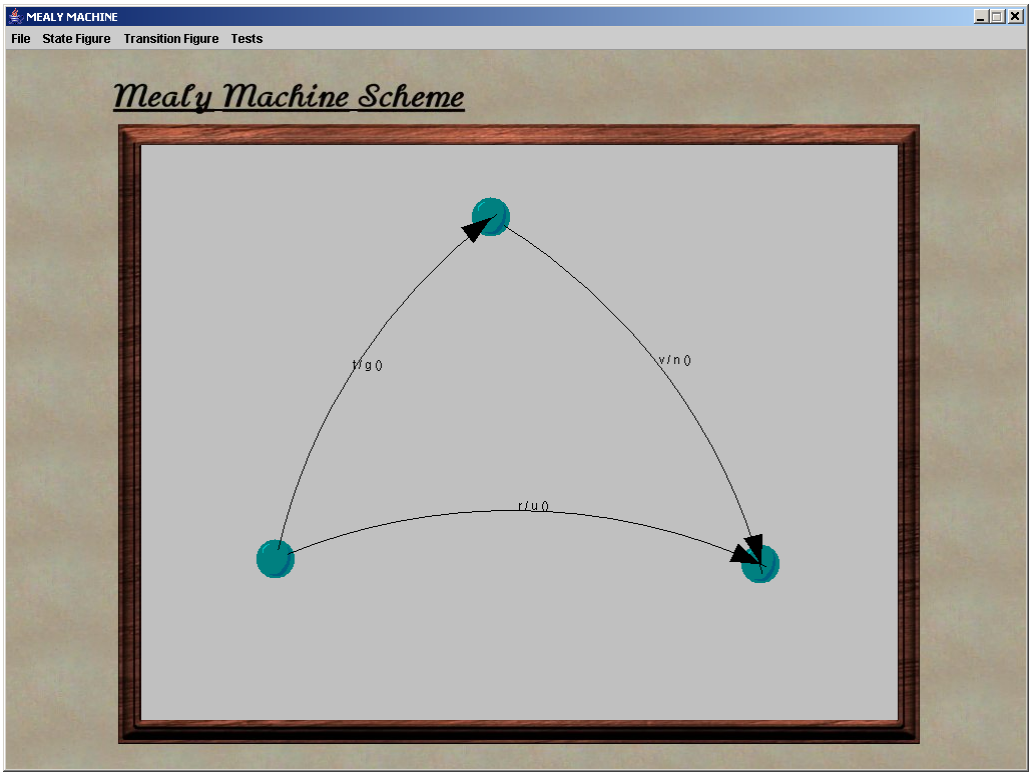
5.4.1.1.4.- Eliminar transiciones

Para eliminar transiciones, seleccionamos del menú la opción TRANSITION_FIGURE>Delete_Transition



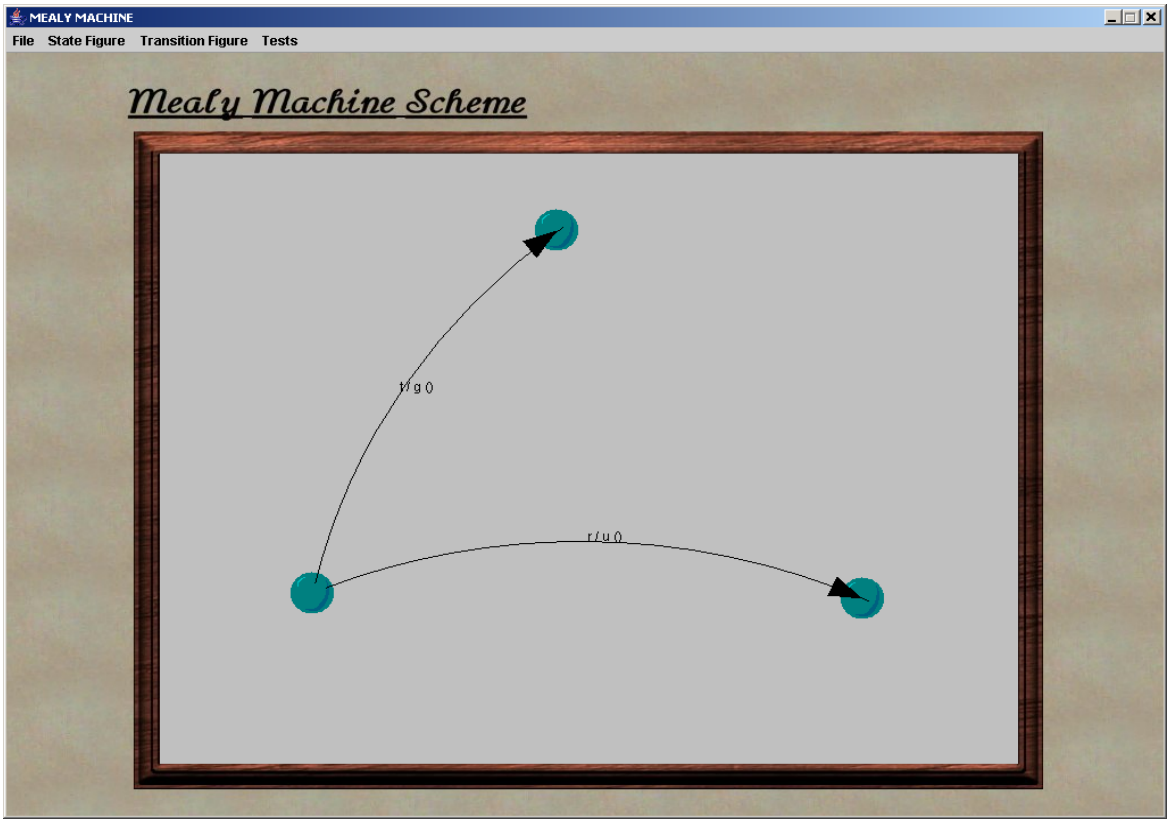
Se seleccionará la transición haciendo click sobre el nombre de ésta. Cada vez que se desee eliminar una transición, debe realizarse el proceso completo aquí descrito.

Se muestra a continuación un ejemplo de eliminación de transiciones.



Esquema del que deseamos eliminar la transición de la derecha.

Esquema tras la eliminación

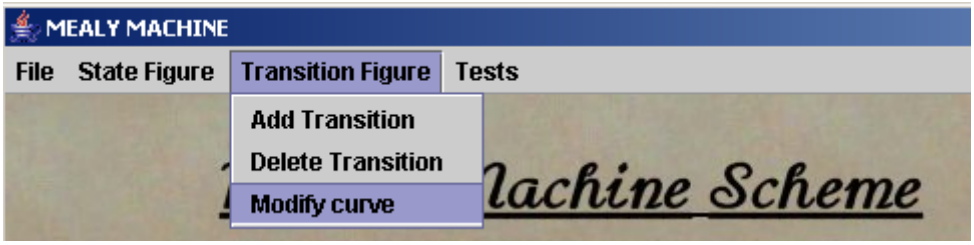


5.4.1.1.5.- Modificar curvatura

A lo largo de la realización de la herramienta se ha tenido en cuenta la posibilidad de querer crear transiciones que tengan en común un mismo estado como origen. De hecho, es una situación normal y la mayoría de este tipo de especificaciones funciona de esta manera.

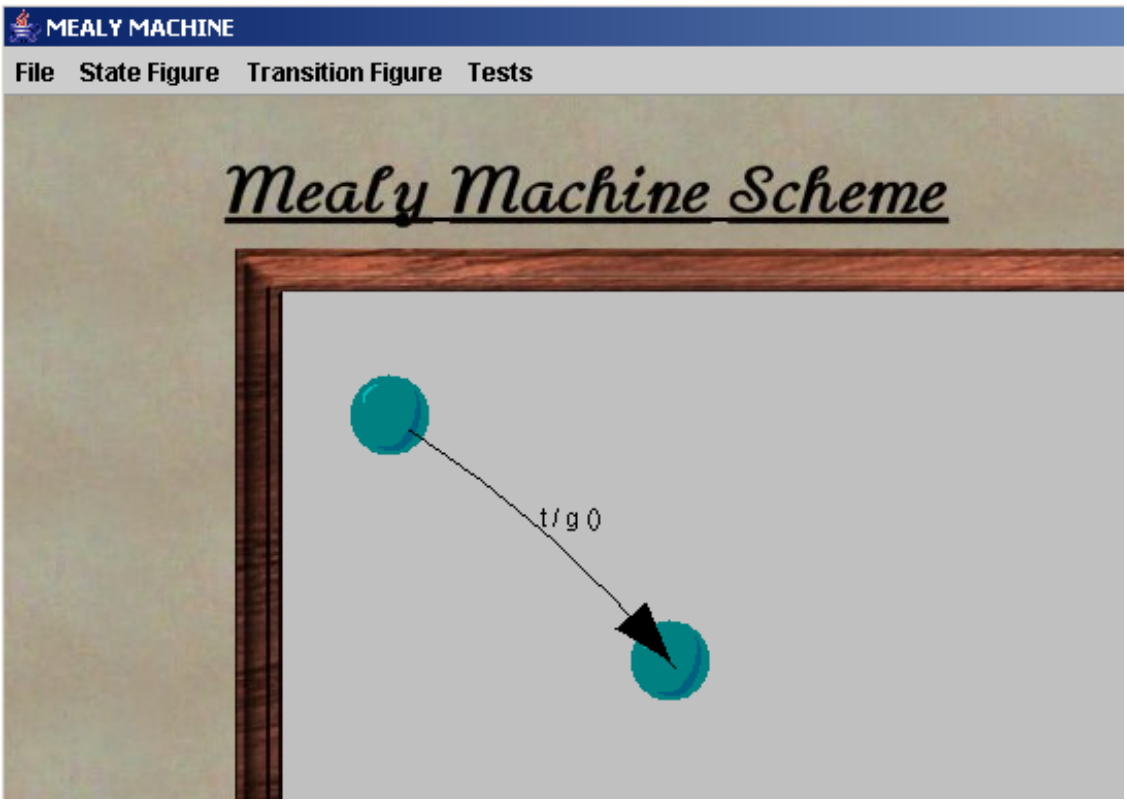
Por esta razón, se ha implementado una solución para permitir que la visualización de la especificación creada resulte lo más clara posible de cara al usuario. Dicha solución no es otra que permitir modificar la curvatura de las flechas que gráficamente representan a las transiciones.

Para modificar la curvatura, seleccionamos en el menú TRANSITION_FIGURE>Modify_curve.

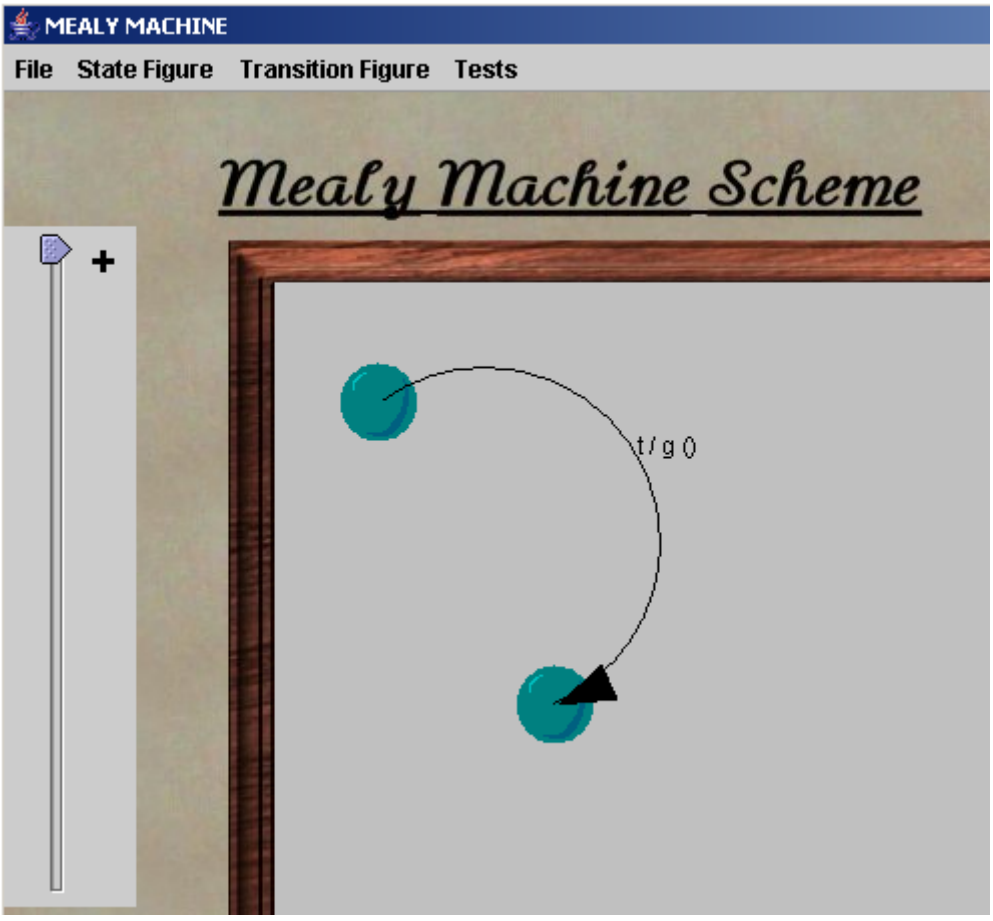


Se deberá seleccionar entonces la transición cuya curvatura se quiere modificar y mediante el “Slide” que aparecerá a la izquierda, fijar la curvatura que se desee. Podrá verse cómo la transición va modificándose en función del movimiento del citado Slide.

Una muestra de modificación de estados se muestra a continuación:



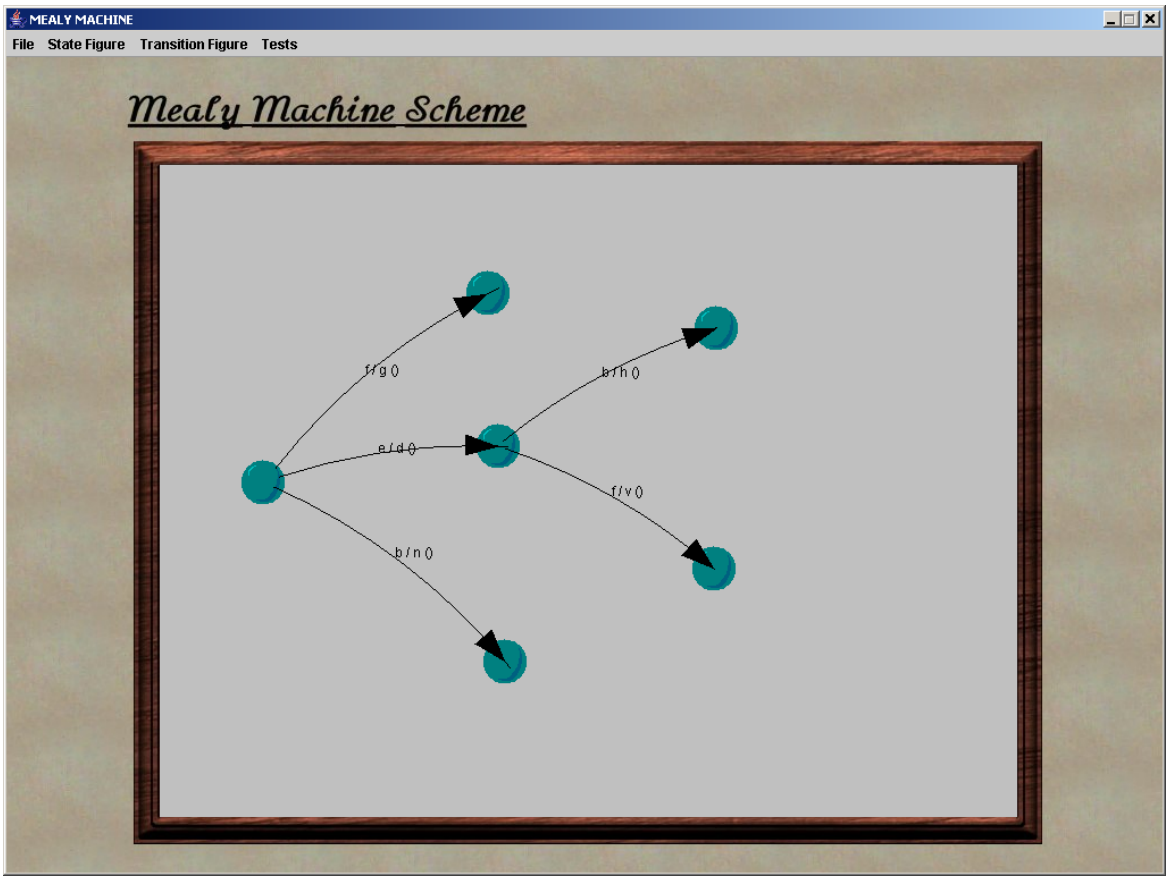
Transición a la que se desea modificar la curvatura



Transición con la curvatura modificada

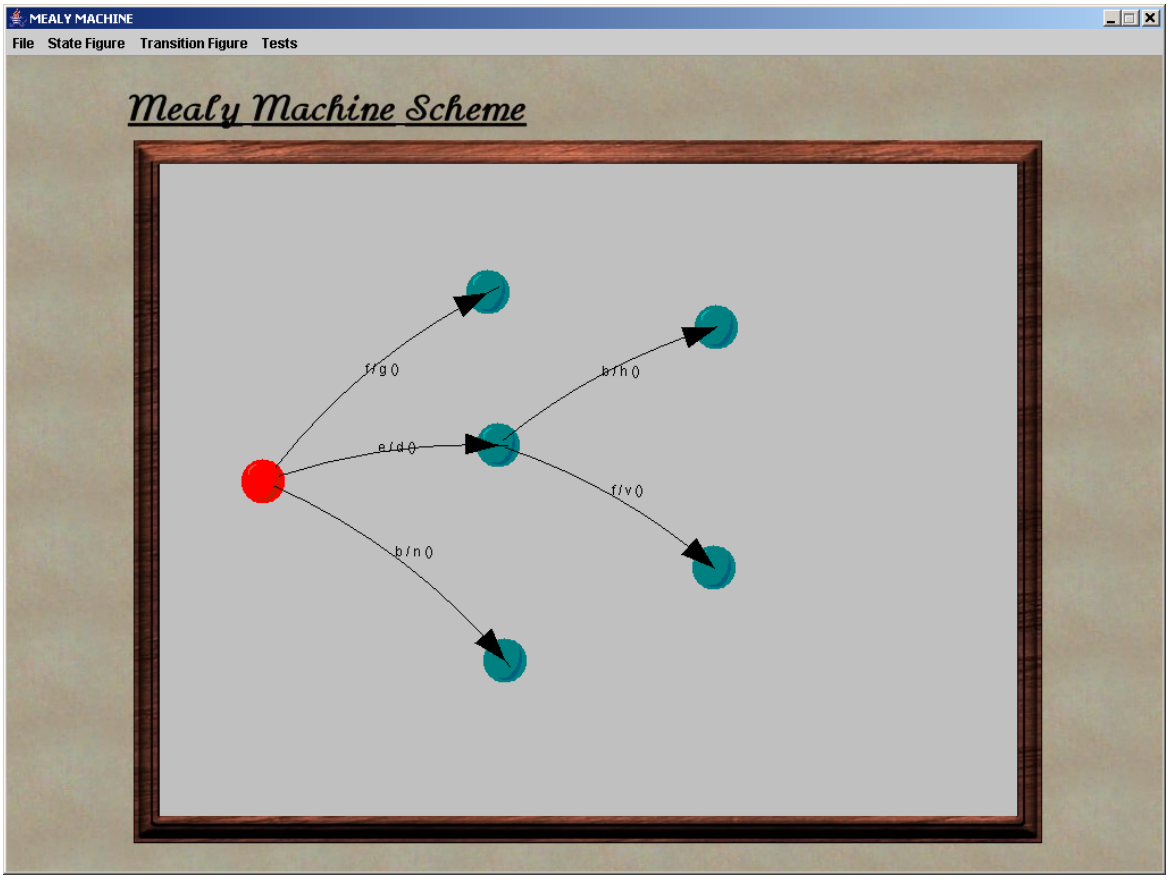
5.4.1.1.6.- Seleccionar estado inicial

Si se han seguido todos los pasos hasta ahora explicados se ha disponer ya de una especificación o implementación consistente en una serie de estados y sus respectivas transiciones. Un ejemplo de una especificación creada se muestra en la siguiente imagen:



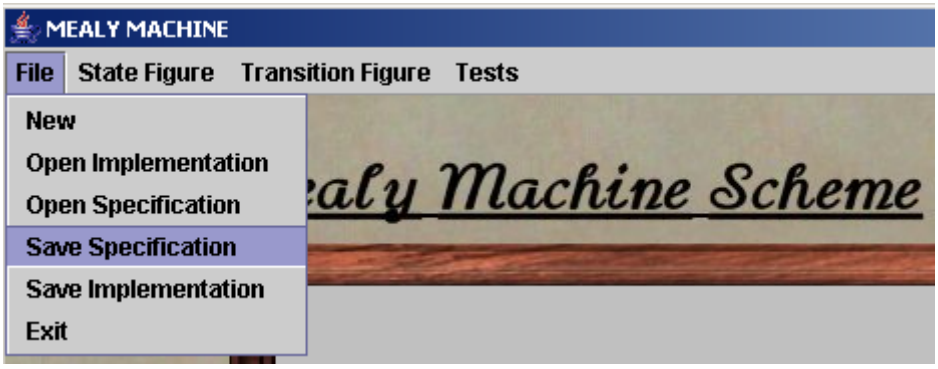
Ahora bien, si lo que se desea a continuación es realizar una derivación de la especificación diseñada, es requisito imprescindible la selección de un estado inicial. Para ello bastará con seleccionar del menú STATE_FIGURE>Select_Initial_State.

Se deberá seleccionar a continuación, de entre los estados existentes, aquel que va a desempeñar el papel de estado inicial. Bastará con pulsar sobre el estado elegido para desempeñar dicha función. El color del estado seleccionado cambiará a rojo como indicativo de su rol, tal y como se muestra a continuación:



5.4.1.2.- Guardar una implementación o especificación.

Para guardar el esquema construido en un archivo, tenemos la opción de guardarlo como implementación o como especificación. Se guardará como un archivo .IMPL (para implementaciones) o tipo .ESPEC (para especificaciones).

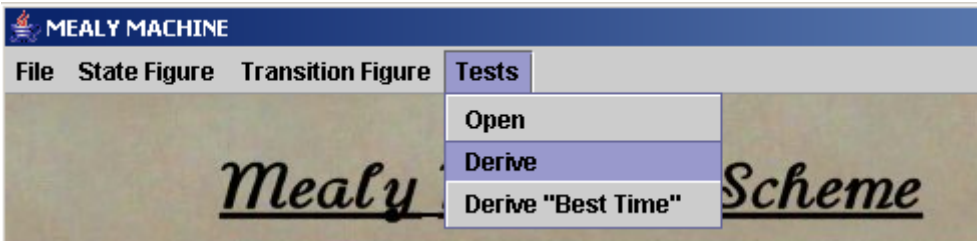


5.4.1.3.- Abrir una implementación o especificación.

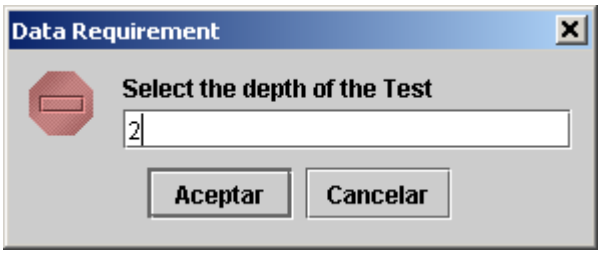
El programa permite cargar en el editor de esquemas un esquema que se haya guardado previamente, mediante FILE->Open_Implementation y FILE->Open_Specification.

5.4.1.4- Derivar test.

Una vez creada la especificación, el siguiente paso lógico, es su derivación y posterior visualización de los tests resultantes de dicha derivación. Para mostrar estos tests, basta con seleccionar del menú TESTS->Derive.



Se pedirá entonces al usuario la introducción de la profundidad máxima de los tests derivados, tal y como se muestra a continuación:

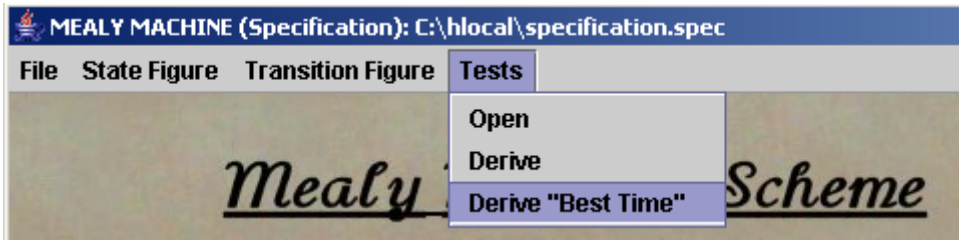


Tras la petición anterior se procederá a la visualización de la ventana DERIVACIÓN DE ESPECIFICACIONES Y APLICACIÓN DE TESTS que nos servirá de interfaz para la aplicación y derivación de tests (Véase sección: 5.4.2).

5.4.1.5- Derivar test con la opción BEST TIME

Cuando el usuario ha creado una especificación no determinista cuyas transiciones tienen asociadas propiedades de tiempo fijo, tiene la opción de derivar este esquema y aplicarle una implementación (Véase apartado 5.4.2.1- Aplicar test a Implementación) mediante el método “Best Time” que consiste, en caso de tener varias transiciones desde un mismo estado en la implementación, en elegir la de menor tiempo posible para ver si satisface el test considerado en cada caso.

Si el usuario quiere activar esta opción no debe derivar de la manera explicada en el apartado 5.4.1.4, sino seleccionando en el menú TESTS->Derive_”Best_Time”:



Una vez seguidos estos pasos, aparecerá la ventana de derivación de especificaciones y aplicación de tests, del mismo modo que si hubiéramos usado la opción de derivar convencional.

5.4.2- Derivación de especificaciones y aplicación de los tests.

Una vez que hemos llegado a la pantalla de visualización, veremos una pantalla similar a esta:

La pantalla esta dividida en tres zonas:

- Zona de identificación de tests: Cada fila es uno de los posibles tests resultado de la derivación y queda identificada por un número que se corresponde con el orden que ocupa en la zona de visualización de tests. Además, dicho test tiene su profundidad y una casilla que permitirá seleccionarlo o no a la hora de aplicarlo a la implementación (Véase apartado 5.4.2.1.-Aplicar test a implementación)
- Zona de visualización de tests: Cada fila representa un posible test gráficamente, indicando las entradas y salidas que se van obteniendo en la derivación.

- Zona de resultados de los tests: Muestra el resultado de aplicar los tests a una implementación que previamente hemos guardado.
- Esta pantalla nos permite seleccionar diferentes opciones que veremos a continuación.

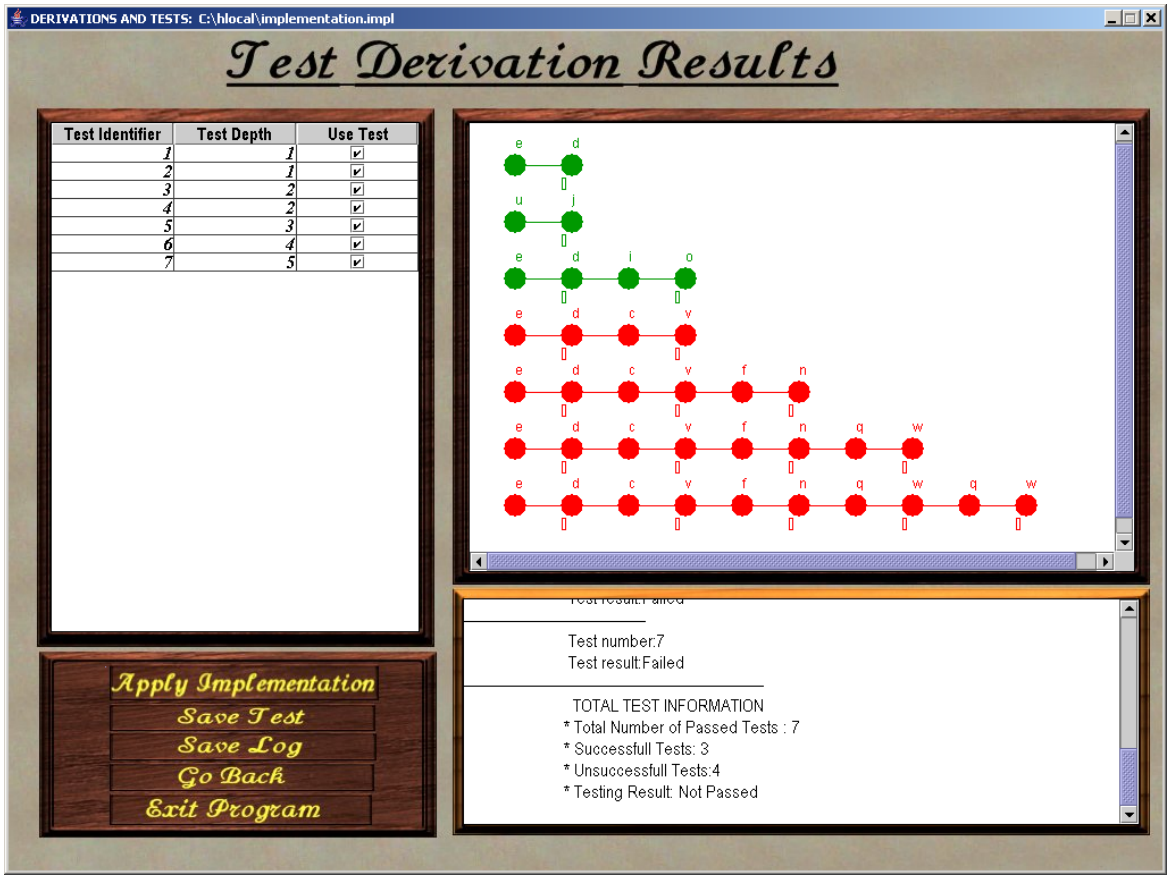
5.4.2.1.- Aplicar test a implementación

El objetivo final de la creación de especificaciones y sus correspondientes derivaciones de tests, no es más que la comprobación de si la implementación creada es conforme con la especificación diseñada.

Para aplicar un test a una implementación guardada, pulsamos sobre el botón APPLY IMPLEMENTATION.

Nos aparecerá una ventana para seleccionar la implementación que queremos usar con las derivaciones que ya tenemos hechas. Es importante darse cuenta de que debe seleccionarse una implementación, ya que conceptualmente es el esquema que permite que se le apliquen los tests.

Como resultado de la aplicación de los tests, obtenemos una pantalla similar a la siguiente:



Se puede ver que la representación gráfica de las derivaciones se colorea de acuerdo a si ese test se ha pasado o no en la implementación seleccionada (verde si el test se pasó y rojo si no se pasó).

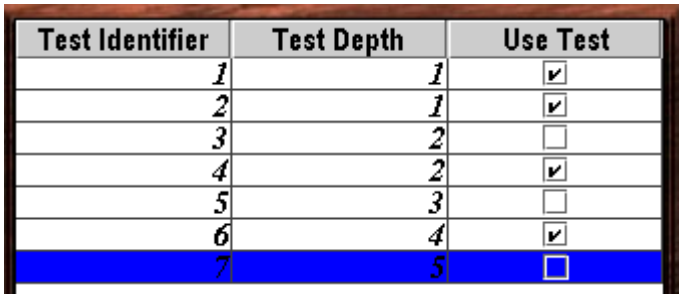
Para más detalle acerca de qué tests se han pasado y el resultado de cada uno, puede consultarse la “zona de resultados de los tests”.

5.4.2.2.- Salvado de tests y log

Puede resultar interesante guardar los tests derivados correspondientes a una especificación concreta así como el resultado obtenido tras su aplicación a la implementación correspondiente.

- Si se quiere guardar los tests derivados, actuamos como se indica a continuación:

Seleccionamos qué tests queremos guardar, indicándolo en la casilla correspondiente, como indica la siguiente imagen:



Test Identifier	Test Depth	Use Test
1	1	☑
2	1	☑
3	2	☐
4	2	☑
5	3	☐
6	4	☑
7	5	☐

En esta imagen se han seleccionado los tests 1,2,4 y 5 para guardar.

Se pulsa sobre el botón SAVE TEST, donde se nos permite guardar los tests como un archivo de extensión .TESTS.

Posteriormente, podremos abrir este archivo directamente desde la ventana de edición de esquemas, sin necesidad de realizar la derivación completa.

- Si se quiere guardar un log del resultado de la aplicación de los tests sobre la implementación, podemos pulsar sobre el botón SAVE LOG, que nos guardará la información que aparece en la "zona de resultados de los tests" en un fichero de texto, con extensión .LOG.

6.- Bibliografía

- **Artículo "Towards Testing Stochastic Timed Systems"**

  Manuel Núñez and Ismael Rodríguez
  Dept. Sistemas Informáticos y Programación, Facultad de Informática, Universidad Complutense de Madrid, E-28040 Madrid, Spain,
  {mn,isrodrig}@sip.ucm.es

- **Artículo "Conformance Testing Relations for Timed Systems"**

  Manuel Núñez and Ismael Rodríguez
  Dept. Sistemas Informáticos y Programación, Facultad de Informática, Universidad Complutense de Madrid, E-28040 Madrid, Spain,
  {mn,isrodrig}@sip.ucm.es